МЕЖГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКО-АРМЯНСКИЙ УНИВЕРСИТЕТ

На правах рукописи

Гукасян Цолак Гукасович

МЕТОДЫ И ПРОГРАММНЫЕ СРЕДСТВА ДЛЯ ВЫЯВЛЕНИЯ ЗАИМСТВОВАНИЙ В ТЕКСТАХ НА АРМЯНСКОМ ЯЗЫКЕ

Специальность 05.13.11 —
«Математическое и программное обеспечение вычислительных машин, комплексов и компьютерных сетей»

Диссертация на соискание учёной степени
кандидата технических наук

Научный руководитель:
к.ф.-м.н.
Турдаков Денис Юрьевич

Москва — 2021

Оглавление

Введение

Определение степени уникальности работ является одной из самых серьезных проблем в научных исследованиях. Неуникальные, заимствованные работы (заимствованием считается как правильно процитированный текст, так и текст без указания оригинального автора), которые остаются нераскрытыми, могут иметь серьезные негативные последствия по нескольким причинам.

Заимствованные исследовательские работы препятствует научному процессу, например, искажая механизмы отслеживания и исправления результатов [1]. Если исследователи расширят или пересмотрят более ранние результаты в последующих исследованиях, то статьи, содержащие заимствования из исходной статьи, останутся неизменными. Неправильные результаты могут распространиться и повлиять на последующие исследования или практическое применение [2]. Исследования показывают, что некоторые частично или полностью заимствованные работы цитируются по крайней мере так же часто, как и оригинал. Это проблематично, поскольку число цитирований является широко используемым показателем эффективности исследований, например, для принятия решений о финансировании или найме. Отсутствие надежных механизмов выявления и предотвращения случаев карьерного продвижения путем сплагиаченного труда может привести к кризисным ситуациям в различных отраслях общественной жизни (в образовательной [1] [2] и судебной[3] системах, например). С образовательной точки зрения заимствования наносят ущерб приобретению и оценке компетенций. Было выявлено, что учащиеся армянских вузов в целом осведомлены, какие именно действия считаются плагиатом, но продолжают их совершать из-за отсутствия мер пресечения [3]. Кроме того, заимствованные работы тратят ресурсы. В Германии в рамках краудсорсингового проекта VroniPlag [4] было расследовано более 200 случаев предполагаемого академического плагиата (по состоянию на июль 2019 года). Опыт VroniPlag, а также других [4], показывает, что расследования уникальности работа часто требуют сотен рабочих часов от затронутых учреждений,

[1] https://ru.armeniasputnik.am/society/20190821/20137912/Epopeya-s-AGEU-zakonchilas-molodoy-io-rektora-Ruben-Ayrapetyan-pokinet-svoy-post-.html

[2] https://ru.armeniasputnik.am/society/20200525/23163731/Esli-moy-zam-pokryvaet-plagiat-dissertatsii-on-dolzhen-otvetit—-Araik-Arutyunyan-.html

[3] https://news.am/rus/news/514896.html

[4] http://www.vroniplag.de/

и поэтому очень важно наличие автоматической системы обнаружения заимствований, с помощью которой можно будет сократить затраты.

Быстрое развитие информационных технологий, особенно Интернета, сделало заимствование работ легче, чем когда-либо. В 2015 году было проведено исследование образовательной политики Армении в направлении усиления академической добросовестности, которое подтвердило, что незаконные заимствования в курсовых, бакалаврских и магистерских работах являются одним из самых распространенных нарушений [5]. Заимствованием, кроме дословного копирования, считается сокрытие заимствований путем перефразирования и перевода. Уникальность работы искусственно увеличивают также с помощью технических приемов, которые используют слабые места методов извлечения текста системы обнаружения заимствований и меняют исходный документ таким образом, чтобы его текст визуально не менялся, но доля обнаруженных заимствований получалась маленькой.

Исследование и разработка методов выявления заимствований сейчас является довольно популярной, если судить по количеству опубликованных статей в последние годы [1]. Тем не менее, для многих языков не существует специализированной системы обнаружения заимствований. В таких случаях приходится прибегать к использованию инструментов, не адаптированных к определенному языку, однако эти решения как правило не учитывают особенности языка и не показывают достаточный уровень качества обработки. Обнаружению заимствований для армянского языка посвящена работа Томеян и др.[6]. Система, предложенная в работе Томеян и др., позволяет пользователю находить случаи замены омоглифов, а также заимствования путем прямого копирования в коллекции, загруженной пользователем. Система также предоставляет опцию поиска заимствований с заменой синонимов, однако для ее работы пользователь должен сам заполнять список синонимов в системе перед проверкой. В работе делается попытка использования машинного перевода для обнаружения заимствований из других языков, но описанный механизм работы не до конца автоматизирован и требует ручное добавление переведенных текстов. Из-за отсутствия экспериментов невозможно делать выводы о качестве и производительности данной системы. Помимо Томеян и др., собственную систему обнаружения заимствований имеет ВАК РА. Обе системы не предусматривают поиск скрытого текста, когда цвет текста совпадает

с фоновым, случаев замены текста изображением, замены синонимов и парафраза.

Учитывая недостаток исследований и решений в этой области для армянского языка, адаптация и разработка методов выявления заимствований в армянских текстах очень актуальна. В настоящее время, армянские университеты вынуждены либо отказываться от проверки текстов на уникальность, либо использовать универсальные инструменты, которые не адаптированы под армянский язык и часто способны находить только случаи обычного копирования. Например, Российско-Армянский университет в Ереване использует систему Антиплагиат.ру [5], однако изучение этой системы показало, что она неспособна выявлять замену синонимов, парафраз, использование технических приемов для армянского языка. Выявление перечисленных видов заимствований требует использование таких инструментов, как языковые модели, оптическое распознавание текста и т.д., предназначенных специально для армянского языка.

В своем обзоре исследований проблемы заимствований [1] делит направления на выбор политики в отношении заимствований и инструменты их нахождения. Упорядочение этих направлений по уровню абстракции, на котором они решают проблему заимствований, дает трехуровневую модель:

1. Исследование методов, анализирующие текстовое сходство на лексическом, синтаксическом и семантическом уровнях, а также сходство нетекстовых элементов контента;
2. Разработка систем обнаружения заимствований, реализующих методы из 1-го уровня и готовых к эксплуатации. К подобным работам относятся [7–9];
3. Статьи данного уровня исследуют отношение студентов и учителей к заимствованиям, анализируют их распространенность в учебных заведениях, обсуждают влияние институциональной политики и т.д. (например, [10–13]).

Уровни модели взаимозависимы и необходимы для всестороннего анализа феномена заимствований. Системы обнаружения заимствований зависят от надежных методов обнаружения, а они, в свою очередь, не имели бы практической ценности без готовых к производству систем, в которых используются. Примене-

[5] https://www.antiplagiat.ru

ние этих систем имеет смысл только при наличии политической основы, регулирующей отношение к заимствованиям.

В этой диссертации рассматриваются методы и системы обнаружения заимствований. С технической точки зрения в литературе различают два подхода к обнаружению заимствований: внешние и внутренние. Внешние методы сравнивают текст проверяемого документа с проверочным набором потенциальных источников. Когда нет проверочной базы, поиск заимствований производится с помощью внутренних методов, которые основываются исключительно на имеющемся документе для нахождения подозрительных участков (например, с помощью стилометрического подхода, анализируя нарушения стиля написания текста).

Целью данной работы является исследование и разработка методов и программных инструментов установления степени уникальности текстов для армянского языка. Объектом исследования данной диссертации являются тексты на литературном армянском языке, в частности, курсовые, выпускные квалификационные работы, диссертации. Предмет исследования – уникальность текстов. В рамках этой работы степень уникальности определяется как процент текста, не встречающийся в других работах.

Для достижения поставленной цели необходимо было решить следующие **задачи**:

1. На основе анализа существующих решений разработать и реализовать внутренние стилометрические методы нахождения заимствований для армянского языка.
2. На основе анализа существующих решений разработать и реализовать методы борьбы с техническими методами маскировки заимствований для армянского языка.
3. На основе анализа существующих решений разработать и реализовать метод нахождения нечетких дубликатов для армянского языка.
4. На основе анализа существующих решений разработать и реализовать метод определения парафраза в армянских текстах для внешних методов нахождения заимствований.
5. На основе реализованных методов, создать программную систему для оценки степени уникальности текстовых документов на армянском языке.

Для достижения поставленной цели и решения вышеуказанных задач были изучены и применены методы машинного обучения, математической статистики, информационного поиска нечетких дубликатов, математического анализа и линейной алгебры. Для программной реализации алгоритмов и разработки системы использовались методы объектно-ориентированного программирования.

Основные положения, выносимые на защиту:

1. Предложен подход к извлечению векторных представлений слов, полностью основанных на признаках на уровне подслов (символов и морфем), который для языков с богатой морфологией позволяет смягчить проблему разреженности данных и сокращает количество параметров в модели. На основе таких векторных представленияй слов получены модели лемматизации и морфологического анализа текстов, требующие гораздо меньше памяти и при этом не уступающие в точности моделям, имеющим в несколько раз больше параметров.
2. Разработаны методы и программные инструменты для автоматизации процесса построения размеченных наборов данных для языков с ограниченными ресурсами. На основе предложенных и существующих подходов для армянского языка впервые созданы размеченные наборы текстов для задач распознавания именованных сущностей, обнаружения парафраза, векторного представления слов, стилометрического анализа, и исправления ошибок автоматического распознавания текстов. Для первых трех из перечисленных задач созданы тестовые наборы с ручной разметкой. Разработанные наборы данных позволили создать программные инструменты для соответствующих задач, превосходящие по точности существующие аналоги.
3. С использованием перечисленных выше инструментов разработана и внедрена программная система для оценки степени уникальности текстовых документов на армянском языке, которая позволяет обнаружить полное и частичное дублирование, парафраз, техническую маскировку, а также выполняет поиск заимствований как в проверочной базе документов, так и в Интернете.

Научная новизна. Представлен новый метод генерации парафразов предложения с помощью обратного машинного перевода в несколько итераций и ручной проверки корректности результатов перевода[14]. Разрабатан метод автома-

тической генерации обучающих и тестовых примеров для задач нахождения и исправления ошибок оптического распознавания текстов [15]. Представен новый подход к использованию Викиданных и статей Википедии для полной автоматизации процесса генерации обучающих примеров для задачи распознавания именованных сущностей [16].

Предложены модификации модели векторов fastText на основе подслов, которые решают проблему разреженности данных для языков с богатой морфологией и существенно сокращают размер этих моделей без серьезной потери точности в задачах лемматизациии морфологического анализа [17; 18].

Теоретическая и практическая значимость. Основная практическая значимость диссертации заключается в разработанной системе оценки уникальности текстов, которая может быть применена в работе высших учебных заведений, ВАК РА и других похожих организаций. Для армянского языка впервые разработаны программные инструменты, позволяющие выполнять внутренний стилометрический анализ текстов на наличие заимствований, обнаруживать парафраз, исправлять ошибки в результатах оптического распознавания текстов.

Впервые для армянского языка созданы тестовые наборы данных с ручной разметкой для задач распознавания именованных сущностей [16], определения парафраза [14], внутреннего стилометрического анализа текстов [19], а также оценки качества векторных представлений слов [20]. Созданные размеченные наборы текстов могут быть использованы в будущих исследованиях для разработки и оценки качества инструментов обработки армянских текстов.

Предложенные автоматические методы генерации размеченных данных позволяют сократить использование человеческих и других ресурсов при создании обучающих и тестовых данных для соответствующих задач, могут быть применены для создания размеченных датасетов для других языков.

Апробация работы. Результаты данной работы докладывались на конференциях, форумах:

1. Science and Technology Convergence (STC) Forum 2018, Ереван, РА;
2. Открытая конференция ИСП РАН им. В.П. Иванникова 2018, Москва, РФ;
3. XIV Годичная научная конференция РАУ, 2019, Ереван, РА;
4. Международная конференция ”Иванниковские чтения 2020, Орел, РФ;

Публикации. По теме диссертации опубликовано 7 печатных работ, в том числе три статьи [14;16;18] в изданиях и сборниках научных конференций, индексируемых в Scopus, две статьи [17;19] в рецензируемых научных журналах из перечня ВАК РФ по специальности 05.13.11, и две статьи [15;20] в других изданиях.

Личный вклад. Предлагаемые в диссертации инструменты, текстовые наборы данных и исследования разработаны и выполнены автором или при его непосредственном участии. Автор имеет решающий вклад в планировании совместных работ [14-16;18-20], разработке и адаптации методов автоматической разметки и обработки текстов, принимал непосредственное участие в сборе, подготовке и ручной разметке текстов, планировании и проведении экспериментов. В публикации [18] автору принадлежит основная часть, совместно проводилось измерение качества разработанных моделей.

Внедрение результатов:

1. Разработанная система по проверке документов на уникальность была внедрена в 2021 году в Российско-Армянском университете для проверки курсовых работ учащихся;
2. Программные библиотеки для стилометрического анализа армянских текстов[6] и для исправления ошибок в тексте[7] были опубликованы на PyPI (каталог программного обеспечения, написанного на языке программирования Python).
3. Тестовый набор для задачи распознавания именованных сущностей pioNER был использован в статьях [21–23], опубликованных в сборниках авторитетных научных конференций EMNLP и LREC.

Объем и структура работы. Диссертация состоит из введения, пяти глав, заключения и двух приложений. Полный объём диссертации составляет 188 страниц с 41 рисунком и 46 таблицами. Список литературы содержит 186 наименований.

В первой главе представляется обзор существующих и используемых в работе определений и типологий заимствований. Вторая и третья главы посвящены исследованию и разработке инструментов выявления заимствований. Вторая глава описывает внутренние методы нахождения подозрительных участков в тексте, включая стилометрический анализ и обнаружение попыток технической маски-

[6] https://pypi.org/project/IntrinsicAnalysis/

[7] https://pypi.org/project/armcor/

ровки заимствований. Третья глава описывает внешние методы выявления заимствований, приводятся результаты исследования и разработки моделей нахождения нечетких дубликатов и парафраза.

В четвертой главе описываются исследования по разработке вспомогательных инструментов обработки текстов на армянском языке, включая инструментов для лемматизации, синтаксического и морфологического анализа, распознавания и классификации именованных сущностей, векторного представления слов, а также обработки результатов оптического распознавания текстов с целью обнаружения и исправления ошибок.

Пятая глава посвящена программной реализации системы обнаружения заимствований. Разделы данной главы описывают архитектуру системы и применяемые технологии для индексации проверочной коллекции документов, поиска источников в сети, реализации асинхронного выполнения трудоемких задач.

Глава 1. Определение и типология заимствований

Так как целью данной работы является исследование и разработка методов и программных инструментов установления степени уникальности текстов для армянского языка, необходимо определить какие именно участки текста считаются неуникальными, то есть заимствованием. Формы заимствования с политической и юридической точки зрения могут быть разные:

1. Незаконные заимствования;
2. Переиспользование своих более ранних работ (самоплагиат[24]);
3. Заимствование с некорректным указанием источника (например, когда студент не знает, как правильно цитировать источники в конкретном стиле);
4. Законные заимствования (с согласия оригинального автора).

В диссертации не изучаются политические и юридические аспекты этого вопроса. Также не учитывается отношение автора к этому действию (непреднамеренное, умышленное). Определение неуникального фрагмента текста, применяемое в этой работе, включает все эти виды заимствований.

В соответствии с определением Фишмана плагиатом считается заимствование идей, текста, графической, звуковой и другой информации без надлежащего признания источника, приносящего пользу в обстановке, где ожидается оригинальность [25]. Определение включает в себя все формы интеллектуального вклада в академические документы независимо от их представления, например текст, рисунки, таблицы и формулы, а также их происхождения. В этой диссертации рассматривается изучение только текстовых заимствований. Другие определения академического плагиата часто включают понятие кражи [7; 26–28], то есть требуют намерения и ограничивают возможности повторного использования чужого контента. При анализе участка текста с точки зрения исследования его уникальности, в работе не ставится вопрос проверки законности заимствования или намерений автора.

Заимствования встречается во многих формах, и задача обнаружения каждой из форм имеет разную степень сложности, поэтому помимо определения также важно изучение типологий заимствований. Исследователями было предложено множество типологий заимствований. Типология [29], основанная на типоло-

гии [30], различает только две формы заимствований: буквальный и интеллектуальный. Буквальные заимствования включают в себя близкие копии и модифицированные копии, тогда как интеллектуальные заимствования включают перефразирование, обобщение, перевод и плагиат идей. Уокер [31] придумал типологию с точки зрения автора, которая различает следующие формы заимствований:

1. Имитация перефразирования (копирование текста с нарушением правил пунктуации цитирования);
2. Незаконное перефразирование (пересказ скопированного текста без цитирования);
3. Заимствование с согласия автора;
4. Дословное копирование (без ссылки);
5. Переиспользование уже опубликованного материала этого же автора (самоплагиат);
6. Написание текста другим автором по заказу;
7. Кража (например, копирование задания другого учащегося без его согласия).

Фольтинек и др. [1] в своем обзоре также исследовали определения видов заимствований и типологии. Согласно этому обзору дословное копирование относят к формам плагиата во всех изученных типологиях. Авторы одной из работ [26] как отдельных форм заимствований в академических текстах дополнительно выделили частичное копирование небольших фрагментов текста, перевод, и две формы перефразирования, различающиеся тем, меняется структура предложения или нет. Веласкес и др. [9] разделяют техническую маскировку от дословного копирования, объединяют перевод и перефразирование в одну форму, и кроме этого, также как Вебер-Вульф [32] и Чоудхури и Бхаттачарья [33], выделяют преднамеренное неправильное использование ссылок как отдельную форму заимствования. Заимствование идей тоже многими авторами выделяется как отдельная форма [33–37].

Типология Мозгового и др. [38] объединяет другие классификации в пять форм академических заимствований:

1. Дословное копирование;
2. Сокрытие случаев заимствования путем перефразирования;
3. Технические приемы, использующие слабые места существующих систем обнаружения заимствований;

4. Умышленное неточное использование ссылок;
5. Сложные формы заимствования (заимствование идей, перевод).

Некоторые различия между формами заимствования, важные с точки зрения политики, с технической точки зрения менее важны. Поскольку в данной работе изучаются технологии обнаружения неуникальных фрагментов текста, технически несущественными свойствами заимствований считаются:

– Наличие разрешения от первоначального автора повторно использовать контент;
– Совпадение авторов подозрительного участка текста и его потенциального источника;
– Соблюдение правил использования ссылок.

Типология заимствований, используемая в [1], основана на общепринятых слоях естественного языка: лексика, синтаксис и семантика. Они классифицируют формы заимствований в соответствии с языковым слоем, на который они влияют:

1. Заимствование с сохранением символов:
    a Буквальное заимствование (копирование и вставка);
    b Буквальное заимствование с указанием источника;
2. Заимствование с сохранением синтаксиса:
    a Техническая маскировка;
    b Замена синонимов;
3. Заимствование, сохраняющее семантику:
    a Перевод;
    b Парафраз;
4. Заимствование, сохраняющее идеи:
    a Структурное заимствование;
    b Использование только концепций и идей;
5. Работа другого автора, выполненная по заказу.

Заимствование, сохраняющее синтаксис, как правило является результатом использования простых методов замены слов (например, замена слов синонимами). Заимствование, сохраняющее семантику, является более сложной формой изменения заимствованного участка текста, в которой помимо слов, также модифицируется структура предложения, но при этом смысл текста не меняется. К заимствованию идей относятся случаи, где источник полностью описывается другими словами, используя только его концепцию. Данную форму заимствования

сложно выявить и доказать. Последняя форма заимствований из типологии описывает наем третьей стороны для написания подлинного текста. Эту форму заимствования невозможно обнаружить путем сравнения подозрительного документа с вероятным источником. В настоящее время единственный технический вариант обнаружения случаев написания по заказу – это сравнение стилометрических характеристик проверяемого документа с документами, определенно написанными предполагаемым автором.

В данной работе рассматриваются методы обнаружения первых 3-х форм заимствований: буквальные заимствования, техническая маскировка, замена синонимов и парафраз. Исследование методов обнаружения перевода оставляется на будущее.

Глава 2. Внутренние методы обнаружения заимствований

Задача выявления заимствований в текстовых документах состоит в обнаружении подозрительных участков текста и последующем подтверждении подозрений с помощью детального анализа. С технической точки зрения в литературе различают два подхода к обнаружению заимствований: внешние и внутренние.

Внешние методы сравнивают текст проверяемого документа с проверочным набором потенциальных источников. Однако вполне возможны случаи, когда источник заимствований отсутствует в проверочном наборе. Кроме этого, не всегда имеется проверочная база для применения внешних методов. В таких случаях поиск заимствований производится с помощью внутренних методов, которые основываются исключительно на имеющемся документе для нахождения подозрительных участков в нем. Учитывая отсутствие публично доступных оцифрованных проверочных коллекций документов для армянского языка и распространенность на практике трудно обнаруживаемых заимствований путем перевода, становится актуальным исследование возможности применения внутренних методов поиска заимствований.

Внутренний метод обнаружения заимствований анализирует исключительно входной документ, то есть не выполняет сравнения с документами в справочной коллекции. К таким методам обнаружения подозрительных участков текста можно отнести стилометрический подход, целью которого состоит в том, чтобы выявить изменения в стиле написания, рассматривая эти изменения как индикаторы потенциального заимствования. Другим направлением внутреннего анализа является поиск попыток технической маскировки заимствований. При технической маскировке, используя слабые места методов извлечения текста системы обнаружения заимствований, исходный документ преобразовывается таким образом, чтобы его текст визуально не менялся, но доля обнаруженных заимствований получалась маленькой. Заранее зная какие именно технические приемы применяются, можно проанализировать документ и обнаружить участки со следами их применения. Подозрительные участки текста, выделенные с помощью методов внутреннего анализа, далее могут быть представлены рецензенту для дополнительной проверки.

В этой главе представлены исследования по адаптации и применению стилометрических методов внутреннего анализа к армянским текстам, а также методов и механизмов по обнаружению технических приемов для маскировки заимствований.

2.1 Стилометрический подход

Данный раздел посвящен применению внутренних стилометрических методов в задаче обнаружения текстовых заимствований для армянского языка. Были исследованы два варианта постановки задачи: обнаружение изменения стиля в документе и обнаружение границ нарушений стиля. Для этих задач в рамках были созданы синтетические примеры с заимствованиями для академического, художественного и новостного жанров текста, и на полученных примерах была проверена эффективность алгоритмов иерархической кластеризации и других моделей по обнаружению нарушений стиля из серии конференций PAN[1].

Для установления наличия заимствований в тексте, методы внутреннего анализа подвергают его стилометрическому анализу. В этом контексте стиль текста определяется закономерностью использования словоформ, знаков препинания, стиля речи, а также уровнем читабельности. Предполагается, что каждый автор обладает собственным уникальным стилем написания текста, и фрагменты, где наблюдается отхождение от этого стиля, предположительно являются заимствованиями из работ других авторов.

В рамках данной работы на армяноязычных текстах проверяется применимость существующих моделей обнаружения нарушений стиля в документах. Эффективность моделей оценивается на автоматически сгенерированных наборах примеров из трех жанров текстов – академических, художественных, и новостных. Представляются результаты для двух конфигураций постановки задачи: бинарная классификация, отвечающая на вопрос, содержит ли текст нарушения стиля, и определение границ стилистически однородных участков в тексте. Наборы данных, исходные код и другие ресурсы, необходимые для тестирования моде-

[1]PAN - научные мероприятия и открытые соревнования (https://pan.webis.de/), посвященные стилометрии и форензике текстов, проводимые ежегодно в рамках конференции CLEF (http://clef2021.clef-initiative.eu/)

лей, доступны на GitHub[2]. Результаты данного исследования были опубликованы в статье [19].

2.1.1 Обзор литературы

Исследованию и разработке стилометрических методов были посвящены несколько конференций PAN. В них встречаются разные постановки задач внутреннего анализа, из которых основными являются обнаружение изменения стиля в документе (style change detection)[39; 40], обнаружение границ нарушений стиля (style breach detection)[41], кластеризация по авторству (author clustering)[41; 42].

2.1.1.1 Обнаружение изменения стиля

Обнаружение изменения стиля формулируется как задача классификации, где нужно определить содержит ли текст отклонения от преобладающего в нем стиля. Для обзора существующих методов по обнаружению изменения стиля текста были изучены работы участников конференции PAN 2018 и 2019 годов [39; 40]. На тестовых выборках наилучшие результаты (табл. 1 и 2) показали модели Nath et al. [43], Zlatkova et al. [44], Hosseinia et al. [45] и Safin et al [46].

Таблица 1 — Результаты методов обнаружения изменения стиля с PAN 2018 [39].

Метод	Accuracy	Время выполнения
Zlatkova et al.	0.893	01:35:25
Hosseinia et al.	0.825	10:12:28
Safin et al.	0.803	00:05:15
Khan	0.643	00:01:10
Schaetti	0.621	00:03:36
C99-BASELINE	0.589	00:00:16
rnd2-BASELINE	0.560	-
rnd1-BASELINE	0.500	-

[2]https://github.com/tsolakghukasyan/style-change-analysis-1

Таблица 2 — Результаты методов обнаружения изменения стиля с PAN 2019 [40].

Метод	Accuracy	Время выполнения
Nath et al.	0.848	00:02:23
Zuo et al.	0.604	00:25:50
BASELINE-RND	0.600	-
BASELINE-C99	0.582	00:00:30

В Zlatkova et al. документ делится на несколько сегментов, для каждого из которых выполняется извлечение лексических, синтаксических признаков, и далее на каждом из этих групп признаков применяются 4 классификатора – SVM, случайный лес, AdaBoost и многослойный персептрон. Каждому классификатору присваивается вес на основе достоверности предсказания, и на основе этого далее каждой группе признаков сопоставляется вектор с вероятностями предсказания класса. Эти векторы вместе с результатами градиентного бустинга на основе TF-IDF подаются на вход логистической регрессии, результатом которой является ответ, присутствует ли в тексте изменение стиля или нет.

Nath et al. использует метод, основанный на алгоритме пороговой кластеризации. Для извлечения признаков документ делится на окна, и каждое окно преобразуется в вектор признаков на основе нормализованной частоты выбранных токенов. Для определения расстояния между векторами авторы выбрали меру расстояния Матусита. Идея алгоритма пороговой кластеризации состоит в том, чтобы построить список расстояний между окнами и итеративно выбирать окна с наименьшим расстоянием так, чтобы при формировании кластера ближайшие элементы включались первыми. После этого включаются окна, которые находятся дальше. Результатом алгоритма является число кластеров, которое обозначает количество авторов в проверяемом тексте.

Метод Hosseinia et al. основан на идее определения изменения стиля в тексте на основе синтаксического анализа. Для каждого предложения в документе строится синтаксического дерево разбора, далее последовательность этих деревьев передается на вход двух рекуррентных нейронных сетей, первая из которых обрабатывает последовательность в прямом порядке, как в документе, а вторая – в обратном. Далее, при помощи функции схожести оценивается разность между результатами сетей, на основе которой определяется вероятность изменения стиля текста. Так как данная модель сильно зависима от качества синтаксического

анализа текстов, а в случае армянского языка точность таких анализаторов сравнительно низкая, она не рассматривалась в рамках данной работы.

В работе Safin et al., как и в модели Zlatkova et al., используется ансамбль классификаторов, каждый из которых по некоторому признаку определяет, встречается ли в документе изменение стиля текста или нет. Первый классификатор использует 19 статистических признаков (число предложений, частота уникальных слов, длина текста и т.д.), для второго классификатора каждый текст представляется в виде 3000-мерного вектора, который содержит информацию о частоте символьных N-грамм в тексте. Третий классификатор применяется на векторном представлении текста, который содержит векторные представления N-грамм (N=1,…,6), и размерность которого составляет более 3 миллионов. В конце вычисляется взвешенная сумма результатов классификаторов, которая используется как оценка вероятности присутствия в тексте изменения стиля.

2.1.1.2 Обнаружение границ нарушений стиля

Задача выявления границ нарушений стиля заключается в определении моно- или мульти-авторства исследуемого документа и разбиении документа на стилистически однородные фрагменты в случае мульти-авторства с указанием границ смен стиля. Данная задача была предложена участникам PAN в 2017 году [41].

Лучший результат (рисунок 2.1) показал метод Karaś et al. [47], в котором документ разбивается по параграфам, затем каждый параграф представляется в виде вектора признаков, полученного путем конкатенации tf-idf значений по униграммам слов, по 3-граммам, по стоп-словам, по размеченным частям речи и пунктуационным символам. Для выявления отклонений от стиля используется статистический подход – тест Вилкоксона над попарными векторами-строками. Наименьшие 30% значений тестов считаются аномальными и ставится граница в начало крайнего параграфа соответствующего теста.

Другие методы из этой конференции, Khan et al. [48] и Safin et al. [49] были менее эффективны в данной задаче, и поэтому не рассматривались в нашей работе для адаптации и применения к армянскому языку. В этих методах сравниваются

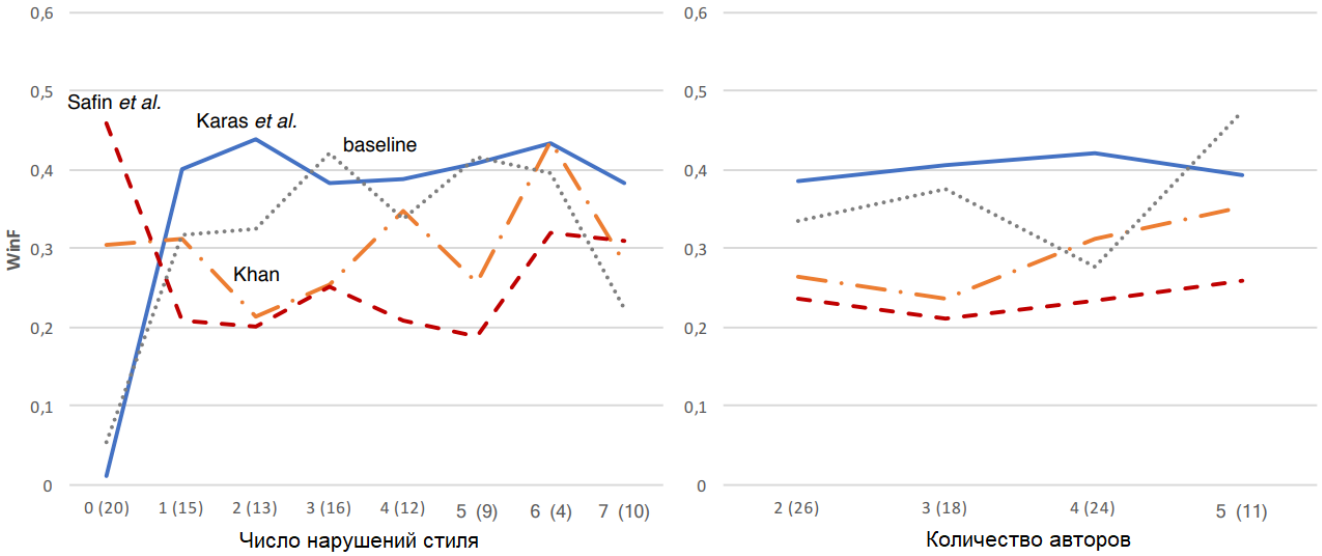


Рисунок 2.1 — Результаты методов обнаружения границ нарушений стиля с PAN 2017 [41].

представления рядом стоящих предложений. В Khan et al. документ разделяется по предложениям и рассматриваются скользящие окна из трех предложений, имеющие одно общее предложение. Они используют статистические признаки на основе синтаксических, лексических характеристик текста. Алгоритм Safin et al. применяет модель Skip-Thought, основанную на предсказании предыдущего и следующего предложения, с архитектурой кодировщик-декодировщик и с использованием GRU сетей. Предложение считается отклонением, если среднее расстояние его векторного представления от остальных векторов больше заданного допустимого. Помимо того, что данная модель работала значительно медленно по сравнению с остальными, ее адаптация к армянскому языку была бы проблематичной также потому, что для предобучения и эффективной работы используемой нейронной сети понадобилось бы большое количество текстовых данных.

2.1.1.3 Кластеризация по авторству

Задача авторской кластеризации заключается в группировании небольших моно-авторских документов по группам авторов. В рамках данной работы рассматривались модели, предназначенные для решения той вариации этой задачи, в которой нужно каждый документ необходимо присвоить кластеру одного авто-

ра. Данная задача авторской кластеризации рассматривалась в рамках открытых соревнований PAN 2016 и 2017 [40; 41].

В условиях неизвестности точного количества авторов лучший результат показали алгоритмы Gómez-Adorno et al. [50], García-Mondeja et al. [51] и Kocher et al. [52], использующие иерархическую кластеризацию (табл. 3). В модели Gómez-Adorno et al. для представления документа используются N-граммы слов и символов, над которыми далее применяется иерархическая агломеративная кластеризация, где количество кластеров определяется путем максимизации индекса Calinski Harabaz (отношение среднего значения дисперсии между кластерами и дисперсия внутри кластера).

Таблица 3 — Результаты методов кластеризации по авторству с PAN 2017 [41]

Метод	B^3 F	B^3 rec.	B^3 prec.	Время выполнения
Gómez-Adorno et al.	0.573	0.639	0.607	00:02:06
Garcia et al.	0.565	0.518	0.692	00:15:49
Kocher & Savoy	0.552	0.517	0.677	00:00:42
Halvani & Graner	0.549	0.589	0.569	00:12:25
Alberts	0.528	0.599	0.550	00:01:46
BASELINE-PAN16	0.487	0.331	0.987	50:17:49
Karaś	0.466	0.580	0.439	00:00:26
BASELINE-Singleton	0.456	0.304	1.000	-
BASELINE-Random	0.452	0.339	0.731	-

В García-Mondeja et al. для векторного представления документов используется модель мешка слов. Схожести между векторами определяется по трем функциям расстояния – косинусное, Дайса и Жаккара. Они использовали β-compact графовую кластеризацию - метод построения ориентированного графа, где вершины i и j соединяются в один кластер, если они являются β-схожими и вершина j наиболее схожая из всех остальных вершин. Порог схожести β определяется на стадии тренировки, при котором максимизируется метрика FBcubed.

В своем решении Kocher et al. используют Single-link иерархическую кластеризацию, где для слияния текущих кластеров сравниваются их наиболее близкие вектора. Для представления документа вектором алгоритм использует частоты наиболее частых токенов (слов и пунктуации) и наиболее частых символьных 6-грамм. Для обозначения схожести используется Канберровское расстояние, и

для сравнения полученных значений вводится понятие “достаточно маленькое расстояние”.

Все указанные алгоритмы используют агломеративную парадигму, оценивая схожесть всех пар документов и используя эти значения для последовательного формирования кластеров. Применение методов кластеризации для группировки небольших текстов по стилю также было исследовано в работе [53], где применяется метод k-средних на коллекции анонимных электронных писем. С учетом того, что документы, на которых проверялись эти модели кластеризации по авторству, имели размер, сравнимый с параграфами текста, в данной работе посчитали целесообразным применение этих моделей в задаче определения нарушений границ стиля.

2.1.2 Адаптация методов к армянскому языку

Для определения изменения стиля в данной работе рассматривается алгоритм Nath et al., а для задачи нахождения границ нарушений стиля – алгоритм Karaś et al. и иерархическую модель кластеризации. В данной работе для рассматриваемых моделей в качестве единиц стилистически однородных участков текста используются параграфы, считая маловероятной нарушения стиля внутри одного параграфа [54]. Соответственно, в алгоритме Karaś et al. и иерархической кластеризации документ представляется как список параграфов.

Для кластеризации применяется агломеративная модель, в которой новые кластеры создаются рекурсивным объединением схожих мелких кластеров. Схожесть между кластерами определяется по методу Варда как прирост суммы квадратов расстояний объектов до центра кластера в случае их объединения. На каждом шаге алгоритма объединяются те два кластера, которые приводят к минимальному увеличению дисперсии. В результате работы алгоритма каждому параграфу присваивается номер кластера. Для обнаружения нарушения стиля между параграфами последовательно сравниваются номера их кластеров. В случае если два соседних параграфа принадлежат разным кластерам, между ними ставится граница смены стиля.

2.1.2.1 Признаки

В задаче определения изменения стиля модель Nath et al. используется в своем оригинальном варианте, без изменений набора признаков. В задаче нахождения границ нарушений стиля в моделях Karaś et al. и кластеризации помимо общепринятых распространенных признаков из решений PAN, дополнительно извлекается собственный набор признаков для описания синтаксических особенностей параграфа. Эффективность синтаксической информации в стилометрических задачах была установлена в работах [55–57]. Суммарно, используются различные символьные, лексические, морфологические и синтаксические признаки (полный список приводится в Приложении А):

1. Символьные: извлекаются две группы символьных признаков: первая группа описывает наличие и частоту используемых суффиксов и префиксов, вторая - описывает наличие и частоту знаков пунктуации, а также стиль их комбинирования с пробелами (например, наличие пробела перед знаком пунктуации);
2. Лексические
   - Общие характеристики, такие как наличие неформальных слов, терминов на латинском/кириллице, наличие и частота использования длинных и коротких слов;
   - Использование сокращений, использование сокращений вместо полных форм, предпочтительные формы сокращений и стиль их склонения, а также выбор заглавных или строчных букв при написании;
   - Формы написания количественных и порядковых числительных, стиль написания дат;
   - Наличие и частота нетипичных для других документов N-грамм слов (с низким значением idf, вычисленном на коллекции текстов [20]);
3. Синтаксические
   - Общие закономерности на основе длин предложений (отдельно рассматривается и посимвольная длина, и количество слов в предложении), как, например, минимальная, средняя, макси-

мальная длина предложений, наличие и частота длинных и коротких предложений, количество предложений в параграфе;
- Предпочтения к использованию определенных частей речи, наличие и частота использования конкретных 2-грамм и 3-грамм частей речи;
- Наличие и частота конкретных синтаксических зависимостей в предложении, использование простых или сложных предложений;

4. Читабельность: индексы читабельности текста, адаптированные к армянскому языку: Flesch reading ease, Flesch-Kincaid, SMOG, Coleman-Liau, automated readability index, Dale-Chall, difficult words, Linsear write formula и Gunning fog.

Изначально, во время данного исследования в открытом доступе не было морфологических и синтаксических анализаторов текста, а также инструмента распознавания и классификации именованных сущностей, необходимых для извлечения признаков. Поэтому было проведено исследование по изучению и созданию этих инструментов для армянского языка. Позже была опубликована библиотека Stanza[58], показывающая высокую точность обработки армянского языка, поэтому именно она была применена в итоговых экспериментах для морфологического и синтаксического анализа. Для токенизации текста на слова и предложения использовалась библиотека UDPipe 2.0[59].

2.1.2.2 Эксперименты

Были проведены эксперименты, чтобы оценить качество работы алгоритмов определения изменения стиля и границ нарушений стиля в документе. Для этих экспериментов были автоматически сгенерированы синтетические тестовые примеры для трех жанров текстов: академического, художественного и новостного. Также, чтобы оценить эффективность этих алгоритмов, их результаты на этих данных были сравнены со случайными и тривиальными базовыми моделями.

Наборы данных Для построения набора данных были использованы документы с одинаковой тематикой, написанные разными авторами. Построение каждого примера происходило автоматически, путем объединения параграфов исходных документов (метод 1). Один из документов выбирался в качестве основного, и далее некоторые из его параграфов случайным образом заменялись на параграфы других документов одинакового содержания. В процессе генерации примеров вероятность замены параграфа на заимствование из другого документа контролировалась с целью, чтобы получить примеры с разным содержанием заимствований. Также многие сгенерированные примеры в итоге были отфильтрованы, чтобы избежать присутствия очень близких примеров в полученном наборе.

```
for all DOCUMENT ∈ GROUP do
    for i = 1 to get_paragraph_count(DOCUMENT) do
        if random() < STYLE_BREACH_RATIO then
            SOURCE = select_document(GROUP - DOCUMENT)
            DOCUMENT.replace_paragraph(i, SOURCE.get_paragraph(i))
        end if
    end for
end for
```

Метод 1: Метод генерации примеров с нарушением стиля.

В качестве исходных данных выбирались следующие наборы данных:

1. Защищенные кандидатские диссертации РА

   Работы, представленные в открытом доступе на официальном сайте[3], доступны только в формате PDF. Извлеченные тексты были обработаны (удаление вводных глав, списка литературы, сокращений, нумерации, сносок, номеров страниц) и далее сгруппированы на основе направления работы. Выбор данного корпуса основывался на предположении, что каждая из работ написана одним автором;

2. Энциклопедические статьи

   В качестве источников для генерации примеров были использованы статьи об известных личностях из Википедии и онлайн-версии Армянской энциклопедии[4];

3. Учебники

[3] http://etd.asj-oa.am/

[4] http://www.encyclopedia.am/

В качестве другого источника примеров академического жанра, были выбраны материалы из учебники старших школ и университетские пособия. Были выбраны учебники по истории армянского народа 7-9 классов на армянском языке и пособие “История Армении” для вузов[5];

4. Художественные тексты

   Для генерации примеров с текстом из литературного жанра были использованы произведения, которые имеют одинаковую сюжетную линию, но написаны разными авторами: произведения “Фазан” Акселя Бакунца и сценарий к фильму “Этот зеленый, красный мир”, написанный Грантом Матевосяном, по мотивам данного произведения;

5. Новостные статьи

   Чтобы найти новостные тексты, описывающие одно и то же событие, но из разных источников, использовался агрегатор новостей NewsHub[6].

В результате было сгенерировано 144 примера из учебников, 84 новостных, 50 художественных, 400 из диссертаций и 44 из энциклопедий. Для примеров из диссертаций, учебников и художественной литературы контролировалось количество параграфов (5, 10, 25 и 50) и процент заимствований для дальнейшего исследования влияния этих параметров на качество работы алгоритмов (Таблица 4).

Таблица 4 — Количество примеров в сгенерированных тестовых наборах.

Жанр		Количество примеров								
		Все	Количество параграфов				Процент заимствований			
			5	10	25	50	0	0-20	20-40	40-50
Академический	Диссертации	400	100	100	100	100	70	82	173	75
	Учебники	144	14	33	46	51	10	41	65	22
Художественный		50	11	22	17		10	14	19	6

Результаты Обнаружение изменения стиля: в данной конфигурации были протестированы алгоритмы Nath et al., Zlatkova et al., и дополнительно, алгоритмы обнаружения границ нарушений стиля – Karaś et al. и кластеризацию. Последние три модели почти на всех примерах давали положительный ответ, то есть фактически работали как тривиальный классификатор. По этой причине в данном разделе более подробно изучаются только результаты модели Nath et al. (Таблица 5).

[5] https://lib.armedu.am/

[6] https://newshub.am/

Для каждого тестового набора результаты приводятся отдельно. Учитывая преобладание положительных примеров в наборах, для получения полной картины поведения алгоритмы также изучается доля ложно положительных предсказаний (FPR), помимо точности и полноты.

Таблица 5 — Качество обнаружения изменения стиля модели Nath et al.

Жанр текстов		Точность	Полнота	F1	Специфичность	FPR
Академический	Диссертации	0.9002	0.8105	0.853	0.3432	0.6567
	Учебники	0.9217	0.8217	0.8688	0.4	0.6
	Энциклопедии	0.4074	0.55	0.468	0.3333	0.6666
Художественный		0.8437	0.675	0.75	0.5	0.5
Новости		0.0417	0.1428	0.0645	0.7012	0.2987

Среди академических текстов алгоритм работает с примерно одинаковым уровнем доли ложно положительных, и дает правильные предсказания лишь на около трети полностью оригинальных примеров. В этом плане заметно лучше результаты для новостных текстов, доля ложно положительных ниже 30%. Однако для этих примеров присутствует другая проблема: алгоритм показывает склонность к классификации большинства новостных примеров как полностью оригинальных. Для энциклопедических примеров также присутствует данная проблема.

Чтобы понять насколько действительно эффективен алгоритм, он сравнивается со случайным классификатором (Рис. 2.2). Для сравнения в качестве метрики используется специфичность, которая показывает какой процент действительно оригинальных документов был предсказан таковым алгоритмом. Данный анализ показал, что для длинных документов (с количеством параграфов 25 и более), независимо от жанра текста результаты алгоритма работает значительно хуже базового метода. Это связано с тем, что с ростом размера текста алгоритм показывает тенденцию находить присутствие заимствований.

Дополнительно также было изучено влияние количества заимствований в тексте документа на качество предсказаний (Рис. 2.3). Для всех категорий примеров наблюдается четкая корреляция роста качества предсказания с ростом процента заимствований.

Обнаружение границ нарушений стиля: в данной конфигурации были протестированы алгоритм Karaś et al. и агломеративная кластеризация (AC). Аналогично [40] в качестве базового метода используется случайный классификатор,

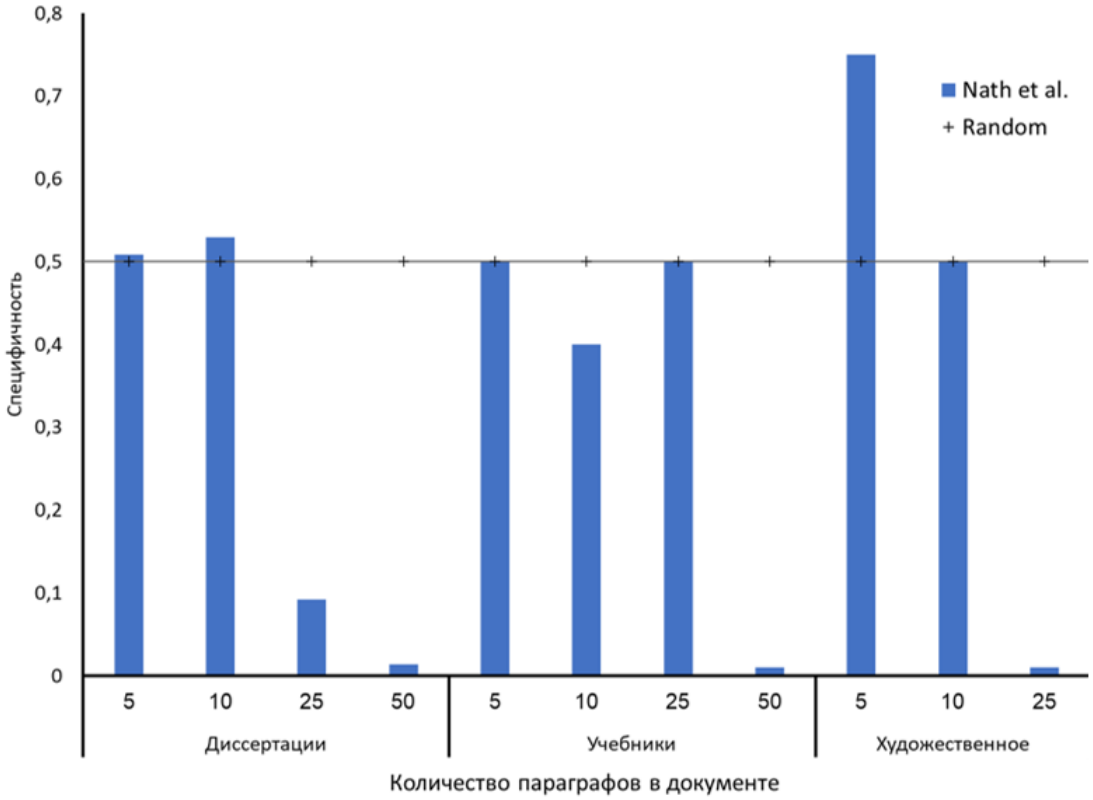


Рисунок 2.2 — Уровень специфичности обнаружения изменения стиля в тексте в зависимости от его жанра и длины.

однако с немного другой конфигурацией: в наших экспериментах используется базовый метод, который для каждой границы параграфов с вероятностью 0.25 предсказывает присутствие нарушения стиля.

Для оценки качества этих моделей используются точность (WinP) и полнота (WinR), заданные по формулам:

$$WinP = \frac{TP}{TP + FP} \tag{2.1}$$

$$WinR = \frac{TP}{TP + FN} \tag{2.2}$$

Точность определяется как соотношение правильно поставленных моделью границ от всех границ (TP), поставленных моделью (TP+FP), а полнота – соотношение правильно поставленных моделью границ от всех реальных границ (TP+FN).

Результаты алгоритмов для каждого тестового набора иллюстрированы на Рис. 2.4. Чтобы определить, насколько уверенно можно утверждать превосходство Karaś et al. и AC над случайным классификатором, был вычислен 90% до-

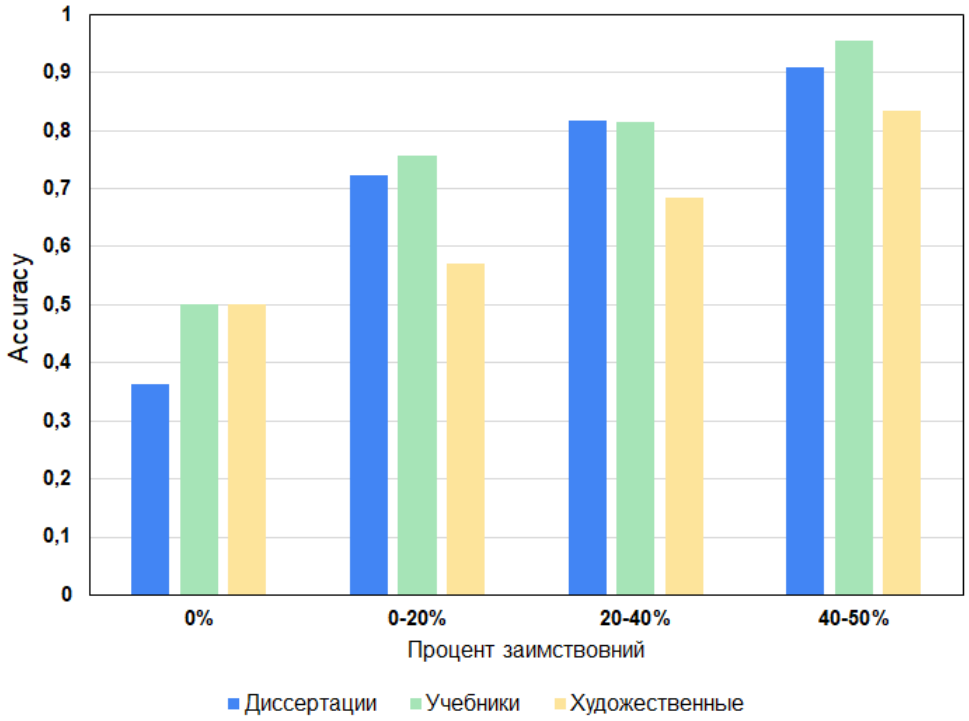


Рисунок 2.3 — Зависимость accuracy от процента заимствований для модели Nath et al.

верительный интервал. В целом, почти на всех текстах, кроме художественных, кластеризация обошла Karaś et al. Также, для художественных текстов преимущество данного алгоритма над базовым методом несущественное, а для энциклопедий и новостных текстов можно говорить, что его качество заметно хуже. Кроме диссертаций и художественных текстов, алгоритм Karaś et al. показал результаты хуже базового метода. В случае учебников и энциклопедий его точность оказалась близкой к нулю. Для этих категорий текстов данный алгоритм редко находил границы нарушений стиля.

Для алгоритма кластеризации качество проверялось для количества кластеров 2, 3, и 4 (Рис. 2.5). За исключением новостных текстов, в среднем самая высокая точность была получена при ограничении количества кластеров на 2. С учетом наличия многих тесно связанных признаков в описаниях параграфов, а также ограниченного количества параграфов в отдельном документе, для повышения точности работы было также протестировано сокращение размерности признакового описания с помощью PCA, следуя примеру[60], и для художественных текстов, учебников и диссертаций были получены заметные улучшения в точности (Рис. 2.6).

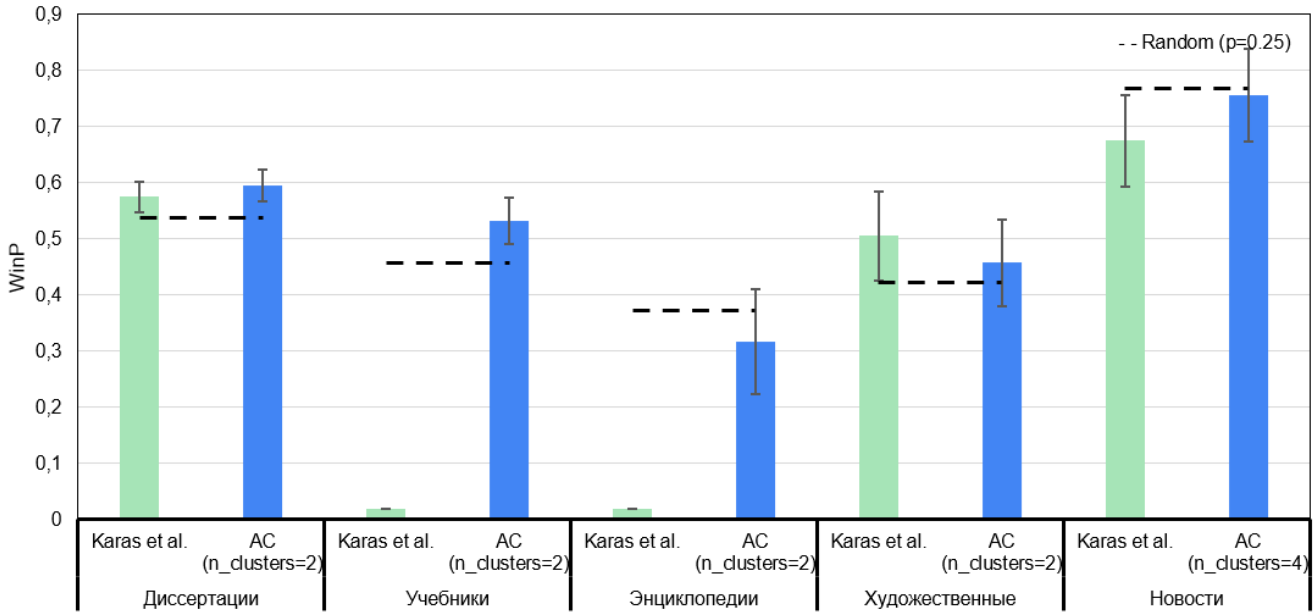


Рисунок 2.4 — Сравнение наиболее точных (90% доверительный интервал) моделей обнаружения границ нарушений стиля и случайного классификатора для каждого жанра.

Дополнительно было изучено влияние длины текста и количества заимствований на качество предсказаний алгоритма кластеризации (Рис. 6 и Рис. 7). С точки зрения полноты результатов, наблюдается закономерность по его уменьшению параллельно с ростом длины документа и процента заимствований. В случае точности нет явной корреляции между значениями по этой метрике и количеством параграфов в тексте. Только в случае текстов учебников наблюдается аномально низкая точность для маленьких документов с 5 параграфами. При этом чем меньше заимствованных участков текста в документе, тем менее точно работает алгоритм.

2.1.3 Заключение

В данном разделе была исследована эффективность стилометрических методов внутреннего анализа для нахождения заимствований в текстах на армянском языке. Были изучены и для трех категорий текстов (академических, художественных, новостных) протестированы алгоритмы, основанные на лучших моделях с соревнований PAN 2017-2019. Было получено, что для длинных докумен-

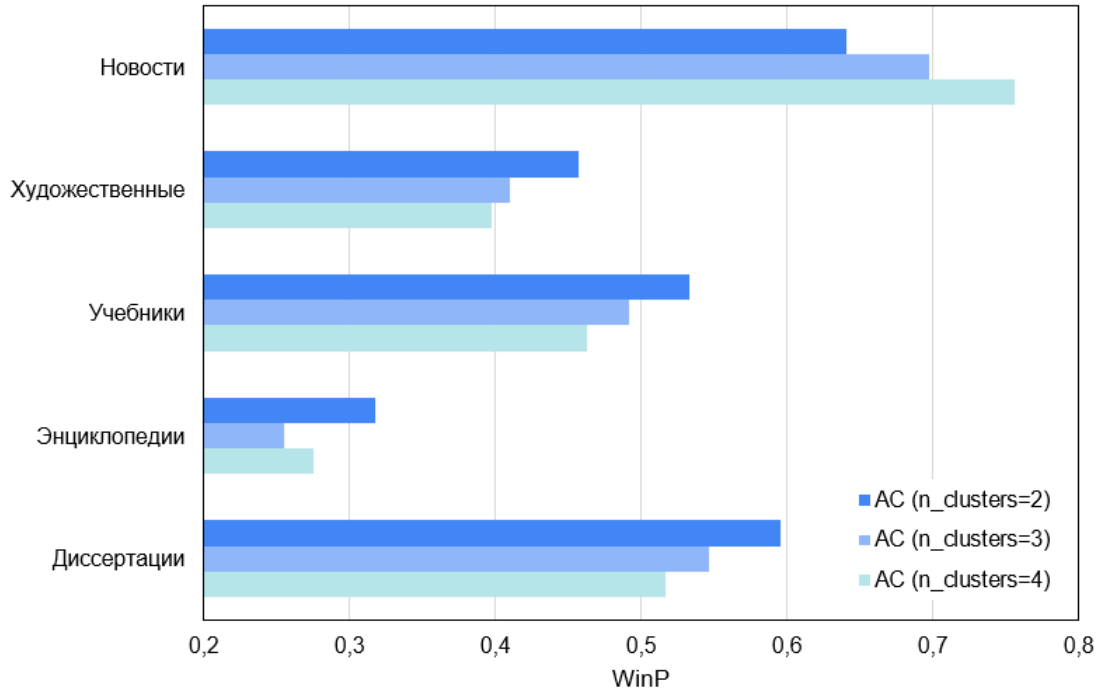


Рисунок 2.5 — Зависимость точности модели AC от количества кластеров.

тов алгоритмы нахождения изменения стиля на основе кластеризации показывают низкую специфичность и поэтому неэффективны. В задаче определения границ нарушений стиля для конкретных категорий текстов алгоритмы достигают качества выше случайного классификатора, однако все еще недостаточно высокого для их применения на практике. Дополнительно была установлена эффективность применения PCA на входных признаках для сокращения их размерности.

2.2 Выявление технических трюков

Помимо стилометрического подхода, другим методом поиска подозрительных участков является выявление попыток маскировки заимствований путем технических трюков, включая скрытый текст, замену омоглифов, вставку изображений вместо текста и т.д. Данный раздел посвящен изучению этих методов маскировки заимствований, исследованию существующих подходов по их обнаружению и обсуждению решений для армянского языка.

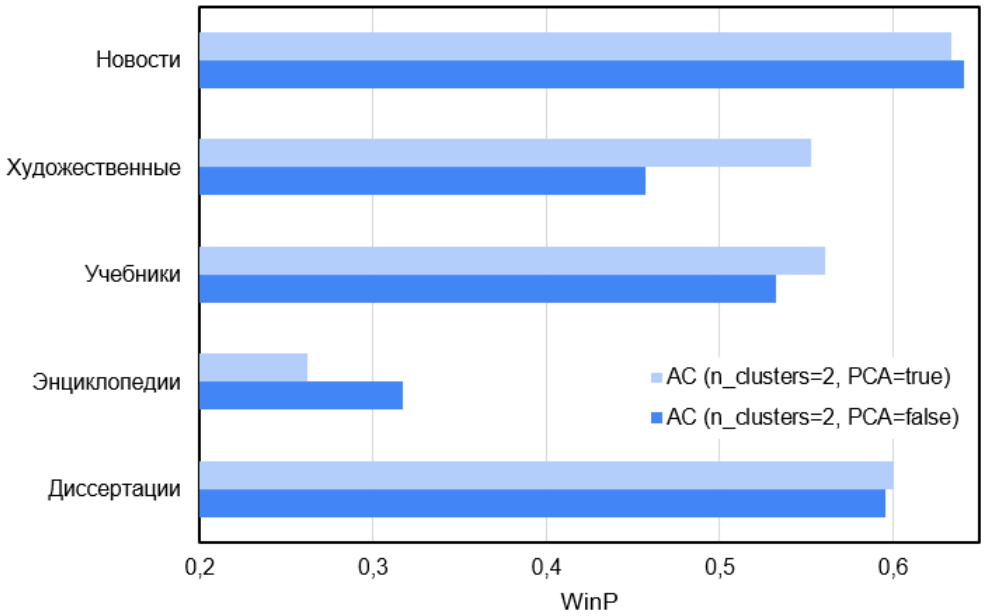


Рисунок 2.6 — Влияние PCA на точность модели AC.

2.2.1 Скрытый текст и вставка изображений

Вставка скрытого невидимого текста в документ (например, белый текст на белом фоне) является одним из распространенных вариантов искусственного увеличения процента уникальности документа. Цель таких вставок – искусственное уменьшение процентного соотношения заимствований путем добавления в документ незаметных псевдоуникальных фрагментов текста. Как правило вставленный текст является бессмысленным набором символов, которые системами автоматического обнаружения заимствований рассматриваются как уникальные.

Данный метод маскировки встречается в файловых форматах .docx, .pdf, .html. Чтобы найти подобные фрагменты можно сравнить цвет текста с цветом фона, и при совпадении выделить фрагмент как подозрительный. Такой подход используется, например, компанией Google во время индексации документов для поисковой системы [61]. Выявить скрытый текст можно также с помощью оптического распознавания (OCR) документа и дальнейшего сравнивая полученного результата с содержанием текстового слоя документа (метод 2). Участки текстового слоя, отсутствующие в результате OCR, выделяются как потенциальное скрытое заимствование. Помимо белого текста, такой подход также позволяет обнаруживать фрагменты текста, вынесенные за границы страницы (в формате .docx, например).

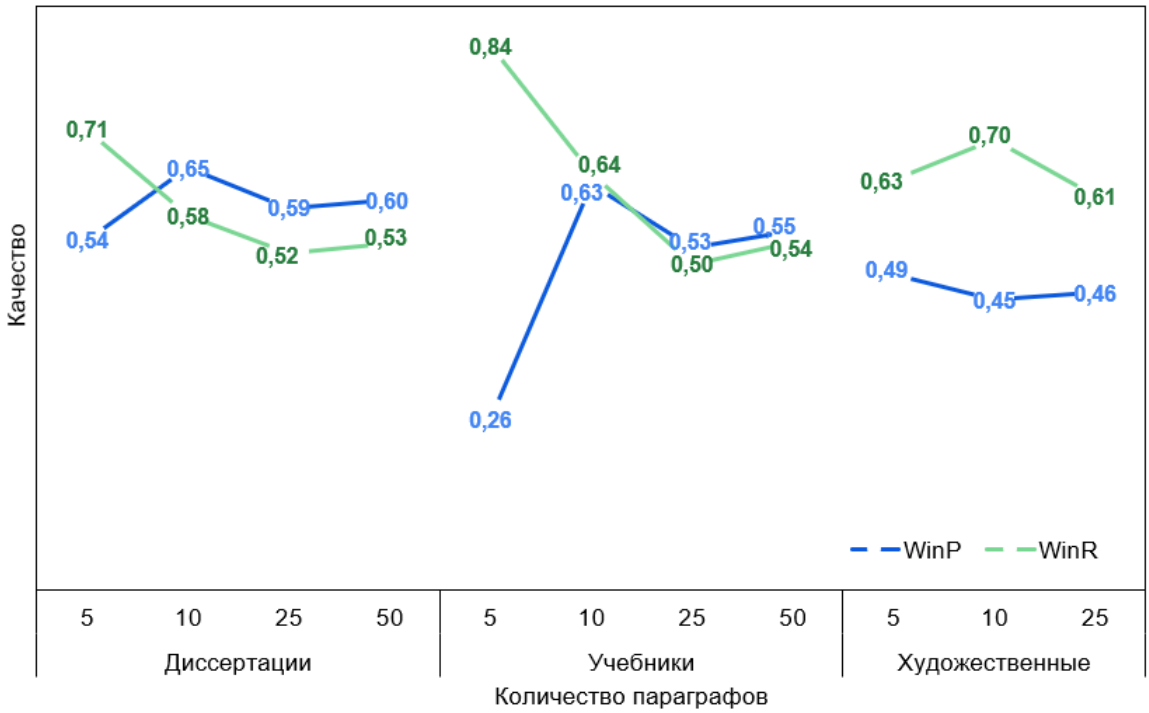


Рисунок 2.7 — Зависимость качества обнаружения границ изменений стиля от длины документов для модели AC (n_clusters=2).

Другим распространенным способом скрытия заимствований является замена текста его изображением в надежде, что таким образом система выявления заимствований не обработает данный участок текста и следовательно уникальность документа не пострадает. Для борьбы с этим видом технической маскировки необходимо использование оптического распознавания текста (OCR). С помощью вышеописанного подхода на основе OCR одновременно со скрытым текстом можно также выявить случаи вставки изображения текста вместо самого текста. В данном случае, в качестве подозрительных отмечаются те участки текста, которые отсутствуют в текстовом слое файла, но присутствуют в результате OCR.

Для сравнения результата OCR с текстовым слоем документа и нахождения отличающихся участков текста были изучены разностные алгоритмы Майерса [62], терпения и гистограммы, которые сравнивают два текста и выделяют несовпадающие участки.

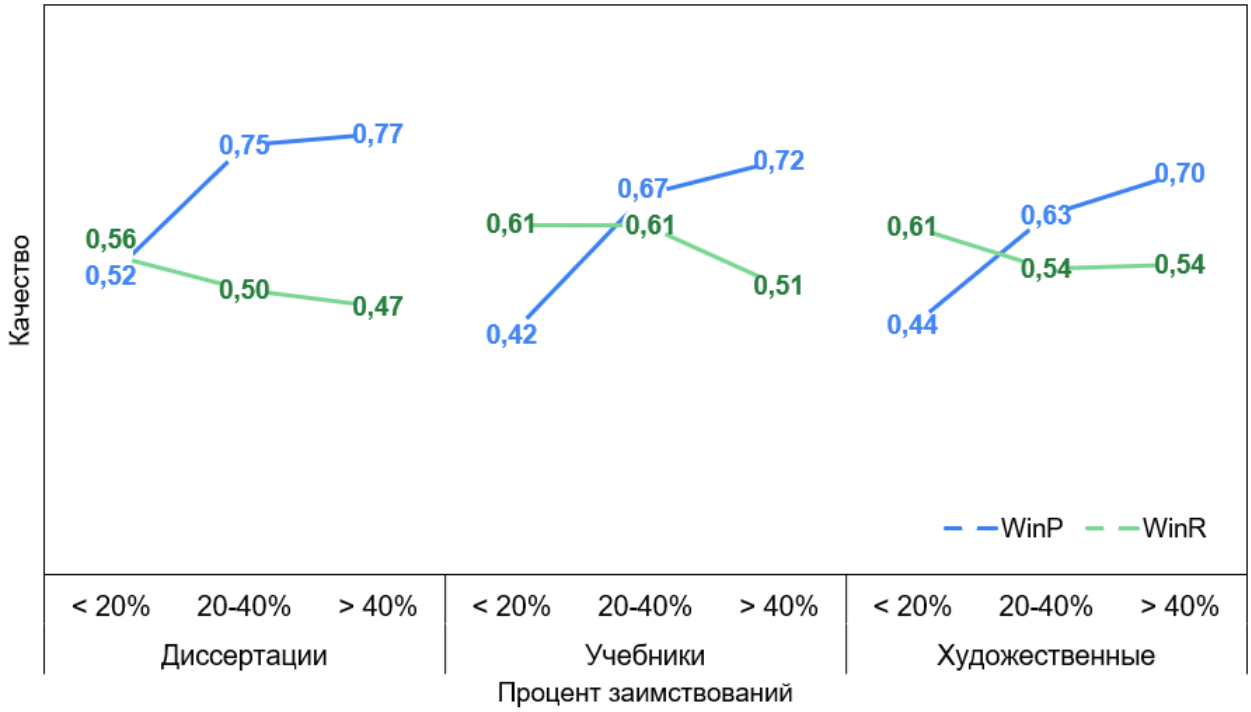


Рисунок 2.8 — Зависимость качества обнаружения границ нарушений стиля от процента заимствований в документах для модели AC (n_clusters=2).

2.2.1.1 Алгоритм Майерса

Алгоритм Майерса пытается найти последовательность изменений, которая превращает первый текст во второй. Для этого строится прямоугольная сетка из квадратов, на верхней части которой помещаются символы первого текста, а слева – второго. Далее выделяются «специальные» квадраты, для которых символы сверху и слева совпадают, и в этих квадратах добавляется диагональная линия, идущая из верхнего левого угла в правый нижний. Далее строится кратчайший путь из левого верхнего угла сетки в правый нижний. Алгоритм построения пошаговый, где на каждом шаге путь продвигается только на один шаг вдоль границы квадратов. Когда путь достигает вершины сетки, в которой начинается диагональ «специального» квадрата, то автоматически путь дополняется до конечной вершины этой диагонали. Построение завершается, когда путь доходит до нижней правой точки сетки. После этого на основе вершин построенного пути восстанавливается последовательность изменений, необходимых для преобразования первого текста во второй. Шаг по вертикали соответствует добавлению соответствующего вертикального символа в текущую позицию исходной строки, шаг по

```
TEXT = DOCUMENT.get_text()
OCR_TEXT = DOCUMENT.get_text_ocr()
for i = 1 to TEXT.get_page_count() do
    PAGE = TEXT.get_page(i)
    OCR_PAGE = OCR_TEXT.get_page(i)
    DIFF = get_patience_diff(PAGE, OCR_PAGE)
    for all FRAGMENT ∈ DIFF do
        if FRAGMENT ∉ OCR_PAGE then
            report_hidden_text(FRAGMENT)
        else
            report_image_insertion(FRAGMENT)
        end if
    end for
end for
```

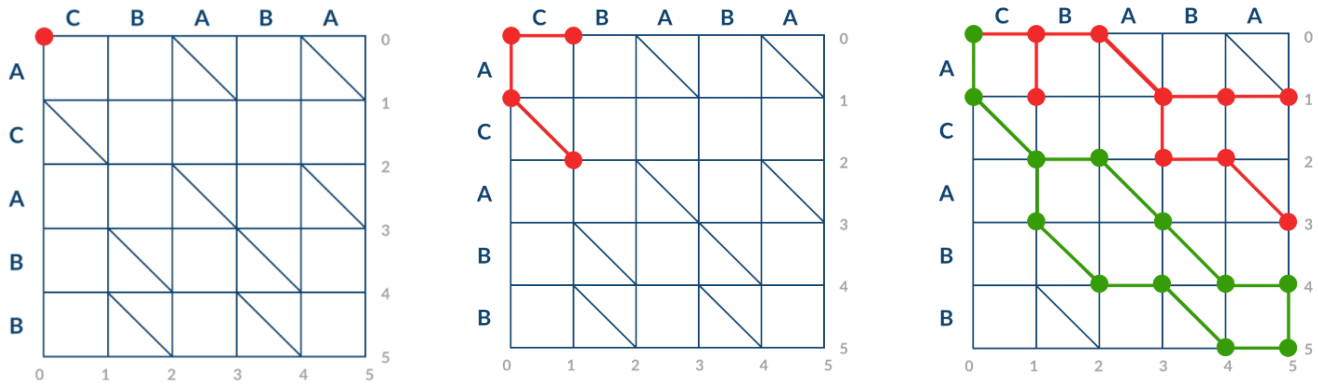
Метод 2: Метод обнаружения скрытого текста и вставок изображений путем сопоставления текстового слоя документа с результатом автоматического распознавания. Для автоматического распознавания текстов используется Tesseract OCR[8].

горизонтали – удалению соответствующего горизонтального символа, а шаг по диагонали – сохранению текущего символа и переход на следующую позицию в строке [63].

2.2.1.2 Алгоритмы терпения и гистограммы

Для сопоставления двух строк и нахождения общих и отличающихся участков, алгоритм терпения, описанный в блоге Брэма Коэна [64], рекурсивно решает задачу нахождения самой длинной общей подстроки. Он учитывает только подстроки, которые являются общими для обоих текстов и появляются только один раз в каждом. После вычисления самой длинной общей подстроки обоих текстов алгоритм начинает аналогичным образом отдельно обрабатывать те участи текста, которые не входят в общую подстроку. Участки текста, которые не попадают в список найденных подстрок, составляют разность двух текстов.

Метод гистограммы является дополненной формой алгоритма терпения и поддерживает выявление редко встречающихся общих элементов. Когда между



а) Исходная позиция. б) Первый шаг. в) Последний шаг.

Рисунок 2.9 — Шаги построения последовательности редактирования строки CBABA в ACABB.

двумя последовательностями имеется уникальный общий элемент, метод гистограммы ведет себя точно так же, как и алгоритм терпения Брэма Коэна.

2.2.1.3 Сравнение разностных алгоритмов

Для разных примеров алгоритмы Майерса, терпения и гистограммы генерируют разные результаты сравнения. Различия заключаются в количестве найденных изменений, порядке измененных участков, а также в обнаруженных добавлений и удалений участков текста. В нашем случае, при выборе наиболее подходящего алгоритма важно было учесть его чувствительность к небольшим изменениям в тексте. Результат OCR может слегка отличаться от соответствующего участка из текстового слоя (например, из-за ошибки распознавания), но такие случаи желательно игнорировать, и поэтому используемый разностный алгоритм должен уметь приоритизировать большие изменения.

Результаты исследования [65] свидетельствуют о том, что результаты метода гистограммы сравнительно лучше по сравнению с методом Майерса с точки зрения ожидаемого результата человеком. Они изучают работу алгоритмов на примерах программного кода, где нужно выявить модифицированные строки кода. Задачу поиска модификаций в тексте на естественном языке можно решить аналогичным образом, если разбить сравниваемые тексты на параграфы и записать каждый параграф на отдельной строке.

2.2.2 Замена омоглифов

Символ – это наименьший элемент письменного языка, имеющий смысловое значение, а глиф – это конкретное изображение, которое представляет символ или часть символа. Омоглифами называются два глифа, которые нельзя различить мгновенным визуальным осмотром. Например, ‘o’ (U+0585) и ‘o’ (U+006F) являются омоглифами. Также, в зависимости от системы рендеринга, используемого шрифта и размера шрифта, два разных символа, глифы которых обычно различимы, могут быть очень похожими. В таблице 6 представлены примеры омоглифов символов армянского алфавита.

Таблица 6 — Примеры символов армянского алфавита и омоглифов.

Глиф	Символ	Омоглифы
h	U+0570	h (U+0068), …
o	U+0585	o (U+006F), o (U+043E), …
u	U+057D	u (U+0075), …
n	U+0578	n (U+006E), …
S	U+054F	S (U+0053), …
.	U+2024	. (U+002E), …
:	U+0589	: (U+003A), …

Юникод, который является универсальным стандартом кодировки символов, потенциально может содержать более одного миллиона символов, включая огромный набор букв, знаков препинания, математических символов, технических символов, стрелок и диакритических знаков, охватывающих все существующие письменные языки. Следовательно, в пространстве Юникода есть много визуально похожих символов, что делает возможным новые типы визуальной маскировки заимствований. При такой маскировке символы в тексте заменяются другими символами (омоглифами), которые визуально похожи.

Например, в армянском языке чтобы скрыть незаконное заимствование, злоумышленник может во всех словах заменить все символы ‘o’ (U+0585), ‘h’ (U+0570) на их латинские омоглифы ‘o’ (U+006F), ‘h’ (U+0068). Слова «հայերեն» и «հայերեն» кажутся идентичными человеку, однако во втором слове армянский символ U+0570 заменен латинским U+0068, и если система обнаружения заимствований не предназначена для работы с такими изменениями, то этот трюк может быть использован для скрытия заимствований в тексте. Помимо единичных

символов, Юникод также содержит 5 лигатур армянского языка: ﬓ, ﬔ, ﬕ, ﬖ, ﬗ, которые могут быть использованы для замены отдельных глифов соответствующих символов. Перечисленные лигатуры редко встречаются в современной литературе, поэтому злоумышленник может потенциально использовать их для искажения текста и усложнения его обработки системой выявления заимствований.

Исследования показывают, что многие системы обнаружения заимствований не находят сходство между исходным текстом и текстом, зашифрованным с помощью омоглифов [66]. Гиллам и др. [67] использовали замену омоглифов в качестве стратегии обфускации для тестирования систем обнаружения заимствований. Их результаты показали, что шесть из семи протестированных систем не смогли обнаружить какое-либо сходство между источником и замененным текстом. В экспериментах Вебер-Вульф и др. [68] также 13 из 15 систем обнаружения заимствований не смогли найти сходство между оригинальным текстом и его версией, где заменены омоглифы.

Исследования методов обнаружения замены омоглифов весьма актуальны, так как данный визуальный обман применяется не только против систем выявления заимствований в документах, а также считается серьезной проблемой веб-безопасности [69; 70].

При работе с текстом имеется доступ к кодам символов и, следовательно, можно программным путем автоматически проверять принадлежность символов к нужному алфавиту. Например, обрабатывая «hայերեն» как строку символов, легко обнаружить, что он состоит из смешанного алфавита (Таблица 7).

Таблица 7 — Разбор строки «hայերեն», маскирующего слово «հայերեն».

Глиф	Символ	Название
h	U+0068	LATIN SMALL LETTER H
ա	U+0561	ARMENIAN SMALL LETTER AYB
յ	U+0575	ARMENIAN SMALL LETTER YI
ե	U+0565	ARMENIAN SMALL LETTER ECH
ր	U+0580	ARMENIAN SMALL LETTER REH
ե	U+0565	ARMENIAN SMALL LETTER ECH
ն	U+0576	ARMENIAN SMALL LETTER NOW

Однако присутствие символов нескольких алфавитов еще недостаточное основание для выделения слова как подозрительного. Многие валидные слова в

естественном языке также могут содержать символы другого алфавита (например, научные термины, как «γ-ճառագայթում», или падежные формы собственных имен, как «MP3-ի»). Поэтому важно дополнительно установить, что присутствие подозрительного символа является именно результатом замены омоглифом.

Есть несколько стратегий защиты от маскировок с применением омоглифов. Одна из стратегий применение Punycode [71], который безвозвратно преобразует символы, отличные от ASCII, в символы ASCII (например, «München» преобразуется в «Mnchen-3ya»). Данная стратегия эффективна для преобразования доменных имен в браузерах, но не подходит для автоматического обнаружения омоглифов в системах проверки уникальности документов. Другой подход – раскраска символов путем присвоения разных цветов символам разных алфавитов [72], также может быть применен для увеличения веб-безопасности, но, как и Punycode, не помогает с обнаружением именно случаев замены омоглифов и требует визуального контроля.

Другой стратегией, позволяющей находить именно замену именно омоглифов, является построение и использование списков омоглифов. Имея списки омоглифов, далее можно с помощью простых правил обнаружить их замену в тексте. Фу и др. [69] составляют списки омоглифов UC-SimList для символов Юникода на основе визуального и семантического сходства между каждой парой символов. Для оценки визуального сходства они используют расхождение Кульбака–Лейблера ядерных оценок плотности (KDE) двумерных изображений глифов. предлагают альтернативное решение проблемы определения степени сходства между двумя глифами: они считают два изображения совершенно непохожими, если они случайны по отношению друг к другу, другими словами, если невозможно описание одного с помощью другого. Руководствуясь этим, они применяют нормализованное расстояние сжатия (NCD) для оценки степени сходства двух глифов на основе теории сложности Колмогорова для измерения случайности. Авторы также проводят эксперимент и показывают на наборе из примерно 6200 символов Юникода, что существует обратная зависимость между значением NCD и степенью сходства глифов.

В работе [66] представлены два подхода к обнаружению заимствований в текстах, содержащих маскировку омоглифами. Первый подход использует список визуально похожих символов Confusables[9] для замены омоглифов в тексте визу-

[9] http://unicode.org/reports/tr36/confusables.txt

ально идентичными символами ASCII. Второй подход использует показатель подобия, вычисленный с использованием нормализованного расстояния Хэмминга. Несмотря на то, что с первым подходом вероятны случаи, когда замена на ASCII производится также для символов, которые не были задуманы как омоглифы, эмпирическое тестирование на наборах данных из PAN-2015 показывает, что оба подхода одинаково хорошо работают для обнаружения заимствований в текстах, содержащих маскировку омоглифами.

Несколько списков омоглифов находятся в свободном доступе (например, homoglyphs.net). Список омоглифов армянского алфавита может быть получен из Unicode Confusables[10]. Используя этот список, можно искать замену омоглифов в текстах на армянском языке: если в слове встречаются символы, принадлежащие другому алфавиту, то оно отмечается как попытка маскировки заимствования в случае, когда все подобные символы в слове имеют омоглиф в армянском алфавите и при их замене соответствующим армянским символом, получаемое слово имеется в заранее подготовленном словаре (метод 3).

```
1: REPLACED = NULL
2: for all CHAR ∈ WORD do
3:     if CHAR ∉ ARMENIAN_ALPHABET and CHAR ∈
       HOMOGLYPHS_OF_ARMENIAN then
4:         REPLACED = WORD.replace(CHAR, get_armenian_homoglyph(CHAR))
5:     end if
6: end for
7: if REPLACED ∈ DICTIONARY then
8:     highlight_as_suspicious(WORD)
9: end if
```

Метод 3: Метод обнаружения замены омоглифов.

2.3 Выводы

В данной главе были изучены внутренние методы обнаружения заимствований, в частности стилометрический анализ и выявление технических методов маскировки.

[10]https://www.unicode.org/Public/security/8.0.0/confusables.txt

Для создания инструмента стилометрического анализа армянских текстов были проведено исследование лучших подходов с серии конференций PAN, в результате которого был выбран и реализован подход на основе иерархической кластеризации. Для признакового описания стиля написания текста для армянского языка были скомпилированы лингвистические ресурсы (списки аббревиатур, редких, жаргонных слов), которые могут быть использованы в дальнейшем в разработке других моделей обработки текстов. Экспериментами было установлено, что реализованные стилометрические методы работают лучше, чем случайные базовые решения, но тем не менее не показывают достаточно высокий уровень точности для применения на практике. По этой причине, помимо стилометрического анализа также необходимо искать решения проблемы обнаружения заимствований путем внешних методов.

Также, были исследованы распространенные методы технической маскировки заимствований и искусственного увеличения степени уникальности работы и методы их обнаружения. На основе существующих подходов для других языков, адаптированы и предложены механизмы и средства выявления скрытого текста, вставки изображений, и замены омоглифов в текстах на армянском языке. Эти методы внутреннего анализа, вместе со стилометрическими, были реализованы в программной системе по обнаружению заимствований в текстах.

Глава 3. Внешние методы обнаружения заимствований

Внешние методы обнаружения заимствований сравнивают подозрительные документы с проверочным набором документов с целью нахождения потенциальных источников заимствований. В качестве проверочной базы могут служить как заранее подготовленная коллекция документов, так и ресурсы из Интернета. В этих методах как правило сначала выполняется глобальный анализ сходства, где извлекаются все документы, которые превосходят установленный порог сходства с проверяемым документом, и затем выполняется локальный, более детальный анализ документов с целью нахождения заимствованных участков текста. Результатом проверки является список обнаруженных заимствований с указанием источника.

Данная глава посвящена внешним методам выявления заимствований. В первом разделе описываются глобальные методы, в том числе алгоритмы нахождения нечетких дубликатов, методы генерации эффективных запросов для нахождения дубликатов в Интернете. Во втором разделе описаны локальные методы анализа сходства, включая исследование и разработка моделей обнаружения парафраза для армянского языка.

3.1 Глобальные методы анализа сходства

Глобальные методы анализа сходства используются для сравнения текстовых документов и применяются в задаче нахождения дубликатов документов. Такими методами являются методы отпечатков, которые работают на основе выделения некоторого подмножества подстрок текста и их хеширования. В этих методах текст документа делится на последовательности токенов (например, на N-граммы слов или символов), которые затем преобразуются в числовую форму и формируют отпечаток документа. Сходство документов оценивается путем сравнения их отпечатков.

3.1.1 Метод отпечатков

Для нахождения полных дубликатов можно использовать хеширование MD5 или SHA, однако такой упрощенный подход не позволит уловить намного более распространенное, почти полное дублирование текстов. Такие неполные копии документов в литературе известны как нечеткие дубликаты.

Одним из общепринятых подходов к обнаружению нечетких дубликатов текстовых документов – это обнаружение перекрывающихся отпечатков. Совпадение одного или нескольких отпечатков двух документов указывает на возможное повторное использование текста между этими документами. Подход с использованием отпечатков с точки зрения производительности эффективнее, чем прямое сравнение текстов целиком, так как сравниваются только выбранные небольшие наборы отпечатков. Лулу и др. [73] выделяют 2 основные группы алгоритмов извлечения отпечатков:

1. С частичным совпадением: K-gram, 0 mod p, отсеивание (winnowing). Эти алгоритмы проходят по тексту используя скользящее окно, передвигаясь на один токен в каждом шаге. Алгоритм K-gram, также известный как метод шинглов, таким образом составляет отпечаток на основе всех множеств k-токенов (токен может быть словом или символом). Алгоритм 0 mod p, в отличие от K-gram, использует сокращенный набор отпечатков, используя только отпечатки, делимые на простое число p. Алгоритм отсеивания использует дополнительное скользящее окно над всеми отпечатками, выбирая отпечаток с наименьшим хешем на каждом шаге;
2. Непересекающиеся отпечатки: Hash-breaking, Discrete Cosine Transformation. В отличие от первой группы, эти алгоритмы разбивают текст на непересекающиеся участки. Hash-breaking работает аналогично алгоритму 0 mod p;

Помимо перечисленных подходов на основе отпечатков, есть другие глобальные методы обнаружения нечетких дубликатов (LSH, Fuzzy, LSA). На практике в задаче нахождения заимствований широко применяется метод шинглов, который более подробно описан в следующем разделе. В статье [74] провели сравнительный анализ вышеперечисленных алгоритмов с точки зрения полноты и точности, и установили эффективность метода шинглов по сравнению с остальными.

3.1.2 Метод шинглов

Метод шинглов позволяет находить нечеткие дубликаты, когда содержание одного документа почти идентично содержанию другого. Данный метод позволит найти нечеткие дубликаты, где разница документов составляет лишь небольшой участок текста (например, когда заменены только дата и имя автора документа, добавлены/удалены небольшие фрагменты текста), или когда тексты почти совпадают за исключением маленького количества слов в предложениях (например, замена всех вхождений слова его синонимом).

В этом разделе представляется описание метода из [75]. Для данного натурального числа k и последовательности токенов в документе d, k-шинглы определяются как множество всех последовательностей из k соседних токенов документа d. Для задач информационного поиска и обнаружения нечетких дубликатов типичным значением для k является 4. Например, 4-шинглами текста «Ունեցիր թեկուզ փոքր խելք, բայց քոնը» будут «Ունեցիր թեկուզ փոքր խելք», «թեկուզ փոքր խելք, բայց», «փոքր խելք, բայց քոնը». Два документа считаются нечеткими дубликатами, если наборы шинглов, сгенерированные из них, почти совпадают.

Обозначим через $S(d_j)$ набор шинглов из документа d_j. Для наборов шинглов $S(d_1)$ и $S(d_2)$ двух документов вычислим коэффициент Жаккара, который измеряет степень перекрытия между множествами:

$$J(S(d_1),S(d_2)) = \frac{|S(d_1) \cap S(d_2)|}{|S(d_1) \cup S(d_2)|} \tag{3.1}$$

Используя коэффициент Жаккара для наборов $S(d_1)$ и $S(d_2)$ как оценку сходства документов d_1 и d_2, с помощью установленного порога (например, 0.9) можно определить являются ли документы нечеткими дубликатами или нет [75]. Пороговое значение для степени совпадения необходимо подобрать таким образом, чтобы полнота системы обнаружения заимствований оставалась высокой, так как ложноположительные случаи потом можно будет отфильтровать на этапе детального анализа документов.

В реализации данного алгоритма, чтобы избежать попарного вычисления коэффициентов Жаккара, используется хеширование. Для каждого шингла вычисляется хеш (64-битный, например), и таким образом для каждого документа d_j получается набор таких хеш-значений $H(d_j)$. Пусть π будет случайной переста-

новкой 64-битных целых чисел в 64-битные целые числа. Обозначим через $\Pi(d_j)$ множество переставленных хеш-значений в $H(d_j)$, и тогда для каждого $h \in H(d_j)$ существует соответствующее значение $\pi(h) \in \Pi(d_j)$. Пусть $x_{\pi j}$ наименьшее целое число в $\Pi(d_j)$. Тогда $J(S(d_1), S(d_2)) = P(x_1^\pi = x_2^\pi)$.

Таким образом, тест на коэффициент Жаккара наборов шинглов является вероятностным: сравниваются вычисленные значения x_i^π из разных документов d_i. Если значения совпадают, то документы потенциально могут быть нечеткими дубликатами. На практике, данный процесс повторяется независимо для 200 случайных перестановок π. Набор из 200 результирующих значений x_i^π называется эскизом $\psi(d_i)$ документа d_i. В этом случае для пары документов d_i, d_j коэффициент Жаккара уже оценивается как $|\psi_i \cap \psi_j|/200$, и, если это превышает установленный порог, считается, что d_i и d_j являются нечеткими дубликатами [75].

3.1.3 Веб-поиск

Многие авторы используют в своих работах заимствования из Интернет-ресурсов с открытым доступом, и поэтому актуальна разработка методов выявления таких заимствований. Исследованиям методов поиска источников были посвящены серии конференций PAN. На PAN 2015[76], последней из серии, было представлено 5 подходов. В этих решениях выделялись следующие подзадачи: сегментация входного текста на части, извлечение ключевых словосочетаний, формулировка запроса, управление поиском и фильтрация результатов. Ниже приводится описание предложенных решений на основе обзорной статьи Хагена и др.[76].

3.1.3.1 Сегментация текста

Для обнаружения заимствований из Интернета текст подозрительного документа сначала сегментируется на небольшие участки, и далее каждый участок текста обрабатывается индивидуально. Это делается с целью увеличения точно-

сти поиска, так как разные фрагменты текста могут быть заимствованы из разных источников. Участниками PAN 2015 были использованы разные подходы для разбиения текста на участки: фрагменты из 500 слов[77], параграфы [78; 79], или отдельные предложения и заголовки [80; 81]. За исключением [79], в результате сегментации получаются неперекрывающиеся фрагменты. Проблема использования фрагментов из 500 слов или фиксированного количества строк заключается в том, что типичные заимствования не имеют фиксированной длины. С другой стороны, применение предложений или перекрывающихся фрагментов приводит к сравнительно большому количеству запросов. По этим причинам использование параграфов в качестве фрагментов для дальнейшего формирования запросов и их обработки является разумным компромиссным решением.

3.1.3.2 Извлечение ключевых словосочетаний

Из каждого полученного фрагмента текста извлекаются «ключевые словосочетания», чтобы с их помощью формулировать запросы. Цель извлечения ключевых словосочетаний – отбор только тех фраз или слов, которые увеличивают вероятность нахождения исходных документов, соответствующих подозрительному документу. Извлечение ключевых словосочетаний также служит средством ограничения количества формулируемых запросов, тем самым снижая общие затраты на использование поисковой системы. От выбора алгоритма извлечения ключевых фраз зависит производительность системы и полнота результатов: чем меньше ключевых слов извлекается, тем точнее должен быть отбор, иначе полнота результатов будет низкая, но и с увеличением количества ключевых фраз, также растет число запросов, что увеличивает время обработки текста. По этой причине, базовые решения, как например, извлечение и использование всех словосочетаний не подходят и необходимо применение более селективного подхода.

В большинстве решений из PAN 2015 выполняется предобработка проверяемого текста и удаляются стоп-слова. В качестве ключевых словосочетаний некоторые участники используют отдельные слова, а другие – целые фразы. В частности, Хан [80] и Конг и др. [81] извлекают существительные и глаголы в качестве ключевых слов. Авторы [77] используют слова с наибольшим значением tf-idf. Ра-

ви Н. и Гупта [78] также используют отдельные слова в качестве ключевых слов (глаголы, существительные и прилагательные), также используя tf-idf для оценки важности.

3.1.3.3 Формулировка запроса

Большинство участников PAN 2015 составляют запрос путем конкатенации первых k токенов с наибольшим tf-idf коэффициентом, где варьируется в зависимости от метода. Следующие k токенов затем используются в следующем запросе и т.д. Таким образом, для отдельных фрагментов создаются в основном неперекрывающиеся запросы. В решениях Хан [80] и Конг и др. [81] каждый запрос формируется из ключевых слов (с ограничением максимального количества слов в запросе), извлеченных из отдельного предложения. В решении [77] используют более сложную схему: сначала идентифицируют 10 слов с наивысшими показателями tf-idf для каждого фрагмента, затем выбирают три предложения соответствующего фрагмента с этими ключевыми словами и составляют запросы на основе ключевых слов в этих предложениях. Рави Н. и Гупта [78] составляют по два запроса на параграф из слов с наибольшим tf-idf коэффициентом. Сухомель и Брандейс [79] используют следующие запросы: запросы с 6 словами на уровне документа, запросы из их сочетаний, и отдельные запросы с top-10 словами по tf-idf для каждого фрагмента текста.

3.1.3.4 Управление поиском

После формирования запросов на основе ключевых словосочетаний, они дополнительно адаптируются под API используемой поисковой системы, так как поисковые системы имеют ряд ограничений насчет формата и содержания запроса (например, системы позволяют лишь ограниченное число слов в запросе). Также важен порядок отправления запросов и обработки результатов, так как в зависимости от нее можно динамически корректировать поиск на основе результатов

каждого запроса. Корректировка может включать в себя удаление, переформулировку запросов, или формирование новых на основе обратной связи по релевантности, полученной из результатов поиска. Например, в [77] отбрасывают запрос, если более 60% его слов содержится в уже загруженном документе. Рави Н. и Гупта [78] удаляют повторяющиеся запросы. В решении Сухомель и Брандейс [79], если фрагмент текста уже был сопоставлен с источником, то остальные запросы, сформированные на его основе, не отправляются и удаляются из списка запросов. В зависимости от того, сколько источников заимствования содержится в длинном абзаце, из-за этого потенциально можно упустить другие источники заимствования (например, когда использовались два источника, а один уже был найден).

Еще одним интересным аспектом исследования является планирование самих запросов. Результаты экспериментов показывают, что ранняя отправка запросов уровня документа уже гарантирует некоторую полноту результатов (например, [79]). Также, присутствие заимствований в начале документа, как правило, менее вероятно, поэтому отправка запросов в том порядке, в котором соответствующие фрагменты встречаются в документе, может быть неоптимальным подходом с точки зрения обеспечения высокого уровня полноты за меньшее число запросов.

3.1.3.5 Фильтрация результатов

После получения результатов поиска имеет смысл дополнительно их отфильтровать с целью удаления менее релевантных документов, для которых нецелесообразно выполнение детального сравнения с проверяемым документом. Такое сокращение набора кандидатов позволяет экономить ресурсы на последующем более трудоемком этапе подробного сравнения. В решениях PAN 2015 в большинстве случаев количество результатов для каждого запроса не превышало 10, поэтому проблема фильтрации результатов не стояла. К тому же, в случае агрессивной фильтрации результатов, используемый алгоритм должен быть точным, что еще больше усложняет стратегию поиска. Поскольку предобработка результатов поиска не является самым критичным узким местом, слишком большое сокращение результатов не кажется самой многообещающей стратегией.

3.1.3.6 Обсуждение

Таблица 8 — Сравнение алгоритмов поиска.

	Конг (2015)	Рафией	Рави Н.	Сухомель	Уильямс (2014)	Kong (2013)	Пракаш
Предобработка	Нет	Удаление стоп-слов	Нет	Удаление пунктуации	Удаление стоп-слов	Нет	Удаление стоп-слов
Фрагменты	Предложения	Участки из 500 слов + предложения	Параграфы	Параграфы	Участки из 5 предложений	Предложения	Параграфы
Формирование запроса	Конкатенация top-10 существительных, глаголов на основе tf-idf	Конкатенация top-10 существительных, прилагательных, глаголов на основе tf-idf	Конкатенация существительных, прилагательных, глаголов с высоким tf-idf	Конкатенация top-6 словарных биграмм и триграмм на основе tf-idf	Конкатенация существительных, прилагательных, глаголов	Конкатенация токенов с высоким tf-idf, с дополнительным взвешиванием в зависимости от положения предложения	Конкатенация 10 самых «информативных» существительных
Количество запросов на уровне документа	Нет	Нет	Нет	Да	Нет	Нет	Нет

Таблица 9 — Производительность и экономическая эффективность алгоритмов поиска источника [76].

Подход	Год	F1	Точность	Полнота	Кол-во запросов	Кол-во загрузок	Кол-во запросов до 1-го обнаружения	Кол-во загрузок до 1-го обнаружения)	#(Нет обнаружения)	Время выполнения
Elizalde	2013	0.16	0.12	0.37	41.6	83.9	18.0	18.2	4	11:18:50
Elizalde	2014	0.34	0.40	0.39	54.5	33.2	16.4	3.9	7	04:02:00
Foltynek	2013	0.11	0.08	0.26	166.8	72.7	180.4	4.3	32	152:26:23
Gillam	2013	0.06	0.04	0.15	15.7	86.8	16.1	28.6	34	02:24:59
Haggag	2013	0.38	0.67	0.31	41.7	5.2	13.9	1.4	12	46:09:21
Han	2013	0.36	0.55	0.32	194.5	11.8	202.0	1.7	12	20:43:02
Kong	2013	0.01	0.01	0.59	47.9	5185.3	2.5	210.2	0	106:13:46
Kong	2014	0.12	0.08	0.48	83.5	207.1	85.7	24.9	6	24:03:31
Kong	2015	0.38	0.45	0.42	195.1	38.3	197.5	3.5	3	17:56:55
Lee	2013	0.40	0.58	0.37	48.4	10.9	6.5	2.0	9	09:17:10
Prakash	2014	0.39	0.38	0.51	60.0	38.8	8.1	3.8	7	19:47:45
Rafiei	2015	0.12	0.08	0.41	43.5	183.3	5.6	24.9	1	08:32:37
Ravi N	2015	0.43	0.61	0.39	90.3	8.5	17.5	1.6	8	09:17:20
Suchomel	2013	0.05	0.04	0.23	17.8	283.0	3.4	64.9	18	75:12:56
Suchomel	2014	0.11	0.08	0.40	19.5	237.3	3.1	38.6	2	45:42:06
Suchomel	2015	0.09	0.06	0.43	42.4	359.3	3.3	39.8	4	161:51:26
Williams	2013	0.47	0.60	0.47	117.1	12.4	23.3	2.2	7	76:58:22
Williams	2014	0.47	0.57	0.48	117.1	14.4	18.8	2.3	4	39:44:11
Zubarev	2014	0.45	0.54	0.45	37.0	18.6	5.4	2.3	3	40:42:188

В таблицах 8 и 9 показаны характеристики пяти алгоритмов нахождения заимствований, которые были предложены в соревновании PAN по информационному поиску в 2015 году, а также лучших решений прошлых лет. В настоящее время не существует единой формулы для организации поиска и экономической эффективности в одну оценку. Например, самый быстрый подход также имеет наибольшее количество случаев отсутствия обнаружений, или высокая полнота

достигается при большом количестве запросов. Как можно видеть на таблице 9, не существует единого детектора, который показывал бы лучший результат по всем метрикам.

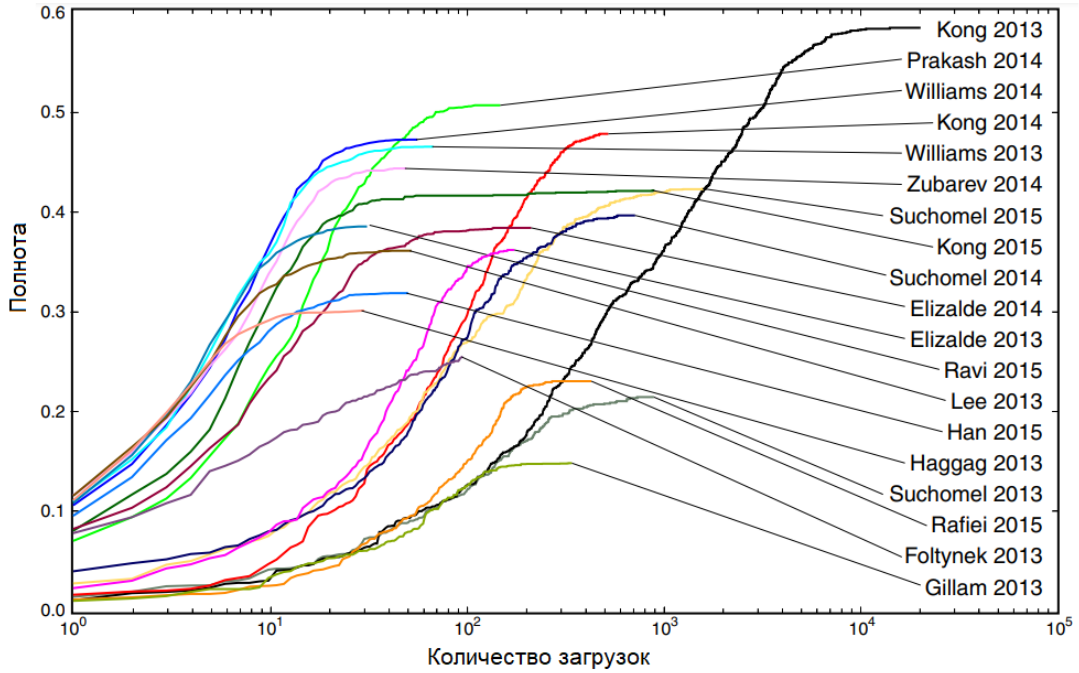


Рисунок 3.1 — Зависимость полноты от количества загрузок для разных моделей [76].

Целью данных алгоритмов поиска является достижение максимально высокого уровня полноты при разумной рабочей нагрузке (нагрузка определяется количеством запросов и загруженных результатов). Поскольку загрузка результатов поиска менее затратна, чем отправка запросов к API поисковых систем, наиболее интересными показателем является уровень полноты результатов в зависимости от количества отправленных запросов. Видно, что полнота постепенно увеличивается в течение всего процесса загрузки результатов, а не при самых первых загрузках (Рис. 3.1). С точки зрения экономии ресурсов на отправку запросов и загрузку результатов, можно заметить, что некоторые подходы с низкой рабочей нагрузкой достигают более высокие уровни полноты при одинаковом количестве загрузок, в то время как подходы с большим количеством загрузок обычно достигают своих лучших конечных уровней полноты только при гораздо большем количестве загрузок.

В системе обнаружения заимствований для конечного пользователя наиболее важной метрикой качества является полнота результатов поиска источников. Если система не обнаружит источник на этапе глобального поиска, далее на этапе

детального анализа уже невозможно будет выявить заимствованные из него участки текста. Именно поэтому алгоритм поиска источников в первую очередь должна ориентироваться на достижение высокого уровня полноты, используя умеренное количество запросов и загрузок. Наилучший уровень полноты достигается с помощью алгоритма Конга и др. 2013 года [82], но, поскольку два других показателя производительности данного подхода имеют самые низкие результаты, было решено использовать вместо этого алгоритм Пракаша [83], который также имеет сравнительно высокий уровень полноты, но при гораздо меньшем количестве обращений к поисковой системе.

3.1.4 Метрики оценки качества

В качестве метрик для количественной оценки алгоритма обнаружения заимствований используются точность и полнота. Дополнительно, вводится понятие гранулярности, которая определяет, правильно ли распознаны смежные заимствованные фрагменты. Низкая степень гранулярности упрощает проверку человеком автоматически обнаруженных подозрительных участков. Эти меры качества применяются как изолированно, так и комбинированно, предоставляя единую общую оценку эффективности алгоритма [84].

Используя определение [84], пусть d_{plg} обозначает документ, содержащий заимствование. Примером заимствования в d_{plg} является кортеж $s =< s_{plg}, d_{plg}, s_{src}, d_{src} >$, где s_{plg} – это заимствование в d_{plg}, а s_{src} – его исходный вариант в некотором исходном документе d_{src}. Аналогично, обнаруженное заимствование для документа d_{plg} обозначается как $r =< r_{plg}, d_{plg}, r_{src}, d'_{src} >$, где r ассоциирует предположительно заимствованный фрагмент текста r_{plg} в d_{plg} с фрагментом текста r_{src} в d'_{src}. Мы говорим, что r обнаруживает s тогда и только тогда, когда $r_{plg} \cap s_{plg} = \emptyset$, $r_{src} \cap s_{src} = \emptyset$ и $d'_{src} = d_{src}$. Для документа d_{plg} предполагается, что разные заимствованные фрагменты d_{plg} не пересекаются. В отношении обнаруженных заимствований такое ограничение не применяется. S и R обозначают наборы истинных и обнаруженных случаев заимствования. Используя эти обозначения, микро-усредненная точность и полнота обнаружений R при истинных случаях S определяются следующим образом:

$$prec_{micro}(S,R) = \frac{|\bigcup_{(s,r)\in(S\times R)}(s \sqcap r)|}{|\bigcup_{r\in R} r|} \tag{3.2}$$

$$rec_{micro}(S,R) = \frac{|\bigcup_{(s,r)\in(S\times R)}(s \sqcap r)|}{|\bigcup_{s\in S} s|} \tag{3.3}$$

а макро-усредненная точность и полнота следующим образом:

$$prec_{macro}(S,R) = \frac{1}{|R|}\sum_{r\in R}\frac{|\bigcup_{s\in S}(s \sqcap r)|}{|r|} \tag{3.4}$$

$$rec_{macro}(S,R) = \frac{1}{|S|}\sum_{s\in S}\frac{|\bigcup_{r\in R}(s \sqcap r)|}{|s|} \tag{3.5}$$

где:

$$s \sqcap r = \begin{cases} s \cap r, & \text{если } r \text{ заимствован из } s \\ \emptyset, & \text{иначе.} \end{cases} \tag{3.6}$$

Гранулярность характеризует способность алгоритма обнаруживать случай заимствования $s \in S$ целиком. Чем выше значение гранулярности, тем чаще алгоритм находит заимствование по частям, а не целиком. Заимствования, обнаруженные идеальным алгоритмом, должны однозначно совпадать с истинными случаями из S. Гранулярность вычисляется следующим образом:

$$gran(S,R) = \frac{1}{|S_R|}\sum_{s\in S_R}|R_s| \tag{3.7}$$

где $S_R \subseteq S$ – истинные заимствования, обнаруженные в R, а $R_s \subseteq R$ – обнаруженные участки, соответствующие истинному заимствованию s. Область значений $\text{gran}(S,R) - [1, |R|]$, где 1 достигается при взаимно однозначном соответствии истинных и обнаруженных случаев, а $|R|$ соответствует худшему случаю.

Точность, полнота и гранулярность позволяют по отдельности оценивать разные аспекты алгоритмов обнаружения заимствований. Эти меры также могут быть объединены в единую оценку качества обнаружения, называемую $plagdet$, следующим образом:

$$plagdet(S,R) = \frac{F_\alpha}{\log_2(1 + gran(S,R))} \tag{3.8}$$

где F_α обозначает взвешенное гармоническое среднее значение точности и полноты. На практике, как правило, используется $\alpha = 1$, то есть точность и полнота одинаково взвешены [84].

В статье [85] изучается производительность $plagdet$, основного средства оценки систем обнаружения плагиата, на перефразированных вручную наборах данных о плагиате. Авторы обнаружили подверженность этой метрики ошибкам при определенных условиях. Есть свидетельства того, что люди, выполняющие неограниченное по длине задание, склонны делать заимствованные тексты короче. Резюмирование и перефразирование – основные методы, которыми пользуются студенты для преобразования текстов. Также, в случаях непреднамеренного плагиата и плагиата идей детали исходных текстов обычно опускаются или забываются. В обоих случаях получаются заимствования, которые по размеру меньше оригинала. В подобных ситуациях plagdet в своем оригинальном варианте, где вычисляется пересечение обнаруженного плагиата и истинного значения, не является эффективной мерой качества.

3.1.5 Локальные методы анализа сходства

В предыдущем разделе были рассмотрены алгоритмы обнаружения нечетких дубликатов документов. Данный раздел посвящен локальным методам анализа сходства, которые применяются для выявления заимствований на уровне небольших участков текста. В группу этих методов входят как алгоритмы нахождения лексически близких заимствований, так и обнаружение парафраза.

Помимо заимствования путем простого копирования, часто авторы прибегают к разным механизмам преобразования текста с целью скрытия факта заимствования. Такими механизмами могут быть перестановка слов в предложении, изменение окончаний, замена синонимов, изменения синтаксической структуры или парафраз. Перестановка слов или изменение окончаний могут быть легко обнаружены путем нормализации текста с помощью лемматизации и представления в виде «мешка слов». Полное совпадение или высокое значение оценки сходства будет считаться признаком заимствования. Для оценки сходства могут быть использованы коэффициенты Жаккара или Шимкевича-Симпсона, представленные

ниже. Обнаружение замены синонимов и парафраза требует более сложных решений.

В случае синонимов сходство текстов можно установить используя списки синонимов. Например, для многих языков доступны такие системы, как WordNet[1], которые могут для извлечения синонимов, однако для армянского языка аналогичного ресурса нет в открытом доступе. В этой работе задача обнаружения замены синонимов рассматривалась как частный случай парафраза, для обнаружения которого было проведено исследование, описанное в последнем разделе этой главы.

3.1.5.1 Коэффициент Жаккара

Одним из самых простых методов оценки лексической близости текстов является коэффициент Жаккара, который для двух фрагментов текста A и B считается следующим образом:

$$J(A,B) = \frac{A \cap B}{A \cup B} \tag{3.9}$$

где $|A \cap B|$ - множество слов, встречающихся в обоих фрагментах, а $|A \cup B|$ - объединение. Область значений коэффициента Жаккара – $[0, 1]$, где 0 соответствует случаю, когда фрагменты не содержат ни одного общего слова, а 1 достигается в случае, когда множества слов фрагментов совпадают. Если сходство превышает установленный порог, то первый фрагмент текста считается потенциальным заимствованием второго.

Недостаток коэффициента Жаккара в том, что разница длин фрагментов может значительным образом повлиять на его значение, и даже когда A является подстрокой B, значение коэффициента может быть близким к 0, если B намного длиннее A. Такое поведение не желательно при поиске заимствований, где лишь часть текста заимствована.

[1] https://wordnet.princeton.edu/

3.1.5.2 Коэффициент Шимкевича-Симпсона

Для двух фрагментов текста A и B коэффициент Шимкевича-Симпсона (также встречается под названием «коэффициент перекрытия», overlap coefficient) считает отношение мощности множества общих слов в двух фрагментах к мощности наименьшего множества слов:

$$O(A,B) = \frac{A \cap B}{min(|A|, |B|)} \tag{3.10}$$

Область значений этого коэффициента также $[0, 1]$, где 0 соответствует случаю, когда фрагменты не содержат ни одного общего слова, а 1 достигается в случае, когда одно из множеств является подмножеством другого. Данный коэффициент устраняет недостаток коэффициента Жаккара, когда при большой разнице длин фрагментов штрафовалась оценка близости фрагментов. Когда A является подстрокой B, значение коэффициента равно 1.

Как и в случае коэффициента Жаккара, для коэффициента Шимкевича-Симпсона тоже, если сходство превышает установленный порог, то первый фрагмент текста считается потенциальным заимствованием второго.

3.1.5.3 Метод отпечатков

Помимо обнаружения дубликатов документов, метод отпечатков также применяется для обнаружения нечетких дубликатов на локальном уровне. Данный метод, предложенный в [86], представляет каждый участок текста (предложение или параграф) наименее частыми символьными 4-граммами (использование 4-грамм вместо N-грамм другой длины было рекомендовано авторами как наиболее эффективный выбор), так как меньшие значения не обеспечивают хорошего различения между предложениями, а при больших значениях увеличивается чувствительность к лексическому совпадению текстов (алгоритм будет более строго отбирать нечеткие дубликаты). На практике при сравнении текстов учитываются 3 наименее частые 4-граммы. Например, для строки S = «Երկինք մի մաքուր, արև մի բոսոր» набор из 4-грамм будет включать, «րկին», «կինք», «մաքո», «աքու» и

т.д. (пробелы и короткие слова игнорируются). Далее выделяются 3 наименее частые 4-граммы («Երկի», «բnun», «nunp» в данном случае). Отпечаток строки формируется путем конкатенации этих 4-грамм: «Երկիբnunnunp». Два предложения считаются одинаковыми, если их отпечатки совпадают.

Данный метод в состоянии обнаружить заимствования, слегка отличающиеся от первоисточника, однако все равно чувствителен к переформулировкам и поэтому не способен обнаружить случаи, где присутствует замена синонимов или парафраз [87].

3.1.5.4 Обнаружение парафраза

Коэффициенты Жаккара, Шимкевича-Симпсона, а также метод отпечатков не способны выявлять заимствования, содержащие замену синонимов или парафраз. Замену синонимов можно было бы обнаружить при наличии ресурсов как WordNet, которые предоставляют списки синонимов. Однако для армянского языка в открытом доступе такого ресурса не было обнаружено. По этой причине в этой работе замена синонимов рассматривается как частный случай парафраза и решается вместе с задачей обнаружения парафраза.

Обнаружение парафраза – это задача проверки семантической идентичности пары фрагментов текста. Поскольку формального определения парафраза нет, исследователи полагались на методы, основанные на данных, при решении задач его обнаружения. Для таких методов необходимо наличие корпусов с перефразированными текстами.

Обзор литературы показал, что общедоступных ресурсов по обнаружению перефразирования для армянского языка не существует. По этой причине в рамках этой работы было проведено исследование по созданию корпуса парафразов предложений, который затем был использован для обучения и оценки моделей обнаружения парафраза. Результаты исследования опубликованы в статье [14].

Одной из основных проблем создания корпуса парафраза является получение семантически похожих, но лексически удаленных пар предложений. Существует несколько подходов к созданию парафразов, которые могут быть сгруппированы в (i) моноязычное перефразирование экспертами, (ii) полуавтоматиче-

ское перефразирование с последующим редактированием экспертами, (iii) полностью автоматическое перефразирование. Федерман и др. [88] провели исследование для сравнения этих методов и пришли к выводу, что использование машинного перевода для генерации парафраза – это хорошо работающий подход, который по сравнению с экспертами значительно дешевле и приводит к более разнообразным результатам. В то же время они рекомендуют постобработку полученных парафразов экспертами, чтобы улучшить их плавность (fluency) и адекватность (adequacy). Основываясь на этих рекомендациях, мы применили аналогичный подход и использовали обратный перевод и последующую ручную проверку для создания парафразов армянских предложений. Наш подход отличался от подхода Федермана и др. [88] тем, что мы дважды повторяли этап обратного перевода, чтобы добиться большего лексического разнообразия, а затем на этапе постобработки эксперты только проверяли плавность и адекватность сгенерированных примеров, чтобы исключить неправильные предложения из набора данных.

За исключением метода генерации парафразов эта работа в основном следовала рекомендациям Долана и др. [89] и Пивоваровой и др. [90]. Долан и др. описывают создание корпуса MSRP, состоящего из 5801 пары предложений из новостей и используемого для оценки моделей обнаружения английских парафраз. Методы, которые мы используем в нашей работе для извлечения предложений, также используются в MSRP и описаны в [91] и [92]. Работа Пивоваровой и др. описывает корпус ParaPhraser для русского языка, состоящий из заголовков новостных статей и основанный на [93], где используется метрика сходства для извлечения кандидатов парафраза.

Помимо наборов данных, в этой работе также были разработаны модели обнаружения перефразирования для армянского языка. Принимая во внимание тот факт, что модели машинного обучения, в частности BERT [94], показывают state-of-the-art результаты в задачах обнаружения парафраза за последние несколько лет [95], было решено использовать тонкую настройку Multilingual BERT (M-BERT) для обнаружения перефразирования. M-BERT поддерживает армянский язык, и решение использовать его также объясняется отсутствием моноязычного BERT для армянского языка, обучение которого с нуля было бы затруднительно из-за стоимости и отсутствия больших текстовых корпусов.

Таблица 10 — Оценка качества (F1) моделей обнаружения парафраза на датасете MRPC [94].

Модель	MRPC 3.5k
Pre-OpenAI SOTA	86.0
BiLSTM+ELMo+Attn	84.9
OpenAI GPT	82.3
$BERT_{BASE}$	88.9
$BERT_{LARGE}$	89.3

Обзор литературы Для задачи обнаружения парафраза в последние годы наиболее высокую точность показывают подходы на основе машинного обучения, в частности глубокие нейронные сети [94; 96–98]. На датасете MRPC модель на основе BERT [94] показывает лучший результат по метрике F1 (таблица 10). BERT - языковая модель на основе нейронной сети Transformer [99], которая может быть предобучена на большом количестве неразмеченных текстовых данных и далее использоваться для извлечения контекстных представлений токенов текста.

Для обучения моделей машинного обучения требуются примеры парафразов. Для автоматической генерации таких примеров есть несколько подходов. В статье [100] использовали нейросетевой машинный перевод для генерации парафразов путем перевода двуязычных пар предложений для обучения векторных представлений предложений. Помимо машинного перевода изучались и другие методы генерации парафраза: основанные на правилах [101], на основе обучения с подкреплением [102], seq2seq [103–106].

Многие предыдущие исследования были сосредоточены на поиске естественных парафразов предложений [107–109]. Были попытки создать корпуса на основе субтитров фильмов, как многоязычный корпус Opusparcus [110] для шести языков (немецкий, английский, финский, французский, русский, шведский). Этап извлечения предложений корпуса TMUP аналогичен нашему и основан на двух разных механизмах перевода Google PBMT и NMT [111; 112]. Некоторые языки, такие как арабский, имеют особые правила преобразования, и с их помощью можно перефразировать автоматически, как показано в [113].

Создание обучающих и тестовых наборов данных Корпус парафразов был создан для решения проблемы обучения и оценки моделей их обнаружения. В этом разделе описаны процесс выбора исходного набора предложений, метод генера-

ции парафразов на основе обратного перевода, и детали ручной постобработки полученных примеров.

Выбор предложений Для этой задачи использовались новостные тексты, состоящие из статей, написанных за последние 10 лет, извлеченных с новостных сайтов Hetq (12122 статьи) и Panarmenian (12497 статей). Набор текстов включал статьи на разные темы (политика, спорт, экономика и др.). Эти тексты были разбиты на предложения, и далее из полученного первоначального набора предложений были отфильтрованы предложения, содержащие ссылку или информацию о странице или разделе (например, «Հայաստանի Հանրապետության արտաքին առևտուրը 2007 թվականին, Վիճակագրական Ժողովածու, Երևան , 2008 , էջ 9697:»).

Для некоторых текстов границы предложений были определены неправильно, и в результате получались либо слишком длинные, либо слишком короткие предложения. Чтобы такие пары не появлялись в нашем итоговом наборе, были удалены все предложения, содержащие менее 6 токенов и более 22 токенов (не считая стоп-слов). Кроме того, если предложение содержало три или более одинаковых слова подряд, оно также удалялось.

Генерация парафраза путем обратного перевода Метод, используемый для генерации семантически похожих предложений, основан на переводе предложений с армянского на английский и наоборот. Google Translate – один из немногих доступных переводчиков для армянского языка, демонстрирует относительно высокую точность и поэтому был выбран для перевода. Также рассматривался перевод с армянского на русский язык, однако точность перевода была заметно хуже, чем с английского.

Предложения, выбранные в предыдущем разделе, были переведены туда и обратно (Рис. 3.2). Процесс обратного перевода повторялся дважды. Это было сделано по той причине, что после одной итерации сгенерированные предложения все еще сохраняли высокий уровень лексического и морфосинтаксического сходства, в то время как 3 и более итераций приводили к более высокой доле ошибочных переводов.

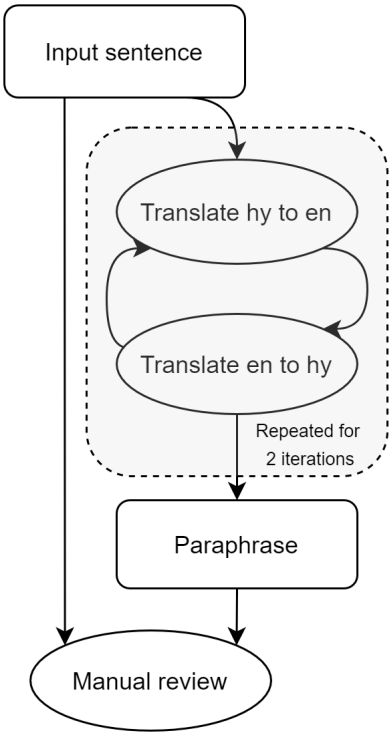


Рисунок 3.2 — Схема генерации парафраза путем перевода из армянского (hy) в английский (en) и обратно.

Таблица 11 — Примеры сгенерированных парафразов (выделены совпадающие слова)

Исходное предложение	Полученный парафраз
Կոռուպցիան չարիք են համարում բոլորը՝ չինովնիկից *մինչև* բանվոր:	*Կոռուպցիան* բոլորի համար *չարիք* է համարվում՝ պաշտոնյաներից *մինչև* աշխատակիցներ:
Քաղաքացիներից մեկն էլ «Հետք»-ին ուղարկած նամակում *նույնիսկ* նշել էր, *որ Հայաստանում Լեհաստանի դիվանագիտական ներկայությունը* վերանայման կարիք ունի:	Մի քաղաքացի *նույնիսկ* «Հետքին» գրեց, *որ Լեհաստանի դիվանագիտական ներկայությունը Հայաստանում* պետք է վերանայվի:
Կարինեն սովորել է *Նոյեմբերյանի պետական* քոլեջի հաշվապահության բաժնում, *վերջերս* ստացել է իր դիպլոմը:	*Կարինեն* սովորում էր հաշվապահություն *Նոյեմբերյանի պետական* քոլեջում և *վերջերս* դիպլոմ ստացավ:

Исходное предложение и его перевод рассматривались как пара предложений в нашем наборе. Из полученного набора были удалены те пары, где в словах переведенного предложения содержались символы двух разных языков.

При наличии идеального переводчика использование этого метода позволило бы получить любое количество пар. Однако Google Translate нередко допускает ошибки перевода, некоторые из которых приводят к бессмысленным переводам или переводам, которые больше не являются парафразом исходного предложения. Поэтому пришлось дополнительно размечать полученные данные, чтобы отделить пары с парафразом от остальных, содержащих неправильный перевод или не парафраз.

Из сгенерированных пар предложений вручную были отфильтровали те, которые содержали синтаксически или семантически некорректные переводы, частично переведенные предложения (т.е. содержали слова на другом языке) или несколько предложений вместо одного. Таким образом, из 4405 рассмотренных пар предложений 1450 были удалены. Остальные 2955 были дополнительно проверены, как описано в следующем разделе.

Ручная постобработка После удаления ошибочных пар была выполнена разметка остальных пар, чтобы решить, это парафраз или нет, в основном полагаясь на суждение аннотаторов. Чтобы повысить согласованность в наборе данных, аннотаторам были даны рекомендации как отличать парафраз от непарафраза, примерно основанные на степени семантического сходства текстов SemEval 2012 года [114]. Пары со степенями сходства 5 («Полностью эквивалентны») и 4 («В основном эквивалентны, но некоторые несущественные детали отличаются») были размечены как парафраз, а степени от 0 («По разным темам») до 3 («Примерно эквивалентны, но некоторые важные информация отличается/отсутствует») были размечены как непарафраз.

Каждому аннотатору был предоставлен список конкретных примеров того, что не следует рассматривать как перефразирование, включая примеры очень близкие к парафразу:

1. Частично перекрывающиеся предложения, как например:
   - Այսօր 100%-ով վերականգնվել է էլեկտրամատակարարումը - հայտարարել է նախարար Խորխե Ռոդրիգեսը:
   - Այսօր էլեկտրաէներգիան վերականգնվել է 100% -ի չափով:

2. Пары, где строго одностороннее логическое следствие. Например:
   - Բայց, միևնույն ժամանակ, դա պարտավորեցնում է, որ էլ ավելի շատ պարապեմ:
   - Բայց, միևնույն ժամանակ, դա ինձ ստիպում է ավելին անել:
3. Пары, где контексты одинаковые, но речь идет о разных сущностях. Например:
   - Լինդան ռուսաստանցի երգչուհի է, որը կատարում է էլեկտրոնային և էթնիկ ոճի երաժշտություն:
   - Սվետլանան ռուս երգչուհի է, ով նվագում է էլեկտրոնային և էթնիկ երաժշտություն:

   Или:
   - Հիշեցնենք, որ նա մեղադրվում է ՀՀ քրեական օրենսգրքի 300.1-րդ հոդվածի 1-ին մասով:
   - Նրան մեղադրանք է առաջադրվել ՀՀ քրեական օրենսգրքի 311-րդ հոդվածի 1-ին մասով:

Набор парафразов был разделен на 2 подмножества: 1573 пар для обучения и 1382 для тестирования, которые затем были проверены вручную с использованием вышеописанного руководства. После ручного изучения 1339 из 1573 обучающих примеров были признаны «парафразом» (85%). Для тестового набора каждая пара была проверена минимум двумя аннотаторами. Разногласия разрешались третим аннотатором. Согласованность разметки, измеренная с помощью Каппа Коэна, варьировалась от 0,55 до 0,65 между парами аннотаторов, что сопоставимо с оценками согласованности для наборов данных MRPC и Paraphraser. После разметки 1021 тестовая пара была помечена как парафраз из 1382 (74%). Суммарно, после ручного просмотра 80% автоматически сгенерированных пар предложений были подтверждены как перефразирование. Остальные пары, которые были сочтены непарафразом, по-прежнему имели высокое семантическое сходство и были примерно эквивалентны, но с некоторыми важными отличающимися деталями (Рис. 12).

Следуя [88], мы также проверили разнообразие сгенерированных парафразов, вычислив среднее расстояние редактирования на уровне слов между исходным предложением и парафразом. Полученный корпус парафразов для армянского языка (названный ARPA) продемонстрировал более высокий уровень разнообразия по сравнению с MRPC и ParaPhraser (Таблица 13). Следует отметить, что

Таблица 12 — Примеры результатов обратного перевода, которые были размечены как непарафраз.

Исходное предложение	После обратного перевода
Եթե նման ցանկություն ունեն, ազատվելու են գնան, անփոխարինելի մարդ չկա՝ ինձնից սկսված»,- վստահեցրեց ՀՀ վարչապետը:	Եթե նրանք ունենան նման ցանկություն, նրանք կազատվեն, ինձանից անփոխարինելի մարդ չկա», - վստահեցրեց վարչապետը:
Այլ կերպ ասած՝ ինչից շատ ունենք, դա էլ ցույց ենք տալիս:	Այլ կերպ ասած, մենք ցույց ենք տալիս ավելին, քան ունենք:
Ճիրակի մարզում տարիներ շարունակ պատկերը մնացել է նույնը:	Ճիրակում նկարը տարիներ շարունակ մնացել է նույնը:
Այն հեղինակել է «Ազատություն Լևոն Հայրապետյանին» քաղաքացիական նախաձեռնությունը:	Այն հեղինակել է «Ազատություն» -ը ՝ Լևոն Հայրապետյանի քաղաքացիական նախաձեռնության համար:
Նրանց տեղը զբաղեցրել է ֆրանսահայ Քրիստիան Ջաղիկյանը:	Նրանց տեղը գրավեց ֆրանսահայ քրիստոնյա Ջաղիկյանը:

Таблица 13 — Уровень разнообразия парафразов в корпусах для английского, русского и армянского языков.

Набор данных	Уровень разнообразия парафразов	
	Обучающие примеры	Тестовые примеры
MRPC	6.79	7.01
ParaPhraser.ru	5.02	5.51
ARPA	8.70	8.66

оценка разнообразия не учитывала знаки препинания и стоп-слова, чтобы лучше отразить значимые изменения.

Отрицательные примеры Кроме парафразов и почти парафразов, полученные наборы были дополнены автоматически сгенерированными отрицательными парами. Для обучающего набора было сгенерировано 2660 пар предложений, половина из которых были последовательными предложениями, которые, предположительно, могли частично совпадать. Другая половина была получена путем отбора двух случайных предложений из текстов. Аналогичным образом было добавлено относительно небольшое количество отрицательных пар в тестовый набор (150 последовательных и 150 случайных) для лучшего представления пространства предложений. По сравнению с русским и английским наборами данных (таблица 14) полученный тестовый набор имеет сопоставимый размер и содержит такое же количество парафразов.

Таблица 14 — Распределение парафразов и непарафразов в корпусах для английского, русского и армянского языков.

Набор данных	Парафразы		Непарафразы		Общее число примеров
	Кол-во примеров	Средний коэффициент Жаккара	Кол-во примеров	Средний коэффициент Жаккара	
Тестовый набор					
MRPC	1147	0.438	578	0.322	1725
ParaPhraser.ru	1137	0.317	762	0.169	1899
ARPA	1021	0.327	661	0.172	1682
Обучающий набор					
MRPC	2753	0.444	1323	0.325	4076
ParaPhraser.ru	4255	0.306	2947	0.119	7202
ARPA	1339	0.320	2894	0.056	4233

Модель обнаружения парафраза на основе M-BERT Основываясь на успехе моделей BERT в задаче обнаружения парафраза, для армянского языка была разработана аналогичная модель (рисунок 3.3). Были проведены эксперименты для сравнения моделей, обученных на наборе данных ARPA, и моделей, обученных на переводах английских и русских корпусов. Качество обнаружения парафраза этих моделей дополнительно сравнивалось с результатами инструментов обнаружения парафраза для английского и русского языков. Для последних двух моделей

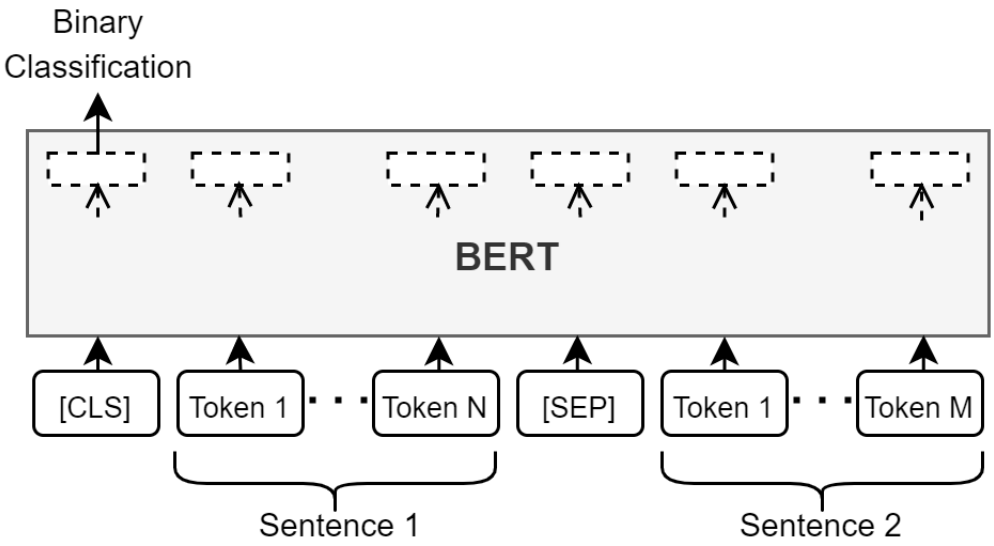


Рисунок 3.3 — Архитектура модели обнаружения парафраза.

примеры нашего тестового набора переводились на соответствующий язык с помощью Google Translate. Полный список рассмотренных моделей приведен ниже:

a M-BERT, с тонкой настройкой на следующих наборах:
   i MRPC, переведенный на армянский;
   ii ParaPhraser.ru, переведенный на армянский;
   iii ARPA;
   iv Все вышеописанные наборы вместе;
b Инструмент обнаружения парафраза на основе RUBERT (тестировалась на примерах ARPA, переведенных на русский);
c BERT-Base обученный на MRPC и протестированный на примерах ARPA, переведенных на английский.

Гиперпараметры: при тонкой настройке M-BERT мы использовали скорость обучения 0.00002, дропаут 0.5 и размер batch 32. Длина последовательности была ограничена 64 токенами.

3.1.5.5 Результаты и обсуждение

Результаты работы описанных моделей приведены в Таблице 15. В целом модели M-BERT, дообученные на примерах ARPA, дали наилучшие результаты.

Результаты RUBERT оказались достаточно близкими к наилучшей модели и даже были заметно выше по полноте. Это говорит о том, что, возможно, также стоит изучить создание набора данных путем обратного перевода на русский язык. Английская модель BERT смогла обнаружить переведенный парафраз для 65.6% примеров, но ее точность была худшей среди всех моделей.

Таблица 15 — Результаты оценки качества моделей обнаружения парафраза на тестовом наборе ARPA.

Модель	Оценка качества (95% доверительный интервал)			
	F1	Accuracy	Полнота	Точность
a.i. tr-MRPC	0.801 ± 0.0141	0.699 ± 0.0283	0.993 ± 0.0051	0.672 ± 0.0212
a.ii. tr-ParaPhraser	0.838 ± 0.0018	0.771 ± 0.0021	0.977 ± 0.0054	0.734 ± 0.0016
a.iii. ARPA	0.837 ± 0.0028	0.775 ± 0.0028	0.952 ± 0.0089	0.747 ± 0.0023
a.iv. Combined	0.840 ± 0.0017	0.776 ± 0.0017	0.971 ± 0.0056	0.741 ± 0.0013
b. RUBERT	0.837	0.764	0.998	0.721
c. BERT	0.779	0.656	1.0	0.638

Качество обнаружения на сложных примерах: при разметке пар предложений, дополнительно отмечались непарафразы, очень близкие по смыслу или лексически. Это в основном были семантически близкие примеры, которые часто было трудно отличить от парафраза. В рамках экспериментов дополнительно была вычислена accuracy моделей на этих примерах (таблица 16).

Таблица 16 — Accuracy моделей на сложных примерах.

Модель	Accuracy на сложных примерах
a.i. tr-MRPC	3.00%
a.ii. tr-ParaPhraser	4.17%
a.iii. ARPA	9.05%
a.iv. Combined	4.55%

Модели показали очень низкое качество предсказания для данного множества примеров. M-BERT, настроенная на обучающем наборе ARPA, показала наилучший результат – 9.05%. Такое низкое качество, однако, неудивительно, поскольку примеры было трудно разметить даже для аннотаторов.

Сравнение с другими языками: Результаты, полученные с помощью ARPA, сравнивались с результатами лучших моделей обнаружения парафраза на основе BERT для английских и русских наборах данных (Таблица 17). Учитывая значительно меньший состав обучающей выборки, был достигнут сопоставимый результат с точки зрения полноты. Однако точность обученной модели была значительно ниже. Помимо размера обучающей выборки, это потенциально может

быть вызвано качеством параметров M-BERT для армянского языка. Стоит отметить, что наилучшие результаты для английского и русского языков были получены при использовании моноязычных моделей BERT.

Таблица 17 — Сравнение качества обнаружения парафраза моделей на основе BERT для английского, русского и армянского языков.

Набор данных	Модель BERT	F1	Accuracy	Полнота	Точность
MRPC	$BERT_{BASE}$	88.9	83.5	99.38	80.39
ParaPhraser	$RUBERT$	87.9	84.9	91.60	84.48
	$M-BERT_{BASE}$	83.4	79.3	86.84	80.22
ARPA	$M-BERT_{BASE}$	83.7	77.5	95.20	74.70

3.2 Выводы

В этой главе были изучены внешние методы обнаружения заимствований. Были исследованы глобальные и локальные методы обнаружения нечетких дубликатов текстов, в частности алгоритмы поиска нечетких дубликатов документов в проверочной коллекции, методы нахождения полных и нечетких дубликатов на уровне небольших участков текста (коэффициенты Жаккара и Шимкевича-Симпсона, метод отпечатков, модели обнаружения парафраза).

Среди рассмотренных алгоритмов наиболее подходящим для применения в системе обнаружения заимствований был выбран метод шинглов, реализация которого с помощью хеширования позволяет выполнять быстрый поиск схожих текстов. Также были изучены подходы к поиску источников заимствований в Интернете, в частности, решения из серии соревнований PAN по информационному поиску. Был выбран алгоритм Пракаша и др.[83], как наиболее эффективный алгоритм достижения приемлемого уровня полноты результатов при умеренном количестве запросов к поисковой системе, который был адаптирован и реализован в системе поиска заимствований в текстах на армянском языке (Глава 5).

Для детального анализа текстов в системе обнаружения заимствований были изучены модели обнаружения парафраза, методы на основе отпечатков, коэффициентов Жаккара и Шимкевича-Симпсона. Изученные методы детального анализа были адаптированы и реализованы в программной системе (Глава 5). Предло-

жен полуавтоматический подход к генерации парафразов предложений на основе обратного перевода и проверки его результатов на корректность. Используя этот подход, для армянского языка создан не имеющий аналогов эталонный корпус парафразов с высоким уровнем лексического разнообразия. Используя предобученную нейронную сеть M-BERT, с помощьюь ее дообучения на созданном корпусе для армянского языка впервые разработан программный инструмент обнаружения парафраза.

Глава 4. Вспомогательные методы обработки текстов.

В этой главе описаны методы обработки текстов, которые имеют вспомогательную роль в системе обнаружения заимствований. Такими методами являются:

- Способы векторного представления слов, которые используются для признакового описания текстов во большинстве задач обработки текстов;
- Методы лемматизации, которые используются для нормализации текста во время индексации, стилометрического анализа и на этапе детального анализа;
- Методы исправления ошибок автоматического распознавания текста;
- Методы распознавания именованных сущностей, используемые для признакового описания текстов в стилометрическом анализе.

4.1 Векторные представления слов для армянского языка

В этом разделе описано исследование, посвященное векторным представлениям слов для армянского языка. Результаты исследования опубликованы в статье [20]. Была проведена внутренняя и внешняя оценка качества существующих векторных моделей слов армянского языка. Вдобавок, представляются новые модели векторов, обученные с помощью методов GloVe[115], fastText[116], CBOW, SkipGram[117; 118]. Для внутренней оценки адаптируется к армянскому языку и применяется тест аналогий. Для внешней оценки, модели векторов тестируются в задачах морфологической разметки и классификации текстов. Разметка выполняется с помощью искусственной нейронной сети, используя для обучения банк деревьев ArmTDP[119]. Для классификации текстов был создан корпус новостных статей, разбитых в 7 классов. Наборы тестовых данных с открытым доступом выложены в Интернете для использования в дальнейших исследованиях.

В синтаксических задачах, как правило, лучшие результаты показали модели fastText, в то время как модели GloVe были лучше в задачах, чувствительных к семантической информации. Что касается вопросов аналогии, то векторы Common Crawl[120] от Facebook были лучшими с большим отрывом, но относи-

тельно плохо справлялись с задачами морфологической разметки и классификации текстов. В последних задачах предложенные модели CBOW, fastText и GloVe продемонстрировали заметно более высокую точность. В целом, результаты также показывают, что точность ответов на вопросы по аналогии не является хорошим показателем эффективности векторных представлений в других задачах.

4.1.1 Введение

Векторные представления слов являются фундаментальной частью многих систем обработки естественного языка. Они используются в моделях лемматизации текстов, синтаксического и морфологического анализа, распознавания именованных сущностей и т.д. Для многих языков существуют различные общедоступные модели для отображения слова в непрерывное векторное пространство, и выбор модели для конкретной задачи, часто может быть проблематичным, поскольку эффективность модели сильно зависит от характера задачи. В статье [20] 2019 года, оценивается и сравнивается эффективность предобученных моделей представлений слов, доступных для армянского языка, с целью определения их эффективности при решении различных задач. Кроме того, для оценки будущих моделей создаются эталонные наборы данных и методы.

Первой попыткой обучения плотных представлений армянских слов была модель SkipGram 2015 года от YerevaNN, обученная на версии армянской Википедии 2 октября 2015 года. В 2017 году Facebook выпустил модели fastText, обученные на Википедии для 90 языков [116], включая армянский. Год спустя Facebook выпустил еще одну партию векторных представлений fastText, обученных на Common Crawl и Википедии [120]. Другие общедоступные представления включают 4 модели, выпущенные в 2018 году в рамках проекта pioNER[16], использующие метод GloVe [115] и обученные на энциклопедии, художественной литературе и новостных данных. В дополнение к этим 7 моделям представляется новые предобученные представления слов GloVe, fastText, CBOW. Все упомянутые модели были включены в оценочные эксперименты. Универсального способа сравнить качество векторных представлений слов не существует [121]. Методы оценки сгруппированы в две категории: (1) внутренние, которые проверяют отно-

шения слов непосредственно через векторы слов, и (2) внешние, которые проверяют векторы слов в других задачах, таких как морфологическая разметка, синтаксический анализ, распознавание сущностей, категоризация текста и т.д. Для внутренней оценки в статье используется набор задач на аналогии слов. Набор тестов семантико-синтаксических отношений слов (далее именуемый «задача на аналогии слов») был впервые представлен в 2013 году для оценки векторов английских слов Миколовым и др. [118], затем адаптирован для других языков: чешский[122], немецкий [123], итальянский [124], французский, хинди, польский [120] и т.д. Для армянского языка командой YerevaNN были собраны вопросы на аналогии[1], но они содержат в основном семантические аналогии и не являются прямой адаптацией задачи на аналогии словом. В статье адаптируется и используется полностью переведенная и адаптированная версия задачи.

При внешней оценке эффективность моделей зависит от характера задачи (например, синтаксическая или семантическая). Имея это в виду, были выбраны 2 задачи для наших экспериментов: морфологическая разметка и классификация текстов. Первый предназначен для оценки применимости моделей в задачах, связанных с морфологией, синтаксисом, а второй в основном ориентирован на измерение их семантических свойств. Эксперименты по морфологическому анализу выполняются на глубокой нейронной сети с использованием набора данных ArmTDP [119]. Для классификации текста был создан датасет из более чем 12000 новостных статей, разделенных на 7 классов: спорт, политика, искусство, экономика, несчастные случаи, погода, общество.

Основными вкладами работы являются:

1. тест аналогий, адаптированный к армянскому языку,
2. корпус категоризированных новостных текстов,
3. GloVe, fastText, CBOW, SkipGram представления слов,
4. оценка эффективности существующих и предлагаемых векторных представлений для армянского языка.

Наборы данных и векторы доступны на GitHub[2].

Данный раздел имеет следующую структуру: 2-й подраздел описывает существующие и предлагаемые модели, 3-й и 4-й подразделы посвящены внутренней и внешней оценке соответственно, а в конце каждого раздела представлены результаты экспериментов.

[1] https://github.com/YerevaNN/word2vec-armenian-wiki/tree/master/analogies

[2] https://github.com/ispras-texterra/word-embeddings-eval-hy

4.1.2 Предобученные модели

Существующие модели: Уже существует несколько обученных моделей для армянского языка.

- fastText Wiki[3]: опубликованные Facebook в 2017 году, эти вложения были обучены в Википедии с использованием архитектуры SkipGram [117] с размером окна 5, размером 300 и символьными N-граммами до длины 5.
- fastText CC[4]: опубликованные Facebook в 2018 году, эти векторные представления были обучены в Википедии и Common Crawl с использованием архитектуры CBOW с размером окна 5, размером 300 и N-граммами символов до длины 5.
- SkipGram YerevaNN[5]: опубликованные YerevaNN в 2015 году, эти векторные представления с размерностью 100 были обучены на армянской Википедии с использованием модели Skip-Gram.
- GloVe pioNER[6]: Четыре модели, выпущенные командой ИСП РАН, обученные для размеров векторов 50, 100, 200 и 300, с использованием алгоритма GloVe с размером окна 15. Они были обучены на Википедии, частях Армянской советской энциклопедии и EANC, новостях и текстах блогов.

Предлагаемые модели: Помимо существующих представлений, были разработаны 3 новые модели:

- 200-мерные векторы GloVe, обученные с размером окна 80;
- 300-мерные векторы CBOW и SkipGram, обученные с размером окна 5 и минимальным порогом частоты слов 5;
- 200-мерные векторы fastText, обученные с использованием архитектуры SkipGram, размера окна 3 и символьных N-граммы до длины 3.

Данные для обучения этих моделей были собраны из различных источников (рис. 1):

а Википедия;

[3]https://fasttext.cc/docs/en/pretrained-vectors.html
[4]https://fasttext.cc/docs/en/crawl-vectors.html
[5]https://github.com/YerevaNN/word2vec-armenian-wiki
[6]https://github.com/ispras-texterra/pioner

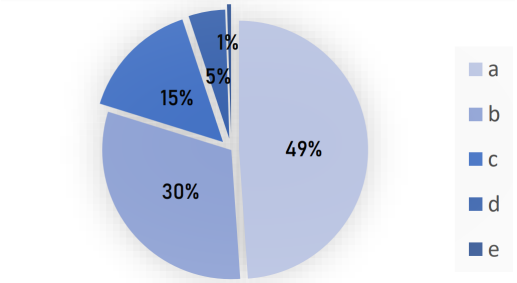


Рисунок 4.1 — Состав обучающего набора текстов.

b тексты с новостных сайтов по темам: экономика, международные события, искусство, искусство, спорт, право, политика, а также блоги и интервью;

c HC Corpora, собранная Хансом Кристенсеном в 2011 году из общедоступных блогов и новостей с помощью поискового робота;

d художественные тексты, взятые из открытой части корпуса EANC[125];

e оцифрованная и отрецензированная часть Армянской советской энциклопедии[7] (по состоянию на февраль 2018 г.), взятая из Wikisource.

Тексты были предварительно обработаны, все слова были приведены к нижнему регистру, а знаки препинания, цифры – удалены. Окончательный набор данных состоял из 90,5 миллионов токенов. Используя набор данных, были обучены векторные представления слов с помощью методов GloVe, fastText, CBOW и Skip-gram. Указанные гиперпараметры были выбирались на основе результатов по вопросам аналогии.

Все модели fastText имеют форматы .text и .bin. Чтобы попытаться воспользоваться возможностью fastText генерировать векторы для слов вне словарного запаса, мы также отдельно протестировали модели .bin.

4.1.3 Внутренняя оценка

Для внутренней оценки векторов слов использовалась адаптация задачи на аналогии слов, введенной Миколовым и др. в 2013 году для векторов слов английского языка. Исходный набор данных содержит 19544 семантических и синтаксических вопросов, разделенных на 14 разделов. Вопрос представляет из себя две

[7] https://hy.wikisource.org/wiki/Հայկական_սովետական_հանրագիտարան

пары слов с одинаковым отношением аналогии: первое слово находится в таком же отношении со вторым, как третье - с четвертым. Разделы делятся на семантические и синтаксические категории: первые 5 разделов проверяют семантические аналогии, остальные - синтаксические.

Чтобы сделать аналогичный набор для наших экспериментов, оригинальные примеры в основном напрямую переводились на армянский, используя словари[8][9] и Википедию в качестве справочной информации. Во время перевода возникло несколько проблем. В некоторых парах страна-столица, таких как Алжир - Алжир, страна и ее столица имеют одинаковый перевод (Ալժիր), поэтому все такие пары были удалены, чтобы ограничить задачу строго решением аналогий. В других разделах примеры, не имевшие четкого армянского перевода (например, he-she, his-her), были заменены другой парой слов. Некоторые названия стран или городов, например, Таиланд, имеют несколько переводов на армянский (Թաիլանդ, Թայլանդ), и для разрешения подобных ситуаций использовалась Википедия. Чтобы сделать локализованную адаптацию, категорию city-in-state заменили регионами Армении и их столицами, аналогично адаптациям для других языков [120]. Раздел comparative был полностью удален, поскольку сравнительные прилагательные в армянском языке являются выражениями из нескольких слов. В разделе present-participle использовались причастия с *-ող* (ենթակայական). При переводе аналогий страна-национальность мы использовали соответствующие демонимы вместо названия этнического большинства.

Полученный тестовый набор состоит из 15646 вопросов и 13 разделов (таблица 18). Как и в оригинале, полученная задача на аналогии слов не сбалансирована по количеству вопросов в категориях и размерам семантических и синтаксических частей. Например, отношения страна-столица (например, Աթենք Հունաստան = Կահիրե Եգիպտոս) составляют более 70% семантических отношений.

Точность векторов в вопросе определяется следующим образом (WV_i обозначает вектор слов i-го слова в вопросе):

1. Вычислить $p = WV_2 - WV_1 + WV_3$.
2. Проверить $p \approx WV_4$ (это верно, если p является ближайшим к вектору 4-го слова среди всех слов).

[8] https://bararanonline.com/
[9] https://hy.wiktionary.org/

Таблица 18 — Разделы адаптированной задачи аналогий слов.

Раздел	Вопросы	Тип примеров
capital-common-countries	506	Семантический
capital-world	4369	Семантический
currency	866	Семантический
city-in-state	56	Семантический
family	506	Семантический
gram1-adjective-to-adverb	992	Синтаксический
gram2-opposite	812	Синтаксический
gram3-superlative	1122	Синтаксический
gram4-present-participle	1056	Синтаксический
gram5-nationality-adjective	1599	Синтаксический
gram6-past-tense	1560	Синтаксический
gram7-plural	1332	Синтаксический
gram8-plural-verbs	870	Синтаксический

Если второе уравнение выполняется, то говорят, что модель правильно предсказывает вопрос.

Полученные результаты. Чтобы оценить точность векторов, использовалась функция evaluate_word_analogies в gensim. При вычислении точности словарь моделей был ограничен до 400 000 слов сверху. Точность рассчитывается как для каждого раздела (Таблица 19), так и для всех примеров вместе. Также была вычислена средняя точность по всем разделам (Таблица 20). Предлагаемые представления fastText и GloVe правильно предсказали наибольшее количество синтаксических и семантических примеров соответственно. Тем не менее, оценка GloVe явно завышена из-за ее показателей на примерах страна-столица.

Таблица 19 — Точность (Accuracy, %) векторных представлений слов на разделах адаптированной задачи аналогий.

Модель	capital-common	capital-world	curr.	city-state	family	gram1	gram2	gram3	gram4	gram5	gram6	gram7	gram8
fastText[Wiki,.text]	5.34	0.78	0	0	4.54	14.91	30.29	39.57	7.19	44.21	23.71	29.35	0.45
fastText[Wiki,.bin]	5.34	0.77	0	0	4.54	16.53	30.29	27.98	7.19	47.52	23.71	29.35	0.45
fastText[CC,.text]	32.61	11.42	2.77	7.14	13.83	22.07	30.66	43.76	4.45	41.58	18.33	19.96	5.51
fastText[CC,.bin]	72.53	39.28	11.55	48.21	47.83	25.2	36.21	49.64	19.7	8.53	23.08	41.89	41.03
fastText[new,.text]	27.66	8.1	0.1	1.79	16.2	28.02	41.74	48.3	23.95	54.59	50.51	53.67	6.09
fastText[new,.bin]	27.67	8.1	1.03	1.79	16.2	30.14	41.74	58.82	23.95	54.59	50.51	53.67	6.09
SkipGram[YerevaNN]	39.32	10.66	2.07	8.93	5.73	4.03	0.61	3.74	1.23	23.57	0.12	5.78	0.8
SkipGram[new]	36.17	17.37	2.3	3.57	17.79	7.56	12.43	16.39	1.7	37.77	4.93	16.81	10.34
CBOW[new]	28.65	13.04	1.5	5.36	29.05	10.48	14.77	17.91	5.49	24.26	6.98	28.6	11.83
GloVe[pioNER d50]	8.1	1.06	0.11	0	6.71	3.32	2.46	3.29	0.94	11.75	1.02	6.98	1.26
GloVe[pioNER,d100]	10.67	1.67	0.46	3.57	10.27	4.53	5.41	4.72	1.51	16.51	1.15	7.43	3.9
GloVe[pioNER,d200]	10.67	2.28	0.8	7.14	11.66	3.52	8.74	7.13	1.32	15.69	1.02	5.7	2.87
GloVe[pioNER,d300]	10.87	2.05	0.46	5.36	11.46	3.22	7.88	5.79	0.94	13.75	0.7	4.72	1.49
GloVe[new]	75.3	49.14	2.19	23.21	15.8	6.55	11.2	12.74	2.27	47.71	1.85	20.49	5.4

Таблица 20 — Суммарная и средняя точность (Accuracy, %) векторных представлений слов на семантических и синтаксических разделах адаптированной задачи аналогий.

Модель	Семантические	Синтаксические	Все	Семантические (средн.)	Синтаксические (средн.)	Все (средн.)
fastText[Wiki,.text]	1.33	24.88	15.39	2.13	22.87	14.89
fastText[Wiki,.bin]	1.33	25.53	15.78	2.13	23.71	15.41
fastText[CC,.text]	12.08	24.3	19.38	13.55	23.29	19.54
fastText[CC,.bin]	38.9	35.96	37.14	43.88	35.65	38.81
fastText[new,.text]	9.29	41.11	28.29	10.77	38.35	27.74
fastText[new,.bin]	9.3	42.48	29.11	10.95	39.93	28.79
SkipGram[YerevaNN]	11.37	5.98	8.15	13.34	4.98	8.19
SkipGram[new]	16.72	14.69	15.51	15.44	13.49	14.24
CBOW[new]	13.92	15.66	14.96	15.52	15.04	15.22
GloVe[pioNER d50]	1.93	4.36	3.38	3.19	3.87	3.61
GloVe[pioNER,d100]	2.93	6.13	4.84	5.33	5.64	5.52
GloVe[pioNER,d200]	3.55	6.07	5.06	6.51	5.74	6.04
GloVe[pioNER,d300]	3.33	5.11	4.39	6.04	4.81	5.28
GloVe[new]	41.88	15.35	26.04	33.12	13.52	21.06

В среднем векторы от Facebook, обученные на Common Crawl, давали более высокие результаты в семантических разделах и в целом по всем разделам. Как свидетельствует заметная разница в точности между форматами .bin и .text, способность .bin генерировать векторы для слов вне словарного запаса значительно повысила его производительность.

4.1.4 Внешняя оценка

Второй способ сравнения качества векторов - использование их в задачах обработки текстов и измерение их влияния на качество выполнения задачи. Результат сравнения зависит от характера этих задач и настроек [121]. С этой целью мы провели эксперименты над двумя разными задачами: морфологическим анализом и классификацией текстов.

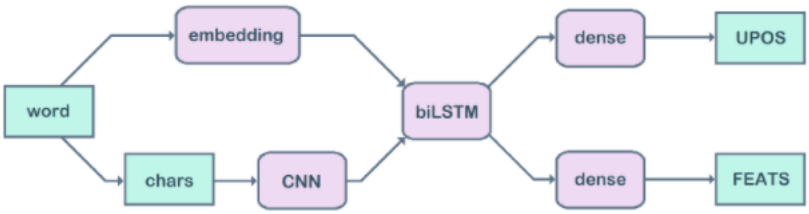


Рисунок 4.2 — Архитектура морфологического анализатора.

4.1.4.1 Морфологический анализ

Чтобы проверить способность векторов описывать морфологические свойства слов, они были использованы в качестве входных признаков в морфологическом анализаторе на основе нейронной сети, обученной на датасете ArmTDP [119]. Предложения в датасете имеют морфологическую разметку, и задача анализатора заключается в том, чтобы предсказать следующие 2 метки:

1. UPOS: универсальный тег части речи.
2. FEATS: список морфологических признаков (число, падеж, одушевленность и т.д.).

В качестве анализатора использовалась нейронная сеть с двунаправленным рекуррентным слоем LSTM на уровне предложения (рис. 4.2). Предобученные представления использовались в качестве входных данных для сети. Помимо предобученного вектора слова, входные данные сети включали символьные признаки, извлеченные через сверточный слой. Анализатор был обучен совместно предсказывать 2 метки для каждого входного токена: его UPOS и конкатенацию тегов FEATS.

ArmTDP версии 2.3 предоставляет только обучающие и тестовые наборы. Из исходной обучающей выборки 20% примеров случайным образом были отделены в валидационный набор, а остальные 80% использовались для обучения сети в течение 200 эпох, сохраняя параметры после каждой эпохи. Для оптимизации использовался градиентный спуск с начальной скоростью обучения 0.6 и спадом по времени 0.05. Обратное распространение ошибок на векторные представления слов было заблокировано во время обучения.

Полученные результаты. Эксперименты для всех моделей повторялись 10 раз и затем результаты усреднялись. Полученные рзультаты представлены в Таблице 21. Модели fastText от Facebook показали низкие результаты по сравнению

с другими. Поразительным наблюдением является то, что, в отличие от аналогий, здесь векторы fastText .text с большим отрывом превзошли .bin, особенно по FEATS. В целом предложенные модели продемонстрировали высочайшую точность. Лучшим был fastText, за ним следуют CBOW и GloVe.

Таблица 21 — Точность (Accuracy, %) морфологического анализа на основе разных моделей векторного представления.

Модель	Валидационный набор		Тестовый набор	
	UPOS	FEATS	UPOS	FEATS
fastText[Wiki,.text]	92.99	83.83	89.54	81.03
fastText[Wiki,.bin]	89.27	75.96	84.35	71.14
fastText[CC,.text]	91.86	79.85	88.38	75.44
fastText[CC,.bin]	91.54	77.78	87.59	73.29
fastText[new,.text]	94.64	85.89	93.35	83.99
fastText[new,.bin]	91.43	80.1	87.55	74.78
SkipGram[YerevaNN]	91.44	80.04	87.45	75.68
SkipGram[new]	93.9	86.45	91.6	83.77
CBOW[new]	93.87	85.39	92.72	84.07
GloVe[pioNER d50]	91.78	82.42	89.12	80.18
GloVe[pioNER,d100]	91.78	82.95	89.06	80.51
GloVe[pioNER,d200]	92.34	83.77	88.90	80.07
GloVe[pioNER,d300]	91.70	83.19	89.09	80.78
GloVe[new]	94.09	86.23	92.98	83.97

4.1.4.2 Классификация текстов

Чтобы проверить качество векторов слов в задаче классификации, с новостного сайта ilur.am было собрано 12428 статей на 7 тем (рис. 4.3): искусство, экономика, спорт, несчастные случаи, политика, общество и погода (всего 2667299 токенов). 80% текстов использовались в качестве обучающих данных, а остальные 20% - в качестве тестовых данных. Все тексты были предварительно обработаны путем удаления стоп-слов. В качестве признаков классификатора использовалось среднее значение векторов слов текста с влизвешиванием tf-idf. Затем был применен «один против остальных» классификатор логистической регрессии с либлинейным решателем.

Полученные результаты. accuracy, точность, полнота и F1, показанные в таблице 22, были рассчитаны для каждой модели. В целом, предложенные векторы

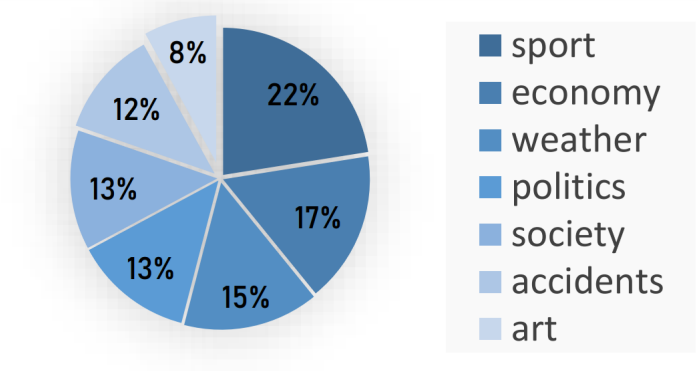


Рисунок 4.3 — Распределение тем в наборе текстов для классификации.

Таблица 22 — Точность классификации текстов на основе разных моделей векторного представления.

Модель	Accuracy	Точность	Полнота	F1
fastText[Wiki,.text]	66.63	60.38	63.02	58.96
fastText[Wiki,.bin]	65.79	59.78	63.97	58.15
fastText[CC,.text]	65.75	59.15	63.68	57.03
fastText[CC,.bin]	65.51	58.96	63.43	56.94
fastText[new,.text]	63.34	56.97	59.81	55.18
fastText[new,.bin]	60.12	53.66	55.4	51.96
SkipGram[YerevaNN]	64.34	57.87	59.73	56.28
SkipGram[new]	66.68	60.84	63.56	59.83
CBOW[new]	67.92	61.94	65.2	60.94
GloVe[pioNER d50]	64.26	57.57	58.39	54.4
GloVe[pioNER,d100]	65.91	60.15	62.34	58.91
GloVe[pioNER,d200]	68.16	62.13	65.54	60.6
GloVe[pioNER,d300]	67.85	61.7	65.36	60.43
GloVe[new]	69.77	63.93	66.55	63.13

GloVe получили самые высокие баллы по всем показателям, превзойдя fastText, SkipGram, CBOW. Несколько удивительно, что многомерные векторы GloVe из проекта pioNER, которые плохо работали на аналогиях, находятся здесь среди лучших. Также стоит отметить, что среди моделей fastText векторы .bin снова показали более низкие результаты, чем .text.

4.2 Лемматизация

Метод шинглов (см. 3.1.2) в своем исходном варианте чувствителен к окончаниям слов, что для языков с богатой морфологией является недостатком, так как автор может с легкостью поменять окончания нескольких слов в тексте и таким образом обмануть систему проверки. По этой причине, для увеличения эффективности метода шинглов целесообразна предварительная нормализация текста путем стемминга или лемматизации. Лемматизация – процесс приведения слов из текста к их словарной форме. Лемматизация является общим и важным компонентом многих систем обработки текста и имеет широкий спектр приложений, включая классификацию текста, кластеризацию, маркировку последовательностей, поиск информации, изучение представления слов и т.д. В рамках этой работы было выполнено исследование по разработке методов лемматизации для армянского языка.

В этом разделе описана модель легкого и точного лемматизатора для армянского языка с использованием глубокого обучения. На основе анализа существующих методов, основанных на глубоком обучении и показывающих точные результаты на языках с богатой морфологией и маленьким обучающим корпусом, в качестве базы для лемматизатора выбрана нейронная сеть COMBO. Для замены тяжелых по памяти и количеству параметров предобученных моделей векторов слов, предлагается несколько альтернатив на основе подслов. Разработаны и протестированы две модификации алгоритма fastText, использующие только внутренние символьные N-граммы и суффиксы слова. Другая альтернатива - использование в качестве вектора слова среднее векторов его подслова, полученных в результате BPE кодирования. Последнее позволяет уменьшить размер модели от 1 Гб до несколько Мб, а размер предобученных моделей векторов уменьшается примерно

в 100 раз, при этом без потери точности. Точность разработанного лемматизатора находится на уровне лучших решений для языков с небольшим банком деревьев. Разработанный лемматизатор превосходит существующие аналоги, достигая точности, сравнимой с state-of-the-art для малоресурсных языков.

Одним из основных вкладов этого исследования является демонстрация эффективности представлений подслов BPE [126] при замене обычных представлений на уровне слов. Это помогает значительно уменьшить размер и количество параметров моделей, позволяя применять их в средах с ограниченными ресурсами. Для этих подслов был обучены представления на основе коллекции армянских текстов, включая Википедию, новостные статьи и художественную литературу. Помимо них, предлагаются две другие облегченные альтернативы модели fastText, в которых для генерации векторного представления используются только символьные N-граммы и суффиксы, которые также демонстрируют сопоставимые результаты. Результаты исследования были опубликованы в нескольких статьях [17; 18].

4.2.1 Введение

Лемматизация играет ключевую роль в системах полнотекстового поиска, особенно для языков с богатой морфологией, как армянский. Основная цель лемматизации - смягчить разреженность лексических данных, сгруппировав разные формы одного и того же слова в одну. В этом смысле лемматизация похожа на стемминг, однако исследования показывают, что использование последнего как правило менее эффективно. Поскольку армянский язык является морфологически богатым языком и не имеет больших текстовых корпусов, наличие инструментов нормализации текста особенно важно для разработки систем точной обработки текста.

Для армянского языка было проведено не так много исследований в этой области. Попытки решить задачу в рамках CoNLL Shared Task 2018 [127] не дали достаточно точных результатов из-за небольшого набора обучающих данных. С тех пор была выпущена вторая версия набора данных ArmTDP [119], что сделало возможным разработку более точных лемматизаторов. В 2018 году YerevaNN

выпустила синтаксический анализатор восточноармянского языка, который показывал точность лемматизации 88.05% на тестовом наборе ArmTDP v2.3[10]. За последний год были опубликованы новые инструменты и модели, UDPipe 2.0 [59] и Stanza [58], позволяющие с более высокой точностью определять леммы слов.

Подходы к лемматизации можно разделить на 3 категории: поиск по словарю, подход на основе правил, и машинное обучение. Первый требует наличие словаря с большим словарным запасом. Помимо того, что подходящего ресурса для армянского языка нет, поиск в словаре не может использоваться для внесловарных слов, а также не умеет обрабатывать неоднозначные случаи. Методы, основанные на правилах, требуют лингвистических знаний, а разработка этих правил требует много времени и ресурсов. В настоящее время методы, основанные на машинном обучении, являются преобладающим подходом к лемматизации, о чем свидетельствует CoNLL Shared Task 2018, где наиболее эффективные системы используют машинное обучение. Одна из проблем последних систем состоит в том, что они часто используют большие наборы параметров и могут занимать много памяти. В результате эти системы не могут работать в средах с ограниченными ресурсами.

При исследовании данной задачи была поставлена цель разработать легкий и точный лемматизатор для армянского языка с использованием глубокого обучения. В рамках этого исследования также изучался вопрос применимости подходов глубокого обучения к лемматизации армянских текстов, а именно:

- state-of-the-art для языков с богатой морфологией и ограниченными ресурсами обучения;
- совместное обучение лемматизации с другими анализаторами;
- возможности обучения легких признаков вместо векторов слов, требующих большого объема параметров и памяти;
- применение обучения с частичным привлечением учителя.

4.2.2 Обзор моделей лемматизации

Работы по созданию лемматизаторов можно разделить на три категории:

1. поиск по словарю;

[10] https://github.com/UniversalDependencies/UD_Armenian-ArmTDP

2. подходы, основанные на правилах;
3. методы на основе машинного обучения.

4.2.2.1 Поиск по словарю

Лемматизаторы на основе словаря хранят множество словоформ для каждой лексемы и определяют лемму слова на основе этих множеств. Для английского языка существует Wordnet, бесплатный и общедоступный ресурс, в первую очередь предназначенный для хранения семантических отношений между словами, но также предоставляющий информацию о словоизмененных формах слов. Последняя особенность позволяет использовать его в качестве лемматизатора.

Для армянского языка существует несколько общедоступных словарей, которые, однако, не могут быть напрямую использованы для лемматизации. Онлайн-словари, такие как bararanonline.com, nayiri.com, хранят только словарные формы слов. Викисловари на английском и армянском языках предоставляют словоизменения для определенных категорий слов, однако они также не могут напрямую использоваться для лемматизации, потому что не для всех слов содержат таблицу словоизменений, и даже в этом случае получение этих форм потребует дополнительного анализа страницы статьи. Последний шаг проблематичен с точки зрения автоматизации с помощью программы, так как изучение этих ресурсов выявило несоответствия разметки между разными категориями слов.

Основным недостатком поиска по словарю является невозможность обработки словоформ, отсутствующих в словаре. Кроме того, для некоторых словоформ, которые этот подход не учитывает, лемма может быть неоднозначна. Существуют варианты поиска по словарю, когда вместо словоформ в качестве ключей словарь хранит пары словоформ и их части речи [128]. Последний подход помогает обрабатывать часть неоднозначных случаев, но одновременно добавляет зависимость от частеречного анализатора.

4.2.2.2 Лемматизация на основе правил

Данная группа лемматизаторов использует правила, подобные if-else условиям или деревьям решений, для выполнения лемматизации, в частности, для предсказания преобразования аффиксов входной словоформы для получения ее леммы. Правила можно создать вручную, что отнимает много времени и требует лингвистических навыков. Однако существуют методы, которые автоматически извлекают эти правила, используя наборы данных с разметкой лемм.

В работе [129] представили модель, способную преобразовывать слова в обоих направлениях: от формы к лемме и обратно. Преобразования, используемые в модели, реализованы с помощью конечных автоматов. Авторы статьи [130] предложили другое решение, основанное на правилах, использующее преобразования суффиксов (например, удаление окончания входной словоформы и добавление другого для получения леммы). В их системе используются ripple-down правила, чтобы предсказать, какие преобразования с суффиксами следует выполнить. Правила создаются на основе набора пар словоформа-лемма.

По сравнению с поиском по словарю эти подходы позволяют обрабатывать внесловарные словоформы. В то же время, из-за алгоритма извлечения правил [130], полученный лемматизатор, например, не может разрешить случаи, когда лемма словоформы зависит от контекста, в котором она появляется. Для языков с богатой морфологией, таких как армянский, эти системы правил могут стать особенно большими, что делает их дальнейшее обновление (например, добавление новых правил) проблематичным. Еще один серьезный недостаток методов, основанных на правилах, заключается в том, что полученная информация не может быть перенесена на другие языки.

4.2.2.3 Машинное обучение

Лемматизаторы на основе машинного обучения имеют ряд преимуществ перед поиском по словарю и правилами: они могут обрабатывать внесловарные словоформы, разрешать неоднозначные случаи на основе контекста, не требуют

лингвистических знаний, а разработанная система может быть переобучена и перенесена на другую язык. Для таких систем необходимо наличие размеченного корпуса, однако некоторые системы на основе правил также полагаются на наборы данных с разметкой лемм, чтобы автоматически извлекать правила. С помощью глубокого обучения также можно использовать неразмеченные текстовые наборы для обучения более эффективных представлений текстов и параметров нейронной сети.

Как правило, методы, основанные на машинном обучении, не рассматривают лемматизацию как задачу классификации лемм, и это обусловлено большим размером словаря лемм. Вместо этого машинное обучение используется для предсказания преобразования входного слова, необходимого для получения его леммы. В существующих подходах это преобразование представлено в различных формах, которые можно разделить на редакционное предписание и трансдукция последовательности символов.

Морфетт [131] и Лемминг [132], два популярных лемматизатора, предсказывают редакционные предписания, которые определяют, как входное слово должно быть преобразовано для построения его леммы. Примером такого преобразования является вставка или удаление символов. Обе модели полагаются на частеречную разметку при предсказании редакционного предписания. Частеречный анализатор Морфетте использует классификацию методом максимальной энтропии, в то время как Лемминг использует усредненный персептрон. Метод из [133] также предсказывает сценарии редактирования и использует более сложную нейронную сеть и глубокое обучение для прогнозирования дерева редактирования от входного слова до его леммы. Эта модель использует два набора признаков: предобученные представления слов и признаки на уровне символов, извлеченные с помощью двунаправленного слоя GRU. Объединение этих признаков обрабатывается другим двунаправленным слоем GRU для предсказания дерева редактирования. Лемматизатор, предложенный в [59], использует дополнительный обучаемый вектор в качестве входа в сеть и 4 двунаправленных слоя LSTM на уровне предложения вместо GRU. Кроме того, лемматизатор обучается совместно с морфологическим анализатором аналогичной архитектуры.

Авторы статей [134–136] трактуют лемматизацию как задачу трансдукции последовательности символов. Одна из таких систем, Lematus [137], использует систему нейросетевого машинного перевода [138] для перевода последователь-

ности символов слова в последовательность символов леммы. Система основана на архитектуре кодировщик-декодировщик и может быть сконфигурирована так, чтобы также учитывать контекстную информацию. Последнее осуществляется путем добавления соседних символов из контекста входного слова на вход кодировщика. Кодировщик использует двухуровневые двунаправленные слои GRU, а декодировщик – однонаправленный слой GRU. Аналогичное решение из CoNLL Shared Task 2018, предложенное группой Turku NLP, добавляет морфологические метки к входным данным вместо контекстных символов [139]. Этот лемматизатор основан на другой нейросетевой системе перевода – OpenNMT, которая имеет архитектуру кодировщик-декодировщик с механизмом внимания и использует LSTM вместо GRU. Однако это решение имеет недостаток, заключающийся в том, что оно полагается на часть речи и морфологические признаки входного слова. Другой точный лемматизатор [128] использует комбинацию нескольких подходов. Чтобы присвоить лемму известным словам, используется словарь, где ключ дополнительно хранит часть речи. Если нужной пары слово-часть речи нет в словаре, используется другой словарь, отображающий только слова в леммы. Для слов вне словарного запаса обучается классификатор, предсказывающий совпадают ли входное слово и его лемма, или является ли лемма строчной формой ввода, или требуется ли более сложное преобразование для получения леммы. В последнем случае снова используется архитектура кодировщик-декодировщик с механизмом внимания, с двунаправленными слоями LSTM в кодировщике и однонаправленными слоями LSTM в декодировщике.

Анализ решений с CoNLL Shared Task 2018 выявил, что для языков с маленьким набором обучающих данных самыми эффективными системами для лемматизации оказались те, которые полагались на трансдукцию последовательностей. За исключением COMBO [140], эти системы основывались на нейронную сеть архитектуры кодировщик-декодировщик. Для языков с маленьким набором обучающих данных лемматизатор COMBO занял второе место после лемматизатора Turku NLP. Учитывая большое количество параметров в архитектурах кодировщик-декодировщик, длительное время обучения и сложность настройки гиперпараметров, в этой работе предпочтение было отдано нейронной сети COMBO для лемматизации.

4.2.3 Нейронная сеть COMBO

4.2.3.1 Архитектура нейронной сети

COMBO – это нейронная сеть, которая использует сверточные и рекуррентные слои для анализа предложений и предсказания лемм и морфосинтаксических меток слов. Входное предложение передается в сеть как последовательность токенов на уровне слов. Сначала извлечение признаков выполняется для каждого токена отдельно, затем извлеченный вектор пропускается через повторяющиеся слои уровня предложения для получения контекстного представления. Затем контекстное представление используется в качестве входных данных для слоев, предназначенных для конкретной задачи.

Входные признаки: для каждого слова входного предложения используются два набора признаков: его предобученное векторное представление и признаки, извлеченные из символов слова. Предобученные представления слов кодируют информацию о семантике слова и контекстах, в которых оно встречается, в то время как символьные признаки предназначены для извлечения информации о внутренней структуре слова. Последний набор признаков особенно эффективен при лемматизации и морфосинтаксической разметке, где последовательности символов, такие как префиксы и суффиксы, играют важную роль.

Для предобученного вектора добавляется дополнительный полносвязный слой, преобразующий его в вектор более меньшей размерности. Для символьных признаков сначала последовательность символов дополняется в начале и в конце специальными «символами» START и END, затем последовательность пропускается через 3-слойную расширяющуюся сверточную сеть. Чтобы получить вектор фиксированного размера, к выходным данным сверточной сети применяется глобальный max-pooling. Полученные наборы признаков конкатенируются, чтобы сформировать окончательный вектор признаков.

Слои нейронной сети уровня предложений: после вычисления вектора признаков он проходит через два слоя двунаправленных LSTM, которые работают на уровне предложений. Эти LSTM применяются к каждому слову входной после-

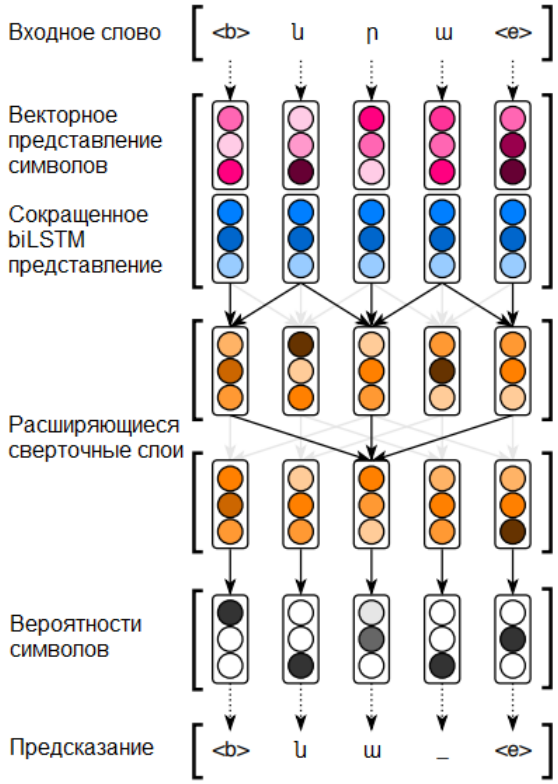


Рисунок 4.4 — Архитектура лемматизатора [140].

довательности, а выходные используются в качестве контекстных представлений входных слов.

4.2.3.2 Архитектура лемматизатора

Как упоминалось выше, лемматизатор COMBO выполняет трансдукцию последовательности символов входного слова в последовательность символов леммы. Подобно извлечению входных признаков, здесь также последовательность символов дополняется специальными метками `START` и `END`. Входные символы кодируются с помощью обучаемых векторов. Контекстное представление входного слова сокращается с помощью полносвязного слоя и конкатенируется к вектору каждого входного символа, формируя их окончательный вектор. Затем к последовательности векторов символов применяется 3-слойная расширяющаяся сверточная сеть. Результатом конечного слоя свертки является последовательность векторов, размер которых дополнительно изменяется до размера словаря символов с использованием одномерного сверточного слоя. Наконец, к каждому

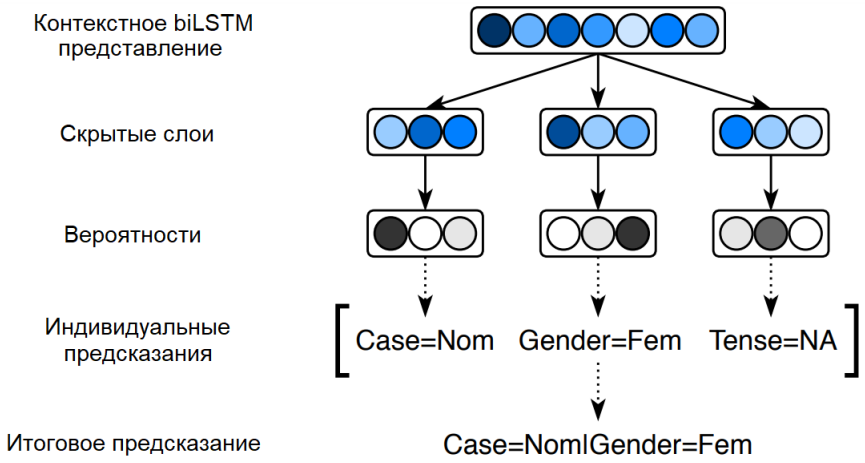


Рисунок 4.5 — Архитектура морфологического анализатора [140].

полученному вектору в последовательности применяется softmax, чтобы определить для него наиболее вероятный символ (Рис. 4.4). Все предсказанные символы между символами `START` и `END` конкатенируются, чтобы сформировать лемму входного слова.

4.2.3.3 Архитектура частеречного и морфологического анализатора

Архитектура морфологического анализатора относительно проще (рисунок 4.5). Для каждого морфологического признака поверх контекстного представления добавляется отдельный полносвязный слой, а затем применяется softmax, чтобы найти наиболее вероятное значение этого признака. При предсказании признаков используется гранулярный подход, то есть предсказание значения каждого признака рассматривается как отдельная, независимая задача классификации.

4.2.3.4 Архитектура анализатора синтаксических зависимостей

Анализатор зависимостей сначала применяет два отдельных полносвязных слоя к контекстному представлению, чтобы получить два новых представления для каждого входного слова. Первое представление кодирует ввод в роли вершины в синтаксической зависимости, а второе представление кодирует ввод в роли

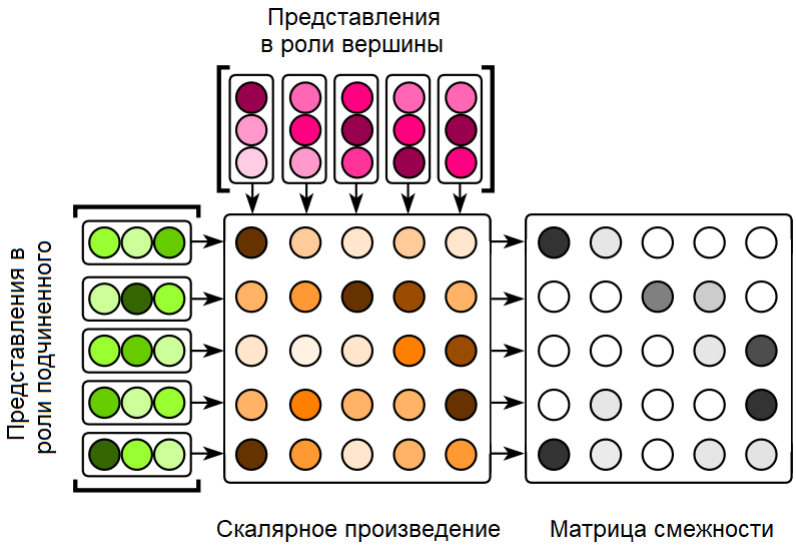


Рисунок 4.6 — Архитектура синтаксического анализатора [140].

подчиненного слова. Затем конструируются две матрицы: первая в столбцах хранит векторы всех слов в роли вершин, а вторая – векторы в роли подчиненных в виде строк. Далее вычисляется матрица смежности путем произведения этих матриц (Рис. 4.6). Для соответствующего слова входного предложения вершина синтаксической зависимости предсказывается путем применения функции softmax к соответствующей строке матрицы смежности.

Для предсказания типа зависимости снова используются вышеописанные представления слова, полученные посредством преобразования их контекстного представления. Используя softmax-оценки вершин зависимостей в качестве весов, для каждого слова в роли подчиненного вычисляется средневзвешенное значение векторов вершин. Средний вектор конкатенируется с представлением в роли подчиненного и передается на слой softmax, который предсказывает тип зависимости.

4.2.4 Эксперименты

4.2.4.1 Обучающие данные

Во всех экспериментах для тестирования качества моделей использовался датасет ArmTDP v2.3. В таблице 23 показано количество предложений и токе-

Таблица 23 — Статистика обучающей и тестовой выборок ArmTDP v2.3.

	# (Предложения)	# (Токены)
Обучающая выборка	560	~11.5k
Тестовая выборка	470	~11.5k

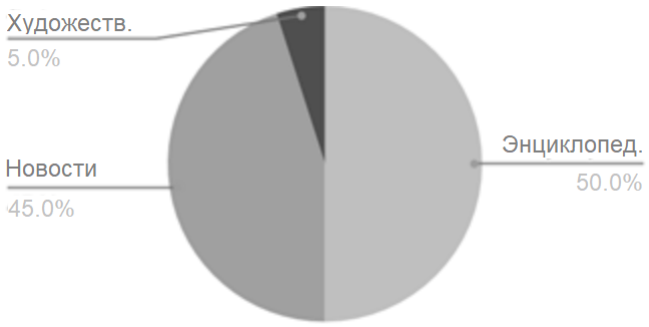


Рисунок 4.7 — Распределение источников в неразмеченном наборе текстов.

нов в каждом наборе датасета. Для обучения с учителем используются случайно выбранные 90% предложений из обучающей выборки ArmTDP, а остальные 10% используются для валидации.

ArmTDP v2.3 поставляется в формате CoNLL-U и содержит ручную разметку части речи каждого токена (метка UPOS), морфологических признаков (метка FEATS), синтаксической зависимости и ее типа (метки HEAD, DEPREL).

Для обучения без учителя использовался набор текста, состоящий из 91 миллиона токенов. Коллекция включает армянскую Википедию, художественную литературу, а также новости и статьи в блогах из более чем десятка различных источников. Распределение текстов по источникам показано на рисунке 4.7. Данные используются для обучения репрезентации и самообучения.

4.2.4.2 Параметры обучения нейронной сети

Для всех компонентов сети в качестве функции потерь используется категориальная кросс-энтропия. Итоговая функция потерь – это взвешенная сумма потерь всех компонентов. Для компонентов используются следующие веса:

– лемматизатор - 0.05;
– теггер части речи - 0.05;

– морфологический теггер - 0.2;
– зависимость - 0.2;
– тип зависимости - 0.8.

Для обучения используется оптимизатор Adam с начальной скоростью обучения 0.002, batch size – около 2500 слов, максимальное количество эпох обучения – 400. Когда точность на валидационной выборке перестает увеличиваться, скорость обучения уменьшается в 2 раза.

4.2.4.3 Базовые методы

Модели лемматизации сравниваются по трем показателям: accuracy на тестовой выборке, accuracy для внесловарных слов, и количество параметров. Accuracy - это процент правильно лемматизированных слов. Здесь термин «внесловарный» (OOV) относится к словам тестовой выборки, которые не появляются в обучающей выборке.

Для оценки эффективности нейросетевых моделей, они сравниваются с несколькими базовыми решениями. Первый базовый метод – тождественное отображение (identity), то есть для каждого слова в качестве леммы предсказывается оно само. На тестовом наборе этот метод точно предсказывает 52.6% лемм (и только 27.34% для OOV).

В качестве второго базового метода был выбран поиск по словарю с использованием обучающей выборки для построения словаря словоформа-лемма. Когда встречается OOV слово, данный метод прибегает к первому базовому методу. Для словоформ с несколькими возможными леммами выбирается наиболее распространенная лемма. Этот базовый метод показывает 74.89% accuracy на всех тестовых примерах и 27,34% на OOV примерах.

В качестве основного базового метода и отправной точки для изучаемых моделей использовался оригинальный лемматизатор COMBO, обученный отдельно от других задач и только на размеченных данных. Цель исследования заключалась в том, чтобы улучшить качество этой модели, в то же время уменьшив ее размер и количество параметров. Для этого базового метода использовались векторы fastText с размерностью 200, обученные с использованием архитектуры SkipGram

Таблица 24 — Результаты базовых методов.

Метод	Accuracy	Accuracy на OOV	Кол-во параметров
Identity	52.60	27.34	-
Словарь	74.89	27.34	-
COMBO	87.72	66.42	1.3×10^8

и символьных N-грамм размером от 1 до 3. Представления слов, используемые в последующих экспериментах, были изучены с использованием того же набора гиперпараметров.

Результаты базовых методов представлены в таблице 24. В рамках исследования ставилась цель не только опередить эти методы, но достигнуть точности 90%, что примерно соответствует результатам лучших моделей из CoNLL Shared Task 2018 для маленьких датасетов.

4.2.5 Совместное обучение

Многие системы машинного обучения фокусируются на прогнозирование одного целевого значения, и потери соответственно оптимизируются только в зависимости от точности прогнозирования этого целевого значения. Однако было показано, что качество модели в задаче часто может быть улучшена, если ее совместно обучить другой связанной задаче. Этот подход также известен как многозадачное обучение. Идея совместного обучения состоит в том, что введение связанной задачи помогает сети изучить параметры, которые обобщаются для более чем одной задачи. Обычно это делается через совместное использование параметров сетей для этих задач. Этот подход, известный как жесткое совместное использование параметров, обновляет параметры в зависимости от потери каждой задачи либо путем вычисления общей потери на каждой итерации, либо путем чередования задач между итерациями. Другой, менее распространенный подход заключается в использовании отдельного набора параметров для каждой задачи и регулировании расстояния между параметрами соответствующих слоев.

Впервые введенное в [141], совместное обучение с тех пор успешно применяется во многих системах обработки естественного языка. Хотя можно использовать общие параметры для прогнозирования множества различных значений

Таблица 25 — Результаты совместного обучения.

Конфигурация	Accuracy	Accuracy на OOV	Кол-во параметров
lemma	87.72	66.42	1.3×10^8
+upos+feats	90.77	75.25	1.3×10^8
+upos+feats+head+deprel	90.33	74.75	1.3×10^8

[142], наиболее распространенным подходом является использование небольшого количества тесно связанных задач [143; 144].

В контексте этой работы совместное обучение лемматизатора с другими задачами может помочь улучшить его качество, не увеличивая при этом сложность нейронной сети. Как описано в предыдущем разделе, слои векторного представления входного слова и извлечения символьных признаков используются всеми задачами совместно. Параметры двухуровневого biLSTM, используемого для обучения контекстному представлению, также одинаковы для разных задач. Остальные параметры уже отдельны и зависят от задачи. Во время обучения оптимизация выполняется с использованием общей потери всех целевых задач. В частности, используется взвешенная сумма потерь, как описано в исходной статье [140].

В экспериментах исследуются две конфигурации совместного обучения:

– Лемматизация и морфологический анализ (столбцы LEMMA, UPOS, FEATS);
– Лемматизация, морфологический анализ и анализ зависимостей (столбцы LEMMA, UPOS, FEATS, HEAD, DEPREL).

Результаты сравниваются с лемматизатором, обученным отдельно. Эксперименты показывают, что совместное обучение помогает повысить accuracy лемматизации. В этом случае сочетание лемматизации и морфологического анализа дало наилучшие результаты. Результаты представлены в таблице 25.

Стоит отметить, что хотя обученная нейронная сеть имеет больше слоев, чем лемматизатор, дополнительные слои не используются во время предсказания леммы. Таким образом, повышение точности достигается при том же конечном количестве параметров и размере модели лемматизатора.

4.2.5.1 Векторные представления

Как правило, векторные представления слов содержат порядка 108 параметров и обычно составляют значительную часть параметров системы. В COMBO только векторы fastText составляет до 90% параметров всей модели. Поэтому при разработке более легкого (с точки зрения количества параметров) лемматизатора в качестве основной цели было выбрано исследование небольших представлений слов. Хотя существуют и другие средства уменьшения размера модели, такие как прореживание сети (англ. pruning) или квантизация, известно, что эти подходы влекут за собой потерю точности и поэтому в этой работе не рассматриваются.

fastText [116] - библиотека для классификации текстов и обучения векторных представлений. В последнем режиме fastText учит представления слов с использованием символьных N-грамм, обучая нейронную сеть вида SkipGram или CBOW на неразмеченных текстах (Рис. 4.8). Представление каждого слова вычисляется как сумма его отдельного вектора и векторов его символьных N-грамм. Отсюда вытекает преимущество fastText по сравнению с другими моделями векторного представления слов, заключающееся в том, что он может вычислять представление для слова вне словарного запаса (OOV), используя его символьные N-граммы. Как видно, fastText хранит и использует отдельный вектор для целого слова, но во внесловарных случаях такой вектор отсутствует, что приводит к ухудшению качества получаемого вектора слова. Кроме того, в результате хранения векторов для целых слов, модели fastText обычно требуют много памяти для хранения и обработки (модели векторов от Facebook, обученные на Common Crawl, весят 7.3 ГБ и 4.5 ГБ в форматах .bin и .vec соответственно [120]). Это становится особенно проблематично для морфологически богатых языков, учитывая многочисленность словоформ. В результате, для таких языков получаются модели, имеющие большое количество параметров и требующие много памяти. По сравнению со словоформами, словарь подслов имеет, как правило, меньший размер и позволяет получать модели со значительно более маленьким числом параметров.

В последние годы появилось несколько мощных альтернатив предобученным векторам слов, таких как ELMo [145], Flair [146] и BERT [94]. Эти альтернативы используют языковую модель, основанную на символах слова, для создания вектора признаков слова. Тем не менее, они требуют больших обучающих кор-

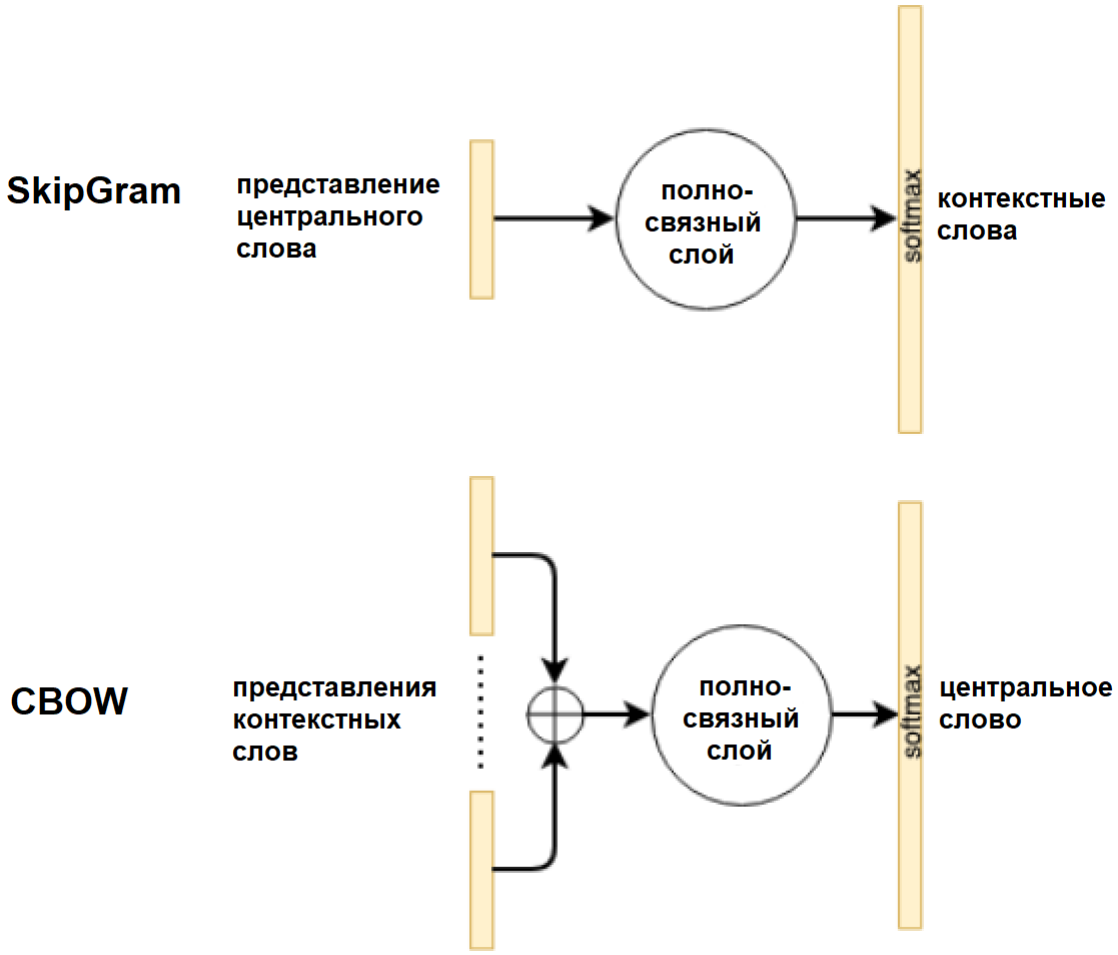


Рисунок 4.8 — Нейронные сети CBOW и SkipGram.

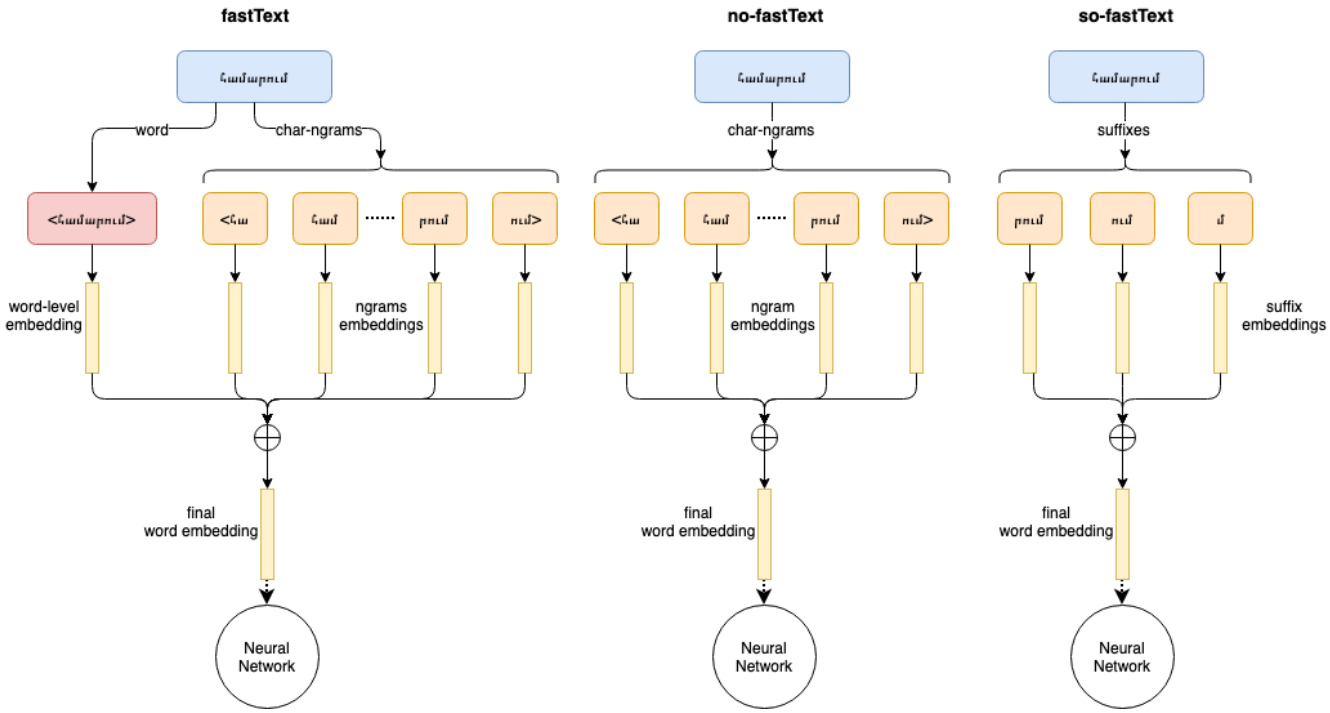


Рисунок 4.9 — Архитектура исходной модели fastText и предлагаемых no-fastText, so-fastText.

пусов, большой вычислительной мощности и все еще требуют много времени на обучение и настройку (более недели). BERT особенно велик по размеру и количеству параметров. Для сравнения: общедоступные многоязычные модели BERT имеют 110 миллионов параметров, примерно столько же, сколько fastText. По этим причинам вышеупомянутые модели в данной работе не рассматривались. Вместо этого основным направлением исследований были модификации существующих моделей предобучения векторов слов, в которых изучались различные варианты замены слов подсловами.

В этой работе предлагаются модификации исходной модели fastText – no-fastText, so-fastText, BPE-fastText (рисунок 4.9). Первый учитывает только N-граммы символов слова во время обучения и генерации вектора слов, второй аналогичен первому, но использует только N-граммы в конце слова (суффиксы). В двух других экспериментах векторы слов заменены усредненными предобученными представлениями их подслов. Для выделения подслов используется BPE кодирование [126] и две модели векторов этих подслов: предобученные векторы из проекта BPEmb [147] и новые векторы, обученные на наших неразмеченных данных.

fastText В режиме без учителя fastText обучает представления слов. При обучении используется нейронная сеть с архитектурой CBOW или SkipGram. SkipGram принимает в качестве входных данных векторное представление слова и применяет полносвязанный слой с softmax активацией для прогнозирования его контекстных слов. Сеть хранит отдельный вектор для каждого слова и ограниченное количество векторов для N-грамм символов. Представление каждого слова вычисляется как сумма его вектора и векторов N-грамм его символов:

$$V = w^T E_w + \sum_{k=min}^{max} \sum_{i=1}^{n-k+1} s_{ki}^T E_s \tag{4.1}$$

$$Output = V \star E' \tag{4.2}$$

где W – множество слов; S – множество символьных N-грамм; $w \in {}_{+}^{|W|\times 1}$ – унитарный код слова; $s_{ki} \in {}_{+}^{|S|\times 1}$ – унитарный код символьного N-грамма; $E_w \in {}^{|W|\times dim}$ и $E_s \in {}^{|S|xdim}$ - матрицы векторов слов и символьных N-грамм соответственно; k – длина N-граммы (min и max – гиперпараметры); E’ $\in {}^{dim\times|W|}$ - параметры выходного слоя. В реализации модели символьные N-граммы отображаются во множество меньшего размера с помощью хеширования. Когда встречается незнакомое слово, его представление вычисляется как сумма векторов N-грамм.

В fastText вместо softmax выходные значения обрабатываются отдельно, как в задаче бинарной классификации. Матрицы векторов и параметры выходного слоя обучаются путем обратного распространения ошибки с использованием негативного семплирования и стохастического градиентного спуска.

Ngrams-only fastText В ngrams-only fastText (no-fastText), первой рассматриваемой модификации fastText, во время обучения и генерации представления слова игнорируется вектор целого слова и учитываются только N-граммы символов. Вектор вычисляется следующим образом:

$$V = \sum_{k=min}^{max} \sum_{i=1}^{n-k+1} s_{ki}^T E_s \tag{4.3}$$

В остальном модель идентична fastText. Данная модель основана на предположении, что символьные N-граммы слова несут достаточно информации, чтобы

восстановить его значение. По сравнению со словами, словарный запас символов N-грамм ограничен, и с использованием хеширования они отображаются в фиксированное число векторов. Следовательно, модифицированная модель имеет значительно меньше параметров, чем исходная.

Suffixes-only fastText Вторая предложенная модификация (suffixes-only fastText; so-fastText) аналогична первой, но сжимает модель еще больше. Она исключает внутренние N-граммы символов из модели, оставляя только N-граммы, которые в конце слова, то есть суффиксы. Представление слова вычисляется как сумма векторов его суффиксов s_k:

$$\sum_{k=min}^{max} s_k^T E_s \tag{4.4}$$

Предположение, лежащее в основе этой модели, заключается в том, что суффиксы могут содержать достаточно информации о форме слова, необходимой для морфологических задач [148]. Несмотря на то, что изменения позволяют еще больше сократить словарный запас, с учетом реализации и использования хеширования количество параметров остается неизменным по сравнению с no-fastText. В то же время, в этой модификации теоретически менее выражена проблема коллизий из fastText и no-fastText, где разные N-граммы отображаются в один и тот же вектор.

Подслова BPE BPE (англ. byte-pair encoding) изначально задумывалось как алгоритм сжатия данных[126], но в последнее время оно успешно применяется в нескольких задачах обработки естественного языка. Первоначальная цель алгоритма заключалась в том, чтобы поэтапно заменить частую пару байтов другим байтом, отсутствующим в данных. В задачах обработки текста он используется аналогичным образом: частые пары символов объединяются в новую запись, которая добавляется к существующему словарю символов, который сохраняется на протяжении всего использования алгоритма. Эти операции слияния можно применять до тех пор, пока в тексте не будет повторяться пара символов. Этот текст имеет роль обучающей выборки для создания словаря, который затем можно использовать для сжатия других текстов. Здесь тексты на самом деле не сжимаются, а их первоначальная символьная сегментация заменяется другой сегментацией, основанной на полученном словаре. Чтобы обучить BPE, необходим только набор

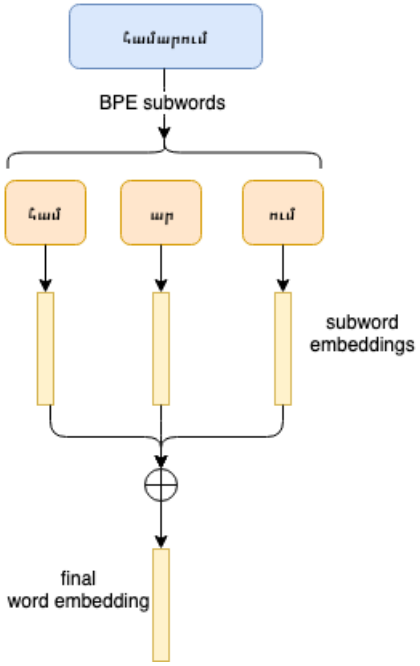


Рисунок 4.10 — Схема построения вектора слова на основе подслов BPE.

неразмеченных текстовых данных, достаточно большой для обучения осмысленных сегментов. Данные для обучения не требуют какой-либо конкретной предобработки или токенизации.

В этих экспериментах BPE используется для извлечения информации о подсловах из входного токена лемматизатора. В частности, входное слово кодируется с использованием BPE, а символы полученной формы служат в качестве подслов. Предполагается, что ключевая информация о входном слове, необходимая для его лемматизации, уже содержится в этих подсловах.

Подобно модификациям fastText, основная цель - уменьшить размер словаря векторных представлений. Однако в этих экспериментах представление слова заменено средним из представлений его подслов (рис. 4.10). Для последних представлений используются предварительно обученные модели из проекта BPEmb, а также новая модель, обученная на наших данных с помощью GloVe.

Представления подслов BPEmb - это набор предобученных векторных представлений для подслов BPE. Используя Википедию, авторы обучили BPE кодировку для 275 языков, включая армянский, с количеством операций слияния 1000, 3000, 5000, 10000, 25000, 50000, 100000, 200000. Затем, используя кодировку, они сегментировали данные и обучали векторные представления на нем. Для каждого языка доступны 25-, 50-, 100-, 200-, 300-мерные вложения. Для обучения представлений они использовали алгоритм GloVe.

Благодаря небольшому размеру словаря эти модели подслов с точки зрения количества параметров и занимаемой памяти значительно легче обычных представлений на уровне слов. (Размер словаря каждой модели — это сумма количества операций слияния и размера словаря начальных символов.)

В наших экспериментах были проверены представления размерности 50, 100, 200 и размеры словаря 10000, 25000, 50000. Модели обучались в многозадачной конфигурации. Результаты представлены в таблице 26.

Таблица 26 — Качество лемматизации (Accuracy) тестовой выборки для разных конфигураций размерности векторов BPEmb и размера словаря.

	vs=10000	vs=25000	vs=50000	average per dim
dim=50	90.39	90.29	90.61	90.43
dim=100	89.17	90.45	90.52	90.05
dim=200	90.00	90.19	89.84	90.01
average per vs	89.85	90.30	90.32	

Поскольку целью было уменьшить размер модели при сохранении того же уровня точности, большие значения размерности и операций слияния не тестировалось. Аналогичным образом, маленькие размерность и словарный запас были сочтены недостаточными для точной лемматизации, о чем также свидетельствуют полученные результаты.

Кроме BPEmb, также были обучены новые векторные представления для подслов, используя нашу коллекцию неразмеченных данных. Преимущество этой коллекции заключается в наличии дополнительных текстов из более широкого круга источников, а не только из Википедии. Однако для BPE сегментации текста используются те же модели, что и в BPEmb. Для обучения снова используется GloVe с фиксированным размером контекста 80 слов.

Таблица 27 — Качество лемматизации (Accuracy) тестовой выборки для разных конфигураций размерности новых векторов и размера словаря.

	vs=25000	vs=50000
dim=50	90.86	90.34
dim=100	90.84	90.10

По результатам предыдущего эксперимента тестировались только модели с размером словаря 25000, 50000 и размерностью 50, 100. Результаты представлены в таблице 27.

Таблица 28 — Сравнение качества (Accuracy) лемматизаторов на основе разных моделей векторного представления слов.

Модель векторов	Accuracy	Accuracy на OOV	Кол-во параметров векторов	Общее кол-во параметров
fastText	90.77	75.25	1.2×10^8	1.3×10^8
no-fastText	90.36	73.96	4.0×10^7	5.0×10^7
so-fastText	89.55	72.31	4.0×10^7	5.0×10^7
BPEmb	90.61	74.92	2.5×10^6	1.2×10^7
BPE-custom	90.86	75.62	1.2×10^6	1.1×10^7

В таблице 28 представлены результаты совместно обученного лемматизатора COMBO с использованием fastText и его сравнение с результатами, в которых fastText заменен вышеописанными альтернативами. С no-fastText и so-fastText достигается двукратное уменьшение размера модели, однако качество также ухудшается (особенно с so-fastText). Подходы на основе BPE позволили сократить количество параметров в векторной модели почти в 100 раз и уменьшить общее количество параметров лемматизатора примерно в 10 раз. Примечательно, что представления для подслов BPE, обученные в рамках этой работы, поддерживают тот же уровень точности, что и fastText, при этом требуя значительно меньше параметров.

В рамках этой диссертации была опубликована статья [18], где помимо моделей so-fastText и no-fastText представлена BPE-fastText, которая использует аналогичную архитектуру для обучения и вычисления векторов слов, но в качестве подслов использует морфемы BPE.

4.2.5.2 Обучение с частичным привлечением учителя

Распространенным использованием частичного привлечения учителя в NLP являются представления слов, предварительно обученные на неразмеченных данных. Первые попытки произвести непрерывные представления слов основывались на факторизации матрицы совместной встречаемости. Начиная с 2011 г. [115;

117; 149], предпочтение было отдано векторным представлениям слов, полученным с помощью нейронных сетей. word2vec, GloVe, fastText, в частности, быстро стали предпочтительным методом обучения представлений и показали state-of-the-art результаты для различных задач. Как упоминалось выше, нейронная сеть, используемая в этой работе, также использует аналогичное векторное представление во входных данных. Недавно было предложено несколько методов предобучения представлений на основе языковых моделей, которые показали свою эффективность как с векторами слов, так и без них [150].

Другой известный метод частичного привлечения учителя - это самообучение, когда модель обучается на размеченных данных, затем эта модель используется для разметки новых данных, которые затем используются для повторного обучения модели. Этот метод был предложен в [151] и является одним из первых методов обучения с частичным привлечением учителя. Также существует вариант самообучения, когда вместо жестких меток переобучение модели выполняется на выходных оценках вероятности целевых меток [152].

Помимо вышеперечисленного, существуют и более сложные техники. Одним из таких подходов является обучение с несколькими представлениями, при котором отдельные модели обучаются предсказывать одну и ту же целевую метку, но каждая имеет доступ только к подмножеству входных признаков [153]. Предсказания этих отдельных моделей затем используются для обучения друг друга. Более новый подход, называемый обучением с перекрестным просмотром, использует вспомогательные модули, которые принимают ограниченное представление о промежуточном представлении в качестве входных данных и на неразмеченных данных обучают их прогнозировать оценки целевых меток, предсказываемых основным модулем прогнозирования [154]. В основе кросс-ориентированного обучения лежит идея, что вспомогательные модули помогают лучше изучить промежуточные представления.

В этой работе обучение с частичным привлечением учителя используется для решения проблемы маленького размера обучающей выборки и рассматривается в первую очередь как средство увеличения данных. Одним из способов достижения этого является добавление случайного шума к входным данным (например, введение шума в векторы слов). Он основан на предположении, что обучение модели на зашумленных данных помогает модели лучше обобщать. Хотя его эффективность в задачах визуального распознавания установлена [155], этот метод

менее распространен при обработке текста. Самообучение успешно применялось в лемматизации [140], а также в других задачах обработки последовательностей слов, таких как морфосинтаксический анализ [156].

В этих экспериментах самообучение используется как подход к обучению с частичным привлечением учителя. В эксперименте предыдущая лучшая модель лемматизатора использовалась для разметки текстов из неразмеченного корпуса, с общим размеров около 250000 токенов. Затем модель была повторно обучена на новых размеченных данных. Результаты представлены в таблице 29.

Самостоятельное обучение не улучшило точность лемматизатора. Напротив, качество немного ухудщилось.

Таблица 29 — Сравнение моделей лемматизации с и без самообучения.

Вариант обучения	Accuracy	Accuracy на OOV	Кол-во параметров
С учителем	90.86	75.62	1.1×10^7
С самообучением	90.35	75.44	1.1×10^7

4.2.6 Обсуждение

Было проведено исследование подходов по повышению точности модели лемматизации за счет совместного обучения и самообучения, а также уменьшения размера модели за счет альтернативных векторных представлений. Полученная модель демонстрирует точность 90,86% на тестовом наборе ArmTDP v2.3, что превосходит установленные базовые показатели и превосходит единственный известный опубликованный результат - 88,05% анализатора восточноармянского языка.

Анализ предсказанных тестовых выборок показал, что бывают случаи, когда внутренний символ леммы предсказан неверно. Это происходит в основном с символами, которые имеют низкую частоту (например, ձ), и это стало возможным благодаря архитектуре COMBO. Чтобы избежать потери точности в таких случаях, в будущем имеет смысл посвятить работу улучшению нейронной сети или, возможно, обучению и применению векторов также для символов.

Другим возможным направлением исследования является обучение fastText на основе сегментов BPE вместо символьных N-грамм. Кроме того, в этой работе в качестве метода сегментации рассматривается только BPE, однако существуют и другие методы, такие как Morfessor, которые умеют сегментировать слова на семантически и морфологически значимые подслова.

Кроме того, важно проверить эффективность предложенных альтернативных представлений слов в различных архитектурах лемматизаторов, а также других морфосинтаксических задач, таких как частеречная разметка. Теоретически предлагаемые представления на основе подслов также должны быть устойчивыми к шумному вводу (например, неправильно написанному слову). Подтверждение этой гипотезы также является интересным направлением будущей работы.

Примененный метод самообучения не улучшил качество лемматизатора. В будущем стоит поэкспериментировать с другими подходами к увеличению обучающих данных. В частности, для армянского языка возможно имеет смысл попытаться распарсить Викисловарь, используя подход [157], или использовать преобразователи Apertium для генерации обучающих данных.

4.3 Исправление ошибок автоматического распознавания текстов

В этом разделе рассмотрена проблема автоматического распознавания текста на армянском языке, в частности, решение задачи исправления ошибок автоматического распознавания. Результаты были опубликованы в статье [15]. Используется двухэтапный подход к решению задачи: (i) обнаружение ошибок распознавания с помощью многослойного персептрона и (ii) исправление ошибок с помощью преобразователя последовательности на основе сверточной нейронной сети. Предложенные методы постобработки позволяют уменьшить количество ошибок в словах из статей Советско-Армянской Энциклопедии, распознанных Tesseract OCR, на 23.5%, достигая результатов, сравнимых с коммерческими решениями данной задачи.

4.3.1 Введение

Важным этапов в процессе определения уникальности документа является извлечение текста из проверяемого документа. Для текстовых файлов и других форматов возможно извлечь текст путем прямого чтения файла или его текстового слоя, однако даже в таких случаях содержание может быть искажено, например, для скрытия заимствования или искусственного увеличения количества псевдоуникальных участков текста. По этой причине наиболее надежным способом извлечение текста из документа является его оптическое распознавание (OCR).

Для армянского языка выбор инструментов OCR небольшой. Из некоммерческих проектов, открытый доступ предоставляет Tesseract, который, однако, часто допускает много ошибок распознавание для армянского языка. Разработка более точного инструмента распознавания – это трудоемкий процесс, для которого необходимо отдельное и объемное исследование. В рамках этой работы было выбрано другое направление улучшения результатов распознавания – его постобработка с целью нахождения и исправления ошибок.

Для постобработки ошибок OCR в армянских текстах предлагается использование двухэтапного подхода: (i) обнаружение ошибок распознавания с помощью многослойного персептрона и (ii) исправление ошибок с помощью преобразования последовательности на основе сверточной нейронной сети. Предложенные методы постобработки позволяют уменьшить количество ошибок в словах из статей Советско-Армянской Энциклопедии, распознанных Tesseract OCR, на 23.5%, достигая результатов, сравнимых с коммерческими решениями данной задачи.

Оптическое распознавание символов (OCR) – это процесс преобразования рукописных, машинописных или печатных текстовых изображений в машиночитаемый текст. Из-за различных факторов, таких как низкое качество изображения или неточность инструментов OCR, ряд ошибок может возникать на разных этапах OCR, таких как обнаружение текста, обнаружение границ слова, сегментация символов, классификация символов. Возникновение таких ошибок мешает OCR правильно распознавать символы, вызывая орфографические ошибки и затрудняя понимание смысла текста. Следовательно, в задачах, где точность текстовой ин-

формации имеет решающее значение, перед их использованием необходима дополнительная постобработка результатов распознавания текста.

Существуют различные инструменты для распознавания текста на армянском языке, однако немногие из них предоставляют свободный доступ. Tesseract [158] - это инструмент OCR с открытым исходным кодом, который поддерживает множество языков, включая армянский. Проблема с Tesseract заключается в том, что его производительность резко снижается для изображений с низким разрешением или текстов с редкими шрифтами, что приводит к более высокому уровню ошибок распознавания текста (например, «ւոեււակը» вместо «տեսակը»; «փուլից» вместо «փոլից») в старых текстах. Для армянского языка в результатах Tesseract довольно часто встречаются ошибки распознавания символов. Ручная коррекция текстов после распознавания текста требует много времени и человеческих ресурсов, поэтому исследования ведутся в направлении автоматизации этой работы. Учитывая наличие большого количества неоцифрованной литературы и других текстовых ресурсов, создание такого инструмента имеет большое значение для армянского языка. Поэтому в этой работе делается попытка решить эту проблему и исследуются методы автоматизации этапа постобработки. Предложенные методы могут быть использованы для сбора и анализа текстовой информации, при разработке систем обнаружения заимствований и компьютерной лингвистики.

Следуя [159], статье, посвященной постобработке OCR для английского языка, здесь также рассматривается разработка двухэтапного решения, с использованием отдельной модели для обнаружения ошибок и другой модели для их исправления. В этой работе были исследованы различные алгоритмы как для обнаружения ошибок, так и для задач исправления ошибок, уделяя особое внимание подходам, основанным на машинном обучении, учитывая их успех в связанных задачах [160; 161]. Алгоритмы подробно описаны в разделе 2, а в разделе 3 представлены результаты экспериментов и их анализ.

4.3.2 Методы обнаружения и исправления ошибок

В этом разделе описываются этапы постобработки текстов OCR и применяемые в каждом из них методы. Результат OCR сначала обрабатывается предварительно для токенизации, удаления разрывов строк и переноса слов, затем токены классифицируются для обнаружения ошибок, и, наконец, обнаруженные ошибки исправляются и заменяются с помощью метода исправления ошибок (рисунок 4.11). Подходы к решению задач обнаружения и исправления ошибок подробно описаны ниже.

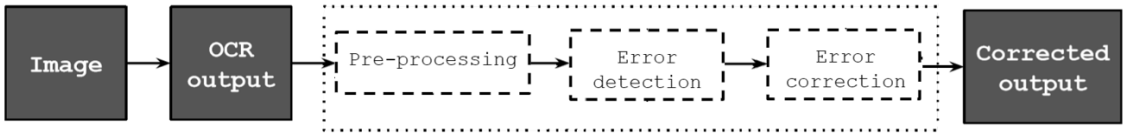


Рисунок 4.11 — Этапы постобработки результатов OCR.

4.3.2.1 Обнаружение ошибок OCR

Первая задача – выявить ошибочные токены в распознанном тексте. Часто встречающиеся ошибочные слова содержат буквы с разным регистром (например, ’բոդԲՈՔայիՆ’), лишние пробелы (например, ’Մ ե թ ո դ ա կ ա ն’), более одного последовательного символа ’ւ’ (этот символ обычно встречается только после символа ’n ’), неправильная классификация похожих символов (например,’ պատերազմի ’).

В качестве базового решения для первой задачи использовался поиск по словарю. Для реализации этого алгоритма был построен словарь на основе текстов армянской Википедии, новостных статей и художественной литературы [20]. В этом методе токен считался ошибочным, если он отсутствовал в словаре, и считался правильным в противном случае.

Второй рассмотренный метод – Наивный байесовский классификатор. Используя слово и его символьные N-граммы (N = 1, 2, 3) в качестве признаков, был обучен полиномиальный Наивный байесовский классификатор, который классифицирует слово как ошибочное (1) или нет (0).

Чтобы классифицировать слова, мы также обучили многослойный персептрон (MLP) с двумя скрытыми слоями и 128 нейронами в каждом слое. Для MLP использовались те же входные признаки, что и в байесовском классификаторе.

4.3.2.2 Исправление ошибок OCR

Вторая задача – это исправление ошибочных токенов, обнаруженных в первой задаче. Были исследованы 3 метода для этой задачи: базовое решение на основе поиска по словарю с использованием расстояния Левенштейна, кодировщик-декодировщик с механизмом внимания и более простой преобразователь последовательности на основе сверточной нейронной сети.

Поиск по словарю: для этого алгоритма был построен словарь частотности слов, используя тот же корпус, что и при обнаружении ошибок. Для каждого ошибочного слова алгоритм выполняет следующие шаги:

1. Вычисляется расстояние редактирования Левенштейна между этим словом и словами в словаре.
2. Слово с минимальным расстоянием возвращается как исправление; если таких слов несколько, возвращается самое распространенное слово.

Дополнительно, также был отдельно построен и протестирован поиск по словарю биграмм используя биграммы токенов. Это было мотивировано наличием примеров, когда токен был неправильно разделен пробелом или между двумя токенами отсутствовал пробел.

Кодировщик-декодировщик: Рассматривая исправление ошибок OCR как задачу трансдукции последовательности входных символов ошибочного токена к последовательности символов исправленного токена, была исследована возможность использования модели кодировщика-декодировщика с механизмом внимания, в частности с использованием нейронной сети Transformer [99]. Входной последовательностью были символы обнаруженного ошибочного токена, а последовательность символов его исправления – выходом декодировщика. Для реализации и обучения сети использовалась библиотека обучения последовательностей OpenNMT-tf [162]. Размер входных и выходных словарей был ограничен 150 символами с общими 32-мерными вложениями.

Rybak et al.: В ходе экспериментов, поскольку первоначальные результаты Transformer были неудовлетворительными, а также с учетом времени и ресурсов, необходимых для правильной настройки и обучения гиперпараметров, было принято решение использовать более простую модель преобразования последовательности, основанную на сверточной нейронной сети, используя архитектуру из [140].

Первоначально разработанная для решения задачи лемматизации текстов, модель [140] также хорошо подходит для этой задачи, учитывая их схожую природу. В этой модели каждый входной токен представляется с помощью символьных признаков, полученных с использованием расширяющейся сверточной сети (CNN), и обучаемого вектора. Эти два представления слова объединяются и обрабатываются biLSTM слоем на уровне предложения, чтобы получить окончательный контекстный вектор входного токена. Размерность этого вектора далее уменьшается с помощью полносвязного слоя, после чего полученный вектор конкатенируется с вектором каждого символа входного токена. Полученные векторные представления символов подаются на вход расширяющейся CNN, последний уровень которой вычисляет вероятности символов исправленного слова.

Полный список гиперпараметров нейронных сетей приведен в Приложении В.

4.3.3 Эксперименты

4.3.3.1 Наборы данных для обучения и тестирования

В качестве набора данных для экспериментов использовались отсканированные страницы и их оцифрованный текст из Армянской советской энциклопедии, доступной в рамках проекта оцифровки, проводимого Армянской Википедией. Выбор этого источника для набора данных был определен, прежде всего, наличием отсканированных изображений его статей и их вычитанных и исправленных текстов. Этот источник также подходил для наших экспериментов по той

причине, что представляет особую сложность для инструментов OCR из-за его старомодного шрифта и структуры страниц.

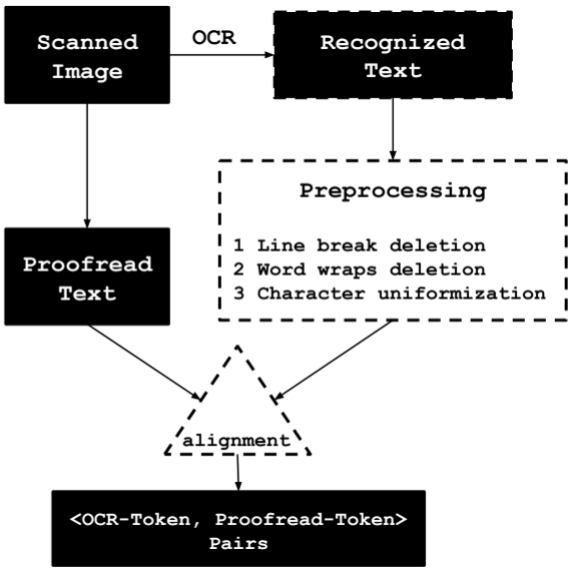


Рисунок 4.12 — Схема генерации примеров для обучения и тестирования.

Чтобы сгенерировать наборы данных для обучения и оценки качества моделей, мы применили OCR к отсканированным изображениям статей и согласовали вывод с исправленными версиями (рисунок 4.12).

На первом этапе создания набора данных с помощью Tesseract извлекался текст из отсканированных изображений, а затем применялась предобработка. Во время предварительной обработки в каждом извлеченном тексте были внесены следующие изменения:

1. Удаление разрывов строк Переносы строк были удалены как в распознанном, так и в исходном (исправленном) тексте.
2. Удаление переноса слов В соответствии с правилами переноса слов в армянском языке, в слове при переносе может появляться буква «ը». Если перед дефисом стоит буква ը или часть слова перед дефисом имеет форму «хըу» (х – одна или несколько букв, у – согласная), то буква ը также удаляется, в других случаях буква ը остается, так как у нас нет информации, была она в слове до переноса слов или нет. Например, слово ‘ջը-րանցքը’ после удаления переноса слов будет иметь следующую форму ‘ջրանցքը’, слово ‘շը-ջանի’ станет ‘շջանի’.
3. Унификация символов При распознавании текста же знак препинания может быть ошибочно заменена другим символом, имеющим одинаковый глиф (например, дефис и тире, или верчакет и точка). В рамках пре-

добработки текстов все омоглифы были приведены к одной и той же форме с помощью библиотеки homoglyphs.

Затем, для сопоставления токенов в исходном и извлеченном текстах, использовалась функцию терминала git diff. На основе выходных данных функции получилось сгенерировать набор данных, состоящий из 2.2 миллиона пар <токен OCR; исправленный токен>. Ошибочные примеры составили примерно 20% набора данных. Набор данных был случайным образом разделен на обучающий (80%), проверочный (10%) и тестовый (10%) наборы, так что соотношение данных по классам сохранялось. Такое же разделение использовалось для задач обнаружения и исправления ошибок. Для последнего использовались только пары, содержащие ошибки.

При обучении нейронных сетей задаче исправления ошибок возникали трудности из-за наличия «сложных» примеров, таких как извлеченные слова, оригиналы которых были на иностранном языке, токены, содержащие числа или ошибку пунктуации, слова с несколькими неправильно распознанными буквами. Чтобы облегчить обучение сети, образцы вышеуказанных типов были либо отфильтрованы из обучающей выборки, либо упрощены. Нейронные сети были обучены и проверены только на этом очищенном наборе, но все модели были протестированы как на очищенном, так и на полном наборе.

4.3.3.2 Результаты и обсуждение

Таблица 30 — Оценка качества моделей обнаружения ошибок.

Модель	Accuracy	Точность	Полнота	F1
Словарь	87.6	69.5	69.4	69.4
Наивный Байес	89.0	73.3	74.2	73.8
MLP	95.2	95.8	80.0	87.5

Обнаружение ошибок: для оценки качества разработанных моделей использовались accuracy, точность, полнота и F1. Результаты описанных моделей для задачи обнаружения ошибок представлены в таблице 30. Базовое решение на основе словаря и Наивный байесовский классификатор (Naive Bayes) проде-

монстрировали относительно низкую точность, которую в первом случае можно было бы улучшить с помощью расширенного словарного запаса.

Лучший результат был достигнут с помощью MLP, который обеспечил точность 95.8% и полноту 80.0%. Это означает, что в среднем 19 из 20 предсказанных ошибок действительно содержали ошибку. При рассмотрении ложноотрицательных результатов около 15% представляли случаи, когда один знак препинания неправильно распознавался как другой («.» вместо «,»). Среди оставшихся ложноотрицательных результатов 36.6% составили токены, содержащие прописные буквы (например, «ԻՆԱՄԱԿԱԼՈՒԹՅՈՒՆ», «օրՈք», «Ճյուղը» и т.д.), некоторые из которых можно было зафиксировать с помощью добавления дополнительных признаков. Также следует отметить, что многие из неправильно классифицированных токенов были бы правильными словами в другом контексте.

Таблица 31 — Результаты методов исправления ошибок.

Модель	Accuracy на полном наборе	Accuracy на очищенном наборе
Словарь	22.00	12.10
Словарь (+биграммы)	25.35	13.81
Rybak et al.	21.62	17.48

Исправление ошибок: нейронная сеть Рыбак и др. показала наивысшую accuracy в этой задаче (таблица 31). Однако на очищенном и упрощенном наборе примеров его качество соответствовало (а в случае биграмм уступало) базовому методу поиска по словарю. После 20000 шагов обучения Transformer достиг лишь 6.95% accuracy, после чего обучение было прекращено.

На полной тестовой выборке лучший результат составил 17.48%. Чтобы лучше понять причину такой низкой точности, был проведен анализ тестовых примеров. Для более чем 35% тестового набора токен OCR и его исправленная версия имели расстояние Левенштейна выше 5. Для этих образцов accuracy исправления ошибок была ниже 5% (рисунок 4.13). Следует отметить, что многие образцы содержат множество неправильно распознанных символов, и их будет сложно исправить даже людям. Примеры таких образцов из тестового набора приведены в таблице 32. Очевидно, что качество OCR оказывает серьезное ограничивающее влияние на общий потенциал методов постобработки. Добавление контекстной информации может потенциально помочь с некоторыми из этих примеров, однако потребуются эффективные механизмы кодирования контекста.

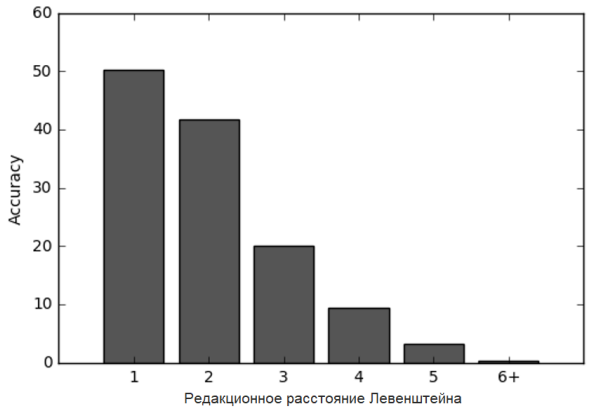


Рисунок 4.13 — Accuracy модели Rybak et al. в задаче исправления ошибок в зависимости от расстояния редактирования между токеном OCR и исправленным токеном.

Таблица 32 — Примеры токенов, особенно сложных для исправления.

Исходный текст	Результат распознавания
գաղութացումը	զաղոյթացւլւմը
քննադատությունը	սնոտդատությունը
այգեգործությամբ	աւգեցող ծուրլամբ
նախագահել	նախազնԻ հել
զրահամեքենա են	գրսւեամեյլենա նկ
Հրաժարվելով	<րամարվեւդվ
տիրույթներից	տիոաւթննոիո
պոլիմերը	ոլոլիմերլւ
ռադիոզգայնությունը	ոաոիոկգայնությանյյ
առանձնացնում	աոլււնյլնացնում

Сравнение с другими инструментами OCR: чтобы проанализировать влияние разработанных методов постобработки на качество текста, для 100 случайно выбранных статей была вычислена средняя пословная вероятность ошибки (WER) между исправленным текстом и выводом OCR до и после применения постобработки. Для постобработки использовалась комбинация наиболее эффективных моделей: MLP для обнаружения ошибок и Rybak et al. для исправления ошибок. Среднее значение WER без постобработки составило 0.51, но с постобработкой улучшилось на 23.5%, снизившись в среднем до 0.39. На рисунках 4.14а и 4.14б показаны образцы выходных данных OCR до и после исправления.

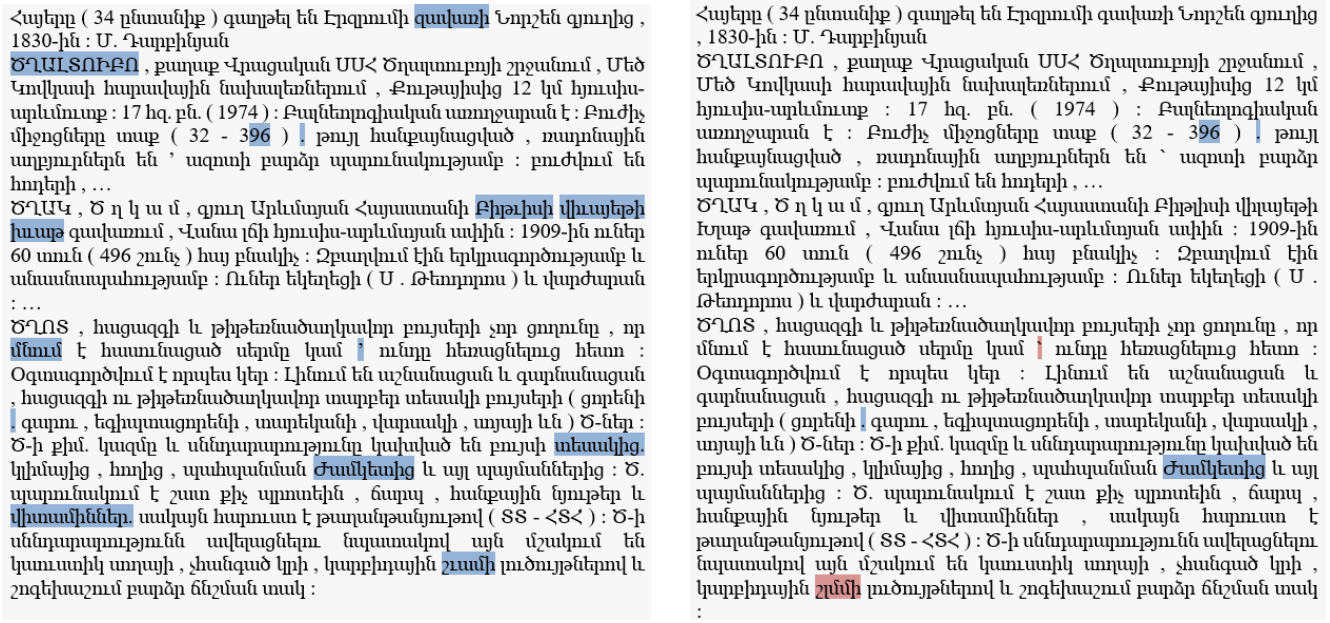
Таблица 33 — Сравнение качества Tesseract с постобработкой с результатом других инструментов.

Модель	WER	Доступность
Google Docs	1.10	Бесплатный
Convertio	0.27	Коммерческий
ABBYY FineReader	0.54	Коммерческий
Tesseract	0.51	Бесплатный
Tesseract + постобработка	0.39	Бесплатный

Чтобы лучше понять эффективность методов постобработки, было проведено сравнение результатов инструментов OCR с результатом Tesseract и постобработки (Таблица 33). Tesseract с применением разработанных методов постобработки продемонстрировал наилучшее качество, за исключением коммерческого продукта Convertio. Стоит отметить, что даже без постобработки качество распознавания Tesseract было сопоставимо с пословной частотой ошибок коммерческих продуктов, таких как ABBYY FineReader. Результат Google Docs OCR оказался на удивление низким, однако анализ его результатов показал, что низкая оценка была вызвана неспособностью правильно читать текстовые столбцы.

4.4 Извлечение именованных сущностей

В этом разделе представлено решение задачи распознавания именованных сущностей для армянского языка. Для решения этой задачи было проведено исследование существующих методов, были разработаны наборы размеченных дан-



а) б)

Рисунок 4.14 — Пример применения разработанных методов к тексту из АСЭ. а) Результат автоматического распознавания без постобработки (ошибки выделены синим). б) Результат автоматического распознавания после постобработки (синим выделены необнаруженные ошибки, фиолетовым - неправильно исправленные).

ных серебряного и золотого стандартов, а также установлены базовые результаты на популярных моделях. В результате был создан корпус именованных сущностей с 163000 токенами, автоматически сгенерированный из Википедии, и еще один корпус новостных предложений с 53400 токенами с ручной разметкой именованных сущностей: людей, организаций и географических объектов. Корпуса использовались для обучения и оценки нескольких популярных моделей распознавания именованных сущностей. Наряду с наборами данных разработаны 50-, 100-, 200-, 300-мерные представления слов GloVe, обученные на коллекции армянских текстов из Википедии, новостей, блогов и энциклопедии. Результаты опубликованы в статье [16].

Стоит подчеркнуть важность тестового корпуса, который может служить эталоном для будущих систем распознавания именованных сущностей, разработанных для армянского языка. Кроме того, чтобы установить применимость основанных на Википедии подходов к армянскому языку, предоставляются результаты оценки для 3 различных систем распознавания именованных сущностей, обученных и протестированных на наших наборах данных. Результаты подтверждают

способность подходов глубокого обучения к достижению относительно высоких значений полноты для этой конкретной задачи, а также эффективность использования плотных представлений на основе символов наряду с обычными векторными представлениями слов.

4.4.1 Введение

Распознавание именованных сущностей - важная задача обработки естественного языка, которая присутствует во многих популярных инструментах обработки текста. Эта область обработки естественного языка активно изучалась в последние десятилетия, и появление глубокого обучения оживило исследования более эффективных и точных моделей. Однако для большинства существующих подходов требуются большие размеченные корпуса. Ранее для армянского языка такая работа не проводилась, и поэтому в этой работе решается несколько проблем, включая создание корпуса для обучения моделей машинного обучения, разработку тестового корпуса золотого стандарта и оценку качества устоявшихся подходов к распознаванию именованных сущностей.

В этом разделе представляется решение задачи распознавания именованных сущностей для армянского языка. Для решения этой задачи было проведено исследование существующих методов, были разработаны наборы размеченных данных серебряного и золотого стандартов, а также установлены базовые результаты на популярных моделях. Представляется корпус именованных сущностей с 163000 токенами, автоматически сгенерированный из Википедии, и еще один корпус новостных предложений с 53400 токенами с ручной разметкой именованных сущностей: людей, организаций и географических объектов. Корпуса использовались для обучения и оценки нескольких популярных моделей распознавания именованных сущностей. Наряду с наборами данных разработаны 50-, 100-, 200-, 300-мерные представления слов GloVe, обученные на коллекции армянских текстов из Википедии, новостей, блогов и энциклопедии.

4.4.2 Наборы данных

4.4.2.1 Автоматическая генерация обучающих данных

Обзор литературы Учитывая затратность создания корпуса именованных сущностей с ручной разметкой, рассматривались альтернативные подходы. Отсутствие корпусов именованных сущностей является общей проблемой для многих языков, что привлекает внимание многих исследователей по всему миру. Схемы передачи разметки на основе проекций оказались очень эффективными (например, [163–165]) с использованием корпусов языка с богатыми ресурсами для создания размеченных данных для языка с низким уровнем ресурсов. В этом подходе разметка языка с высоким уровнем ресурсов проецируется на соответствующие токены текстов параллельного языка с низким уровнем ресурсов. Эта стратегия может применяться к языковым парам, имеющим параллельные корпуса. Однако этот подход не сработает для армянского языка, поскольку у нас не было доступа к достаточно большому параллельному корпусу с богатым ресурсами языком.

Другой популярный подход - использование Википедии. Клести Ходжа и Артур Баксаку используют географические справочники, извлеченные из Википедии, для создания размеченного корпуса для албанского языка [166], а Вебер и Потзл предлагают основанную на правилах систему для немецкого языка, которая использует информацию из Википедии [167]. Однако последний полагается на внешние инструменты, такие как анализаторы части речи, что на момент проведения исследования делало его неподходящим для армянского языка.

Nothman et al. сгенерировали корпус серебряного стандарта для 9 языков, извлекая тексты статей из Википедии с исходящими ссылками и превращая эти ссылки в метки именованных сущностей на основе типа целевой статьи [168]. Сысоев и Андрианов применили аналогичный подход к русскому языку [169; 170]. Основываясь на его успехе для широкого диапазона языков, выбор остановился на этой модели, для автоматической генерации данных и их разметки.

Алгоритм генерации размеченных предложений на основе Википедии В работе использовалась модификация подходов [169] и [168] к автоматической гене-

рации данных для обучения распознавателя именованных сущностей. Этот подход использует ссылки между статьями Википедии для создания последовательностей размеченных токенов именованных сущностей.

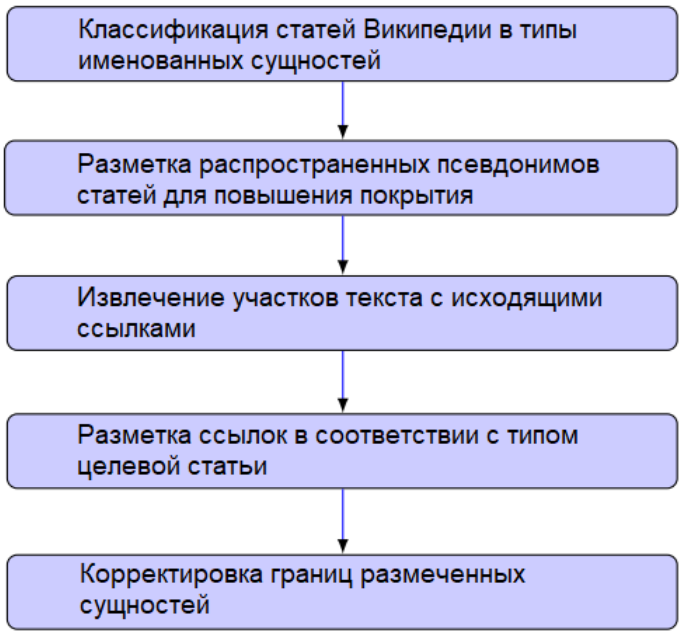


Рисунок 4.15 — Алгоритм автоматической генерации размеченных предложений.

Во-первых, каждой статье Википедии присваивается тип именованной сущности (например, статья Քիմ Քաշքաշյան (Ким Кашкашян) классифицируется как PER (человек), Ազգերի լիգա (Лига Наций) как ORG (организация), Սիրիա (Сирия) как LOC и т.д.). Одно из основных различий между нашим подходом и системой из [168] заключается в том, что мы не полагаемся на ручную классификацию статей и не используем межъязыковые ссылки для классификации статей проекта на разных языках. Вместо этого наш алгоритм классификации использует только значение атрибута `subclass of` первого значения атрибута `instance of` статьи в Викиданных. Значения атрибутов универсальны для языков, и, таким образом, могут использоваться для любого языка Википедии.

Затем исходящим ссылкам в статьях присваивается тип статьи, на которую они ведут. Предложения включаются в обучающий корпус только в том случае, если они содержат хотя бы одну именованную сущность и все слова, написанные с заглавной буквы, имеют исходящую ссылку на статью известного типа. Поскольку в статьях Википедии дается ссылка только на первое упоминание каждой сущности, этот подход становится очень ограничительным, и для включения большего количества предложений выводятся дополнительные ссылки. Это до-

стигается путем составления списка частых альтернативных названий для статей, соответствующих именованным сущностям, а затем поиска фрагментов текста с этим названием для присвоения метки именованной сущности. Названия статьи включают ее заголовок, заголовки страниц значений со статьей и тексты ссылок, ведущих к статье (например, Լենինգրադ (Ленинград), Պետրոգրադ (Петроград), Պետերբուրգ (Петербург)) являются псевдонимами для Սանկտ Պետերբուրգ (Санкт-Петербург)). Такой список названий составляется для всех статей PER, ORG, LOC.

После этого границы ссылок корректируются путем удаления меток для выражений в круглых скобках, текста после запятой и в некоторых случаях разделения на отдельные именованные сущности, если текст со ссылкой содержит запятую. Например, [LOC Աբովյան (քաղաք)] (Абовян (город)) корректируется в [LOC Աբովյան] (քաղաք).

Вместо того, чтобы вручную классифицировать статьи Википедии, как это было сделано в [168], был разработан классификатор на основе правил, который использовал атрибуты статьи Wikidata `instance of` и `subclass of` для поиска соответствующего типа именованной сущности.

Классификация может быть выполнена с использованием только меток `instance of`, но эти метки излишне специфичны для задачи, и построение правил для них потребует более трудоемкой и скрупулёзной работы. Таким образом, статьи классифицировались на основе значения `subclass of` первого значения атрибута `instance of`. Таблица 34 показывает составленное отображение этих значений в типы именованных сущностей. Использование значений `subclass of` более высокого уровня было недопустимым, поскольку их значения часто были слишком общими, что делало невозможным получение правильной категории для сущности.

Используя описанный выше алгоритм, было сгенерировано 7455 размеченных предложений с 163247 токенами на основе версии армянской Википедии от 20 февраля 2018 года.

Сгенерированные данные по-прежнему значительно меньше, чем размеченные вручную корпуса из CoNLL 2002 и 2003. Для сравнения, обучающий набор английского корпуса CoNLL 2003 содержит 203621 токен, немецкий - 206931, а испанский и голландский корпуса из CoNLL 2002 соответственно 273037 и 218737 строк. Меньший размер сгенерированных нами данных можно объяснить строгим

Таблица 34 — Таблица значений атрибутов subclass of и соответствующей метки типа именованной сущности.

Значение атрибута subclass of	Тип сущности
company, business enterprise, company, juridical person, air carrier, political organization, government organization, secret service, political party, international organization, alliance, armed organization, higher education institution, educational institution, university, educational organization, school, fictional magic school, broadcaster, newspaper, periodical literature, religious organization, football club, sports team, musical ensemble, music organisation, vocal-musical ensemble, sports organization, criminal organization, museum of culture, scientific organisation, non-governmental organization, nonprofit organization, national sports team, legal person, scholarly publication, academic journal, association, band, sports club, institution, medical facility	ORG
state, disputed territory, country, occupied territory, political territorial entity, city, town, village, rural area, rural settlement, urban-type settlement, geographical object, geographic location, geographic region, community, administrative territorial entity, former administrative territorial entity, human settlement, county, province, federated state, district, county-equivalent, municipal formation, raion, nahiyah, mintaqah, muhafazah, realm, principality, historical country, watercourse, lake, sea, still waters, body of water, landmass, minor planet, landform, natural geographic object, mountain range, mountain, protected area, national park, geographic region, geographic location, arena, bridge, airport, stadium, performing arts center, public building, venue, sports venue, church, temple, place of worship, retail building	LOC
person, fictional character, fictional humanoid, human who may be fictional, given name, fictional human, magician in fantasy	PER

отбором предложений-кандидатов, а также просто относительно небольшим размером армянской Википедии.

Точность разметки в сгенерированном корпусе во многом зависит от качества ссылок в статьях Википедии. Во время генерации предполагалось, что первые упоминания всех именованных сущностей имеют исходящую ссылку на их статью, однако это не всегда так было в реальных исходных данных, и в результате обучающий набор содержал предложения, в которых не все именованные сущности помечены. Неточности в метках также связаны с неправильно заданными границами ссылок (например, в статье Википедии “Արթուր Ուելսլի Վելինգթոն” (Артур Веллесли) есть ссылка на статью Наполеона с текстом “է Նապոլեոնը” (”Napoleon is ”), когда должно быть “Նապոլեոնը” (”Napoleon”)). Другой тип рас-

пространенных ошибок разметки возникал, когда именованный объект появлялся внутри ссылки, не нацеленной на статью типа LOC, ORG, PER например, “ԱՄՆ նախագահական ընտրություններում” (”Президентские выборы в США”) связана со статьей “ԱՄՆ նախաահական ընտրություններ 2016” (президентские выборы в США, 2016), и в результате метка [LOC ԱՄՆ] (США) потеряна.

Разметка тестовых данных Помимо создания данных для обучения, также обращается внимание на отсутствие набора проверочных данных для распознавания именованных сущностей. Для решения этой проблемы создается корпус золотого стандарта с ручной разметкой именованных сущностей CoNLL: личность, географический объект и организация [171; 172], в том числе в надежде, что он будет использоваться для оценки будущих моделей распознавания именованных сущностей.

Чтобы оценить модели, обученные на сгенерированных данных, вручную был размечен тестовый датасет именованных сущностей, состоящий из 53453 токенов и 2566 предложений, выбранных из более чем 250 текстов новостей из ilur.am[11]. Этот набор данных сопоставим по размеру с тестовыми наборами других языков (Таблица 35). Включены предложения из политических, спортивных, местных и мировых новостей (рисунки 4.17а и 4.17б), охватывающие период с августа 2012 года по июль 2018 года. Набор данных содержит метки для 3 популярных классов именованных сущностей: личности (PER), организации (ORG) и местоположения (LOC), и публикуется в формате CoNLL03 со схемой разметки IOB. Слова и предложения были токенизированы в соответствии со стандартами UD для армянского языка [119].

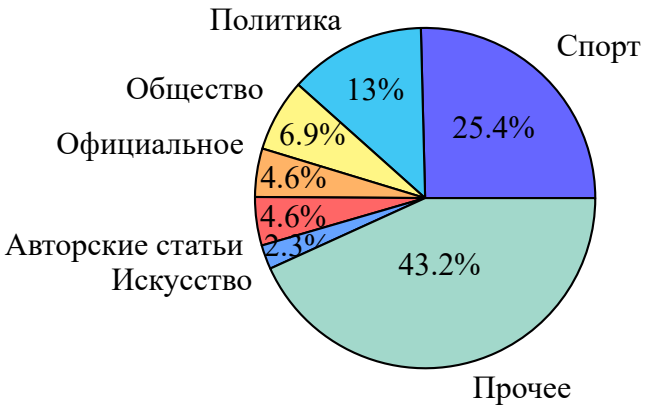
Таблица 35 — Сравнение тестовых наборов для армянского, английского, немецкого и русского языков.

Тестовый набор	#(Токены)	#(LOC)	#(ORG)	#(PER)
Armenian	53453	1306	1337	1274
English CoNLL03	46435	1668	1661	1617
German CoNLL03	51943	1035	773	1195
Spanish CoNLL02	51533	1084	1400	735
Russian factRuEval-2016	59382	1239	1595	1353

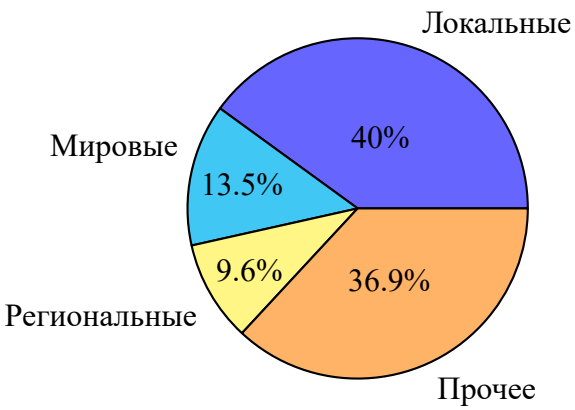
[11] http://ilur.am/news/newsline.html

Рисунок 4.16 — Состав и распределение текстов в тестовом наборе.

а) Категория новостей.



б) География новостей.



Во время разметки использовались категории и рекомендации, предложенные BBN Technologies для соревнования по вопросно-ответным системам TREC 2002[12]. Только именованные сущности, соответствующие категории person BBN, были помечены как PER. К ним относятся собственные имена людей, включая вымышленных персонажей, имена и фамилии, фамилии, уникальные прозвища. Аналогичным образом категории organization name, включая названия компаний, правительственные учреждения, образовательные и академические учреждения, спортивные клубы, музыкальные ансамбли и другие группы, больницы, музеи, названия газет, были отмечены как ORG. Однако, в отличие от BBN, мы не отмечали прилагательные формы названий организаций как именованные сущности. Категории BBN gpe name, facility name, location name были объединены и размечены как LOC.

Сущности других категорий (например, произведения искусства, законы или события) были игнорированы, в том числе те случаи, когда сущность ORG, LOC или PER находилась внутри объекта постороннего типа (например, ՀՀ (RA) в ՀՀ Քրեական Օրենսգիրք (Уголовный кодекс РА) не был размечен как LOC).

Кавычки вокруг названной организации не были включены в разметку, если эти кавычки не были частью полного официального названия этой организации (например, «Նաիրիտ գործարան» ՓԲԸ (ЗАО «Завод Наирит»)).

В зависимости от контекста такие метонимы, как Կրեմլ (Кремль), Բաղրամյան 26 (Баграмян 26), когда использовались как псевдоним соответствующих государственных учрежденияй, размечались как ORG. Аналогичным обра-

[12]https://catalog.ldc.upenn.edu/docs/LDC2005T33/BBN-Types-Subtypes.html

зом, названия стран или городов также были помечены как ORG, когда использовались для обращения к их представляющим спортивным командам.

4.4.3 Модели и эксперименты

В этом разделе описывается ряд экспериментов, направленных на сравнение качества популярных алгоритмов распознавания именованных сущностей на наших данных. Были обучены и протестированы Stanford NER[13], spaCy 2.0[14], а также рекуррентную модель, аналогично [173; 174] использующую двунаправленные ячейки LSTM для извлечения признаков на основе символов и CRF, описанный в статье Гийома Джентиала “Sequence Tagging with Tensorflow” [175].

4.4.3.1 Векторные представления слов

Помимо наборов данных, в этой работе были также были разработаны плотные представления слов для армянского языка, которые мы использовали в экспериментах для обучения и оценки алгоритмов распознавания именованных сущностей. Учитывая способность фиксировать семантические закономерности, использовалась модель GloVe для обучения векторному представлению слов. Был собран датасет армянских текстов, содержащий 79 миллионов токенов из статей Армянской Википедии, Армянской Советской Энциклопедии, подкорпуса Восточно-Армянского национального корпуса [125], более десятка армянских новостных сайтов и блогов. Включенные тексты охватывают такие темы и жанры, как экономика, политика, прогноз погоды, информационные технологии, право, общество и политика, научная и художественная литература.

Подобно оригинальным представлениям, опубликованным для английского языка, для армянского также обучаются 50-, 100-, 200- и 300-мерные векторы слов с размером словарного запаса 400000. Перед обучением все слова в набо-

[13] https://nlp.stanford.edu/software/CRF-NER.shtml
[14] https://spacy.io/

ре данных были приведены к нижнему регистру. Для финальных моделей были использованы следующие гиперпараметры обучения: 15 размер окна и 20 эпох обучения.

4.4.3.2 Модели распознавания и классификации сущностей

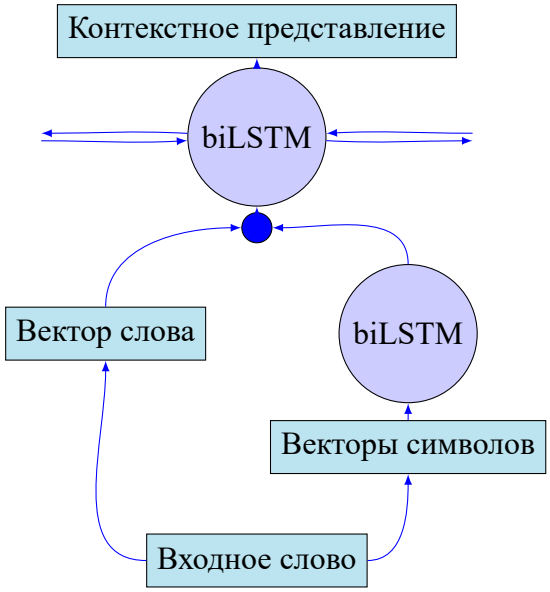
Stanford NER – это классификатор на основе условных случайных полей (CRF), использующий лексические и контекстные признаки, такие как текущее слово, N-граммы уровня символа длиной до 6 в начале и в конце, предыдущие и следующие слова, форма слова и особенности последовательности [176].

spaCy 2.0 использует систему перехода на основе CNN для распознавания именованных сущностей. Для каждого токена представление Блума рассчитывается на основе его строчной формы, префикса, суффикса и формы, а затем с использованием остаточных CNN извлекается контекстное представление этого токена, которое потенциально извлекает информацию из 4 токенов с каждой стороны [177]. Каждое обновление конфигурации системы перехода представляет собой задачу классификации, которая использует контекстное представление верхнего токена в стеке, предшествующих и последующих токенов, первых двух токенов буфера и их крайних левых, вторых крайних левых, крайних правых и вторых крайних правых потомков. К системе применяется наиболее вероятный валидный переход. Авторы заявляют, что этот подход работает в пределах 1% от state-of-the-art для английского языка[15]. В наших экспериментах были протестированы 50-, 100-, 200- и 300-мерные предварительно обученные векторы GloVe. Из-за нехватки времени мы не настраивали остальные гиперпараметры и использовали их значения по умолчанию.

Главной моделью, на которой мы сосредоточились, была рекуррентная модель с верхним слоем CRF, а вышеупомянутые методы служили в основном как базовые решения. Отличительной чертой этого подхода является способ формирования контекстных представлений слов. Для каждого токена отдельно, чтобы уловить особенности формы слова, символьное представление извлекается с помощью двунаправленного LSTM [178]. Это представление объединяется с векто-

[15] https://spacy.io/usage/v2#features-models

Рисунок 4.18 — Извлечение контекстного вектора в алгоритме распознавания именованных сущностей на основе biLSTM+CRF.



ром слова, таким как GloVe, образуя промежуточное представление. Используя другую двунаправленную ячейку LSTM над этими промежуточными представлениями, получается контекстное представление токенов (рисунок 4.18). Наконец, слой CRF размечает последовательность этих контекстных представлений. В наших экспериментах использовалась реализация[16] алгоритма Гийома Женциала. Используется размер символьной biLSTM 100, и размер второй сети biLSTM 300.

4.4.3.3 Обсуждение результатов оценки качества

Эксперименты проводились с использованием схемы разметки IOB, всего с 7 метками классов: O, B-PER, I-PER, B-LOC, I-LOC, B-ORG, I-ORG. Из сгенерированных размеченных предложений, случайным образом были выбраны 80% для обучения, а остальные 20% использовали для валидации (dev). Модели с лучшими показателями F1 на валидационном наборе были протестированы на вручную размеченном тестовом наборе данных (test).

В таблице 36 показаны усредненные оценки качества трех моделей. По мере F1 наилучший результат был достигнут с помощью рекуррентной модели с

[16] https://github.com/guillaumegenthial/sequence_tagging

Таблица 36 — Оценка качества распознавания алгоритмов.

Модель	Dev			Test		
	Точность	Полнота	F1	Точность	Полнота	F1
Stanford NER	76.86	70.62	73.61	**78.46**	46.52	58.41
spaCy 2.0	68.19	71.86	69.98	64.83	**55.77**	59.96
Char-biLSTM+biLSTM+CRF	**77.21**	**74.81**	**75.99**	73.27	54.14	**62.23**

Таблица 37 — Зависимость качества распознавания модели Char-biLSTM+biLSTM+CRF от размерности и параметра обучаемости векторного представления слова.

Векторы слов	train embeddings=False		train embeddings=True	
	Dev F1	Test F1	Dev F1	Test F1
GloVe (dim=50)	74.18	56.46	**75.99**	**62.23**
GloVe (dim=100)	73.94	58.52	74.83	61.54
GloVe (dim=200)	**75.00**	58.37	74.97	59.78
GloVe (dim=300)	74.21	**59.66**	74.75	59.92

использованием batch size 8 и оптимизатора Adam с начальной скоростью обучения 0.001. Обновление векторов слов во время обучения также заметно повысило качество. Были протестированы векторные модели слов GloVe четырех различных размеров (50, 100, 200 и 300), причем векторы размером 50 дали наилучшие результаты (Таблица 37).

Для распознавателя сущностей spaCy 2.0 были протестированы те же модели векторного представления слов. Однако в этом случае качество было наивысшим при 200-мерных векторах (Таблица 38). Учитывая приемущества глубокого обучения, неудивительно, что обе соответстующие модели по полноте превзошли стэнфордский распознаватель, где используется ручное конструирование входных признаков, однако в то же время, последний продемонстрировал заметно более высокую точность.

Таблица 38 — Зависимость результатов spaCy 2.0 от используемой модели векторов.

Векторы слов	Dev			Test		
	Точность	Полнота	F1	Точность	Полнота	F1
GloVe (dim=50)	69.31	71.26	70.27	66.52	51.72	58.20
GloVe (dim=100)	**70.12**	**72.91**	**71.49**	66.34	53.35	59.14
GloVe (dim=200)	68.19	71.86	69.98	64.83	**55.77**	**59.96**
GloVe (dim=300)	70.08	71.80	70.93	**66.61**	52.94	59.00

После сравнения результатов на валидационной и тестовой выборках, становится ясно, что набор автоматически сгенерированных примеров не был идеальным индикатором качества моделей на тестовом наборе золотого стандарта. Более высокие показатели на валидационной выборке часто приводили к более низким тестовым показателям, как видно из результатов оценки для spaCy 2.0 и Char-biLSTM+biLSTM+CRF (таблицы 38 и 37). Анализ ошибок на валидационной выборке показал, что многие из них были вызваны неполнотой разметки, когда распознаватели именованных сущностей правильно предсказали сущности, отсутствующие в аннотациях (например, [ԽՍՀՄ-ի LOC] (СССР), [Դինամոն ORG] (the_Dinamo), [Պիրենեյան թերակղզու LOC] (Пиренейский полуостров) и т.д.). Точно так же модели часто правильно игнорировали несущности, которые неправильно помечены в данных (например, [օսմանների PER], [կոնսերվատորիան ORG] и т.д.).

В целом протестированные модели продемонстрировали относительно высокую точность распознавания токенов, которые начинали именованные сущности, но не смогли этого сделать с описательными словами для организаций и, в определенной степени, местоположений. Матрица ошибок для одной из обученных рекуррентных моделей иллюстрирует это различие (Таблица 39). Частично это может быть связано с качеством сгенерированных данных: слова-дескрипторы иногда излишние размечены (например, [Հավայան կղզիների տեղաբնիկները LOC] (коренные жители Гавайев)), что, вероятно, вызвано несогласованностью стилей добавления ссылок в статьях армянской Википедии (в статье ԱՄՆ մշակույթ (Культура США), ссылка с текстом ”Հավայան կղզիների տեղաբնիկները”(”коренные жители Гавайев”) приводит к статье Հավայան կղզիներ (Гавайи)).

4.5 Выводы

В этой главе были описаны вспомогательные методы обработки текстов. Предложен подход к извлечению векторных представлений слов, полностью основанных на признаках на уровне подслов (символов и морфем), который для языков с богатой морфологией позволяет смягчить проблему разреженности данных

Таблица 39 — Матрица ошибок рекуррентной модели на валидационной выборке.

Истинное значение \ Предсказанное значение	O	B-PER	B-ORG	B-LOC	I-ORG	I-PER	I-LOC
O	26707	100	57	249	150	78	129
B-PER	107	712	6	32	2	4	0
B-ORG	93	6	259	58	8	0	0
B-LOC	226	25	32	1535	5	3	20
I-ORG	67	1	5	3	289	3	19
I-PER	46	5	0	1	6	660	8
I-LOC	145	0	1	13	45	11	597
Precision (%)	97.5	83.86	71.94	81.17	57.23	86.95	77.23

и сокращает количество параметров в модели. На основе таких векторных представленияй слов получены сравнительно легкие (с точки зрения параметров) модели лемматизации текстов, которые не уступают в точности моделям, имеющим в несколько раз больше параметров.

Была выполнена оценка векторных представлений слов армянского языка в 3 различных задачах. С этой целью были созданы эталонные наборы данных для внутренней и внешней оценки представлений слов. Для внутренних тестов был переведен и адаптирован тест аналогий. Для внешней оценки использовалась задачи морфологического анализа и классификации текстов, для которой был создан корпус новостных текстов. Дополнительно, была собрана коллекция неразмеченных текстов, на которой были обучены новые, более эффективные в рассмотренных задачах, модели векторов.

Разработанные векторные модели были использованы для создания лемматизатора текстов. На основе анализа существующих методов, основанных на глубоком обучении и показывающих точные результаты на языках с богатой морфологией и маленьким обучающим корпусом, в качестве базы для лемматизатора выбрана нейронная сеть COMBO. Для замены тяжелых по памяти и количеству параметров предобученных моделей векторов слов, предложены несколько альтернатив на основе подслов. Разработаны и протестированы модификации алгоритма fastText, использующие только внутренние символьные N-граммы, суффиксы и морфемы слова. В качестве морфем слова используются его подслова, получаемые в результате BPE кодирования. Последнее позволяет уменьшить размер модели лемматизации от 1 Гб до нескольких Мб, а размер предобученных

моделей векторов уменьшается примерно в 100 раз, при этом без потери точности. Разработанный лемматизатор превосходит существующие аналоги, достигая точности, сравнимой с state-of-the-art для малоресурсных языков.

Помимо альтернативных моделей векторов и лемматизации текста, разработаны методы и программные инструменты для автоматизации процесса построения наборов данных для языков с ограниченными ресурсами. На основе предложенных подходов созданы новые, не имевшие аналогов, эталонные корпуса для армянского языка для задач распознавания именованных сущностей и исправления ошибок автоматического распознавания текстов. Предложенные методы для исправления ошибок автоматического распознавания текстов позволяют уменьшить количество ошибок в словах на 23.5%, достигая результатов, сравнимых с некоторыми коммерческими решениями этой задачи.

Для создания инструмента распознавания именованных сущностей было проведено исследование и разработаны наборы размеченных данных серебряного и золотого стандартов, а также установлены базовые результаты для существующих методов. В результате был создан корпус именованных сущностей с 163000 токенами, автоматически сгенерированный из Википедии, и еще один корпус новостных предложений с 53400 токенами с ручной разметкой именованных сущностей: людей, организаций и географических объектов. Использованный подход может в дальнейшем использоваться для автоматичекой генерации корпусов для других языков. Важность тестового корпуса состоит в том, что он может служить эталоном для будущих систем распознавания именованных сущностей, разработанных для армянского языка.

Глава 5. Система обнаружения заимствований

В предыдущих главах были описаны математические модели для решения данных задач. Данная глава посвящена программной реализации системы обнаружения заимствований на основе этих моделей. Описывается архитектура и технологии, применяемые в реализованной системе определения уникальности текстовых документов, которая помимо простого копирования и вставки также в состоянии обнаружить скрытые заимствования парафраз.

Сначала приводится краткий обзор существующих систем. Затем описывается общая архитектура системы, его модули и компоненты. В третьем разделе приводится описание метода и программных средств индексации текстовых документов. Четвертый раздел посвящен технологиям поиска заимствований в сети с помощью поисковых систем, а четвертый раздел предоставляет описание используемых методов реализации асинхронности для вычисления трудоемких задач.

5.1 Обзор

Системы обнаружения заимствований обычно выделяют участки подозрительного документа, которые, вероятно, происходят из другого источника, а также указывают источники этих участков. Некоторые системы после обнаружения заимствования, в результатах проверки указывают, как именно были изменены исходные тексты при заимствовании. Поиск источников может выполняться как в заранее подготовленных коллекциях документов, так и через Интернет (например, так работает система Антиплагиат (Рис. 5.1)). Для предоставления этих функций системы обнаружения заимствований реализуют методы, описанные в предыдущих главах.

В данной работе была сделана попытка проанализировать существующие системы проверки документов на уникальность, с целью определения применяемых программных средств для реализации вышеописанных функций. Большинство систем обнаружения заимствований реализуются в качестве веб-

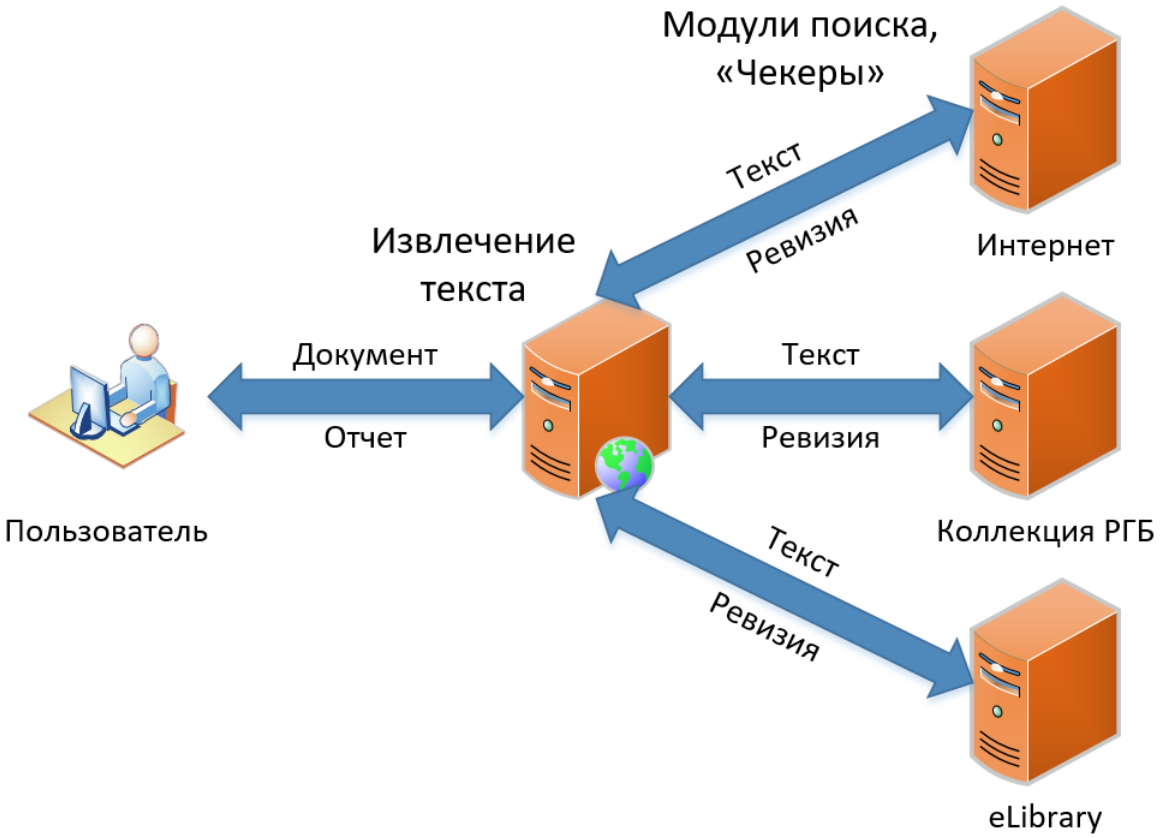


Рисунок 5.1 — Схема поиска заимствований в системе Антиплагиат (источник: habr.com).

приложений. Чтобы быть пригодным для практического применения, этим системам необходимо найти оптимальный компромисс между качеством обнаружения и скоростью обработки, то есть находить заимствования при разумных вычислительных затратах.

Для обзора существующих решений была использована статья [1]. К сожалению, для коммерческих систем обнаружения заимствований редко публикуется информация о применяемых в них методах [9; 179]. По этой причине, сложно оценить, в какой степени исследования по обнаружению заимствований влияют на практические приложения. Тем не менее, исследователи продолжают изучать, какие системы обнаружения дают наилучшие результаты. Вебер-Вульф и ее команда провели исследование этого вопроса в 2004, 2007, 2008, 2010, 2011, 2012 и 2013 годах [68]. В своей последней сравнительной оценке группа с использованием документов, написанных на английском и немецком языках, сравнила 15 систем: Compilatio, Copyscape, Docoloc, Duplichecker, Ephorus, OAPS, PlagAware, Plagiarisma, PlagiarismDetect, PlagiarismFinder, PlagScan, PlagTracker, Strike Plagiarism, Turnitin, Urkund. Последний такой сравнительный анализ Weber-

Wulff провели в 2013 году, после чего подобных систематических и методологически обоснованных оценок эффективности систем обнаружения заимствований не было проведено. Чоудхури и Бхаттачарья [7] в своей работе попытались собрать исчерпывающий список имеющихся в настоящее время систем обнаружения заимствований. Однако их обзор содержит лишь краткие описания систем без сравнения производительности. В статье [8] делается обобщение основных характеристик 17 систем обнаружения заимствований. Более подробный сравнительный анализ был проведен в работе Канджирангат и Гупта [180], где изучаются четыре общедоступные системы. Для сравнения этих систем они использовали тестовые документы, содержащие разные формы заимствования (копирование и вставка, случайные модификации в тексте, перевод на другой язык и обратно, обобщение). Оказалось, что кроме полных дубликатов и случайного модифицированных текстов ни одна из рассмотренных систем не смогла выявить другие формы заимствования. Неспособность программных систем обнаруживать скрытые заимствования в настоящее время является одним из их самых распространённых недостатков [68; 180; 181].

Технологический стек систем обнаружения заимствований как правило включает инструменты полнотекстового поиска, обработки документов и извлечения текста, СУБД. Для полнотекстового поиска используется Apache Solr (например, в PlagScan, Docode 5), ElasticSearch (например, в HyPlag); для управления базами данных – PostgreSQL (Plagiarism Checker от Grammarly, «Антиплагиат»), MySQL (Plagiarism Checker от Grammarly); для извлечения текста из документов – Apache TIKA (Docode 5). В статье [182] компания Антиплагиат описывает детали реализации их системы. Они используют клиент-серверную архитектуру, реализованную на C# и python, в качестве систем управления базами данных – PostgreSQL и MongoDB. Для поиска разработчики Антиплагиата используют собственный поисковик на основе метода шинглов для обнаружения нечетких дубликатов. Извлечение текста из документов выполняется в том числе с помощью оптического распознавания текстов.

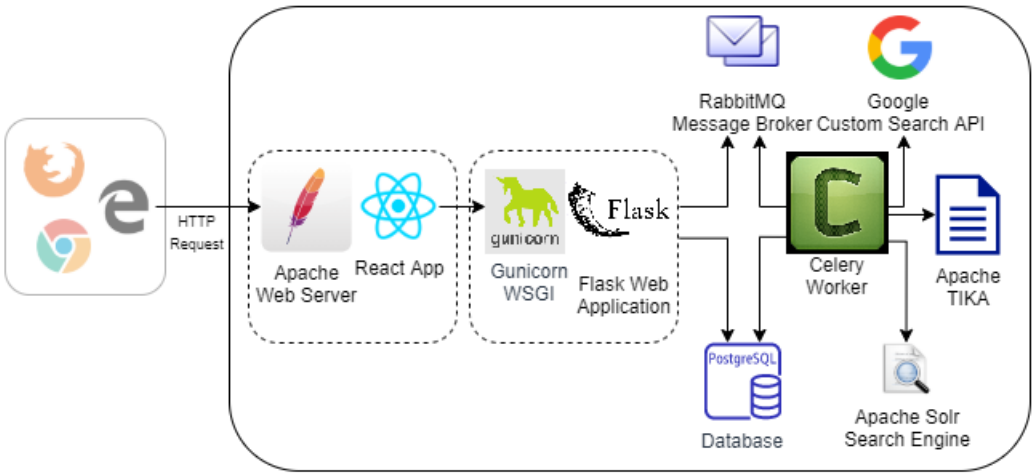


Рисунок 5.2 — Архитектура микросервисов.

5.2 Архитектура

В рамках этой работы была разработана система выявления заимствований (далее - Система), используя микросервисную архитектуру (Рис. 5.2). В качестве отдельных микросервисов были организованы веб-сервер Apache с приложением React, основной код бэкенда вместе с WSGI интерфейсом, база данных, модуль полнотекстового поиска (Apache Solr), модуль извлечения текста из документов (Apache TIKA), брокер сообщений RabbitMQ, и задачи Celery. Разработанная система запускается как многоконтейнерное приложение. Для его описания используется пакетный менеджер Docker Compose.

На рисунке 5.3 изображены основные этапы обработки документа при его проверке. После автоматического распознавания текста документа, в нем исправляются ошибки с помощью методов, описанных в разделе 4.3. Далее исправленный текст проходит проверку на наличие технических приемов для маскировки заимствований, используя подходы из раздела 2.2. Перед применением стилометрических и внешних методов поиска заимствований текст разбивается на фрагменты (параграфы и, опционально, предложения). Для внутреннего анализа применяется метод на основе кластеризации, предложенный в разделе 2.1. Внешний анализ сравнивает фрагменты документа с потенциальными источниками из проверочной базы (далее - База). Изначально База состоит из документов, добавленных администратором Системы. Во время проверки документа из его фрагментов извлекаются ключевые словосочетания, которые затем отправляются в систе-

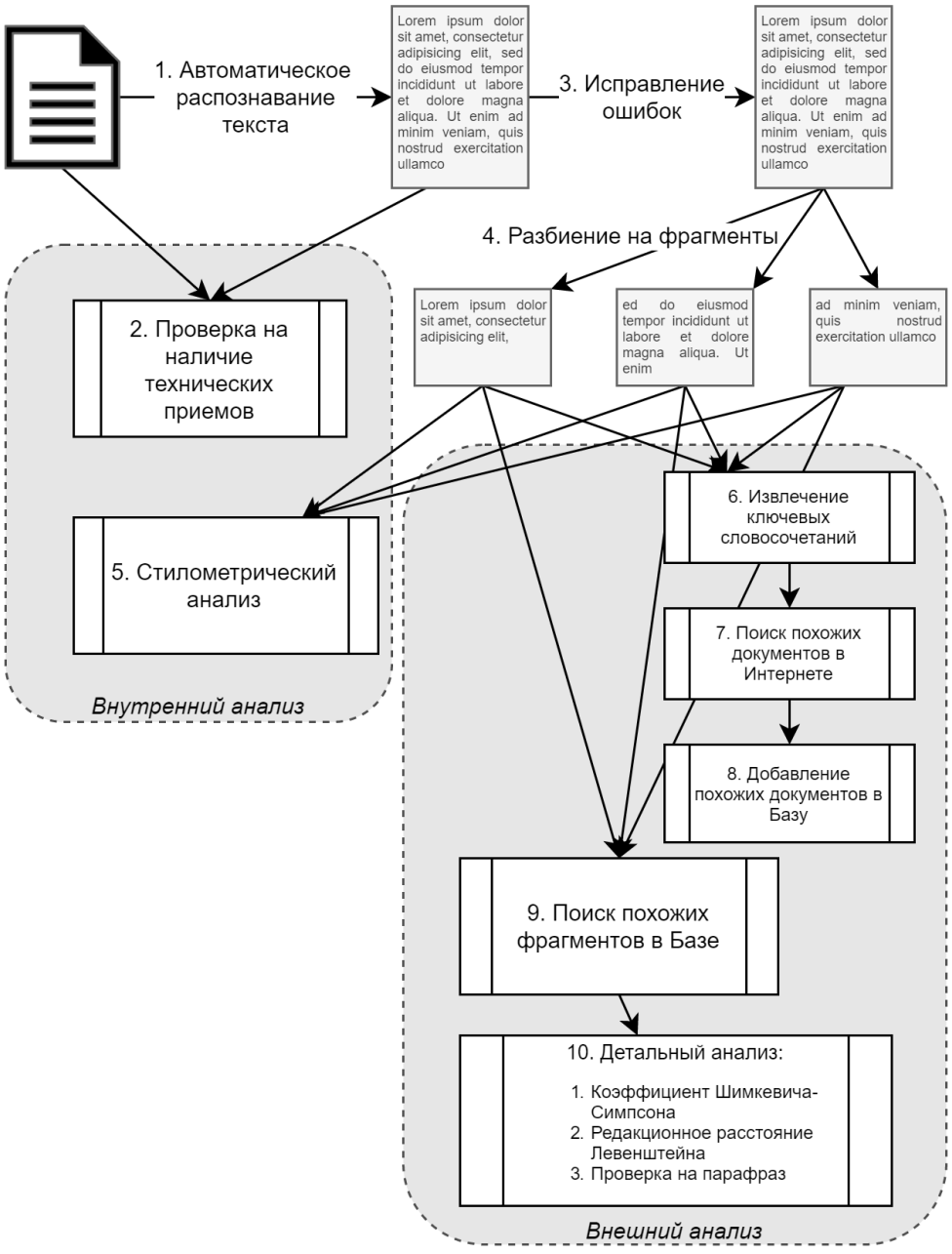


Рисунок 5.3 — Схема использования методов обработки текста в реализованной системе.

му поиска в Интернете, результаты которого фильтруются и добавляются в Базу. Для поиска в Интернет используется реализация алгоритма Пракаща и др. (см. раздел 3.1.3). После завершения поиска в Интернете, каждый фрагмент проверяемого документа целиком отправляется в качестве запроса в систему полнотекстового поиска по документам Базы, которая возвращает список наиболее релевантных фрагментов. Проверяемый фрагмент попарно сравнивается с фрагментами из списка, используя методы детального анализа, после чего отфильтрованные фрагменты возвращаются в качестве потенциальных источников заимствования. Для реализации полнотекстового поиска используется метод шинглов (см. раздел 3.1.2), для детального анализа фрагменто - пороговые классификаторы на основе коэффициента Шимкевича-Симпсона и редакционное расстояние Левенштейна, и нейросетевой метод обнаружения парафраза (см. раздел 3.2).

5.3 Полнотекстовый поиск

Полнотекстовым поиском называется автоматизированный поиск документов, при котором поиск ведётся не по именам документов, а по их содержимому[1]. Задача поиска по мультимедийной информации в рамках этой работы не рассматривалась. Для быстрого поиска по коллекции текстовых документов используется инвертированный индекс. Инвертированный индекс – структура данных, в которой для каждого слова коллекции документов в соответствующем списке перечислены все документы в коллекции, в которых оно встретилось[2].

Данный раздел описывает методы построения инвертированного индекса, программные средства, позволяющие строить и использовать индексы, а также детали предлагаемого решения для поиска текстовых заимствований.

[1] https://www.opengost.ru/iso/01_gosty/01140_gost_iso/0114020_gost_iso/1872-gost-7.73-96-sibid.-poisk-i-rasprostranenie-informacii.-terminy-i-opredeleniya.html

[2] https://ru.wikipedia.org/wiki/Инвертированный_индекс

5.3.1 Обзор методов

В этом разделе описываются методы построения инвертированного индекса. Описание методов взято из [75]. Процесс построения индекса называется индексацией. Основные этапы построения непозиционного индекса изображены на рисунке 5.4. Сначала алгоритм проходит через коллекцию, собирая все пары (слово, docID), затем эти пары сортируются, используя слово как доминирующий ключ и docID как дополнительный ключ. Наконец, из полученного списка строится индекс, собирая для каждого термина список docID и дополнительно вычисляя статистику, такую как частота документов.

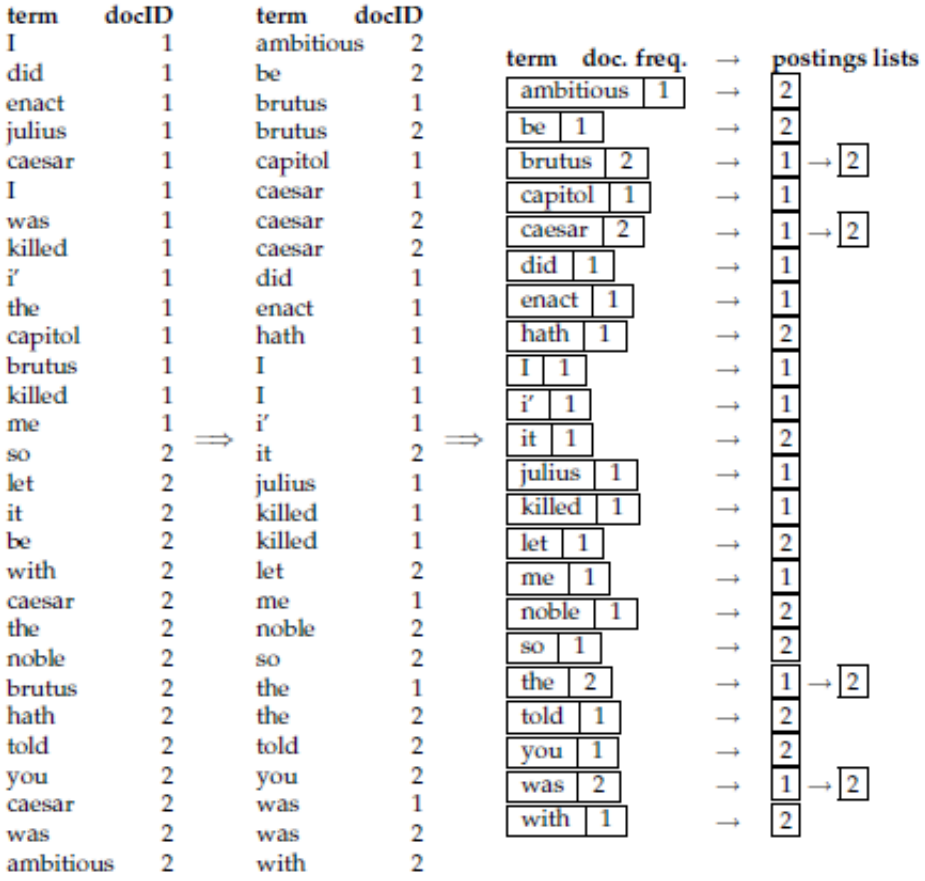


Рисунок 5.4 — Основные этапы построения инвертированного индекса[75].

Для небольших коллекций индексацию можно выполнить в памяти. В этой главе описываются методы для таких коллекций, которые требуют использования вторичного хранилища. Чтобы сделать построение индекса более эффективным, вместо строк для слов используются числовые идентификаторы termIDs, где каждый termID является уникальным идентификационным номером. Отображение слов в termID можно построить по ходу обработки коллекции, или с помо-

щью двухпроходного алгоритма, компилируя словарь на первом проходе и строя инвертированный индекс на втором. В этом разделе рассматриваются алгоритмы построения индекса, выполняющие один проход через данные.

На выбор алгоритма индексации влияют аппаратные ограничения. Следующий раздел содержит небольшой обзор компьютерного оборудования, необходимого для индексации. Далее описывается одномашинный алгоритм индексации индексации на основе блочной сортировки, разработанный для статических коллекций. Также описывается однопроходный алгоритм индексации в памяти, который не хранит словарь в памяти и благодаря этому свойству хорошо масштабируется к большим данным. В случае проверочной коллекции для задачи поиска заимствований актуальна динамическая индексация, позволяющая обновлять уже построенный индекс при поступлении новых документов. Индексация очень больших коллекций выполняется распределенно с помощью компьютерных кластеров из большого числа машин, однако для армянского языка нет такой большой коллекции, поэтому данная проблема недостаточно актуальна и на исследование данного подхода не было обращено внимание в рамках этой работы.

5.3.1.1 Требования к аппаратным средствам

При построении системы обнаружения заимствований, многие решения, связанные с реализацией полнотекстового поиска, основываются на характеристиках аппаратных средств, на которых будет работать система. Данный раздел содержит краткий обзор компьютерного оборудования, используемого для таких систем [75]. При разработке системы на основе полнотекстового поиска необходимо учесть следующие свойства аппаратного обеспечения:

– Доступ к данным в оперативной памяти намного быстрее, чем доступ к данным на диске. Для доступа к байту в памяти требуется несколько тактов (примерно 5×10^{-9} секунд), в то время как для его передачи с диска необходимо значительно больше времени (около 2×10^{-8} секунд). Следовательно, для быстрого поиска необходимо хранить в памяти как можно больше данных, особенно тех данных, к которым нужно часто обращать-

ся. По этой причине выполняется кэширование часто используемых дисковых данных в основной памяти.

– При чтении или записи на диск головке диска требуется некоторое время, чтобы переместиться в ту часть диска, где находятся данные. Это называется временем позиционирования, и оно составляет в среднем 5 мс для обычных дисков. Следовательно, чтобы максимизировать скорость передачи данных, фрагменты данных, которые будут считываться вместе, должны храниться на диске непрерывно.
– Серверы, используемые в информационно-поисковых системах, могут иметь до несколько десятков гигабайт (ГБ) оперативной памяти. Такой объем памяти необходим для эффективного и быстрого поиска с помощью индекса. Когда размер индексируемой коллекции большой, также необходимо дисковое пространство с памятью на несколько порядков больше.

5.3.1.2 Индексация на основе блочной сортировки

Индексация целой коллекции может потребовать до несколько десятков ГБ места в памяти. Поскольку основной памяти часто недостаточно, приходится использовать внешний алгоритм сортировки, который использует диск. Для получения приемлемой скорости основным требованием такого алгоритма является минимизация количества случайных поисков диска во время сортировки – последовательное чтение с диска происходит намного быстрее, чем поиск[75]. Одним из решений является алгоритм индексации на основе блочной сортировки (BSBI), показанный на рисунке 5.5.

```
BSBIndexConstruction()
1  n ← 0
2  while (all documents have not been processed)
3  do n ← n + 1
4     block ← ParseNextBlock()
5     BSBI-Invert(block)
6     WriteBlockToDisk(block, f_n)
7  MergeBlocks(f_1, ..., f_n; f_merged)
```

Рисунок 5.5 — BSBI индексация [75].

BSBI сначала сегментирует коллекцию на части равного размера, затем в памяти сортирует пары (termID, docID) каждой части, после чего сохраняет промежуточные отсортированные результаты на диске и в конце объединяет все промежуточные результаты в окончательный индекс. Алгоритм разбивает документы на пары (termID, docID) и накапливает эти пары в памяти до тех пор, пока блок не заполнится. Размер блока устанавливается заранее и выбирается таким образом, чтобы блок помещался в памяти, чтобы обеспечить быструю сортировку в памяти. После того, как блок заполняется, он инвертируется и записывается на диск. Инверсия состоит из двух шагов: сначала пары (termID, docID) сортируются, и затем группируются на основе termID в список docID. Инвертированный индекс, полученный для прочитанного блока, записывается на диск. На последнем этапе алгоритма BSBI записанные индексы блоков объединяются в один большой индекс. Процесс слияния двух блоков показан на рисунке 5.6, где d_i используется для обозначения i-го документа коллекции.

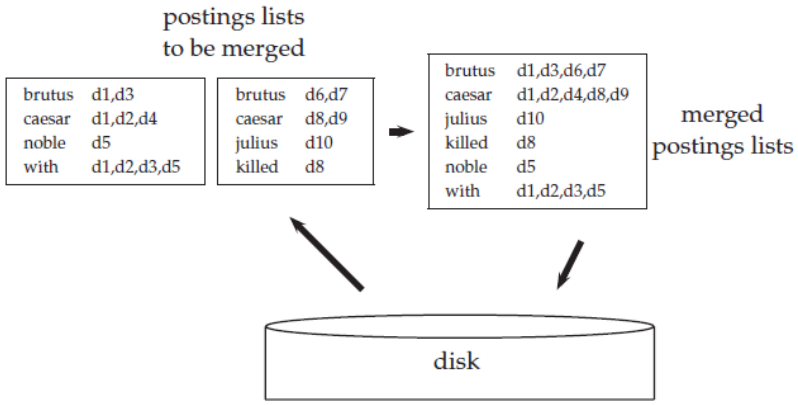


Рисунок 5.6 — Слияние блоков в алгоритме BSBI [75].

Чтобы выполнить слияние, все файлы блоков одновременно открываются для чтения и поддерживаются небольшие буферы для чтения этих блоков, и буфер для записи объединенного индекса. Чтение и запись происходит итеративно: на каждой итерации выбирается самый низкий termID, который еще не был обработан, считываются и объединяются все списки docID для этого termID, и полученный объединенный список записывается в буфер.

Временная сложность алгоритма BSBI равна $O(TlogT)$, потому что этап с наивысшей временной сложностью – это сортировка, а T – это верхняя граница количества элементов, которые мы должны отсортировать (т.е. количествоа пар (termID, docID). Однако, вместе с этим преобладающая часть фактического време-

ни индексации идет на обработку документов для извлечения пар (termID, docID) и на выполнение слияния блоков.

5.3.1.3 Однопроходная индексация в памяти

```
SPIMI-INVERT(token_stream)
 1  output_file = NEWFILE()
 2  dictionary = NEWHASH()
 3  while (free memory available)
 4  do token ← next(token_stream)
 5     if term(token) ∉ dictionary
 6        then postings_list = ADDTODICTIONARY(dictionary, term(token))
 7        else postings_list = GETPOSTINGSLIST(dictionary, term(token))
 8     if full(postings_list)
 9        then postings_list = DOUBLEPOSTINGSLIST(dictionary, term(token))
10     ADDTOPOSTINGSLIST(postings_list, docID(token))
11  sorted_terms ← SORTTERMS(dictionary)
12  WRITEBLOCKTODISK(sorted_terms, dictionary, output_file)
13  return output_file
```

Рисунок 5.7 — Инверсия блока в алгоритме SPIMI. Часть алгоритма, которая читает документы и превращает их в поток пар (слово, docID), была опущена [75].

Индексация на основе блочной сортировки хорошо масштабируется, но нуждается в структуре данных для отображения слов на termID. Для очень больших коллекций эта структура данных может не поместиться в память. Более масштабируемой альтернативой является однопроходная индексация в памяти (SPIMI). SPIMI также обрабатывает данные частями, но в отличие от BSBI использует слова вместо termID, по очереди обрабатывает слова в блоке, записывает словарь каждого блока на диск, а затем использует новый словарь для следующего блока (рис. 5.7). Последним шагом SPIMI является объединение блоков в окончательный инвертированный индекс. Если на диске имеется достаточно большой объем памяти, с помощью алгоритма SPIMI можно индексировать коллекции любого размера.

Разница между BSBI и SPIMI заключается в том, что SPIMI, вместо того чтобы сначала собирать все пары (termID, docID), а затем сортировать их, использует динамический список docID для каждого слова и добавляет новые значения непо-

средственно в этот список. Преимущество такого подхода в том, что не требуется сортировка. Общие требования к памяти для динамически создаваемого индекса блока в SPIMI ниже, чем в BSBI. Когда память исчерпана, индекс блока записывается на диск. Временная сложность SPIMI равна O(T), потому что сортировки не требуется, и все операции в худшем случае линейны по размеру коллекции.

5.3.1.4 Динамическая индексация

Вышеописанные алгоритмы индексации предназначены для статических коллекций документов. Но в случае систем обнаружения заимствований проверочные коллекции могут регулярно обновляться в результате добавления, удаления или модификации документов. Это означает, что необходимо в словарь индекса добавить новые слова, а также обновить списки docID для присутствующих в индексе слов. Самый базовый способ решения этой проблемы – построение нового индекса с нуля и замена старого новым. Это решение подходит для тех случаев, когда изменения в проверочной базе не происходят часто или когда приемлема задержка в обеспечении возможности поиска в новых документах. Также нужно учесть, что для такого подхода необходимо наличие достаточных ресурсов для создания нового индекса, в то время как старый индекс все еще используется для обработки запросов. В том случае, когда задержка неприемлема и новые документы также должны быть учтены во время поиска, одним из решений является хранение двух индексов вместо одного: большой основной индекс и небольшой вспомогательный индекс, в котором хранятся новые документы. В таком случае вспомогательный индекс хранится в памяти. Поиск выполняется по обоим индексам, и результаты объединяются. Каждый раз, когда вспомогательный индекс становится слишком большим, он объединяется с основным индексом. Стоимость этой операции слияния зависит от того, как хранится индекс в файловой системе. Если каждый список docID хранится в виде отдельного файла, то слияние просто состоит в расширении каждого списка основного индекса соответствующим списком docID вспомогательного индекса. К сожалению, хранения каждого списка docID в отдельном файле неосуществима, так как большинство файловых систем не умеют эффективно обрабатывать очень большое количество файлов [75].

Простой альтернативой является хранение индекса в одном большом файл, как объединение всех списков docID. На практике поисковые системы обычно применяют подход, который является компромиссом между этими двумя крайностями, но некоторые из-за сложности динамической индексации, применяют стратегию реконструкции с нуля, т.е. периодически создают новый индекс с нуля и переключают обработку запроса на новый индекс, удаляя старый индекс.

5.3.2 Выбор технологий

Полнотекстовый поиск выполняется с помощью поисковых систем, реализующих описанные в предыдущем разделе методы индексации. Поисковой системой[3] называют совокупность компьютерных программ, которая предоставляет пользователю возможность быстрого доступа к необходимой ему информации при помощи поиска в обширной коллекции доступных данных. Цель поиска – найти наиболее релевантные совпадения с запросами пользователя.

В открытом доступе есть более 20 поисковых систем. В рамках этой работы были изучены Elasticsearch, Apache Solr или Sphinx, которые являются одними из наиболее популярных систем, и предоставляют надежное и эффективное решение для полнотекстового поиска. Исходный код всех трех систем открыт и активно поддерживается сообществом. Solr, Sphinx, Elasticsearch показывают высокую производительность, хорошо масштабируются и предлагают большой набор функций[183] (таблица 40), однако каждая из этих систем имеет свои особенности, которые следует учесть перед использованием в проекте.

Elasticsearch и Apache Solr основаны на Apache Lucene, и поэтому имеют много общих функций. Apache Lucene – это библиотека с открытым исходным кодом, предоставляющая инструменты для выполнения полнотекстового поиска, разработанная в основном на языке программирования Java. Именно с помощью Apache Lucene выполняется индексация данных. Библиотека приспособлена к обработке и индексации больших количеств документов [184].

Сильными сторонами Solr являются наличие большого количества функций полнотекстового поиска: предобработка текста, добавление набора синонимов,

[3] https://bigenc.ru/technology_and_technique/text/3151090

Таблица 40 — Сравнение свойств Elasticsearch, Solr, Sphinx [183].

	Elasticsearch	Solr	Sphinx
Набор поисковых опций	– Полнотекстовый – Автозаполнение – Фасетирование – Многополевой – Синонимы – Fuzzy – Геопространств.	– Полнотекстовый – Автозаполнение – Фасетирование – Многополевой – Синонимы – Fuzzy – Геопространств. – Выделение сниппетов – Правописание	– Полнотекстовый – Автозаполнение – Фасетирование – Многополевой – Синонимы – Геопространств. – Выделение сниппетов – Правописание
Индексация в режиме реального времени	Да	Да	Да
Производительность	Высокая	Высокая	Высокая
Масштабируемость	Высокая	Высокая	Высокая
Схема данных	Нет	Да, но динамическая	Да
Можно использовать как хранилище	Да	Да	Нет
Визуализация данных	С помощью Elastic Stack (ES, Kibana, Logstash)	С помощью плагина Banana	Нет
Машинное обучение	Да	Да	Нет

проверка орфографии, указание важности слов в запросе, поддержка форматов PDF, Word, XML помимо простого текста.

Elasticsearch уместно использовать в проектах, где коллекция данных постоянно обновляется. Solr скорее предназначен для работы с данными, которые не требуют частого изменения. Это обусловлено тем, что в Solr кеши являются глобальными, и следовательно, когда в кеше происходит даже малейшее изменение, вся индексация требует обновления.

Еще одной популярной поисковой системой является Sphinx. Sphinx обеспечивает очень быстрый поиск, однако это не лучший выбор для проектов, имеющих дело с неструктурированными данными (DOC, PDF, MP3 и т.д.), поскольку настройка работы с этими данными трудоемкая и потребует много времени от разработчиков. Учитывая это и другие сложности в настройке, можно сделать вывод, что Sphinx менее удобный в использовании, чем его конкуренты.

Различия между Elasticsearch, Solr и Sphinx минимальны, и все они выполняют свою главную задачу – обеспечивают эффективный и быстрый поиск. С учетом того, что в системах поиска заимствований коллекции меняются нечасто, а также принимая во внимание такие факторы как удобство использования и наличие подробной документации, для реализации системы поиска заимствований в этой работе было решено использовать Solr.

После слияния проектов Lucene и Solr в 2010 году, теперь оба разрабатываются одной и той же командой из Apache Software Foundation. У Solr есть HTTP/XML и JSON API, что делает возможным использовать Solr из всех популярных языков программирования. Также Solr можно очень гибко настраивать и подключать к нему внешние модули.

5.3.3 Настройка Apache Solr

На рисунке 5.8 изображен процесс индексации текста, используемого в реализованной системе. Текст индексируется не целиком, а сначала разбивается небольшие фрагменты (параграфы, предложения). Далее полученный список фрагментов текстов добавляется в коллекцию Solr, где они индексируются. Каждый фрагмент считается отдельной записью, однако информация о родительском

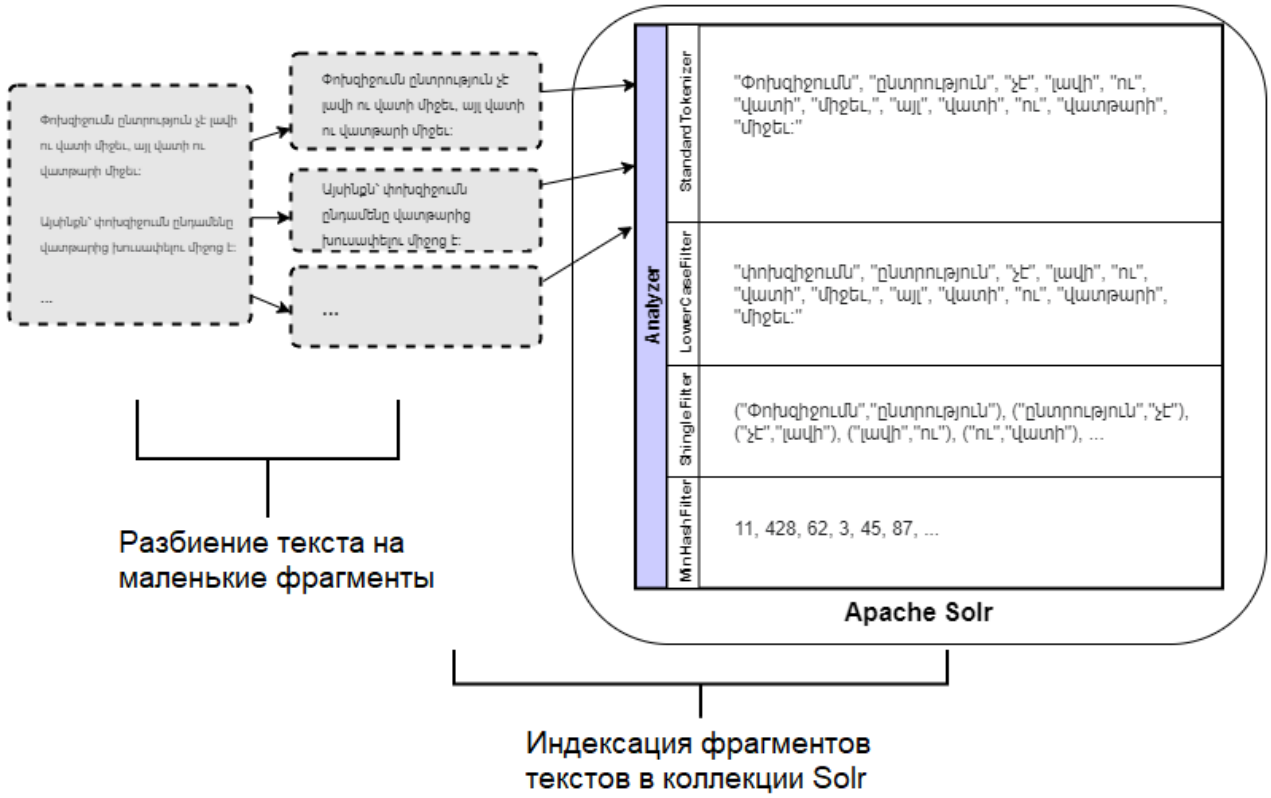


Рисунок 5.8 — Схема индексации текста в Solr.

документе также хранится в Solr. Это реализовано с помощью хранения фрагментов в Solr в качестве вложенных документов. После отправки запроса на добавление фрагментов в индекс, они проходят несколько этапов обработки: текст токенизируется, приводится к нижнему регистру, извлекаются шинглы, и далее для них считается MinHash.

Затем при поиске заимствований в качестве запроса также отправляются небольшие фрагменты текста. Фильтрация кандидатов выполняется путем установления порога на сходство MinHash представлений текстов.

5.4 Поиск в интернете

Поисковые запросы в сети можно сгруппировать в три категории: информационные, навигационные и транзакционные [75]. Информационные запросы позволяют получить общую информацию по широкой теме, такой как лейкемия или Прованс. Обычно нет ни одной веб-страницы, содержащей всю искомую ин-

формацию; действительно, пользователи с информационными запросами обычно пытаются ассимилировать информацию с нескольких веб-страниц.

Навигационные запросы ищут веб-сайт или домашнюю страницу отдельного объекта, который имеет в виду пользователь. В таких случаях пользователь ожидает, что самым первым результатом поиска должна быть домашняя страница этого объекта. Для навигационных запросов, пользователя не интересует множество документов, содержащих термины, связанные с искомым объектом. Транзакционный запрос помогает пользователю, выполняющему транзакцию в Интернете, найти интерфейсы форм для таких транзакций.

От категории запроса зависит какие результаты вернет поиск. Например, для навигационных запросов нужно, чтобы искомый объект был первым в результатах поиска. Такие запросы обычно короткие, не более 1-2 слов. Для информационных запросов пользователю важнее полнота результатов поисковой системы, и, как правило, запросы длиннее. Запросы, отправляемые при поиске заимствований наиболее похожи информационным, так как составляются из списка ключевых слов, а не одного слова, и заимствования могут быть взяты из нескольких источников вместо одного (т.е. полнота результатов важна).

На рисунке 5.9 изображена архитектура типичной поисковой системы в Интернете, включая поисковый робот, а также индексы веб-страниц и рекламы. Пользователь (или клиент) отправляет запрос, который затем ищется в заранее построенных индексах Интернет-документов.

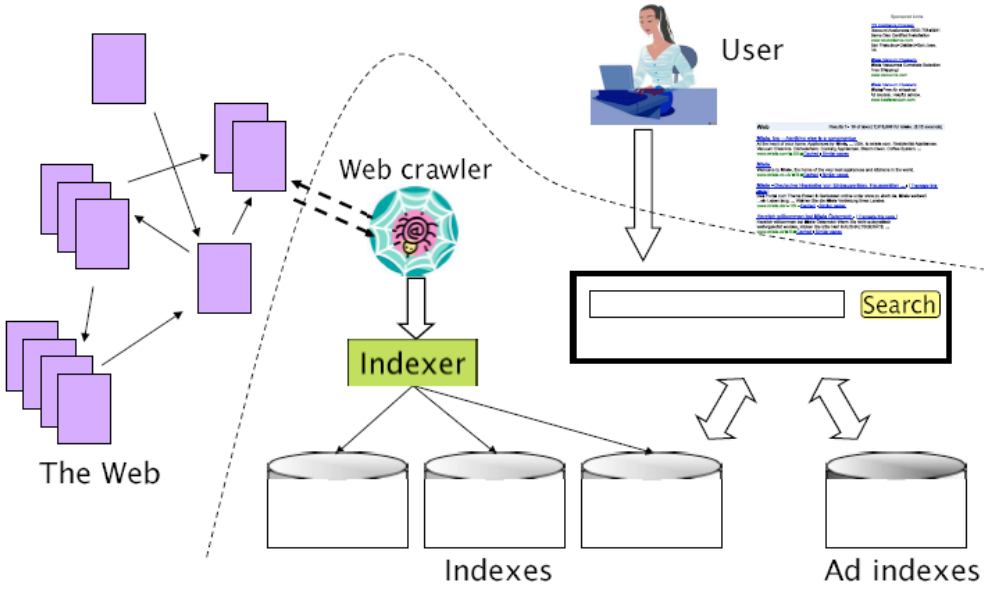


Рисунок 5.9 — Составные части поисковой системы[75].

При реализации поиска заимствований внутренняя организация поисковой системы не так важна. В этом случае более важны допустимые форматы запросов, скорость обработки запросов, размер индекса, наличие удобного программного интерфейса, а также условия предоставления доступа (стоимость, количество запросов, и т.д.). Следующий раздел содержит краткий обзор наиболее популярных поисковых систем с точки зрения удобства применения в системе поиска заимствований.

5.4.1 Выбор технологий

В статье [185] описаны 13 разных API для поиска в Интернете, доступных для использования в программах. Ниже описываются те API, которые являются наиболее релевантными для поиска заимствований:

1. Bing Web Search Microsoft предлагает Bing Search API на основе искусственного интеллекта как часть Microsoft Cognitive Services. Bing Web Search API предоставляет результаты, аналогичные поиска сайта Bing.com поиска. Результаты поиска можно настроить с помощью таких функций, как уровень безопасного поиска, варианты написания, связанные запросы и результаты на основе местоположения.
2. Duck Duck Go DuckDuckGo – поисковая система, которая отличается тем, что не отслеживает своих пользователей и не генерирует результаты на основе предыдущего поведения своих пользователей. DuckDuckGo Instant Answer API предоставляет бесплатный программный доступ ко многим мгновенным ответам поисковой системы, полученным из более чем 100 независимых источников, включая Wikipedia, Wikia, CrunchBase, GitHub, WikiHow, The Free Dictionary.
3. GitHub Search GitHub Search API позволяет пользователям искать на GitHub определенные элементы, относящиеся к пользователям, файлам и т.д. Каждый поиск возвращает до 1000 результатов, а сортировка по умолчанию – наилучшее соответствие. API хорошо документирован и включает пример кода для настройки запросов. Примеры включают: поиск строк кода и выделение совпадающего текста при поиске кода, поиск

репозиториев, написанных на определенном языке. В этой работе изучается только поиск текстов на естественном языке, однако в будущем с помощью данного API можно реализовать и добавить обнаружение заимствований в исходном коде программы.

4. Неофициальный Google News Неофициальный Google News API используется для сбора статей, соответствующих данному набору ключевых слов. API был создан как надежная альтернатива в ответ на закрытие Google News API. API позволяет пользователям получать до 100 статей за один поиск (максимальное количество статей зависит от тарифного плана). Каждая статья содержит заголовок, описание, ссылку на статью, веб-сайт, источник и дату.
5. Google Custom Search Google Custom Search Engine API является программируемая поисковая система с RESTful API, который позволяет разработчикам получать данные результатов поиска в Интернете. Этот API позволяет разработчикам настраивать веб-поиск на поиск по сайту, блогам или коллекции веб-сайтов. Поисковую систему можно настроить как для поиска веб-страниц, так и изображений. Помимо этого, чтобы указать программируемой поисковой системе, какие веб-сайты следует искать, необходимо дополнительно расставлять приоритеты [186]. Custom Search JSON API предоставляет 100 поисковых запросов в день бесплатно, после чего, каждые 1000 запросов стоят 5 долларов. С помощью этого API можно использовать запросы RESTful для получения результатов поиска в формате JSON. Кроме этого есть и другие тарифные планы, более подробно представленные в таблице 41. С учетом размера индекса Google, качества поиска, а также возможностей для дальнейшей более тонкой настройки поисковой система, именно данный API был выбран для использования в системе.

5.5 Извлечение текста из документов

Реализованная система поддерживает множество форматов входного документа: .pdf, .docx, .doc, .html, .txt, .djvu, изображения. За исключением djvu, для

Таблица 41 — Сравнение тарифных планов Google Custom Search [186].

Предложение	Standard Search Element	Non-profit Search Element	Custom Search JSON API	Custom Search Site Restricted JSON API
Стоимость	Бесплатно	Бесплатно	5$ за тысачу запросов	5$ за тысачу запросов
Реклама	Да	Нет	Нет	Нет
Брэндинг Google	Опционально	Да	Нет	Нет
Ежедневный лимит на запросы	Нет	Нет	10000 запросов	Нет
Реализация	JavaScript на стороне клиента	JavaScript на стороне клиента	JSON API на стороне клиента и сервера	на стороне клиента
Доступность	Все	Только некоммерческие	Все	Поиск только по сайту

всех остальных форматов для извлечения текста используется Apache TIKA. Документы формата djvu обрабатываются с помощью утилиты djvutxt.

Схема извлечения текста из документов изображена на рисунке 5.10. С целью выявления случаев попыток обмана системы методами технической маскировки, для каждого документа текст извлекается двумя путями: с помощью оптического распознавания, и с помощью извлечения текстового слоя. Некоторые виды маскировки обнаруживаются путем анализа разницы между двумя полученными текстами. Для корректного разбиения файлов на страницы, некоторые форматы предварительно конвертируются в формат .pdf и только после этого отправляются на вход TIKA для обработки.

5.6 Асинхронное исполнение задач

Чтобы оперативно отвечать на запросы от клиента и избежать тайм-аута соединения, трудоемкие задачи выполняются асинхронно. После поступления соответствующего запроса, в диспетчер сообщений RabbitMQ добавляется запись о новой задаче Celery, которую нужно выполнить. Далее асинхронно запускается процесс исполнения задачи, которая получает необходимые входные данные из параметров и базы данных. После завершения задачи результат записывается в соответствующую таблицу в базе данных. Используемый механизм изображен на

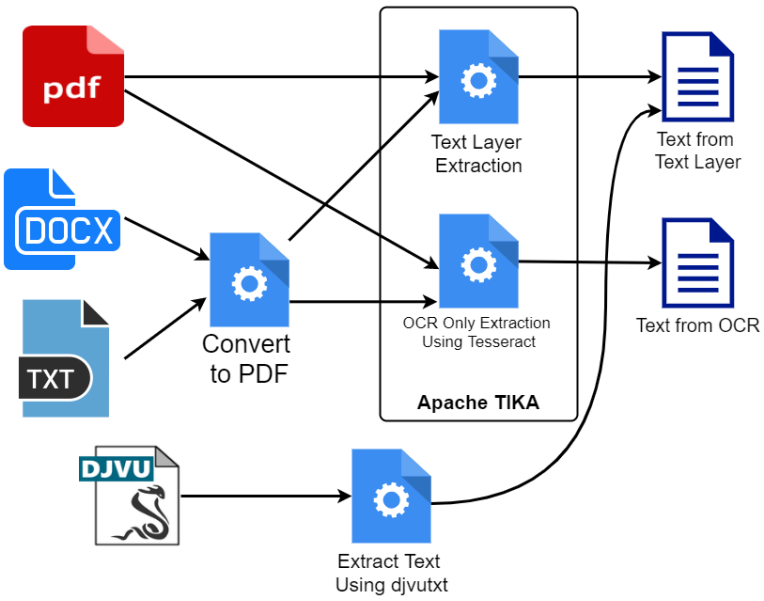


Рисунок 5.10 — Схема извлечения текста из документов.

картинке 5.11. Данный механизм применяется для задач проверки документа на уникальность и добавления новых документов в проверочную коллекцию.

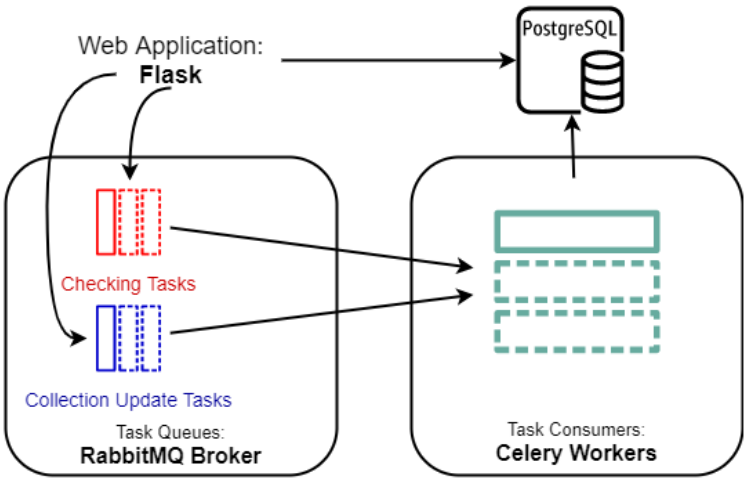


Рисунок 5.11 — Схема асинхронного исполнения задач проверки и обновления коллекции документов.

Заключение

Основные результаты работы заключаются в следующем.

1. Разработаны методы для автоматизации процесса построения наборов данных для языков с ограниченными ресурсами. На основе предложенных подходов созданы новые, не имевшие аналогов, размеченные наборы текстов для армянского языка для задач распознавания именованных сущностей, обнаружения парафраза, векторного представления слов, стилометрического анализа, и исправления ошибок автоматического распознавания текстов. Кроме этого, для задач распознавания именованных сущностей, обнаружения парафраза, векторного представления слов были созданы тестовые наборы текстов с ручной разметкой. Разработанные наборы данных позволили создать программные инструменты для соответствующих задач, превышающие по показателям существующие аналоги.
2. Предложены подходы к извлечению векторных представлений слов, полностью основанных на признаках на уровне подслов (символов и морфем), который для языков с богатой морфологией позволяет смягчить проблему разреженности данных и сокращает количество параметров в модели. На основе таких векторных представленияй слов получены сравнительно легкие (с точки зрения параметров) модели лемматизации и морфологического анализа текстов, которые не уступают в точности моделям, имеющим в несколько раз больше параметров.
3. С использованием перечисленных выше инструментов разработана и внедрена программная система для оценки степени уникальности текстовых документов на армянском языке, которая помимо прямого копирования позволяет обнаружить нечеткие дубликаты, парафраз, техническую маскировку, а также выполняет поиск заимствований как в проверочной базе документов, так и в Интернете.

Список литературы

1. *Foltýnek T.*, *Meuschke N.*, *Gipp B.* Academic Plagiarism Detection: A Systematic Literature Review // ACM Computing Surveys. — 2019. — Oct. — Vol. 52. — P. 1–42. — DOI: 10.1145/3345317.
2. *Gipp B.* Citation-based plagiarism detection // Citation-based plagiarism detection. — Springer, 2014. — P. 57–88.
3. *Hovakimyan D.* Plagiarism as an academic dishonesty: the case of Armenian universities : PhD thesis / Hovakimyan Dianna. — 2014.
4. *Никитов А. В.*, *Орчаков О. А.*, *Чехович Ю. В.* Плагиат в работах студентов и аспирантов: проблема и методы противодействия. — 2012. — URL: https://cyberleninka.ru/article/n/plagiat-v-rabotah-studentov-i-aspirantov-problema-i-metody-protivodeystviya (visited on 12/20/2020).
5. M. Milovanovitch, I. Ceneric, M. Avetisyan, T. Khavanska. — 2015. — URL: http://www.osf.am/wp-content/uploads/2016/01/Integrity-report_final_en_12.11.2015.pdf.
6. *Margarov G.*, *Tomeyan G.*, *Pereira M. J. V.* Plagiarism detection system for Armenian language // 2017 Computer Science and Information Technologies (CSIT). — IEEE. 2017. — P. 185–189.
7. *Hussain S. F.*, *Suryani A.* On retrieving intelligently plagiarized documents using semantic similarity // Engineering Applications of Artificial Intelligence. — 2015. — Vol. 45. — P. 246–258.
8. *Pertile S. d. L.*, *Moreira V. P.*, *Rosso P.* Comparing and combining C ontent-and C itation-based approaches for plagiarism detection // Journal of the Association for Information Science and Technology. — 2016. — Vol. 67, no. 10. — P. 2511–2526.
9. DOCODE 3.0 (DOcument COpy DEtector): A system for plagiarism detection by applying an information fusion process from multiple documental data sources / J. D. Velásquez, Y. Covacevich, F. Molina, E. Marrese-Taylor, C. Rodríguez, F. Bravo-Marquez // Information Fusion. — 2016. — Vol. 27. — P. 64–75.

10. *Чехович Ю. В.*, *Беленькая О. С.* Оценка корректности заимствований в текстах научных публикаций // Научное издание международного уровня-2018: редакционная политика, открытый доступ, научные коммуникации. — 2018. — P. 158–162.

11. *Curtis G. J.*, *Clare J.* How prevalent is contract cheating and to what extent are students repeat offenders? // Journal of Academic Ethics. — 2017. — Vol. 15, no. 2. — P. 115–124.

12. Impact of policies for plagiarism in higher education across Europe: Results of the project / T. Foltýnek, I. Glendinning, [et al.] // Acta Universitatis Agriculturae et Silviculturae Mendelianae Brunensis. — 2015. — Vol. 63, no. 1. — P. 207–216.

13. *Owens C.*, *White F. A.* A 5-year systematic strategy to reduce plagiarism among first-year psychology university students // Australian Journal of Psychology. — 2013. — Vol. 65, no. 1. — P. 14–21.

14. *Malajyan A.*, *Avetisyan K.*, *Ghukasyan T.* ARPA: Armenian Paraphrase Detection Corpus and Models // 2020 Ivannikov Memorial Workshop (IVMEM). — 2020. — P. 35–39. — DOI: 10.1109/IVMEM51402.2020.00012.

15. *Tigranyan S.*, *Ghukasyan T.* Post-OCR Correction of Armenian Texts Using Neural Networks. // “Vestnik” Scientific Journal of Russian-Armenian University. — 2020.

16. pioNER: Datasets and Baselines for Armenian Named Entity Recognition / T. Ghukasyan, G. Davtyan, K. Avetisyan, I. Andrianov // 2018 Ivannikov Ispras Open Conference (ISPRAS). — 2018. — P. 56–61. — DOI: 10.1109/ISPRAS.2018.00015.

17. *ГУКАСЯН Ц.* Векторные модели на основе символьных н-грамм для морфологического анализа текстов. // Труды Института системного программирования РАН. — 2020. — Vol. 32, no. 2. — P. 7–14. — DOI: https://doi.org/10.15514/ISPRAS-2020-32(2)-1.

18. *Ghukasyan T.*, *Yeshilbashian Y.*, *Avetisyan K.* Subwords-Only Alternatives to fastText for Morphologically Rich Languages // Programming and Computer Software. — 2021. — Vol. 47, no. 1. — P. 56–66.

19. *Ешилбашян Е.*, *Асатрян А.*, *Гукасян Ц.* Поиск заимствований в армянских текстах путем внутреннего стилометрического анализа // Труды Института системного программирования РАН. — 2021. — Vol. 33, no. 1. — P. 209–224. — DOI: https://doi.org/10.15514/ISPRAS-2021-33(1)-14.

20. *Avetisyan K.*, *Ghukasyan T.* Word Embeddings for the Armenian Language: Intrinsic and Extrinsic Evaluation. // “Vestnik” Scientific Journal of Russian-Armenian University. — 2019. — No. 1. — P. 59–72.

21. *Jain A.*, *Paranjape B.*, *Lipton Z. C.* Entity Projection via Machine Translation for Cross-Lingual NER // EMNLP/IJCNLP. — 2019.

22. *Kulshreshtha S.*, *García J. L. R.*, *Chang C.-Y.* Cross-lingual Alignment Methods for Multilingual BERT: A Comparative Study // EMNLP. — 2020.

23. *Ali W.*, *Lu J.*, *Xu Z.* SiNER: A Large Dataset for Sindhi Named Entity Recognition // LREC. — 2020.

24. *Кулешова А.В. Чехович Ю.В. Б. О.* По лезвию бритвы: как самоцитирование не превратить в самоплагиат. // Научный редактор и издатель. — 2019. — 4(1–2). — P. 45–51. — DOI: https://doi.org/10.24069/2542-0267-2019-1-2-45-51.

25. *Fishman T.* “We know it when we see it” is not good enough: toward a standard definition of plagiarism that transcends theft, fraud, and copyright. — 2009.

26. *Alfikri Z. F.*, *Purwarianti A.* Detailed analysis of extrinsic plagiarism detection system using machine learning approach (naive bayes and svm) // TELKOMNIKA Indonesian Journal of Electrical Engineering. — 2014. — Vol. 12, no. 11. — P. 7884–7894.

27. Machine learning tool and meta-heuristic based on genetic algorithms for plagiarism detection over mail service / H. A. Bouarara, A. Rahmani, R. M. Hamou, A. Amine // 2014 IEEE/ACIS 13th International Conference on Computer and Information Science (ICIS). — IEEE. 2014. — P. 157–162.

28. *Paul M.*, *Jamal S.* An improved SRL based plagiarism detection technique using sentence ranking // Procedia Computer Science. — 2015. — Vol. 46. — P. 223–230.

29. *Eisa T. A. E.*, *Salim N.*, *Alzahrani S.* Existing plagiarism detection techniques // Online Information Review. — 2015.

30. *Alzahrani S. M.*, *Salim N.*, *Abraham A.* Understanding plagiarism linguistic patterns, textual features, and detection methods // IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews). — 2011. — Vol. 42, no. 2. — P. 133–149.

31. *Walker J.* Student plagiarism in universities: What are we doing about it? // Higher Education Research & Development. — 1998. — Vol. 17, no. 1. — P. 89–106.

32. *Weber-Wulff D.* False feathers: A perspective on academic plagiarism. — Springer Science & Business, 2014.

33. *Chowdhury H. A.*, *Bhattacharyya D. K.* Plagiarism: Taxonomy, tools and detection techniques // arXiv preprint arXiv:1801.06323. — 2018.

34. *Chong M. Y. M.* A study on plagiarism detection and plagiarism direction identification using natural language processing techniques. — 2013.

35. *Hourrane O.*, *Benlahmar E. H.* Survey of plagiarism detection approaches and big data techniques related to plagiarism candidate retrieval // Proceedings of the 2nd international Conference on Big Data, Cloud and Applications. — 2017. — P. 1–6.

36. *Oberreuter G.*, *Velásquez J. D.* Text mining applied to plagiarism detection: The use of words for detecting deviations in the writing style // Expert Systems with Applications. — 2013. — Vol. 40, no. 9. — P. 3756–3763.

37. *Vani K.*, *Gupta D.* Detection of idea plagiarism using syntax–semantic concept extractions with genetic algorithm // Expert Systems with Applications. — 2017. — Vol. 73. — P. 11–26.

38. *Mozgovoy M.*, *Kakkonen T.*, *Cosma G.* Automatic student plagiarism detection: future perspectives // Journal of Educational Computing Research. — 2010. — Vol. 43, no. 4. — P. 511–531.

39. Overview of the Author Identification Task at PAN-2018: Cross-domain Authorship Attribution and Style Change Detection / M. Kestemont, M. Tschuggnall, E. Stamatatos, W. Daelemans, G. Specht, B. Stein, M. Potthast // Working Notes Papers of the CLEF 2018 Evaluation Labs. Vol. 2125 / ed. by L. Cappellato, N. Ferro, J.-Y. Nie, L. Soulier. — CEUR-WS.org, 09/2018. — (CEUR Workshop Proceedings). — URL: http://ceur-ws.org/Vol-2125/.

40. Overview of the Style Change Detection Task at PAN 2019 / E. Zangerle, M. Tschuggnall, G. Specht, M. Potthast, B. Stein // CLEF 2019 Labs and Workshops, Notebook Papers / ed. by L. Cappellato, N. Ferro, D. Losada, H. Müller. — CEUR-WS.org, 09/2019. — URL: http://ceur-ws.org/Vol-2380/.

41. Overview of the Author Identification Task at PAN 2017: Style Breach Detection and Author Clustering / M. Tschuggnall, E. Stamatatos, B. Verhoeven, W. Daelemans, G. Specht, B. Stein, M. Potthast // Working Notes Papers of the CLEF 2017 Evaluation Labs. Vol. 1866 / ed. by L. Cappellato, N. Ferro, L. Goeuriot, T. Mandl. — CEUR-WS.org, 09/2017. — (CEUR Workshop Proceedings). — URL: http://ceur-ws.org/Vol-1866/.

42. Overview of PAN 2016—New Challenges for Authorship Analysis: Cross-genre Profiling, Clustering, Diarization, and Obfuscation / P. Rosso, F. Rangel, M. Potthast, E. Stamatatos, M. Tschuggnall, B. Stein // Experimental IR Meets Multilinguality, Multimodality, and Interaction. 7th International Conference of the CLEF Initiative (CLEF 2016) / ed. by N. Fuhr, P. Quaresma, B. Larsen, T. Gonçalves, K. Balog, C. Macdonald, L. Cappellato, N. Ferro. — Berlin Heidelberg New York : Springer, 09/2016. — ISBN 978-3-319-44564-9. — DOI: 10.1007/978-3-319-44564-9_28.

43. *Nath S.* Style Change Detection by Threshold Based and Window Merge Clustering Methods // CLEF 2019 Labs and Workshops, Notebook Papers / ed. by L. Cappellato, N. Ferro, D. Losada, H. Müller. — CEUR-WS.org, 09/2019. — URL: http://ceur-ws.org/Vol-2380/.

44. An Ensemble-Rich Multi-Aspect Approach for Robust Style Change Detection—Notebook for PAN at CLEF 2018 / D. Zlatkova, D. Kopev, K. Mitov, A. Atanasov, M. Hardalov, I. Koychev, P. Nakov // CLEF 2018 Evaluation Labs and Workshop – Working Notes Papers, 10-14 September, Avignon, France / ed. by L. Cappellato, N. Ferro, J.-Y. Nie, L. Soulier. — CEUR-WS.org, 09/2018. — URL: http://ceur-ws.org/Vol-2125/.

45. *Hosseinia M.*, *Mukherjee A.* A Parallel Hierarchical Attention Network for Style Change Detection—Notebook for PAN at CLEF 2018 // CLEF 2018 Evaluation Labs and Workshop – Working Notes Papers, 10-14 September, Avignon, France / ed. by L. Cappellato, N. Ferro, J.-Y. Nie, L. Soulier. — CEUR-WS.org, 09/2018. — URL: http://ceur-ws.org/Vol-2125/.

46. *Safin K.*, *Ogaltsov A.* Detecting a Change of Style Using Text Statistics—Notebook for PAN at CLEF 2018 // CLEF 2018 Evaluation Labs and Workshop – Working Notes Papers, 10-14 September, Avignon, France / ed. by L. Cappellato, N. Ferro, J.-Y. Nie, L. Soulier. — CEUR-WS.org, 09/2018. — URL: http://ceur-ws.org/Vol-2125/.

47. *Karaś D.*, *Śpiewak M.*, *Sobecki P.* OPI-JSA at CLEF 2017: Author Clustering and Style Breach Detection—Notebook for PAN at CLEF 2017 // CLEF 2017 Evaluation Labs and Workshop – Working Notes Papers, 11-14 September, Dublin, Ireland / ed. by L. Cappellato, N. Ferro, L. Goeuriot, T. Mandl. — CEUR-WS.org, 09/2017. — URL: http://ceur-ws.org/Vol-1866/.

48. *Khan J.* Style Breach Detection: An Unsupervised Detection Model—Notebook for PAN at CLEF 2017 // CLEF 2017 Evaluation Labs and Workshop – Working Notes Papers, 11-14 September, Dublin, Ireland / ed. by L. Cappellato, N. Ferro, L. Goeuriot, T. Mandl. — CEUR-WS.org, 09/2017. — URL: http://ceur-ws.org/Vol-1866/.

49. *Safin K.*, *Kuznetsova R.* Style Breach Detection with Neural Sentence Embeddings—Notebook for PAN at CLEF 2017 // CLEF 2017 Evaluation Labs and Workshop – Working Notes Papers, 11-14 September, Dublin, Ireland / ed. by L. Cappellato, N. Ferro, L. Goeuriot, T. Mandl. — CEUR-WS.org, 09/2017. — URL: http://ceur-ws.org/Vol-1866/.

50. Author Clustering using Hierarchical Clustering Analysis—Notebook for PAN at CLEF 2017 / H. Gómez-Adorno, Y. Alemán, D. Vilariño Ayala, M. Sanchez-Perez, D. Pinto, G. Sidorov // CLEF 2017 Evaluation Labs and Workshop – Working Notes Papers, 11-14 September, Dublin, Ireland / ed. by L. Cappellato, N. Ferro, L. Goeuriot, T. Mandl. — CEUR-WS.org, 09/2017. — URL: http://ceur-ws.org/Vol-1866/.

51. Discovering Author Groups using a B-compact graph-based Clustering—Notebook for PAN at CLEF 2017 / Y. García-Mondeja, D. Castro-Castro, V. Lavielle-Castro, R. Muñoz // CLEF 2017 Evaluation Labs and Workshop – Working Notes Papers, 11-14 September, Dublin, Ireland / ed. by L. Cappellato, N. Ferro, L. Goeuriot, T. Mandl. — CEUR-WS.org, 09/2017. — URL: http://ceur-ws.org/Vol-1866/.

52. *Kocher M.*, *Savoy J.* UniNE at CLEF 2017: Author Profiling Reasoning—Notebook for PAN at CLEF 2017 // CLEF 2017 Evaluation Labs and Workshop – Working Notes Papers, 11-14 September, Dublin, Ireland / ed. by L. Cappellato, N. Ferro, L. Goeuriot, T. Mandl. — CEUR-WS.org, 09/2017. — URL: http://ceur-ws.org/Vol-1866/.

53. Mining writeprints from anonymous e-mails for forensic investigation / F. Iqbal, H. Binsalleeh, B. Fung, M. Debbabi // Digital Investigation. — 2010. — Oct. — Vol. 7. — P. 56–64. — DOI: 10.1016/j.diin.2010.03.003.

54. *Zuo C.*, *Zhao Y.*, *Banerjee R.* Style Change Detection with Feed-forward Neural Networks // Working Notes of CLEF 2019 - Conference and Labs of the Evaluation Forum, Lugano, Switzerland, September 9-12, 2019. Vol. 2380 / ed. by L. Cappellato, N. Ferro, D. E. Losada, H. Müller. — CEUR-WS.org, 2019. — (CEUR Workshop Proceedings). — URL: http://ceur-ws.org/Vol-2380/paper%5C_229.pdf.

55. *Dewang R. K.*, *Singh A. K.* Identification of Fake Reviews Using New Set of Lexical and Syntactic Features // Proceedings of the Sixth International Conference on Computer and Communication Technology 2015. — Allahabad, India : Association for Computing Machinery, 2015. — P. 115–119. — (ICCCT '15). — ISBN 9781450335522. — DOI: 10.1145/2818567.2818589. — URL: https://doi.org/10.1145/2818567.2818589.

56. *Hirst G.*, *Feiguina O.* Bigrams of syntactic labels for authorship discrimination of short texts // Literary and Linguistic Computing. — 2007. — Vol. 22, no. 4. — P. 405–417.

57. Research on Author Identification Based on Deep Syntactic Features / C. Zhao, W. Song, L. Liu, C. Du, X. Zhao // 2017 10th International Symposium on Computational Intelligence and Design (ISCID). Vol. 1. — 2017. — P. 276–279. — DOI: 10.1109/ISCID.2017.159.

58. Stanza: A Python Natural Language Processing Toolkit for Many Human Languages / P. Qi, Y. Zhang, Y. Zhang, J. Bolton, C. D. Manning // ACL. — 2020.

59. *Straka M.* UDPipe 2.0 Prototype at CoNLL 2018 UD Shared Task // CoNLL Shared Task. — 2018.

60. *Gishamer F.* Using Hashtags and POS-Tags for Author Profiling // CLEF. — 2019.

61. *Schneider F.*, *Cutts M.* Systems and methods for detecting hidden text and hidden links. — 2013. — US Patent 8,392,823.

62. *Myers E. W.* AnO (ND) difference algorithm and its variations // Algorithmica. — 1986. — Vol. 1, no. 1–4. — P. 251–266.

63. *Sayfudinova O.* Разностный алгоритм Майерса и наблюдаемые свойства в Kotlin — как их объединить, чтобы облегчить жизнь разработчика [Electronic Resource]. — URL: https://medium.com/nuances-of-programming/%D1%80%D0%B0%D0%B7%D0%BD%D0%BE%D1%81%D1%82%D0%BD%D1%8B%D0%B9-%D0%B0%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC-%D0%BC%D0%B0%D0%B9%D0%B5%D1%80%D1%81%D0%B0-%D0%B8-%D0%BD%D0%B0%D0%B1%D0%BB%D1%8E%D0%B4%D0%B0%D0%B5%D0%BC%D1%8B%D0%B5-%D1%81%D0%B2%D0%BE%D0%B9%D1%81%D1%82%D0%B2%D0%B0-%D0%B2-kotlin-%D0%BA%D0%B0%D0%BA-%D1%81%D0%BE%D0%B5%D0%B4%D0%B8%D0%BD%D0%B8%D1%82%D1%8C-%D0%B8%D1%85-%D1%87%D1%82%D0%BE%D0%B1%D1%8B-%D0%BE%D0%B1%D0%BB%D0%B5%D0%B3%D1%87%D0%B8%D1%82%D1%8C-1ae87e56ae6a (visited on 12/20/2020).

64. *Cohen B.* Patience Diff Advantages [Electronic Resource]. — URL: https://bramcohen.livejournal.com/73318.html (visited on 12/20/2020).

65. *Nugroho Y.*, *Hata H.*, *Matsumoto K.* How different are different diff algorithms in Git? // Empirical Software Engineering. — 2019. — Vol. 25. — P. 790–823.

66. *Alvi F.*, *Stevenson M.*, *Clough P.* Plagiarism Detection in Texts Obfuscated with Homoglyphs // ECIR. — 2017.

67. *Gillam L.*, *Marinuzzi J.*, *Ioannou P.* Turnitoff-defeating plagiarism detection systems. — 2010.

68. Plagiarism detection software test 2013 / D. Weber-Wulff, C. Möller, J. Touras, E. Zincke, H. Berlin // Abgerufen am. — 2013. — Vol. 12. — P. 2014.

69. Safeguard against unicode attacks: generation and applications of uc-simlist / A. Y. Fu, W. Zhang, X. Deng, L. Wenyin // Proceedings of the 15th international conference on World Wide Web. — 2006. — P. 917–918.

70. *Roshanbin N.*, *Miller J.* Finding homoglyphs-a step towards detecting unicode-based visual spoofing attacks // International Conference on Web Information Systems Engineering. — Springer. 2011. — P. 1–14.

71. *Costello A.* RFC3492: Punycode: A Bootstring encoding of Unicode for Internationalized Domain Names in Applications (IDNA). — 2003.

72. *Wenyin L.*, *Fu A. Y.*, *Deng X.* Exposing homograph obfuscation intentions by coloring unicode strings // Asia-Pacific Web Conference. — Springer. 2008. — P. 275–286.

73. *Lulu L.*, *Belkhouche B.*, *Harous S.* Overview of fingerprinting methods for local text reuse detection // 2016 12th International Conference on Innovations in Information Technology (IIT). — IEEE. 2016. — P. 1–6.

74. *Potthast M.* Technologies for reusing text from the web : PhD thesis / Potthast Martin. — Citeseer, 2012.

75. *Manning C. D.*, *Raghavan P.*, *Schütze H.* Introduction to Information Retrieval. — USA : Cambridge University Press, 2008. — ISBN 0521865719.

76. *Hagen M.*, *Potthast M.*, *Stein B.* Source Retrieval for Plagiarism Detection from Large Web Corpora: Recent Approaches // CLEF. — 2015.

77. Source retrieval plagiarism detection based on noun phrase and keyword phrase extraction / J. Rafiei, S. Mohtaj, V. Zarrabi, H. Asghari // PAN. — 2015.

78. *RiyaRavi N.*, *Gupta D.* Efficient Paragraph based Chunking and Download Filtering for Plagiarism Source Retrieval // CLEF. — 2015.

79. *Suchomel S.*, *Brandejs M.* Improving Synoptic Quering for Source Retrieval: Notebook for PAN at CLEF 2015 // CLEF. — 2015.

80. *Han Y.* Submission to the 7th International Competition on Plagiarism Detection //.

81. Source Retrieval and Text Alignment Corpus Construction for Plagiarism Detection / L. Kong, Z. Lu, Y. Han, H. Qi, Z. Han, Q. Wang, Z. Hao, J. Zhang // CLEF. — 2015.

82. Approaches for Source Retrieval and Text Alignment of Plagiarism Detection Notebook for PAN at CLEF 2013 / L. Kong, H. Qi, C. Du, M. Wang, Z. Han // CLEF. — 2013.

83. *Prakash A.*, *Saha S.* Experiments on Document Chunking and Query Formation for Plagiarism Source Retrieval—Notebook for PAN at CLEF 2014 // Working Notes Papers of the CLEF 2014 Evaluation Labs / ed. by L. Cappellato, N. Ferro, M. Halvey, W. Kraaij. — CEUR-WS.org, 09/2014. — URL: http://ceur-ws.org/Vol-1180.

84. An evaluation framework for plagiarism detection / M. Potthast, B. Stein, A. Barrón-Cedeño, P. Rosso // Coling 2010: Posters. — 2010. — P. 997–1005.

85. *Belyy A.*, *Dubova M.*, *Nekrasov D.* Improved Evaluation Framework for Complex Plagiarism Detection // ACL. — 2018.

86. *Yerra R.*, *Ng Y.* A Sentence-Based Copy Detection Approach for Web Documents // FSKD. — 2005.

87. *Kent C. K.*, *Salim N.* Features Based Text Similarity Detection // ArXiv. — 2010. — Vol. abs/1001.3487.

88. *Federmann C.*, *Elachqar O.*, *Quirk C.* Multilingual whispers: Generating paraphrases with translation // Proceedings of the 5th Workshop on Noisy User-generated Text (W-NUT 2019). — 2019. — P. 17–26.

89. *Dolan W. B.*, *Brockett C.* Automatically constructing a corpus of sentential paraphrases // Proceedings of the Third International Workshop on Paraphrasing (IWP2005). — 2005.

90. ParaPhraser: Russian paraphrase corpus and shared task / L. Pivovarova, E. Pronoza, E. Yagunova, A. Pronoza // Conference on Artificial Intelligence and Natural Language. — Springer. 2017. — P. 211–225.

91. *Quirk C.*, *Brockett C.*, *Dolan W. B.* Monolingual machine translation for paraphrase generation // Proceedings of the 2004 conference on empirical methods in natural language processing. — 2004. — P. 142–149.

92. Unsupervised construction of large paraphrase corpora: Exploiting massively parallel news sources / W. Dolan, C. Quirk, C. Brockett, B. Dolan. — 2004.

93. *Wubben S.*, *Van Den Bosch A.*, *Krahmer E.* Paraphrase generation as monolingual translation: Data and evaluation // Proceedings of the 6th International Natural Language Generation Conference. — 2010.

94. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding / J. Devlin, M.-W. Chang, K. Lee, K. Toutanova // NAACL-HLT. — 2019.

95. Roberta: A robustly optimized bert pretraining approach / Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, V. Stoyanov // arXiv preprint arXiv:1907.11692. — 2019.

96. A deep network model for paraphrase detection in short text messages / B. Agarwal, H. Ramampiaro, H. Langseth, M. Ruocco // Information Processing & Management. — 2018. — Vol. 54, no. 6. — P. 922–937.

97. *Wang Z.*, *Mi H.*, *Ittycheriah A.* Sentence similarity learning by lexical decomposition and composition // arXiv preprint arXiv:1602.07019. — 2016.

98. *Ji Y.*, *Eisenstein J.* Discriminative improvements to distributional sentence similarity // Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing. — 2013. — P. 891–896.

99. Attention is All you Need / A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, I. Polosukhin // ArXiv. — 2017. — Vol. abs/1706.03762.

100. *Wieting J.*, *Mallinson J.*, *Gimpel K.* Learning paraphrastic sentence embeddings from back-translated bitext // arXiv preprint arXiv:1706.01847. — 2017.

101. *McKeown K.* Paraphrasing questions using given and new information // American Journal of Computational Linguistics. — 1983. — Vol. 9, no. 1. — P. 1–10.

102. Paraphrase Generation with Deep Reinforcement Learning / Z. Li, X. Jiang, L. Shang, H. Li // Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing. — 2018. — P. 3865–3878.

103. A deep generative framework for paraphrase generation / A. Gupta, A. Agarwal, P. Singh, P. Rai // Proceedings of the AAAI Conference on Artificial Intelligence. Vol. 32. — 2018.

104. *Fu Y.*, *Feng Y.*, *Cunningham J.* Paraphrase Generation with Latent Bag of Words // NeurIPS. — 2019.

105. *Egonmwan E.*, *Chali Y.* Transformer and seq2seq model for Paraphrase Generation // NGT@EMNLP-IJCNLP. — 2019.

106. *Roy A.*, *Grangier D.* Unsupervised Paraphrasing without Translation // ACL. — 2019.

107. *Barzilay R.*, *McKeown K.* Extracting Paraphrases from a Parallel Corpus // ACL. — 2001.

108. *Coster W.*, *Kauchak D.* Learning to Simplify Sentences Using Wikipedia // Monolingual@ACL. — 2011.

109. Extracting Lexically Divergent Paraphrases from Twitter / W. Xu, A. Ritter, C. Callison-Burch, W. Dolan, Y. Ji // Transactions of the Association for Computational Linguistics. — 2014. — Vol. 2. — P. 435–448.

110. *Creutz M.* Open Subtitles Paraphrase Corpus for Six Languages // ArXiv. — 2018. — Vol. abs/1809.06142.

111. Google's neural machine translation system: Bridging the gap between human and machine translation / Y. Wu, M. Schuster, Z. Chen, Q. V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, K. Macherey, [et al.] // arXiv preprint arXiv:1609.08144. — 2016.

112. *Suzuki Y.*, *Kajiwara T.*, *Komachi M.* Building a non-trivial paraphrase corpus using multiple machine translation systems // Proceedings of ACL 2017, Student Research Workshop. — 2017. — P. 36–42.

113. Towards building Arabic paraphrasing benchmark / M. Alian, A. Awajan, A. Al-Hasan, R. Akuzhia // Proceedings of the Second International Conference on Data Science, E-Learning and Information Systems. — 2019. — P. 1–5.

114. SemEval-2012 Task 6: A Pilot on Semantic Textual Similarity / E. Agirre, D. M. Cer, M. T. Diab, A. Gonzalez-Agirre // SemEval@NAACL-HLT. — 2012.

115. *Jeffrey Pennington Richard Socher C. D. M.* GloVe: Global Vectors for Word Representation. // Empirical Methods in Natural Language Processing (EMNLP) 2014. — 2014. — P. 1532–1543.

116. Enriching Word Vectors with Subword Information / P. Bojanowski, E. Grave, A. Joulin, T. Mikolov // Transactions of the Association for Computational Linguistics. — 2017. — Vol. 5. — P. 135–146.

117. Distributed Representations of Words and Phrases and their Compositionality / T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, J. Dean // ArXiv. — 2013. — Vol. abs/1310.4546.

118. Efficient Estimation of Word Representations in Vector Space / T. Mikolov, K. Chen, G. S. Corrado, J. Dean // ICLR. — 2013.

119. ArmTDP: Eastern Armenian Treebank and Dependency Parser. / M. M. Yavrumyan, H. H. Khachatrian, A. S. Danielyan, G. D. Arakelyan. // XI International Conference on Armenian Linguistics, Abstracts. Yerevan. — 2017.

120. Learning Word Vectors for 157 Languages / E. Grave, P. Bojanowski, P. Gupta, A. Joulin, T. Mikolov // ArXiv. — 2018. — Vol. abs/1802.06893.

121. Evaluation methods for unsupervised word embeddings / T. Schnabel, I. Labutov, D. Mimno, T. Joachims // EMNLP. — 2015.

122. *Svoboda L.*, *Brychcin T.* New word analogy corpus for exploring embeddings of Czech words // ArXiv. — 2016. — Vol. abs/1608.00789.

123. *Köper M.*, *Scheible C.*, *Walde S. S. im*. Multilingual Reliability and ”Semantic” Structure of Continuous Word Spaces // IWCS. — 2015.

124. *Berardi G.*, *Esuli A.*, *Marcheggiani D.* Word Embeddings Go to Italy: A Comparison of Models and Training Datasets // IIR. — 2015.

125. EASTERN ARMENIAN NATIONAL CORPUS / K. V.G., D. M.A., L. D.V., P. V.A., P. A.E., R. S.V. // ”Dialog 2009”. — 2009.

126. Byte Pair Encoding: a Text Compression Scheme That Accelerates Pattern Matching / T. Kida, S. Fukamachi, M. Takeda, A. Shinohara, T. Shinohara, S. Arikawa //. — 1999.

127. CoNLL 2018 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies / D. Zeman, J. Hajic, M. Popel, M. Potthast, M. Straka, F. Ginter, J. Nivre, S. Petrov // CoNLL 2018. — 2018.

128. Universal Dependency Parsing from Scratch / P. Qi, T. Dozat, Y. Zhang, C. D. Manning // CoNLL Shared Task. — 2018.

129. *Koskenniemi K.* A General Computational Model For Word-Form Recognition And Production // COLING. — 1984.

130. *Plisson J.*, *Lavrac N.*, *Mladenic D.* A Rule based Approach to Word Lemmatization //. — 2004.

131. *Chrupała G.*, *Dinu G.*, *Genabith J.* Learning Morphology with Morfette // LREC. — 2008.

132. Joint Lemmatization and Morphological Tagging with Lemming / T. Müller, R. Cotterell, A. M. Fraser, H. Schütze // EMNLP. — 2015.

133. *Chakrabarty A.*, *Pandit O.*, *Garain U.* Context Sensitive Lemmatization Using Two Successive Bidirectional Gated Recurrent Networks // ACL. — 2017.

134. *Dreyer M.*, *Smith J.*, *Eisner J.* Latent-Variable Modeling of String Transductions with Finite-State Methods // EMNLP. — 2008.

135. *Nicolai G.*, *Kondrak G.* Leveraging Inflection Tables for Stemming and Lemmatization // ACL. — 2016.

136. *Brück T. vor der*, *Eger S.*, *Mehler A.* Lexicon-assisted tagging and lemmatization in Latin: A comparison of six taggers and two lemmatization models // LaTeCH@ACL. — 2015.

137. *Bergmanis T.*, *Goldwater S.* Context Sensitive Neural Lemmatization with Lematus // NAACL-HLT. — 2018.

138. Nematus: a Toolkit for Neural Machine Translation / R. Sennrich [et al.] // EACL. — 2017.

139. Turku Neural Parser Pipeline: An End-to-End System for the CoNLL 2018 Shared Task / J. Kanerva, F. Ginter, N. Miekka, A. Leino, T. Salakoski // CoNLL Shared Task. — 2018.

140. *Rybak P.*, *Wróblewska A.* Semi-Supervised Neural System for Tagging, Parsing and Lematization // CoNLL Shared Task. — 2018.

141. *Dietterich T. G.*, *Hild H.*, *Bakiri G.* A Comparative Study of ID3 and Backpropagation for English Text-to-Speech Mapping // ML. — 1990.

142. *Collobert R.*, *Weston J.* A unified architecture for natural language processing: deep neural networks with multitask learning // ICML ’08. — 2008.

143. *Zhang Y.*, *Weiss D.* Stack-propagation: Improved Representation Learning for Syntax // ArXiv. — 2016. — Vol. abs/1603.06598.

144. *Plank B.*, *Søgaard A.*, *Goldberg Y.* Multilingual Part-of-Speech Tagging with Bidirectional Long Short-Term Memory Models and Auxiliary Loss // ArXiv. — 2016. — Vol. abs/1604.05529.

145. Deep contextualized word representations / M. E. Peters, M. Neumann, M. Iyyer, M. Gardner, C. Clark, K. Lee, L. Zettlemoyer // NAACL-HLT. — 2018.

146. FLAIR: An Easy-to-Use Framework for State-of-the-Art NLP / A. Akbik, T. Bergmann, D. Blythe, K. Rasul, S. Schweter, R. Vollgraf // NAACL-HLT. — 2019.

147. *Heinzerling B.*, *Strube M.* BPEmb: Tokenization-free Pre-trained Subword Embeddings in 275 Languages // ArXiv. — 2018. — Vol. abs/1710.02187.

148. On the Importance of Subword Information for Morphological Tasks in Truly Low-Resource Languages / Y. Zhu, B. Heinzerling, I. Vulic, M. Strube, R. Reichart, A. Korhonen // CoNLL. — 2019.

149. Natural language processing (almost) from scratch / R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, P. Kuksa // Journal of machine learning research. — 2011. — Vol. 12, ARTICLE. — P. 2493–2537.

150. *Dai A. M.*, *Le Q. V.* Semi-supervised Sequence Learning // NIPS. — 2015.

151. *Scudder H. J.* Probability of error of some adaptive pattern-recognition machines // IEEE Trans. Inf. Theory. — 1965. — Vol. 11. — P. 363–371.

152. *Hinton G. E.*, *Vinyals O.*, *Dean J.* Distilling the Knowledge in a Neural Network // ArXiv. — 2015. — Vol. abs/1503.02531.

153. *Xu C.*, *Tao D.*, *Xu C.* A Survey on Multi-view Learning // ArXiv. — 2013. — Vol. abs/1304.5634.

154. Semi-Supervised Sequence Modeling with Cross-View Training / K. Clark, M.-T. Luong, C. D. Manning, Q. V. Le // EMNLP. — 2018.

155. *Tarvainen A.*, *Valpola H.* Weight-averaged, consistency targets improve semi-supervised deep learning results // CoRR,, vol. abs/1703. — 1780. — Vol. 2017.

156. *McClosky D.*, *Charniak E.*, *Johnson M.* Effective Self-Training for Parsing // HLT-NAACL. — 2006.

157. *Aker A.*, *Petrak J.*, *Sabbah F.* An Extensible Multilingual Open Source Lemmatizer // RANLP. — 2017.

158. *Smith R.* An overview of the Tesseract OCR engine // Ninth international conference on document analysis and recognition (ICDAR 2007). Vol. 2. — IEEE. 2007. — P. 629–633.

159. *Khirbat G.* OCR Post-Processing Text Correction using Simulated Annealing (OPTeCA) // ALTA. — 2017.

160. *Amrhein C.*, *Clematide S.* Supervised ocr error detection and correction using statistical and neural machine translation methods // Journal for Language Technology and Computational Linguistics (JLCL). — 2018. — Vol. 33, no. 1. — P. 49–76.

161. *Mokhtar K.*, *Bukhari S. S.*, *Dengel A.* OCR Error Correction: State-of-the-Art vs an NMT-based Approach // 2018 13th IAPR International Workshop on Document Analysis Systems (DAS). — IEEE. 2018. — P. 429–434.

162. Opennmt: Open-source toolkit for neural machine translation / G. Klein, Y. Kim, Y. Deng, J. Senellart, A. M. Rush // arXiv preprint arXiv:1701.02810. — 2017.

163. *David Yarowsky Grace Ngai R. W.* Inducing multilingual text analysis tools via robust projection across aligned corpora. // In Proceedings of the First International Conference on Human Language Technology Research. Association for Computational Linguistics, Stroudsburg, PA, USA, HLT'01. — 2001. — P. 1–8.

164. *Imed Zitouni R. F.* Mention detection crossing the language barrier. // In Proceedings of the 2008 Conference on Empirical Methods in Natural Language Processing. Association for Computational Linguistics. — 2008. — P. 600–609.

165. *Maud Ehrmann Marco Turchi R. S.* Building a multilingual named entity-annotated corpus using annotation projection. // In Proceedings of Recent Advances in Natural Language Processing. Association for Computational Linguistics. — 2011. — P. 118–124.

166. *Klesti Hoxha A. B.* An Automatically Generated Annotated Corpus for Albanian Named Entity Recognition. // CYBERNETICS AND INFORMATION TECHNOLOGIES. — 2018. — Vol. 18, no. 1.

167. *Weber, Pötzl.* NERU: Named Entity Recognition for German. // Proceedings of GermEval 2014 Named Entity Recognition Shared Task. — 2014. — P. 157–162.

168. Learning multilingual named entity recognition from Wikipedia. / N. J., R. N., R. W., M. T., C. J. R. // Artificial Intelligence. — 2013. — Vol. 194. — P. 151–175.

169. *Sysoev A. A. A. I. A.* Named Entity Recognition in Russian: the Power of Wiki-Based Approach. // Computational Linguistics and Intellectual Technologies: Proceedings of the International Conference "Dialogue 2016". — 2016.

170. Texterra: A Framework for Text Analysis. / T. D., A. N., N. Y., S. A., A. I., M. V., F. D., K. A., K. S. // Proceedings of the Institute for System Programming of RAS. — 2014. — Vol. 26, no. 1. — P. 421–438.

171. *Sang. E. F. T. K.* Introduction to the CoNLL-2002 Shared Task: Language-Independent Named Entity Recognition. // Proceedings of CoNLL-2002. — 2002. — P. 155–158.

172. *Erik F. Tjong Kim Sang F. D. M.* Introduction to the CoNLL-2003 Shared Task: LanguageIndependent Named Entity Recognition. // Proceedings of the CoNLL-2003. — 2003. — Vol. 4. — P. 142–147.

173. Neural Architectures for Named Entity Recognition / G. Lample, M. Ballesteros, S. Subramanian, K. Kawakami, C. Dyer. // Proceedings of NAACL-2016, San Diego, California, USA. — 2016.

174. *Xuezhe Ma E. H.* End-to-end Sequence Labeling via Bi-directional LSTM-CNNs-CRF. // Proceedings of ACL. 2016. — 2016.

175. *Genthial. G.* Sequence Tagging with Tensorflow. [Electronic Resource]. — 2017. — URL: https://guillaumegenthial.github.io/sequence-tagging-with-tensorflow.html (visited on 11/11/2018).

176. *Jenny Rose Finkel Trond Grenager C. M.* Incorporating Non-local Information into Information Extraction Systems by Gibbs Sampling. // Proceedings of the 43nd Annual Meeting of the Association for Computational Linguistics (ACL 2005). — 2005. — P. 363–370.

177. Fast and Accurate Entity Recognition with Iterated Dilated Convolutions. / E. Strubell, P. Verga, D. Belanger, A. McCallum. — 2017.

178. Finding function in form: Compositional character models for open vocabulary word representation. / W. Ling, T. Lúıs, L. Marujo, R. F. Astudillo, S. Amir, C. Dyer, A. W. Black, I. Trancoso // Proceedings of EMNLP. — 2015.

179. *Garg U.*, *Goyal V.* Maulik: A plagiarism detection tool for hindi documents // Indian Journal of Science and Technology. — 2016. — Vol. 9, no. 12. — P. 1–11.

180. Study on Extrinsic Text Plagiarism Detection Techniques and Tools. / D. Gupta [et al.] // Journal of Engineering Science & Technology Review. — 2016. — Vol. 9, no. 5.

181. A Deep Learning Approach to Persian Plagiarism Detection. / E. Gharavi, K. Bijari, K. Zahirnia, H. Veisi // FIRE (Working Notes). — 2016. — Vol. 34. — P. 154–159.

182. *Ивахненко А.* Так устроен поиск заимствований в Антиплагиате. — 2018. — URL: https://habr.com/ru/company/antiplagiat/blog/429634/ (visited on 12/20/2020).

183. *Klimenko A.* ELASTICSEARCH VS. SOLR VS. SPHINX: BEST OPEN SOURCE SEARCH PLATFORM COMPARISON. — URL: https://greenice.net/elasticsearch-vs-solr-vs-sphinx-best-open-source-search-platform-comparison/ (visited on 12/20/2020).

184. *Kılıç U.*, *Aksakalli K.* Comparison of Solr and Elasticsearch Among Popular Full Text Search Engines and Their Security Analysis // Future Internet of Things and Cloud Workshops, 2015 6th International Conference on. — 2016. — P. 163–168.

185. *Culbertson J.* 13 Top APIs for Search. — 2019. — URL: https://www.programmableweb.com/news/13-top-apis-search/brief/2019/04/07 (visited on 12/20/2020).

186. Overview. — URL: https://developers.google.com/custom-search/docs/overview (visited on 12/20/2020).

Благодарности

Хотелось бы выразить благодарность Турдакову Денису за научное руководство, а также поблагодарить Ярослава Недумова, Кирилла Скорнякова, Карена Аветисяна, Еву Ешилбашян, Ариану Асатрян, Артура Маладжяна, Шагане Тигранян, Сергея Королева, Гарника Давтяна, Владимира Майорова за помощь в разработке и экспериментах, Ивана Андрианова, Смбата Гогяна, Марата Яврумяна, Гранта Хачатряна за ценные отзывы и обсуждения, а также Севака Саргсяна, Армана Дарбиняна за оказанную поддержку в процессе написания диссертации.

Список рисунков

Список таблиц

Приложение А

Список использованных признаков для стилометрического анализа армянских текстов.

Уровень	Группа	Признаки

символы	пунктуация	– знаки пунктуации, их частота, например:
		  – наличие знака пунктуации
		  – #[знаки пунктуации] / #[слова] для всех знаков вместе, а также для каждого отдельно
		– комбинации пунктуации и пробельных символов, например:
		  – наличие пробела перед знаком пунктуации
		  – наличие пробела после знака пунктуации
		  – #[наличие пробела перед знаком пунктуации] / #[знаки пунктуации]
		  – #[наличие пробела после знака пунктуации] / #[знаки пунктуации]
		  – вышеперечисленное для каждого знака пунктуации отдельно

	общие	– суффиксы:
		  – наличие конкретного суффикса
		  – #[слова с конкретным суффиксом] / #[слова]
		– префиксы:
		  – наличие конкретного префикса
		  – #[слова с конкретным префиксом] / #[слова]

слова	общие	– частота слов:
		  – наличие длинных слов
		  – #[длинные слова] / #[слова]
		  – наличие редких слов
		  – #[редкие слова] / #[слова]
		  – средняя частота используемых слов
		– наличие терминов на латинском/кириллице
		– наличие жаргона/неформальных выражений
		– написание именованных сущностей

	сокращения	– использование сокращения вместо полной формы (например, թ. вместо թվական, թ-ի вместо թվականի) – стиль склонения сокращений (например, թ. vs թ.-ի vs թ-ի) – сокращения пишутся с маленькой буквы (например, բուհ, բոհ) – сокращения пишутся с большой буквы (например, ԲՈՒՀ, ԲՈՀ) – наличие разделителя в сокращениях (դդ. vs դ.դ. или ստ. vs ս.տ.)
	числительные	– написание количественных (например, տասը հազար vs 10000) – написание порядковых (например, երկրորդ vs ii vs II vs 2-րդ vs 2րդ) – о формат написания дат (например, 12\18\10 vs 12/18/10 vs 12-18-10 vs 12.18.10 vs 12\18\2010 vs 12/18/2010 vs 12-18-2010 vs 12.18.2010 vs 18 դեկտեմբերի, 2010)

	N-граммы	– биграммы:
		  – наличие биграммы с низкой IDF
		  – #[биграммы с низкой IDF] / (#[слов] - 1)
		– триграммы:
		  – наличие траграммы с низкой IDF
		  – #[триграммы с низкой IDF] / (#[слов] - 2)

предложения	общие	– количество предложений
		– средняя длина предложений:
		  – среднее число слов в предложении
		  – среднее число символов в предложении
		– максимальная длина предложений:
		  – максимальное число слов в предложении
		  – максимальное число символов в предложении
		– минимальная длина предложений:
		  – минимальное число слов в предложении
		  – минимальное число символов в предложении
		– длинные предложения:
		  – наличие длинных предложений
		  – #[длинные предложения (с точки зрения числа слов)] / #[предложения]
		  – #[длинные предложения (с точки зрения числа символов)] / #[предложения]
		– короткие предложения:
		  – наличие коротких предложений
		  – #[короткие предложения (с точки зрения числа слов)] / #[предложения]
		  – #[короткие предложения (с точки зрения числа символов)] / #[предложения]

	морфологические	части речи: – #[слова с этой частью речи] / #[слова] – наличие последовательности 2-х конкретных частей речи – #[последовательность 2-х конкретных частей речи] / (#[слова] - 1) – наличие последовательности 3-х конкретных частей речи – #[последовательность 3-х конкретных частей речи] / (#[слова] - 2)

	грамматические	– синтаксические связи, их частота:
		  – наличие синтаксической связи в параграфе (за исключением частых связей, которые повсеместны)
		  – #[предложения, где присутствует синтаксическая связь] / #[предложения]
		– частота простых предложений:
		  – наличие простых предложений
		  – #[простые предложения] / #[предложения]
		– частота сложных предложений:
		  – наличие сложных предложений
		  – #[сложные предложения] / #[предложения]

параграфы	читабельность	– Flesch reading ease – SMOG grade – Flesch-Kincaid grade – Coleman-Liau index – automated readability index – Dale-Chall readability score – difficult words – Linsear write formula – Gunning fog

Таблица 42 — Список использованных признаков для стилометрического анализа армянских текстов.

Приложение Б

Результаты экспериментов по определению качества методов обнаружения границ нарушений стиля для случайного базового метода, Karas et al. и иерархической кластеризации (AC).

	Диссертации				Художественные				Учебники				Энциклопедии				Новости			
Метод (гипер-параметры)	WinP	WR	F1	Достоверность	WinP	WR	F1	Достоверность	WinP	WinR	F1	Достоверность	WinP	WinR	F1	Достоверность	WinP	WinR	F1	Достоверность
Karas (0.3; alpha = 0.01)	0.31545	0.34115	0.3278	0.9885	0.5052	0.6447	0.4637	0.9883	0.0	0.1183	0.0	0.9861	0.0	0.5539	0.0	0.9954	0.6749	0.9891	0.6842	0.9943
AC (n_clusters = 2; PCA = false)	0.5953	0.5854	0.5903	0.9922	0.4574	0.6518	0.4278	0.9869	0.5326	0.5819	0.4701	0.9863	0.3175	0.8727	0.3060	0.9917	0.6406	0.9921	0.6493	0.9941
AC (n_clusters = 2; PCA = true)	0.6007	0.5502	0.5743		0.5525	0.6121	0.4324	0.9888	0.5611	0.5841	0.4726	0.9874	0.2623	0.7736	0.2231	0.9908	0.6336	0.9874	0.6363	0.9933
AC (n_clusters = 3; PCA = false)	0.5465	0.7462	0.631	0.9918	0.4095	0.7961	0.4626	0.9829	0.4915	0.7682	0.5319	0.9838	0.2551	0.9198	0.2871	0.9876	0.6971	0.9458	0.6586	0.9917
AC (n_clusters = 4; PCA = false)	0.517	0.8236	0.6352	0.9911	0.3974	0.8792	0.4810	0.9817	0.4627	0.8272	0.5354	0.9813	0.2755	0.9104	0.2814	0.9854	0.7558	0.9497	0.7169	0.9913
Random (style_-breach_-ratio = 0.25)	0.5368	0.2954	0.38	-	0.4229	0.5730	0.3244	0.9845	0.4569	0.3187	0.2768	0.9835	0.3716	0.7749	0.2918	0.9891	0.7691	0.9434	0.7337	0.9946
Random (style_-breach_-ratio = 0.5)	0.4726	0.5105	0.49	-	0.4187	0.5853	0.3509	0.9849	0.4214	0.4411	0.3339	0.9816	0.2870	0.8701	0.2771	0.9869	0.7099	0.9547	0.6878	0.9935
Random (style_-breach_-ratio = 0.75)	0.5367	0.6879	0.60	-	0.3579	0.7982	0.4273	0.9819	0.4069	0.5545	0.3767	0.9797	0.3299	0.8039	0.2814	0.9884	0.6557	0.9674	0.6435	0.9916

Таблица 43 — Результаты методов обнаружения нарушений стиля.

Приложение В

Гиперпараметры нейронный сетей для нахождения и исправления ошибок автоматического распознавания армянских текстов.

Гиперпараметр		Значение
Оптимизатор	batch size	64
	метод	Adam
	beta1	0.9
	beta2	0.999
	epsilon	1e-7
	скорость обучения	0.01
Число скрытых слоев		2
Функция активации		ReLU
Число нейронов в скрытом слое		128

Таблица 44 — Гиперпараметры многолинейного персептрона для задачи нахождения ошибок распознавания.

Гиперпараметр		Значение
Оптимизатор	batch size	4
	метод	Adam
	beta1	0.8
	beta2	0.998
	train_steps	20000
	скорость обучения	0.01
Слои		[“encoder” : {“layer1” : “Multi-head self-attention “layer2” : “Dense”, “number of units” : 32, “number of heads in multi-head self-attention”: 8, “Dense inner dimension” : 32, “dropout” : 0.1}, “decoder”: {“layer1” : “Multi-head self-attention “layer2” : ”Dense”, “layer 3” : “Multi-head self-attention “number of units” : 32, “number of heads in multi-head self-attention”: 8, “Dense inner dimension” : 32, “dropout” : 0.1 }]
Дропаут		ReLU
Размер словаря входных символов		150
Размер словаря выходных символов		150
embedding_size		32
maximum_features_length		30
maximum_labels_length		30

Таблица 45 — Гиперпараметры кодировщика-декодировщика для задачи исправления ошибок распознавания.

ч[htp]

Гиперпараметр		Значение
Оптимизатор	batch size	100
	метод	Adam
	beta1	0.9
	beta2	0.9
	epsilon	1e-4
	скорость обучения	0.002
Слои		По умолчанию.
Дропаут		0.25
Размер обучаемых векторных представлений		10

Таблица 46 — Гиперпараметры нейронной сети COMBO для задачи исправления ошибок распознавания.