Федеральное государственное автономное образовательное учреждение
высшего образования
«Московский физико-технический институт
(национальный исследовательский университет)»
Физтех-школа Прикладной Математики и Информатики
Кафедра корпоративных информационных систем

Направление подготовки: 03.03.01 Прикладные математика и физика
Направленность (профиль) подготовки: Физика и компьютерные технологии

Исправление опечаток и грамматических ошибок в русскоязычных текстах

(бакалаврская работа)

Студент:
Бунин Дмитрий Алексеевич

(подпись студента)

Научный руководитель:
Сорокин Алексей Андреевич
канд. физ.-мат. наук

(подпись научного руководителя)

Москва
2021

Аннотация

Исследование посвящено автоматическому исправлению опечаток и относится к области компьютерной лингвистики и автоматической обработки текста. Целью работы является исследовать возможность применения модели BERT к исправлению опечаток в русскоязычных текстах на материале корпуса SpellRuEval. Выделим основные задачи исследования:

1. создание комплексной системы исправления опечаток с использованием BERT;
2. изучение степени влияния вариантов модели BERT на улучшение качества решения задачи.
3. изучение влияния других компонентов модели.

Основным результатом работы является достигнутая величина F1-меры в 79.8%, что превышает показатели как достигнутые в предыдущих работах, так и показываемые открытыми инструментами. Без использования BERT удалось добиться качества в 77.4%, что без учета первой модели также является лучшим результатом на данный момент.

Содержание

Глава 1

Введение и обоснование актуальности

Социальные медиа на данный момент являются крупнейшим источником новых текстов. В качестве примера можно рассмотреть комментарии на форумах, посты в соцсетях, подписи к фотографиям, публикации в блогах. Многие тексты из этих источников содержат орфографические и пунктуационные ошибки, что может увеличивать вариативность используемого языка, осложняя автоматическую обработку. Так, согласно работе «Моделирование расширенной лемматизации для русского языка на основе морфологического парсера TnT-Russian» [1] около 8% несловарных слов в корпусе ГИКРЯ (Генеральный Интернет-Корпус Русского Языка, http://www.webcorpora.ru/) являются на самом деле результатом опечаток в словарных словах. Следует также отметить, что многие крупные корпуса, например ГИКРЯ, «Тайга» [2], содержат морфологический разбор предоставляемых предложений, при выполнении которого исправления ошибок не производилось. Это могло значительно повлиять на качество корпусной разметки.

Задача исправления опечаток и грамматических ошибок имеет также широкое применение в поисковых системах для исправления запроса. Так, в работе «Автоматическое исправление опечаток в поисковых запросах без учета контекста» [3] было подсчитано, что около 12% всех поисковых запросов к Яндексу содержат по крайней мере одну ошибку, а в работе «Исправление поисковых запросов в Яндексе» [4] говорится о 15%. Проверка орфографии также является неотъемлемой частью любого современного текстового редактора.

Решаемая задача имеет приложения и в сфере образования. Детектирование и исправление ошибок может помочь при оценке письменных ответов студентов. Например, в работе «How to account for mispellings: Quantifying the benefit of character representations in neural content scoring models» [5] было показано, что предварительное исправление опечаток улучшает качество оценивания.

Недавние исследования демонстрируют, что наличие опечаток может значительно повлиять на качество современных нейросетевых моделей. Авторами работы «Noisy Text Data: Achilles' Heel of BERT» [6] было показано, что добавление в текст случайных опечаток негативно влияет на результаты модели BERT [7] при дообучении на задачах определения тональности (sentiment analysis) и схожести текстов (textual similarity).

В свете вышеизложенных обстоятельств мы считаем достаточно важным изучать методы для исправления опечаток и грамматических ошибок в русскоязычных текстах.

Глава 2

Формальная постановка задачи

Для конкретизации решаемой задачи было выбрано соревнование SpellRuEval — это одна из дорожек, предложенных участникам конференции «Диалог 2016». В состязании предлагалось исправлять опечатки и грамматические ошибки в текстах из социальных медиа.

2.1 Виды ошибок

2.1.1 Классификация с точки зрения языкознания

Для начала обозначим какие виды ошибок в русскоязычных текстах нам могут быть интересны с точки зрения языкознания:

1. Грамматическая ошибка — это ошибка, связанная с нарушением закономерностей и правил грамматики. К таким ошибкам относятся неправильное образование и употребление форм слова, неверное построение словосочетаний и предложений. [8]

2. Орфографическая ошибка — это неправильное написание слова. Она может быть допущена только на письме, обычно в слабой фонетической позиции. Такую ошибку можно только увидеть, но услышать ее нельзя.

3. Опечатка — это ошибка в печатном тексте, обычно в результате случайности.

2.1.2 Классификация по словарности

Можно также выделить несловарные (non-word) и словарные (real-word) ошибки. Первый тип характеризуется тем, что получившееся в результате опечатки слово не является словарным, во втором случае получается словарное слово. Исправлять ошибки первого типа проще. Во-первых, их проще обнаружить, достаточно проверять слова на присутствие в большом словаре. Во-вторых, для их исправления может не понадобиться знание контекста. Ошибки второго типа без понимания контекста исправить нельзя.

2.1.3 Классификация по типу преобразования

Если рассматривать процесс исправления ошибок, то можно выделить различные типы преобразований по тому, как они действуют на смежные слова:

- «один к одному» — одно слово заменяется на какое-то другое слово (*лчше* → *лучше*);

- «один ко многим» — одно слово заменяется на несколько слов (*вобщем* → *в общем*);
- «многие к одному» — несколько слов заменяется на одно слово (*как то* → *как-то*);
- «многие ко многим» — несколько слов заменяются на несколько других слов (*вобщем то* → *в общем-то*).

2.2 Обучающее и тестовое множества

Начнем с рассмотрения предложенных обучающих и тестовых примеров и способа их сбора. Согласно статье [9], посвященной соревнованию, организаторы использовали тексты из ГИКРЯ, собранные на блог-платформе «Живой Журнал» (https://www.livejournal.com/). Было автоматически отобрано около 10000 предложений, содержащих несловарные слова. Собранное множество было дополнено несколькими сотнями предложений, содержащих ошибки некорректного употребления словарных слов (real-word errors). Затем все предложения были перепроверены на предмет содержания ошибок, могло так оказаться, что несловарными были редкие имена, сленговые слова или неологизмы. Итого, было собрано около 5000 предложений.

После этого предложения были направлены разметчикам, которые должны были исправить ошибки и, в случае возникновения трудностей, написать комментарий организаторам. Приведем инструкцию, выданную разметчикам:

Что надо исправлять:

- опечатки: *мнея* → *меня*;
- орфографические ошибки: *митель* → *метель*;
- смысловые ошибки: *компания* → *кампания*;
- намеренно неправильное написание: *хоцца* → *хочется*, *ваще* → *вообще*;
- грамматические ошибки (согласование): *он видят* → *он видит*;
- ошибки с пробелами/дефисами: *както* → *как-то*;
- смешанное использование чисел и букв в числительных: *2-ух* → *двух*;
- использование цифр вместо букв: *в4ера* → *вчера*.

Что не надо исправлять:

- иностранные слова, включая кириллицу;
- неформальные сокращения: *прога* → *программа*;
- пунктуационные ошибки;

- ошибки, связанные с заглавным/строчным написанием;
- неразличение букв «е» и «ё».

По словам организаторов, большинство сложных случаев для исправления составляли разговорные формы слов, такие как «ваще» вместо «вообще» и «щас» вместо «сейчас». В большинстве случаев их можно было заменить на литературную форму без потери смысла. Исключения составляли некоторые выражения для экспрессии, такие как «Ну ты ваще», «Да щас!», где замена на литературную форму делает выражение неупотребимым в том же контексте. Ввиду сложностей различения ситуаций, когда замена возможна, а когда невозможна, было решено во всех случаях приводить слова к литературной форме.

Видим, что все рассматриваемые ошибки являются локальными, то есть затрагивают смежные слова. Более сложные грамматические ошибки не рассматриваются. В таком случае бывает весьма сложно понять совершает человек опечатку, орфографическую ошибку или грамматическую ошибку. В тексте ниже мы не будем различать эти понятия, если на этом не будет сакцентированно внимание.

После полной разметки осталось порядка 2400 исправленных предложений, в которые добавили 1600 корректно написанных предложений. Затем итоговые 4000 предложений разделили пополам на обучающее множество и тестовое.

2.2.1 Статистика по обучающему множеству

Изучим статистику по обучающему множеству.

- Количество предложений: 2000.
- Количество исправлений: 1727.
- Распределение количества ошибок в предложениях:
  - 0: 40.6%;
  - 1: 40.1%;
  - 2: 14.0%;
  - 3: 4.0%;
  - $\geqslant$4: 1.4%.
- Распределение ошибок по типу преобразования:
  - «один к одному»: 80.0%;
  - «один ко многим»: 15.6%;
  - «многие к одному»: 4.1%;
  - «многие ко многим»: 0.2%;

- Распределение ошибок по расстоянию Дамерау-Левенштейна [10] [11]:
  - 1: 77.2%;
  - 2: 15.5%;
  - 3: 4.6%;
  - $\geqslant$4: 2.7%;
- Количество уникальных исправлений, неизвестных словарю: 68.

Такая же статистика для тестового множества не вычислялась для минимизации утечки информации. Кроме того, оказалось, что в итоговом файле было не 2000 тестовых предложений, а 2008.

2.3 Оценка качества

Теперь, когда обучающее и тестовое множества определены, надо понять принципы, на основе которых будет оцениваться качество моделей. Авторы соревнования решили оценивать модель на основе индивидуальных исправлений, то есть основной единицей в измерении качества будет не целое предложение, а отдельные исправления, которых в предложении может быть несколько. В таком случае, возникает проблема выравнивания. В самом деле, чтобы оценивать правильность отдельных исправлений надо понимать какие токены исходного предложения переходят в токены исправленного, что может быть непросто если нет соответствия вида «один к одному». Например, в исходном предложении может быть написано «кто то» вместо «кто-то» или «итак» вместо «и так».

Опишем используемую процедуру выравнивания:

1. Предобработка текста, включающая в себя перевод всего текста в нижний регистр.
2. Токенизация. Организаторы предпочли использовать свой алгоритм, а не один из стандартных. Алгоритм:
   (a) Разделение слов по пробелам.
   (b) Удаление изолированных знаков препинания.
   (c) Удаление знаков препинания на краях оставшихся токенов. Например, запятых на концах слов.
3. Процедура выравнивания между предобработанными предложениями:
   (a) При помощи динамического программирования находится наибольшая общая подпоследовательность на уровне слов. Токены, находящиеся на одинаковых позициях в этой подпоследовательности, считаются соответствующими друг другу.

(b) Группы токенов, располагающиеся между выровненными на предыдущем шаге, выравниваются раздельно в соответствии со своими позициями. Для этого строятся переходы согласно расстоянию Дамерау-Левенштейна. Точками выравнивания в таком случае считаются соответствующие друг другу пробелы.

Рассмотрим работу процедуры на примере из статьи, посвященном соревнованию. В качестве исходного предложения рассмотрим: «помоему кто то из них то же ошипся», а в качестве исправления: «по-моему кто-то из них тоже ошибся». После выполнения первого этапа выравнивания получим следующие соответствия:

- «помоему кто то» → «по-моему кто-то»,
- «из» → «из»,
- «них» → «них»,
- «то же ошипся» → «тоже ошибся».

Во время выполнения второго этапа удастся установить дополнительные соответствия:

- «помоему» → «по-моему»,
- «кто то» → «кто-то»,
- «то же» → «тоже»,
- «ошипся» → «ошибся».

После выполнения процедуры выравнивания можно выделить такую сущность, как исправление — это нетождественное преобразование между элементарными единицами выравнивания в исходном и преобразованном предложениях. Обозначим множество исправлений между исходными и скорректированными моделью предложениями, как S_{sol}. Аналогично, для исходных и эталонных предложений — S_{cor}. Теперь можем посчитать следующие метрики:

$$P = \frac{|S_{cor} \cap S_{sol}|}{|S_{sol}|},$$
$$R = \frac{|S_{cor} \cap S_{sol}|}{|S_{cor}|}.$$

Рассмотрим метрики по отдельности. Метрика P обозначает точность (precision), она показывает, как много исправлений из тех, что выполнила модель, являются корректными. Ее недостаток в том, что она не позволяет понять какую общую долю ошибок модели удалось исправить. Так, например, для получения большой точности можно было бы выполнять только те исправления, в которых велика уверенность.

Метрика R — это полнота (recall), ее смысл в том, чтобы показать общую долю исправленных ошибок. Тем не менее, она не позволяет увидеть как часто исправления выполнялись там, где исправлять не надо было.

В качестве итоговой метрики было взято среднее гармоническое полноты и точности, которое называется F-мерой (или F_1-мерой):

$$F_1 = \frac{2PR}{P + R}.$$

Такая агрегация позволяет сглаженно реагировать на слишком малое значение одной из метрик.

Глава 3

Обзор литературы

3.1 Классические методы

3.1.1 Расстояние Дамерау-Левенштейна

Исправление опечаток — одна из старейших задач в автоматической обработке естественного языка. Работы Левенштейна [11] и Дамерау [10] в середине 60-х позволили построить меру разности двух слов на основании количества элементарных изменений, которые надо внести в одно слово, чтобы получить другое. Такой подход позволяет для слов, отсутствующих в словаре, находить близкие словарные слова. Это делает возможным нахождение по несловарному слову похожих словарных.

Рассмотрим подробнее как устроено расстояние Дамерау-Левенштейна (иногда называется редакционным расстоянием). Для начала выделим список элементарных операций. Пусть имеем строку s. Операции:

- вставка — добавление какого-либо символа из алфавита в строку s;
- удаление — удаление какого-либо символа из s;
- замена — замена какого-либо символа из s на другой символ из алфавита;
- транспозиция — перестановка двух смежных символов.

Тогда рассматриваемое расстояние между строками s и t — это минимальное количество элементарных операций, требуемых чтобы из s получить t.

3.1.2 Модель шумного канала

Рассмотрим классическую схему решения задачи исправления опечаток на основе модели шумного канала. Пусть канал принимает в себя слово s, а на выходе после преобразований получается строка t. Мы, как наблюдатели, видим только t и хотим восстановить s. Для этого мы можем попробовать оценить $P(s|t)$. Несколько преобразуем это выражение, согласно формуле Байеса:

$$P(s|t) = \frac{P(s)P(t|s)}{\sum\limits_{s} P(s)P(t|s)}.$$

Знаменатель нам не интересен, потому что не зависит от s. Тогда для нахождения исходной строки требуется максимизировать числитель. Первый множитель — это

априорная вероятность исправления. Ее можно получить при помощи языковой модели. Второй множитель — это модель шумного канала (или модель ошибок), которая описывает вероятности различных преобразований внутри канала.

Схема была предложена в двух работах: «Context-based spelling correction» [12] и «A Spelling Correction Program Based on a Noisy Channel Model» [13]. Рассмотрим подробнее вторую, так как она доступна публично. Языковая модель была выбрана униграммная, а в качестве модели ошибок было рассмотрено взвешенное расстояние Дамерау-Левенштейна. Дело в том, что в его обычно версии все операции имеют одинаковый вес. Но если мы посмотрим на то, как люди набирают текст на клавиатуре, то при печати разные ошибки совершаются с разной вероятностью. Например, буквы, стоящие близко друг от друга на клавиатуре перепутать гораздо проще, чем те, что находятся друг от друга далеко. Таким образом, требуется научиться взвешивать разные операции в зависимости от их частот. Тогда расстояние — это минимальная сумма весов рассматриваемых элементарных операций.

В качестве элементарных операций авторами работы были выбраны следующие классы:

- удаление буквы s после буквы t;
- вставка буквы s после буквы t;
- замена буквы s на букву t;
- транспозиция соседних букв s и t.

Называя их классами, мы подразумеваем, что операции, над различными буквами s и t рассматриваются раздельно. Для оценки вероятностей всех преобразований был использован EM-алгоритм [14].

Можно заметить, что у этой схемы есть большой потенциал для улучшения. Так, авторы работы «An Improved Error Model for Noisy Channel Spelling Correction» [15] усложнили модель ошибок, значительно повысив качество.

3.1.3 Дальнейшее развитие

После этого можно выделить два общих направления развития в решении задачи исправления опечаток.

Первое направление нацелено на эффективный поиск кандидатов. Нахождение словарных слов на заданном расстоянии Дамерау-Левенштейна может быть непростой задачей для языков с развитой морфологией, например для русского. Для иллюстрации рассмотрим как растет количество словарных слов если увеличивать допустимое расстояние. В качестве словаря возьмем объединение wiktionary [16] и словаря Хагена [17]. В таблице 3.1 ясно видно, что с ростом параметра PP (редакционное расстояние) количество словоформ на заданном расстоянии растет практически экспоненциально и совпадает с точностью до порядка для всех выбранных слов.

Таблица 3.1 – Количества слов на заданном редакционном расстоянии (PP).

Токен	**PP = 1**	**PP = 2**	**PP = 3**	**PP = 4**
делать	7	137	1588	12913
длать	6	159	2312	19063
далеко	7	72	1476	16582
далико	3	126	2089	17763
собака	9	92	1812	17714
сабака	9	153	2067	17758

В работе «Error-tolerant Finite-state Recognition with Applications to Morphological Analysis and Spelling Correction» [18] был предложен алгоритм поиска в глубину по префиксному бору, построенному над словарем. Для уменьшения количества рассматриваемых кандидатов применялась эвристика на нижнюю оценку расстояния. Авторы работы «Fast approximate string matching with finite automata» [19] использовали алгоритм A-star [20] с разработанными для него эвристиками.

Второе направление нацелено на использование контекста для выбора корректного исправления. Так, например, в работе «Combining Trigram-based and Feature-based Methods for Context-Sensitive Spelling Correction» [21] были предложены подходы, основанные на триграммах частей речи, совстречаемости слов. Задача состояла в исправлении словарных ошибок в применении к заранее выбранным множествам слов, которые часто путают, например «their/there/they're». В такой постановке необходимо грамотно использовать контекст.

В работе «Four types of context for automatic spelling correction» [22] исследовалось влияние различных контекстных признаков на качество исправления несловарных слов. В качестве данных были взяты эссе носителей английского языка и изучающих его, как иностранный язык. Опишем предложенный алгоритм:

1. Обнаружение позиций для исправления. Используется проверка по словарю.
2. Для каждой выбранной на предыдущем этапе позиции генерируется список кандидатов на исправление на основании расстояния Дамерау-Левенштейна.
3. Вычисляются признаки кандидатов, в том числе и контекстные.
4. Производится ранжирование на признаках и выбирается самое высоко оцененное исправление.

Посмотрим теперь на то, какие контекстные признаки были испробованы:

- Словарные n-граммы.

  Подсчитывается сумма логарифмов от частот n-грамм, включающих в себя контекст и исправляемое слово, замененное на рассматриваемого кандидата. Были также попытки заменить суммы логарифмов на PMI (поточечная взаимная информация).

- Семантическая связь.

  Этот признак отвечает за то, насколько рассматриваемых кандидат хорошо вписывается в тему, о которой говорится в тексте. Например, имеем токен «carh», а в качестве кандидатов на исправление рассматриваются «car» и «card». Если в тексте речь идет об автомобилях, то больше подходит первый вариант, а если о казино, то второй. Для подсчета самого признака авторы брали сумму нормализованных значений поточечной взаимной информации между кандидатом и словами из контекста. В отличие от предыдущего признака этот способен охватывать больший контекст.

- Déjà vu.

  Признак основан на нормализованном подсчете других форм слова рассматриваемого кандидата. Если кандидат или его вариации встречаются в тексте эссе часто, то это можно рассматривать, как свидетельство в его пользу.

- Тематический список.

  Начало эссе можно рассматривать, как некий набор тематических ключевых слов. Если кандидат на исправление появляется в нем, то это может свидетельствовать о его превосходстве в рамках выбранной тематики эссе.

Как видим, не все эти признаки могут быть применены к отдельным предложениям.

3.2 Методы, применяемые для русского языка

В этом разделе будут рассмотрены методы, которые применялись для исправления опечаток в русскоязычных текстах.

До проведения соревнования SpellRuEval было не так много работ в этом направлении. Можно выделить работы «Исправление поисковых запросов в Яндексе» [4], «Автоматическое исправление опечаток в поисковых запросах без учета контекста» [3], которые концентрируют свое внимание именно на поисковых запросах. Рассмотрим, как во второй работе производится исправление ошибок в несловарных словах:

1. Генерация исправления. По строке q генерируется исправление c на основе модели шумного канала.

2. Выравнивание. Исходная строка и сгенерированное исправление выравниваются, чтобы понять какой токен исходного токена во что переходит.

3. Фильтрация. На этом этапе отбираются только те ошибки, которые авторы намерены были исправлять.

4. Классификация. Для каждого исправления генерируются признаки и на основе логистической регрессии над ними решается стоит применять это исправление или нет.

Посмотрим также на используемые признаки:

- веса q, c в словарной языковой модели;
- веса q, c в символьной языковой модели;
- длины q, c в символах;
- индикаторы присутствия в словаре для q, c;
- язык q, c (русский или английский);
- вероятность написания q, c с заглавной буквы. Данные собраны по большому корпусу;
- взвешенное редакционное расстояние от q до c;
- взаимный словарный контекст между q и c. Строится на основе слов, встречающихся с q и c в 3-граммах.

Как видим, на самом деле некая контекстная информация все же была использована.

Также с исправлением опечаток связана работа «Моделирование расширенной лемматизации для русского языка на основе морфологического парсера TnT-Russian» [1], где исследуется алгоритм исправления опечаток в применении к текстам из социальных медиа, содержащихся в ГИКРЯ. Работа модели состояла из следующий этапов:

1. Обучение взвешенного расстояния Дамерау-Левенштейна на основе EM-алгоритма.
2. Поиск кандидатов в словаре на расстоянии Дамерау-Левенштейна, не превышающем 2.
3. Ранжирование кандидатов при помощи модели ошибок на основе взвешенного расстояния Дамерау-Левенштейна. Модель предполагает рассмотрение всех выравниваний, которые не намного хуже оптимального.

3.2.1 SpellRuEval

Теперь рассмотрим, какие методы применялись на соревновании SpellRuEval, для этого изучим статью победителей соревнования: «Automatic spelling correction for Russian social media texts» [23].

Исправление состояло из трех этапов:

1. Генерация кандидатов.

   Для каждого токена находятся словарные слова на заданном расстоянии Дамерау-Левенштейна. К ним добавляются также слова, имеющие тот же фонетический код, что и у исправляемого. Алгоритм фонетического кодирования был построен специально для данной задачи на основе уже существующего алгоритма Double Metaphone [24]. Дополнительно вносятся около 50 вручную сформированных соответствия.

   Для обработки ошибки вставки пробела рассматриваются группы последовательных слов. Чтобы обработать ошибку удаления пробела, при поиске слов в боре допускается вставка пробела после получения словарного слова и продолжение поиска.

2. Отсечение гипотез.

   После предыдущего этапа для каждого токена или последовательной группы токенов имеется список кандидатов. На основе всех этих данных может быть сформировано исправленное предложение, путем комбинации всех возможных сочетаний кандидатов. В таком случае доступных исправлений предложения будет очень много, и их нужно проредить. Для этого каждый вариант исправления оценивается согласно языковой модели и модели ошибок, которая в свою очередь смотрит на заглавное/строчное написание слова, источник кандидата, присутствие в словаре и другие признаки. Итого, остаются лишь те варианты, которые не хуже лучшего в какое-то фиксированное число раз.

3. Ранжирование.

   Когда осталось не так много возможных предложений-кандидатов по ним вычисляются признаки, над которыми затем выполняется ранжирование при помощи логистической регрессии. Рассмотрим признаки, использованные в лучшей посылке:

   - количество слов в предложении;
   - вывод модели ошибок;
   - вывод языковой модели;
   - количество измененных слов;
   - количество несловарных слов;
   - количество исправлений в словарных словах;
   - количество исправлений на расстоянии Дамерау-Левенштейна, равном одному;
   - количество исправлений, полученных при помощи фонетического кодирования;

- количество исправлений, полученных при помощи вручную созданных соответствий;
- количество исправлений, вставляющих пробел в слово;
- количество исправлений, объединяющих слова;
- количество несловарных слов, которые можно разбить на словарные подстроки;
- взвешенное расстояние Дамерау-Левенштейна.

Стоит также описать какие техники не дали прироста в качестве. Так, для внесения в модель знания о морфологии были использованы вероятности из языковой модели, построенной на частях речи. Были также испытаны признаки, призванные охватить семантику, посчитанные на основе совстречаемостей словоформ, и признаки предлогов.

Глава 4

Описание моделей и методов

В этой главе будут описаны модели и методы, используемые в разработанной системе исправления опечаток. С исходным кодом можно ознакомиться в репозитории, посвященном бакалаврской работе: https://github.com/Mr-Geekman/bd-research.

4.1 Общая схема модели

Модель состоит из трех модулей:

1. модуль генерации кандидатов;
2. модуль выбора позиции;
3. модуль выбора исправления.

Опишем общий алгоритм и роль каждого модуля:

1. Подготовка:
   (a) Токенизация.
   (b) Нормализация.
   (c) Инициализация «черного списка» позиций.
   (d) Генерация кандидатов для каждого токена и списка признаков к нему при помощи соответствующего модуля.
2. Итерации исправления до срабатывания критерия останова или достижения максимального установленного числа итераций:
   (a) Поиск лучшей позиции для исправления и проверка критерия останова при помощи модуля выбора позиции.
   (b) Выбор лучшего исправления в выбранной позиции при помощи модуля выбора исправления.
   (c) Изменение текущего предложения в соответствии с предыдущим шагом.
   (d) Обновление «черного списка» позиций для следующей итерации.

Теперь разберем отдельные элементы.

4.1.1 Токенизация

На данный момент можно выделить три основных вида токенизации:

1. пословная — токенами являются слова;
2. посимвольная — токенами являются отдельные символы;
3. подсловная — токенами являются части слов.

Первый тип является самым базовым. Возможных токенов очень много, а потому если это множество как-то ограничивать, то проблемой может стать наличие несловарных слов.

Второй тип не имеет проблем с неизвестными токенами, но при этом весьма сложен в обработке, так как теперь токенов стало гораздо больше на один и тот же объем текста.

Последний вид является промежуточной формой между первыми двумя. Частые слова являются отдельными токенами, а несловарные можно получить из отдельных частей. Таким образом, удается комбинировать преимущества первых двух методик. К примерам алгоритмов, реализующих такой вид токенизации, можно отнести BPE [25] и WordPiece [7].

В рамках нашей задачи целью является исправление слов на словарные, а потому будем использовать первый тип. Кроме того, это позволит получать консистентные результаты с алгоритмом проверки, так как он тоже оперирует словами.

В качестве алгоритма токенизации была выбрана портированная на язык Python версия токенизатора Moses: https://github.com/alvations/sacremoses.

4.1.2 Нормализация

Нормализация предложений включала в себя две операции: приведение слов к нижнему регистру, замена буквы «ё» на букву «е».

Это не значит, что информация о регистре была окончательно потеряна. В модуль генерации кандидатов передается текст в оригинальном регистре. Тем не менее, все последующие модули работают с уже полностью нормализованным текстом.

Обработка пунктуации не производилась заранее, а отдавалась на откуп модулям.

4.1.3 «Черный список» позиций

Это список позиций, которые нельзя выбирать в качестве исправляемых. Он выполняет две функции:

1. Изначальная блокировка некоторых позиций.

   Было решено не исправлять слова, написанные с большой буквы или большими буквами, если они находятся не на первой позиции. Во-первых, таких слов

оказалось достаточно мало, а во-вторых, это часто имена собственные или аббревиатуры, про которые сложно сказать, правильно они пишутся или нет.

2. Временная блокировка некоторых позиций.

   Предположим, модуль выбора позиции выбрал некую позицию, а затем модуль выбора исправления решил оставить изначальное написание. Если не поместить выбранную позицию в «черный список», то на следующей итерации все повторится. В таком случае, будем помещать выбранную позицию в «черный список» до тех пор, пока в предложении не произойдет какое-то исправление, которое может изменить контекст и результат работы модулей.

4.2 Модуль генерации кандидатов

Рассмотрим подробнее модуль поиска кандидатов. Как было указано выше, он отвечает за поиск кандидатов для исправления и генерацию их признаков для каждого токена в предложении.

Общая схема поиска заимствована из статьи «Automatic spelling correction for Russian social media texts» [23]. Модуль состоит из трех подмодулей, которые находят кандидатов разными способами:

- на основе расстояния Дамерау-Левенштейна;
- на основе фонетического кодирования;
- на основе вручную закодированных соответствий.

Помимо этого, в списке кандидатов обязательно присутствует текущий токен, даже если он отсутствует в словаре.

Модулю необходим словарь для осуществления поиска. Было решено взять объединение словарей Хагена [17] и wiktionary [16] для русского языка, включающее около 2.36 млн. словоформ.

Рассмотрим каждый подмодуль подробнее.

4.2.1 Поиск на основе расстояния Дамеарау-Левенштейна

Модуль был заимствован из библиотеки DeepPavlov [26]. На основе заданного словаря строится ациклический конечный автомат при помощи минимизации префиксного бора. При получении запроса осуществляется алгоритм A-star [20] с использованием h_2-эвристики из статьи «Fast approximate string matching with finite automata» [19].

Для обработки ошибки удаления пробела (при исправлении необходимо вставить пробел), автомат дополняется ребрами из всех конечных состояний в начальное с меткой пробела. Это позволяет при получении словарного слова вставить пробел и продолжить поиск.

Итого, при выполнении поиска кандидатов для токена извлекаются все словарные слова и группы словарных слов, разделенных пробелами, на расстоянии Дамерау-Левенштейна, не превышающем некоторой фиксированной константы.

4.2.2 Поиск на основе фонетического кодирования

Алгоритмы фонетического кодирования — это алгоритмы, которые на основе правил произношения преобразуют слова в кодирующий их текст таким образом, что коды близких по произношению слов похожи или совпадают.

Первым представителем подобных алгоритмов является алгоритм SoundEx [27] [28]. Существует и множество других алгоритмов, некоторые из которых приспособлены для русского языка. Хороший обзор существующих методов представлен в статье «Обзор алгоритмов фонетического кодирования» [29].

В своей модели мы будем использовать алгоритм из лучшего решения SpellRuEval, так как в нем есть несколько особенностей, полезных для исправления опечаток. Каждой букве ставится в соответствие свой фонетический класс согласно таблице 4.1 (нумерация с пропусками кодов 2 и 4 сохранена из оригинальной статьи).

Таблица 4.1 – Соответствие букв и их фонетических кодов.

Код	**Буквы**	**Код**	**Буквы**	**Код**	**Буквы**
1	а, о, ы, у, я	8	г, к, х	13	з, с
3	и, е, ё, ю, я, э	9	л	14	й
5	б, п	10	р	15	щ, ч
6	в, ф	11	м	16	ж, ш
7	д, т	12	н	17	ц

Кроме того, есть несколько правил, согласно которым фонетический класс может меняться:

1. все гласные после *ъ, ѣ, щ, ч, й* переходят в класс 3;
2. все гласные после *ш, ж, ц* переходят в класс 1;
3. сочетания *тс, тьс, тѣс* переходят в класс 17;
4. выбрасывается *т* после сибилянтов и перед согласной (например, для обработки слова *грустный*);
5. в последовательностях из одинаковых классов оставляется лишь один элемент.

У полученного алгоритма есть несколько полезных свойств:

- опускается множественное повторение одной гласной, например, слова «ооочень» и «очень» получат один и тот же код;

- не различаются окончания на «ться», «тся», «цца», например слова «различаться», «различацца», «различатся» получат один и тот же код.

Итого, при выполнении поиска кандидатов для токена достаются все словарные слова, имеющие тот же фонетический код.

4.2.3 Поиск на основе вручную закодированных соответствий

Третий подмодуль для поиска представляет собой таблицу соответствий между токеном и его возможными заменами. Далее опишем процедуру сбора подобных соответствий.

Большая часть соответствий была собрана на основе данных из социальных сетей корпуса «Тайга» [2]. По токенизированным текстам были посчитаны частоты всех несловарных слов. Перебирая слова от самых частых к самым редким было просмотрено около 5000 слов, из которых были составлены 136 соответствий.

Посмотрим на некоторые примеры:

- *че* → *что*;
- *ваще* → *вообще*;
- *щас* → *сейчас*;
- *оч* → *очень*;
- *шоб* → *чтобы, чтоб, что бы, что б.*

Два соответствия были получены на основе обучающей выборки. Их особенность заключается в том, что это словарные слова:

- *тока* → *только*;
- *скока* → *сколько*;

Еще 10 паттернов были закодированы на основе опыта общения в социальных сетях. Примеры:

- *пж* → *пожалуйста*;
- *мб* → *может быть*;
- *норм* → *нормально.*

4.2.4 Объединение токенов

Как уже было описано, модуль умеет обрабатывать ошибки «один ко многим», когда один токен надо разбить на несколько, вставив пробел. Тем не менее, в обучающей выборке присутствуют и ошибки типа «многие к одному», когда надо объединить несколько подряд идущих токенов в один или вставить между ними дефис.

Для решения этой проблемы рассматриваются пары подряд идущих токенов, не являющихся знаками пунктуации. Происходит проверка находится ли в словаре результат удаления пробела между ними или замены его на дефис. Если ответ положительный, то рассматриваемая пара токенов становится отдельным токеном. Список его кандидатов состоит из:

- приписывания второго токена к элементам списка кандидатов для первого токена;
- приписывания первого токена к элементам списка кандидатов для второго токена;
- оригинальных токенов, записанных через пробел,
- объединения токенов с удалением пробела или вставкой дефиса (в зависимости от того, что нашлось в словаре).

Была также предпринята попытка заменить простую проверку удаления пробела или вставки дефиса на полноценный поиск на заданном расстоянии Дамерау-Левенштейна, но это привело лишь к резкому увеличению числа ложных срабатываний, которых и без того было очень много.

Можно также заметить, что этот механизм скорее всего непригоден для объединения трех и более слов, тем не менее, такой случай был всего один на всю обучающую выборку.

4.2.5 Генерация признаков

Как было описано выше, модуль генерации кандидатов помимо создания списка исправлений также вычисляет признаки для них, которые будут использоваться в следующих модулях. Перечислим генерируемые признаки:

- признаки позиции:
  - индикатор написания с заглавной буквы;
  - индикатор написания заглавными буквами;
  - индикатор написания строчными буквами;
  - индикатор первой позиции (чтобы можно было как-то отдельно обработать случаи регистров для нее).
- индикатор содержания пробела;

- индикатор содержания дефиса;
- индикатор нахождения при помощи расстояния Дамерау-Левенштейна;
- индикатор нахождения при помощи фонетического кодирования;
- индикатор нахождения при помощи таблицы соответствий;
- индикатор того, что кандидат является результатом комбинирования смежных токенов;
- индикатор того, что кандидат является изначальным токеном в данной позиции;
- индикатор того, что кандидат является текущим токеном в данной позиции (отличается от изначального, если в позиции произошло исправление);
- индикатор принадлежности словарю.

Отдельная сложность возникает при объединении токенов. В случае, когда токен дополняется кандидатами другого токена, брались признаки для изменяемого токена. Для объединенного токена сложность составляло лишь вычисление признаков регистра, но этот случай достаточно редкий.

4.3 Модуль выбора позиции

Рассмотрим модуль, предназначенный для выбора позиции исправления и проверки критерия останова.

В основу модуля положено языковое моделирование. Для каждой позиции можно посмотреть вероятности различных кандидатов в ней при проходе текста как слева направо, так и справа налево. Затем эти вероятности можно сагрегировать, получив для каждой позиции и каждого кандидата некоторое число, характеризующее насколько данный кандидат хорошо подходит в данной позиции. На основе этих данных можно сравнить насколько в каждой позиции лучший кандидат превосходит текущего и выбрать позицию, отсутствующую в «черном списке» с самым большим разрывом. Такой способ позволяет отдавать приоритет тем позициям, в которых замена может значительнее всего увеличить вероятность. Для проверки критерия останова достаточно посмотреть на полученный разрыв и сравнить его с зафиксированной константой.

Подход можно усложнить, обучив модель, учитывающую помимо вычисленной сагрегированной характеристики и другие параметры, характеризующие позицию. Эта попытка была предпринята, прочитать о ней можно в соответствующем разделе, посвященном экспериментам. Тем не менее, если не обучать модель и зафиксировать языковые модели, то описанный подход требует настроить только способ агрегации и константу критерия останова.

В качестве способа агрегации было решено взять среднее гармоническое. Пусть логарифм вероятности кандидата от левой языковой модели равна s_l, а от правой — s_r, тогда итоговый показатель:

$$s = \frac{2 s_l s_r}{s_l + s_r}.$$

К плюсам именно такого метода можно отнести чувствительность к минимальному значению. Если одна из вероятностей окажется сильно меньше другой, то среднее гармоническое отклонится в меньшую сторону сильнее, чем, например, среднее арифметическое. В то же время, реакция на меньшее значение достаточно сглаженная в отличие от функции минимума.

Также следует описать почему при сравнении позиций берется разность между показателями кандидатов. Дело в том, что на выходе языковых моделей мы получаем логарифмы вероятностей, а их разности характеризуют множитель между вероятностями. Было намерение отдавать приоритет тем позициям, в которых вероятность от исправления может повыситься в наибольшее число раз.

Может появиться мысль оценивать не вероятности кандидатов при условии контекста, а вероятности предложений целиком с подстановкой соответствующих кандидатов. Чтобы сравнить этот подход с описанным в самом начале, рассмотрим как их можно реализовывать на примере модели слева направо (для модели справа налево все аналогично). В обоих подходах достаточно проходиться по предложению слева направо и запоминать текущий левый контекст перед рассмотрением очередной позиции. В подходе из начала раздела достаточно перебрать всех кандидатов, а в подходе, работающим с предложениями целиком, потребуется для каждого кандидата посмотреть, как изменится состояние правой части предложения, что может занять гораздо больше времени. Именно поэтому было решено использовать подход, описанный в самом начале.

В качестве предобработки текста перед применением языковых моделей было решено убрать всю пунктуацию. Также следует прояснить, как обрабатывались кандидаты, состоящие из нескольких токенов. Такое могло произойти если в модуле генерации кандидатов сработал механизм объединения или подмодуль поиска на основе расстояния Дамерау-Левенштейна вставил пробел внутри слова. В таком случае, было решено считать логарифмом вероятности кандидата сумму логарифмов вероятностей отдельных слов внутри него.

После вычисления характеристик каждой позиции в список кандидатов, сгенерированных предыдущим модулем, записываются дополнительные признаки:

- логарифмы вероятностей от левой и правой языковых моделей;
- агрегированные логарифмы вероятностей (среднее гармоническое);
- разница агрегированных характеристик рассматриваемого кандидата и текущего кандидата в данной позиции.

4.3.1 Языковые модели

В этом подразделе будут описаны использованные языковые модели и процедура их обучения.

Рассматриваемый модуль имеет высокие требования к производительности, а потому было решено использовать n-граммные языковые модели, так как они достаточно быстрые в отличие от нейросетевых. Выбор пал на реализацию в библиотеке KenLM [30] [31], которая использует модифицированное сглаживание Кнезера-Нея [32].

Было решено обучать модели на текстах из корпуса «Тайга». Было отобрано 100 млн. предложений из разных источников. В процессе предобработки тексты переводились в нижний регистр, буква «ё» заменялась на «е», пунктуация удалялась. Было получено несколько версий моделей:

- 3-граммная модель;
- 3-граммная модель без данных из социальных сетей;
- 4-граммная модель.

4.4 Модуль выбора исправления

Опишем теперь модуль, отвечающий за выбор конкретного исправления в позиции.

4.4.1 Выбор исправления на основе BERT

Модель BERT была предложена в статье «BERT: Pre-training of deep bidirectional transformers for language understanding» [7]. Одной из задач, используемых для ее предобучения, является маскированное языковое моделирование (MLM от англ. masked language modeling), которое подразумевает предсказание токена не по одностороннему контексту, а по всему окружению. Достигается это за счет того, что модель по архитектуре является энкодером и обрабатывает все предложение целиком. Кажется вполне разумным рассмотреть задачу выбора исправления, как MLM-задачу. Достаточно выбирать самого вероятного кандидата из списка.

При попытке реализовать этот подход требуется решить, что делать в случае, если кандидат представляет из себя несколько токенов, так как токенизация для модели BERT (WordPiece-токенизация) работает на уровне частей слов. Было решено находить логарифм вероятности для каждого WordPiece-токена внутри кандидата, при этом задав остальным подтокенам значение <UNK> (неизвестный токен) и отключив для них механизм внимания. Таким образом, при вычислении вероятности конкретного WordPiece-токена модель будет знать только позиции других токенов, но не будет принимать во внимание их конкретные значения.

В качестве примера рассмотрим предложение «это поделанные документы», где будет осуществляться ранжирование кандидатов «поделанные» и «подделанные» для

второй позиции. Пример проиллюстрирован на рис. 4.1. На первом шаге происходит вычисление логарифмов вероятностей для первых подтокенов, а на втором — для вторых. Как видим, кандидат «подделанные» оказывается лучше, согласно модели.

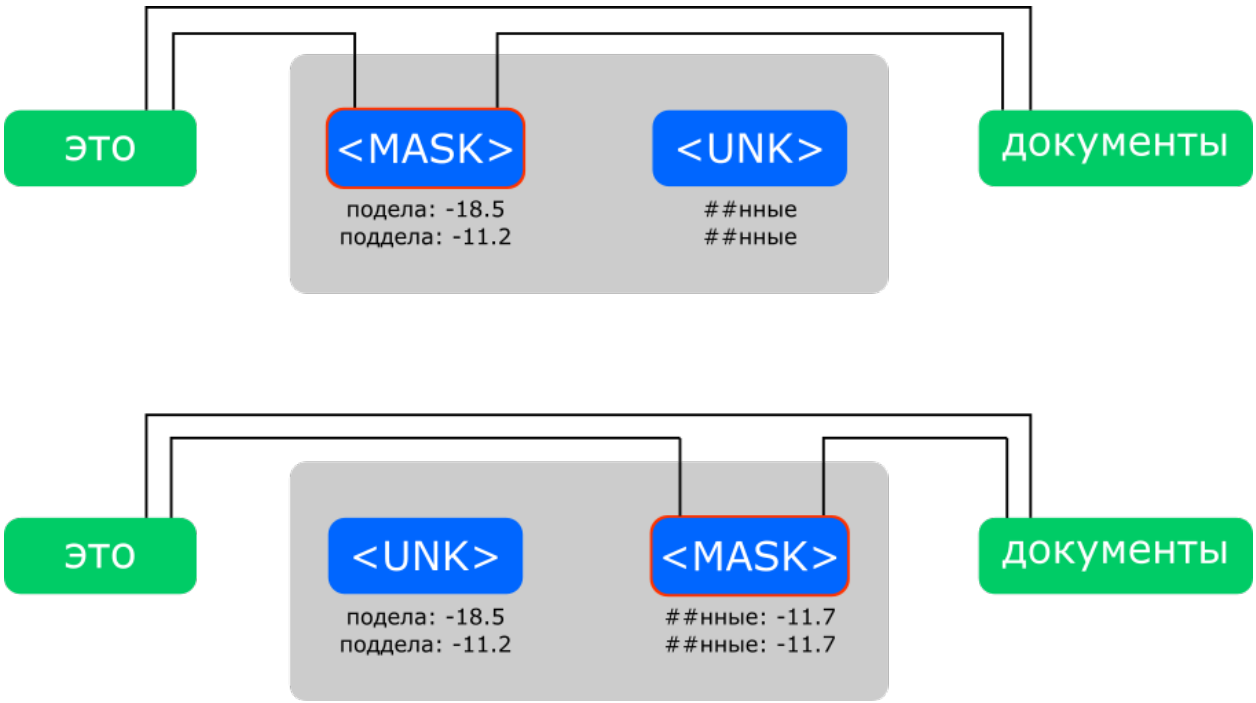

Рис. 4.1 – Иллюстрация работы ранжирования при помощи BERT.

Возможен и другой подход, когда механизм внимания не отключается для остальных токенов. Во-первых, это может усложнить параллелизацию. Во-вторых, это может привести к слишком сильному влиянию на вероятность других WordPiece-токенов кандидата, а требуется, чтобы кандидат выбирался исходя из окружения, а не частей самого себя.

Таким образом, для каждого кандидата в общем случае получается список логарифмов вероятностей его подтокенов. Для выбора лучшего кандидата требуется сагрегировать значения. Были попытки использовать для этого сумму или среднее.

В целях увеличения производительности был реализован способ группировки MLM-задач, чтобы за один запуск модели проверялось как можно больше кандидатов. Пусть имеется список из n токенизированных кандидатов одинаковой длины m: $[[s_{11}, s_{12}, \ldots, s_{1m}], \ldots, [s_{n1}, s_{n2}, \ldots, s_{nm}]]$, таких, что $s_{1i} = s_{2i} = \cdots = s_{ni}$. Тогда при запуске MLM-задачи для их i-го WordPiece-токена будет получаться одинаковый контекст, а значит достаточно будет выполнить запуск модели всего один раз и для каждого кандидата посмотреть на MLM-вероятность. Таким образом, можно группировать MLM-задачи для WordPiece-токенов кандидатов, если кандидаты состоят из одинакового количества подтокенов и все подтокены, кроме рассматриваемых в данный момент, совпадают. Но, как уже было описано выше, при запуске MLM-задачи всем токенам

кандидата кроме рассматриваемого в данный момент присваивается значение <UNK>, а значит для группировки MLM-задач достаточно лишь чтобы соответствующие кандидаты состояли из одинакового числа WordPiece-токенов. Такой подход позволяет для любого количества кандидатов длины m выполнить только m запусков модели.

В качестве предобученных моделей BERT для русского языка рассматривались RuBERT [33] и Conversational RuBERT, который представляет из себя дообученный на данных из Интернета RuBERT.

4.4.2 Выбор исправления на основе множества признаков

Оказалось, что предыдущий подход работает не очень хорошо. Точность была совсем невысокой, а значит модели было тяжело выбрать правильное исправление. Появилась мысль, что модели BERT не хватает знаний о том, как совершаются ошибки, то есть требовалась некая модель ошибок. Для этого было решено добавить больше признаков и обучить некую модель ранжирования, которая бы выбирала из списка кандидатов лучшего.

Опишем используемые признаки:

- признаки, вычисленные при генерации кандидатов;
- признаки, связанные с языковом моделированием (модуль выбора позиции);
- признаки BERT:
  - количество WordPiece-токенов;
  - сумма логарифмов вероятностей для WordPiece-токенов;
  - среднее логарифмов вероятностей для WordPiece-токенов.

Для сбора обучающей выборки была запущена модель, использующая только BERT. Правильные ответы для каждой задачи ранжирования были получены при помощи процедуры выравнивания, использующейся при измерении качества.

В качестве первой модели было решено взять ranking SVM [34], которая представляла из себя линейный SVM [35], обученный предсказывать, что один кандидат строго лучше другого на разностях их признаковых векторов. Один кандидат считался лучше другого если являлся правильным ответом. Использовалась реализация алгоритма в библиотеке scikit-learn [36].

В качестве второй модели был взят CatBoost [37], решающий задачу ранжирования. Этот алгоритм способен улавливать более сложные зависимости, чем SVM, но при этом достаточно хорошо работает при стандартных значениях гиперпараметров. Немного остановимся на принципах его работы. CatBoost — это алгоритм градиентного бустинга, способный качественно предобрабатывать категориальные признаки и использующий «забывчивые» (oblivious) деревья. Их особенность в том, что условие разделения в деревьях выбирается одно на весь уровень, см. рис. 4.2.

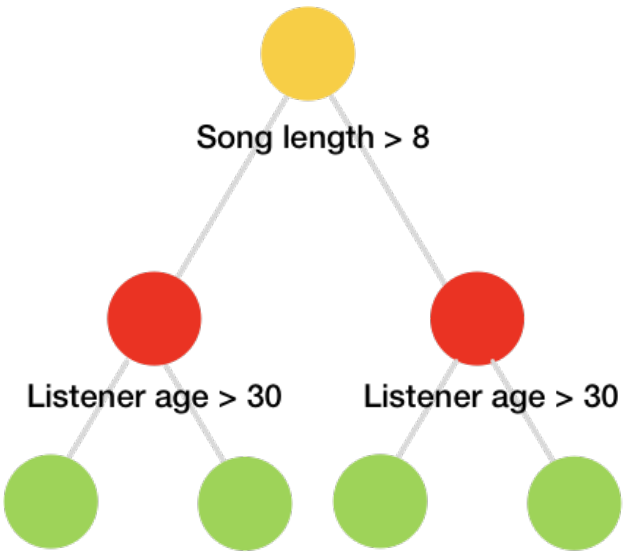

Рис. 4.2 – Иллюстрация «забывчивого» дерева [38].

В качестве отклика был взят индикатор того, что кандидат является верным ответом. Данные были разбиты на группы, где каждая из них отвечает за конкретную задачу ранжирования по кандидатам в определенной позиции. Были испытаны следующие функции потерь:

- `RMSE`.

  Решается задача регрессии по предсказанию отклика. Пусть имеются множество задач ранжирования T и множество кандидатов C. Тогда если рассматривается конкретная задача ранжирования t, то обозначим множество кандидатов, соответствующее ей, как C_t. Получим следующую функцию потерь:

  $$\frac{1}{\sum\limits_{t \in T} |C_t|} \sqrt{\sum_{t \in T} \sum_{c \in C_t} \left(\widehat{y}(c) - y(c)\right)^2},$$

  где $\widehat{y}(c)$ — предсказание модели, а $y(c)$ — истинный отклик.

- `QueryRMSE`.

  Выражение:

  $$\frac{1}{\sum\limits_{t \in T} |C_t|} \sqrt{\sum_{t \in T} \sum_{c \in C_t} \left(\widehat{y}(c) - y(c) - \frac{1}{|C_t|} \sum_{\tilde{c} \in C_t} \left(\widehat{y}(\tilde{c}) - y(\tilde{c})\right)\right)^2}.$$

  Такая функция позволяет не обращать внимания на смещение, так как при ранжировании нам важен порядок, а не абсолютные значения.

- `PairLogit`.

Эта функция потерь отвечает попарному подходу к решению задачи. Внутри каждой группы t генерируются все возможные пары P_t, в каждой из которых первый кандидат лучше, чем второй. Итоговая формула:

$$-\sum_{t \in T} \sum_{(c_p, c_n) \in P_t} \ln \left(\frac{1}{1 + \exp\{-(\widehat{y}(c_p) - \widehat{y}(c_n))\}} \right).$$

- `YetiRank`.

  Данная функция тоже реализует попарный подход. Подробнее о ней можно почитать в соответствующей статье: «Winning The Transfer Learning Track of Yahoo!'s Learning To Rank Challenge with YetiRank» [39].

Сведения о всех доступных функциях потерь размещены на странице документации: https://catboost.ai/docs/concepts/loss-functions-ranking.html.

Глава 5

Описание экспериментов и результатов

Глава посвящена проведенным экспериментам и их результатам.

5.1 Предварительные эксперименты

Для начала посмотрим на эксперименты, которые проводились до того, как тестировать все возможные версии моделей.

5.1.1 Работа модуля генерации кандидатов

Посмотрим на то, насколько хорошо работает модуль генерации кандидатов. При использовании модуля поиска на основе расстояния Дамерау-Левенштейна, максимальное расстояние было взято за единицу, ввиду резкого снижения производительности при увеличении данного значения. В таком случае, следует посмотреть насколько хорошо фонетическое кодирование и вручную заданные соответствия позволяют находить исправления на больших расстояниях. Результаты тестирования для ошибок вида «один к одному» см. в табл. 5.1.

Таблица 5.1 – Работа модуля поиска кандидатов.

Фон. код	Соотв.	PP	Кол. ошибок	Кол. нахождений	Доля успехов
+	-	⩾2	341	110	31.3%
+	-	2	230	65	28.3%
+	-	3	75	28	37.3%
+	-	4	26	15	57.75%
+	-	⩾5	10	2	20%
-	+	⩾2	341	99	29.0%
-	+	2	230	54	23.5%
-	+	3	75	31	41.3%
-	+	4	26	7	26.9%
-	+	⩾5	10	7	70%
+	+	⩾2	341	204	59.8%
+	+	2	230	115	50.0%
+	+	3	75	58	77.3%
+	+	4	26	22	84.6%
+	+	⩾5	10	9	90%

Как видим, результаты работы двух рассматриваемых модулей практически не пересекаются, т.е. один и тот же кандидат редко обнаруживается обоими модулями сразу.

Всего удается находить около 60% всех кандидатов на большом расстоянии Дамерау-Левенштейна, причем наибольшую трудность составляет расстояние, равное двум.

5.1.2 Работа механизма объединения

Аналогично, посмотрим насколько хорошо удается находить ошибки «многие к одному» при помощи механизма объединения последовательных токенов. Рассматривая только ошибки «два к одному» (ошибка другого типа была всего одна для обучающего датасета), получим:

- количество ошибок: 69;
- количество срабатываний: 507;
- количество верных срабатываний (true positive): 40
- точность: 7.9%;
- полнота: 58.0%;

Как видим, доля ложных срабатываний очень велика. Посмотрим на примеры, когда такое происходит (в скобках выделены токены, на которых сработало объединение):

- *В принципе, (я к) этому готова.*
- *Провожаем жизнь (мы с) тобой.*
- *Вы меня извините, (но я) опять про своих.*

Примеры демонстрируют, что токены могут объединяться совсем неподходящим образом.

Как было сказано при описании рассматриваемого механизма, была предпринята попытка объединения по расстоянию Дамерау-Левенштейна, но это лишь удвоило количество ложных срабатываний, почти не изменив TP.

5.1.3 Обучение с учителем для модуля выбора позиции

Как было описано в разделе про модель нахождения позиции, предпринималась попытка выбирать позиции не только на основе языковых моделей, но и при помощи других признаков.

Рассматриваемые признаки:

- признаки позиции из модуля генерации кандидатов.
  - индикатор написания с заглавной буквы;
  - индикатор написания заглавными буквами;

  - индикатор написания строчными буквами;
  - индикатор первой позиции;
- индикатор написания с заглавной буквы и не первой позиции;
- количество кандидатов;
- индикатор того, что текущий кандидат содержит дефис;
- признаки на основе вероятностей языковой модели для текущего кандидата:
  - логарифм вероятности для текущего кандидата от языковой модели «слева направо»;
  - логарифм вероятности для текущего кандидата от языковой модели «справа налево»;
  - сагрегированное значение от языковых моделей для текущего кандидата.
- признаки на основе источника кандидатов:
  - доля кандидатов, пришедших из поиска, на основе Дамерау-Левенштейна;
  - доля кандидатов, пришедших из поиска, на основе фонетического кодирования;
  - доля кандидатов, пришедших из поиска, на основе вручную заданных соответствий;
- признаки на основе доли более хороших кандидатов:
  - доля кандидатов, лучше текущего, на основе языковой модели «слева направо»;
  - доля кандидатов, лучше текущего, на основе языковой модели «справа налево»;
  - доля кандидатов, лучше текущего, на основе сагрегированного показателя от обеих языковых моделей.
- признаки на основе разницы между лучшим кандидатом и текущим:
  - разница между лучшим кандидатом и текущим согласно языковой модели «слева направо»;
  - разница между лучшим кандидатом и текущим согласно языковой модели «справа налево»;
  - разница между лучшим кандидатом и текущим согласно сагрегированному показателю от обеих языковых моделей (признак, который использовался «по-умолчанию» до обучения с учителем).

Решалась задача классификации позиции, как требующей исправления, выбиралась позиция с наибольшей предсказываемой вероятностью. В качестве метрики была выбрана обобщенная F-мера:

$$F_\beta = (1 + \beta^2)\frac{PR}{(\beta^2 P) + R},$$

где P — точность, R — полнота.

Было взято $\beta = 4$, чтобы отдать приоритет полноте, но при этом не классифицировать слишком много позиций, как требующие исправления.

Модель по-умолчанию — это классификация позиции, как требующей исправления, на основе сравнения признака «по-умолчанию» с некоторым фиксированным значением α. Посмотрим, как зависят различные метрики в зависимости от выбора α, на рис. 5.1.

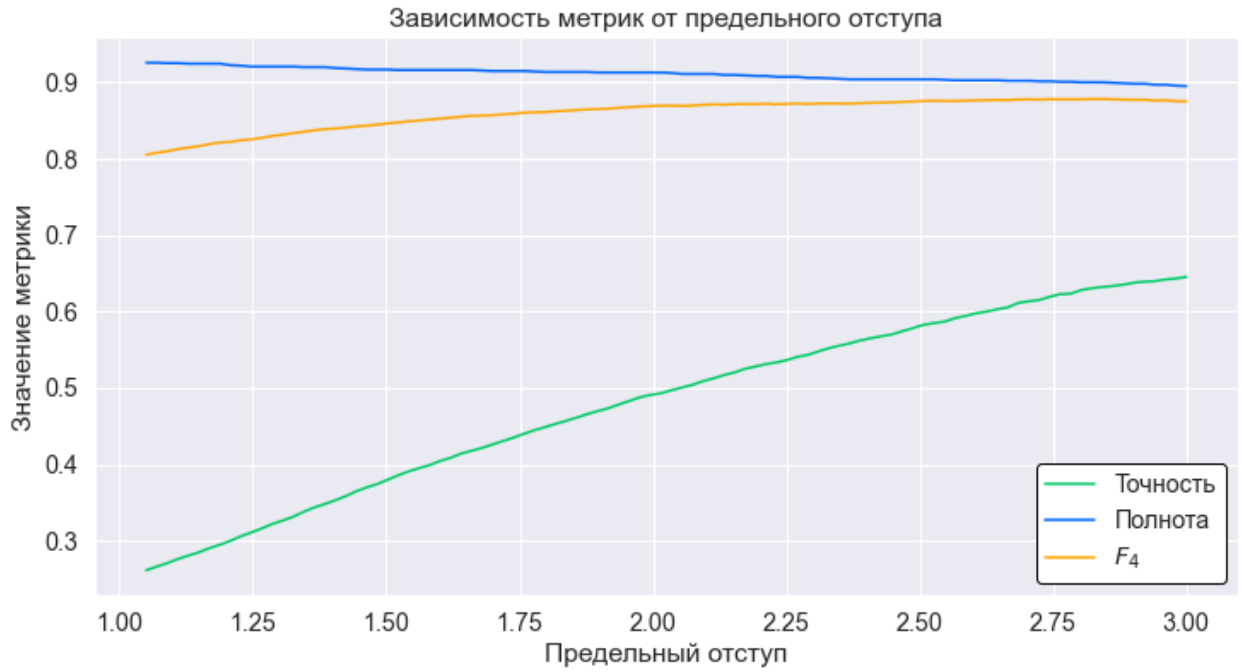

Рис. 5.1 – График зависимости метрик от предельного отступа

Итого, лучшее значение метрики достигаются при $\alpha = 2.84$. Аналогичным образом для сравнения была обучена модель, игнорирующая исправления в заглавных позициях кроме первой. Затем были протестированы несколько моделей, учитывающих все признаки. Лучше всего показала себя логистическая регрессия с использованием взвешивания классов, реализованная в библиотеке scikit-learn. Посмотрим на все полученные результаты в табл. 5.2.

Как видим, модель логистической регрессии, действительно, лучше, чем версия «по-умолчанию». Она не только находит больше ошибок, но и имеет более хорошую точность. Проверим дает ли это прирост качества в применении к самой задаче. Для

Таблица 5.2 – Сравнение версий модели выбора позиции исправления.

Модель выбора позиции	**Точность**	**Полнота**	F_4
По-умолчанию	63.2%	90.0%	87.8%
По-умолчанию + игнор.	65.0%	89.8%	87.8%
Лог. рег.	73.7%	95.6%	94.0%

этого измерим качество модели исправления опечаток с отключенным механизмом объединения токенов и ранжирующей моделью SVM. Результаты см. в табл. 5.3

Таблица 5.3 – Сравнение версий модели выбора позиции исправления в применении к исправлению опечаток.

Модель выбора позиции	**Точность**	**Полнота**	F_1
По-умолчанию	87.55	69.38	77.41
Лог. рег.	87.48	68.27	76.69

Как видим, прироста в качестве нет, хотя прирост на задаче обучения был значительный. Объяснить это можно тем, что хоть модель и находит больше позиций с исправлениями, выполнить исправление не удается. Вполне возможно, что нет подходящих кандидатов. Например, модель понимает, что какое-то слово очень маловероятно согласно языковой модели, а потому там скорее всего опечатка, но подходящего кандидата на исправление нет. Модель «по-умолчанию» такой проблемой не обладает, так как выбирает только те позиции, для которых есть хорошее исправление.

5.1.4 Эксперименты над моделью ранжирования

Рассмотрим, как производилось измерение качества для моделей ранжирования. В частности, покажем, как выбиралась функция потерь для модели на основе CatBoost.

Модели тестировались при помощи кросс-валидации по обучающим данным. В качестве целевой метрики была выбрана частота угадывания самого релевантного кандидата (такую метрику обычно обозначают как precision@1), так как по обучающим данным мы могли получить лишь одного правильного кандидата. Продемонстрируем результаты в табл. 5.4, полученные для случая, когда механизм объединения токенов был отключен.

Отсутствие модели ранжирования означает, что берется модель исправления на основе BERT, то есть выбирается кандидат с самым большим средним логарифмом вероятностей для WordPiece-токенов.

Как видим, добавление дополнительных признаков вместе с ранжирующей моделью увеличивает качество очень значительно. В дальнейшем, при обсуждении модели CatBoost подразумевается, что используется функция потерь `PairLogit`.

Таблица 5.4 – Сравнение различных функций потерь для модели ранжирования CatBoost.

Модель ранжирования	Точность
-	54.7%
SVM	90.3%
CatBoost (`RMSE`)	91.9%
CatBoost (`QueryRMSE`)	92.1%
CatBoost (`PairLogit`)	92.7%
CatBoost (`YetiRank`)	92.4%

5.2 Сравнение разных версий модели

Посмотрим какие результаты демонстрируют различные версии моделей: см. табл. 5.5. Колонка «BERT» обозначает были ли использованы признаки BERT, колонка «Объединение» отвечает был ли использован механизм объединения смежных токенов. Используемые гиперпараметры:

- версия BERT — ConversationalRuBERT;
- версия языковой модели — 3-граммная модель, обученная на всех данных;
- максимальное количество итераций — 5;
- минимальный отступ при выборе лучшей позиции — 2.5;

Таблица 5.5 – Сравнение результатов вариаций модели.

Ранжирование	BERT	Объединение	Точность	Полнота	F_1
-	+	-	48.76	57.89	52.94
SVM	-	-	86.52	67.26	75.68
SVM	-	+	86.48	70.24	77.52
SVM	+	-	87.55	69.38	77.41
SVM	+	+	87.47	72.42	79.24
CatBoost	-	-	86.14	67.00	75.38
CatBoost	-	+	86.04	70.80	77.68
CatBoost	+	-	87.34	69.13	77.18
CatBoost	+	+	**87.70**	**73.23**	**79.81**

Как видим, использование других признаков вместе с ранжирующей функцией сильно улучшает качество. Существенный выигрыш в качестве получается как от добавления механизма объединения, так и от использования признаков BERT. В модели, использующей все улучшения, по-видимому, нет существенной разницы, что использовать для ранжирования: SVM или CatBoost.

5.3 Эксперименты над гиперпараметрами лучшей версии модели

Рассмотрим, как различные гиперпараметры влияют на качество лучшей модели. Исследуемые параметры:

- Версия BERT:
  - RuBERT;
  - Conversational RuBERT.
- Версия языковой модели:
  - 3-граммная модель;
  - 3-граммная модель без данных из социальных сетей;
  - 4-граммная модель.
- Максимальное число итераций.
- Минимальный зазор между лучшим и текущим кандидатами в модели выбора позиции.

Результаты тестирования отражены в табл. 5.6. На первой строке указаны результаты модели по-умолчанию. Языковая модель с пометкой «3*» обозначает версию 3-граммной модели, обученную без данных из социальных сетей. Как видим, прироста в качестве изменения гиперпараметров не дают.

Таблица 5.6 – Сравнение результатов лучшей модели с различными гиперпараметрами.

BERT	**ЯМ**	**Макс. ит.**	**Зазор**	**Точность**	**Полнота**	$\boldsymbol{F_1}$
Conv. RuBERT	3	5	2.5	**87.70**	73.23	**79.81**
RuBERT	3	5	2.5	87.01	72.22	78.93
Conv. RuBERT	3	10	1.5	87.19	**73.38**	79.69
Conv. RuBERT	4	5	2.5	86.86	72.57	79.07
Conv. RuBERT	3*	5	2.5	86.99	70.70	78.00

5.4 Сравнение с существующими решениями

Посмотрим на то, как результаты лучшей модели соотносятся с уже существующими решениями: см. табл. 5.7. Под быстрой моделью в таблице подразумевается лучшая модель, не использующая BERT. Данные взяты со страницы: http://docs.deeppavlov.ai/en/master/features/models/spelling_correction.html [40]. Как видим, даже быстрая версия оказывается лучше других решений.

Таблица 5.7 – Сравнение результатов готовых моделей или сервисов.

Модель	**Точность**	**Полнота**	$\boldsymbol{F_1}$
Яндекс.Спеллер	83.09	59.86	69.59
DeepPavlov	59.38	53.44	56.25
SpellRuEval	81.98	69.25	75.07
Лучшая версия	**87.70**	**73.23**	**79.81**
Быстрая версия	86.04	70.80	77.68

5.5 Недостатки модели

Стоит отметить, что построенное решение имеет ряд недостатков:

1. Низкая скорость работы.

   На процессоре Intel Core i5-3210M, имеющем 2 ядра и 4 потока, версия, использующая BERT способна обрабатывать лишь около одного предложения за две секунды, что гораздо дольше, чем стандартные существующие решения. Около 80% процессорного времени расходуется именно на BERT, а потому в быстрой версии скорость обработки удалось увеличить до примерно 4.5 предложений в секунду, что все равно достаточно мало. Тем не менее, возможно потенциальное ускорение с улучшением параллельной обработки предложений.

2. Большое потребление памяти.

   Модель потребляет порядка 7.5ГБ оперативной памяти. Большая часть уходит на хранение префиксного бора, позволяющего осуществлять поиск на заданном расстоянии Дамерау-Левенштейна. Тем не менее, стоит отметить, что имплементация этого модуля выполнена на языке Python, что открывает широкие возможности для оптимизации.

Глава 6

Выводы и перспективы

Подведем итог проведенного исследования. Была построена модель, которая демонстрирует более высокое качество, чем лучшее решение соревнования SpellRuEval. Несмотря на высокие требования к объему оперативной памяти и вычислительным ресурсам можно выделить несколько применений:

- Создание веб-сервиса с API для исправления опечаток, как это сделано в Яндекс.Спеллере. В таком случае клиент не должен будет беспокоиться о требованиях к аппаратной части;
- Использование модели для предварительной обработки текстов перед применением других методов, чувствительных к опечаткам. В таком случае временные затраты могут не быть столь критичными.

Выделим также направления для будущих исследований, которые кажутся перспективными:

- сбор более крупного и современного датасета для задачи (Интернет-сленг достаточно быстро меняется);
- использование для ранжирования GPT-3 [41], в связи с появлением соответствующей версии модели для русского языка;
- дообучение модели BERT, используемой для ранжирования;
- увеличение производительности и уменьшение расхода памяти;
- увеличение предельного расстояния Дамерау-Левенштейна до двух.

Литература

1. Шаврина Т. О., Сорокин А. А. Моделирование расширенной лемматизации для русского языка на основе морфологического парсера TnT-Russian. — 2015.

2. Шаврина Т. О., Шаповалова О. К методике создания корпусов для машинного обучения: «Тайга», синтаксический корпус и парсер // По материалам Международной Конференции «Корпусная Лингвистика – 2017». — 2017. — С. 78–84.

3. Панина М. Ф., Байтин А. В., Галинская И. Е. Автоматическое исправление опечаток в поисковых запросах без учета контекста // Компьютерная Лингвистика И Интеллектуальные Технологии: По Материалам Ежегодной Международной Конференции «Диалог». — 2013. — № 12(19). — С. 556–567.

4. Байтин А. Исправление поисковых запросов в Яндексе // Российские Интернет Технологии. — 2008.

5. Riordan Brian, Flor Michael, Pugh Robert. How to account for mispellings: Quantifying the benefit of character representations in neural content scoring models. — 2019. — P. 116–126.

6. Kumar Ankit, Makhija Piyush, Gupta Anuj. Noisy Text Data: Achilles' Heel of BERT. — 2020. — P. 16–21. — 2003.12932.

7. BERT: Pre-training of deep bidirectional transformers for language understanding / Jacob Devlin, Ming Wei Chang, Kenton Lee, Kristina Toutanova // NAACL HLT 2019 - 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies - Proceedings of the Conference. — 2019. — Vol. 1, no. Mlm. — P. 4171–4186. — 1810.04805.

8. Азимов Э. Г., Щукин А. Н. Новый словарь методических терминов и понятий (теория и практика обучения языкам). — М. : ИКАР, 2009. — С. 53.

9. SpellRuEval: The first competition on automatic spelling correction for Russian / А. А. Сорокин, А. В. Байтин, И. Е. Галинская и др. // Компьютерная Лингвистика и Интеллектуальные Технологии. — 2016. — С. 660–673.

10. Damerau F. A technique for computer detection and correction of spelling errors // Communications of the ACM. — 1964. — Vol. 7, no. 3. — P. 171–176.

11. Левенштейн В. И. Двоичные коды с исправлением выпадений, вставок и замещений символов // Докл. АН СССР. — 1965. — Т. 163, № 4. — С. 845–848.

12. Mays Eric, Damerau Fred J, Mercer Robert L. Context based spelling correction // Information Processing & Management. — 1991. — Vol. 27, no. 5. — P. 517–522.

13. Kernighan Mark D., Church Kenneth W., Gale William A. A spelling correction program based on a noisy channel model. — 1990. — P. 205–210. — Access mode: https://doi.org/10.3115/997939.997975.

14. Dempster A. P., Laird N. M., Rubin D. B. Maximum Likelihood from Incomplete Data Via the EM Algorithm // Journal of the Royal Statistical Society: Series B (Methodological). — 1977. — Vol. 39, no. 1. — P. 1–22. — Access mode: http://doi.wiley.com/10.1111/j.2517-6161.1977.tb01600.x.

15. Brill Eric, Moore Robert C. An improved error model for noisy channel spelling correction. — 2000. — no. Kukich 1992. — P. 286–293.

16. Викисловарь. — Режим доступа: https://ru.wiktionary.org.

17. Хаген М. Полная парадигма Русского языка. — Режим доступа: http://www.lingvodics.com/dics/details/4309/.

18. Oflazer Kemal. Error-tolerant Finite-state Recognition with Applications to Morphological Analysis and Spelling Correction // Computational Linguistics. — 1996. — Vol. 22, no. 1. — P. 69–89. — 9504031.

19. Hulden Mans. Fast approximate string matching with finite automata // Procesamiento del Lenguaje Natural. — 2009. — Vol. 43, no. 43. — P. 57–64.

20. Hart Peter, Nilsson Nils, Raphael Bertram. A Formal Basis for the Heuristic Determination of Minimum Cost Paths // IEEE Transactions on Systems Science and Cybernetics. — 1968. — Vol. 4, no. 2. — P. 100–107. — Access mode: http://ieeexplore.ieee.org/document/4082128/.

21. Golding Andrew R., Schabes Yves. Combining Trigram-based and feature-based methods for context-sensitive spelling correction. — 1996. — P. 71–78. — Access mode: https://doi.org/10.3115/981863.981873.

22. Flor Michael. Four types of context for automatic spelling correction // TAL Traitement Automatique des Langues. — 2012. — Vol. 53, no. 3. — P. 61–99.

23. Сорокин А. А., Шаврина Т. О. Автоматическое исправление опечаток и орфографических ошибок для русскоязычных социальных медиа // Компьютерная Лингвистика и Интеллектуальные Технологии. — 2016. — С. 688–701.

24. Philips Lawrence. The double metaphone search algorithm // C/C++ users journal. — 2000. — Vol. 18, no. 6. — P. 38–43.

25. Gage Philip. A New Algorithm for Data Compression // The C Users Journal. — 1994. — P. 1–14. — Access mode: https://www.derczynski.com/papers/archive/BPE{_}Gage.pdf.

26. DeepPavlov: Open-Source library for dialogue systems / Mikhail Burtsev, Alexander Seliverstov, Rafael Airapetyan et al. // ACL 2018 - 56th Annual Meeting of the Association for Computational Linguistics, Proceedings of System Demonstrations. — 2015. — P. 122–127.

27. Russell R. C. U.S. Patent 1 261 167A, Index. — 1917.

28. Russell R. C. U.S. Patent 1 435 663, Index. — 1921.

29. Выхованец В. С., Цзяньмин Ду, Сакулин С. А. Обзор алгоритмов фонетического кодирования // Управление большими системами: сборник трудов. — 2018. — № 73.

30. Heafield Kenneth. KenLM : Faster and Smaller Language Model Queries // Proceedings of the Sixth Workshop on Statistical Machine Translation. — 2011. — no. 2009. — P. 187–197. — Access mode: http://www.aclweb.org/anthology/W11-2123{%}5Cnhttp://kheafield.com/code/kenlm.

31. Scalable modified Kneser-Ney language model estimation / Kenneth Heafield, Ivan Pouzyrevsky, Jonathan H. Clark, Philipp Koehn // ACL 2013 - 51st Annual Meeting of the Association for Computational Linguistics, Proceedings of the Conference. — 2013. — Vol. 2. — P. 690–696.

32. Chen Stanley F., Goodman Joshua. An empirical study of smoothing techniques for language modeling // Proceedings of the 34th annual meeting on Association for Computational Linguistics -. — Vol. 213. — Morristown, NJ, USA : Association for Computational Linguistics, 1996. — P. 310–318. — Access mode: http://portal.acm.org/citation.cfm?doid=981863.981904.

33. Kuratov Yuri, Arkhipov Mikhail. Adaptation of deep bidirectional multilingual transformers for Russian language // arXiv. — 2019. — 1905.07213.

34. Joachims Thorsten. Optimizing search engines using clickthrough data // Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. — 2002. — P. 133–142.

35. Boser Bernhard E., Vapnik Vladimir N., Guyon Isabelle M. Training Algorithm Margin for Optimal Classifiers // Perception. — 1992. — P. 144–152.

36. Scikit-learn: Machine Learning in Python / Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort et al. // Environmental Health Perspectives. — 2012. — jan. — Vol. 127, no. 9. — P. 097008. — 1201.0490.

37. Dorogush Anna Veronika, Ershov Vasily, Gulin Andrey. CatBoost: Gradient boosting with categorical features support // arXiv. — 2018. — P. 1–7. — 1810.11363.

38. CatBoost is a high-performance open source library for gradient boosting on decision trees. — Access mode: https://catboost.ai/news/catboost-enables-fast-gradient-boosting-on-decision-trees-using-gpus.

39. Gulin Andrey, Kuralenok Igor, Pavlov Dmitry. Winning The Transfer Learning Track of Yahoo!'s Learning To Rank Challenge with YetiRank // JMLR Workshop. — 2011. — Vol. 14. — P. 63–76. — Access mode: http://download.yandex.vibrow.com/company/to{_}rank{_}challenge{_}with{_}yetirank.pdf.

40. Automatic spelling correction pipelines. — Access mode: http://docs.deeppavlov.ai/en/master/features/models/spelling_correction.html.

41. Language models are few-shot learners / Tom B. Brown, Benjamin Mann, Nick Ryder et al. // arXiv. — 2020. — 2005.14165.