Министерство образования и науки Российской Федерации
Государственное образовательное учреждение высшего
профессионального образования
«Московский физико-технический институт (государственный университет)»

Факультет инноваций и высоких технологий
Кафедра «Анализ данных»

Выпускная квалификационная работа бакалавра
по направлению Прикладные математика и физика

«Использование машинного обучения для исправления опечаток сегментации»

Выполнила:
студентка 4 курса 191 группы
Тигунова Анна Сергеевна

Научный руководитель:
д.ф-м.н. *Бунина Е.И*
Фролов А.В.

г. Москва, 2015

Содержание

Аннотация

Исправление опечаток в поисковых запросах является приоритетной задачей, потому что это позволяет улучшить качество выдачи и помогает пользователям лучше сформулировать то, что они хотят найти. Однако существует много особенностей, которые не позволяют решать эту задачу с помощью традиционных средств обработки текстов. Например, в запросе обычно мало слов, и они слабо связаны между собой. Среди видов опечаток выделяется класс ошибок сегментации, когда пользователь ошибся при расставлении пробелов. Эти опечатки занимают второе место по частоте встречаемости. Несмотря на это, не так много научных работ посвящено именно этому классу опечаток. Недостатком классического подхода, описанного в этих работах, является то, что исправление происходит только на основе статистической вероятности слова, что не всегда является наилучшим. В данной работе представлен новый подход к исправлению опечаток типа сегментации с применением генератора возможных разбиений и алгоритма ранжирования, который выбирает лучшее разбиение запроса. Новизна подхода заключается в том, что в предыдущих исследованиях не использовался алгоритм ранжирования, который работает на основе признаов, характерных именно для опечаток типа склейки-разрезания. Для разработанного метода были проведены эксперименты, которые показали значительный прирост качества исправлений.

Ключевые слова: *исправление опечаток, сегментация, создание признаков, поисковые запросы, машинное обучение, ранжирование*

1 Вступление

Исправление опечаток является важным и приоритетным вопросом из области обработки естественного языка. Существует много алгоритмов, которые занимаются определенными частными случаями, будь то проверка орфографии в длинных художественных текстах, или обработка почтовых сообщений.

Особое место занимает задача исправления опечаток в поисковых запросах. Эта задача гораздо труднее, чем исправление опечаток в обычных текстах, по целому ряду причин.

Во-первых, опечатки встречаются гораздо чаще и являются более трудно исправляемыми, чем в обычных текстах, в потоке запросов к поисковой системе содержится около 10-12% опечаток [2]. Во-вторых, запросы короткие, а это значит, что использовать контекстные признаки в них становится сложнее. Еще одна проблема связана с тем, что часто слова в запросе не связаны грамматически или даже по смыслу. Это, например, может быть какой-то список понятий, которые пользователь вводит в надежде найти хотя бы одно, в то время как в обычных текстах такая последовательность слов никогда не может встретиться. Также очень часто происходит использование слов, которые не встречаются в литературном языке. Не смотря на это, они должны считаться валидными запросами, потому что пользователь именно их и имел в ввиду. Примерами могут служить разговорные слова, просторечия а также неологизмы, которых очень много в поисковых логах, потому что новые термины и понятия очень широко начинают использоваться именно в интернете. [2] Кроме этого, сложно разобраться с запросами, если они содержат названия организаций, сайтов или URL.

В статье [4] проведен достаточно интересный обзор статистики по опечаткам в поисковых запросах. Авторы выяснили, что в среднем на запрос приходится 3 слова, а около 20% запросов являются однословными. Еще более поражает их рассчет процента терминов, которых нет в стандартном словаре. По статистике, приведеенной в статье 73% уникальных слов из запросов не встречаются в словаре, а всего они составляют 20% от потока. Это ещё раз доказывает несостоятельность подходов, основанных только на использовании словарей. Более того, в статье [4] проведена классификация терминов, которые затрудняют исправление опечаток в запросах. На первых местах стоят неологизмы, термины используемые в интернете, названия компаний и продуктов, имена собственные и научные термины.

Исправление опечаток в поисковых запросах имеет много важных преимуществ. Перечислим некоторые из них. Как было экспериментально показано в исследовании [3], исправленные опечатки в запросах помогают в разы увеличить релевантную запросу выдачу. Также исправленные опечатки помогают сократить время пользователя на подбор подходящего запроса, чтобы его результат соответствовал цели поиска. Кроме этого, автоматически исправленный запрос позволяет не перепечатывать его заново, что сильно помогает и экономит время.

Кроме того исправление опечаток важно и в других сферах. Например, авторы [5] замечают, что исправление опечаток применимо при распознавании рукописных текстов и определении надписей на изображениях, так как опечатки сильно могут ударить по качеству определения символов.

Из всего сказанного выше следует, что опечатки в поисковых запросах обладают своей богатой спецификой. В связи с этим, хочется провести какую-то классификацию этих опечаток, чтобы в дальнейшем более детально подойти к их исправлению. Таких классификаций может быть несколько, разные авторы по-разному подходят к этому вопросу. Приведем несколько примеров. В работе [2] указана классификация на типографические ошибки (когда человек набрал слово с ошибкой) и ошибки замены слова (их также можно назвать лексическими, примером может послужить употребление слова в неправильном контексте). В свою очередь типографические ошибки можно разделить [2] на следующие виды:

1. пользователь нажал не на ту клавишу на клавиатуре (очки → лчки)
2. фонетические ошибки (вследствие → вследствии)
3. ошибки написания — орфограммы (куча → кучя)

В то время как с типографическими ошибками достаточно успешно борются, достаточно сложно правильно предложить исправления для ошибок типа замены слова.

Еще один вид классификации предолжили авторы [4], где они делят слова на фонетические ошибки и ошибки связанные с омонимами. Также они указывают, что ошибки могут отличаться на основе того, является ли опечатка валидным словом или такого слова не существует вообще.

В своей работе мы будем пользоваться несколько другой классификацией. Все ошибки в поисковых запросах мы будем делить на следующие типы:

1. ошибки орфографии (пчила)
2. ошибки сегментации (настол)
3. раскладка клавиатуры (vjkjrj)
4. транслитерация (купить донаты)

Как мы далее увидим, данная классификация очень неравномерна, однако она дает очень четкое качественное разделение видов опечаток, потому что в ее основу положены признаки, которые очень существенны при подборе подходов для каждого вида.

Классическим подходом к исправлению опечаток в поисковых запросах является модель канала с шумом, которая описана в работе [3]. Получив на входе запрос Q,

наша задача — найти для него наилучшее исправление C^*, среди всех возможных кандидатов в исправление

$$C^* = \underset{c}{argmax} P(C|Q) \tag{1}$$

Если мы применим правило Байеса, то получим следующую формулу для наилучшего исправления

$$C^* = \underset{c}{argmax} P(Q|C)P(C) \tag{2}$$

В данном представлении множители имеют простую интерпретацию. P(C) — языковая модель, а P(Q|C) — модель канала с шумом

В данной работе мы будем изучать опечатки типа сегментации. Мы тоже будем использовать популярную модель канала с шумом, а также алгоритмы машинного обучения. Мы применим последовательно эти два подхода для генерации 30 лучших вариантов исправления опечатки и выбор наилучшего варианта путем переранжирования. Хотя такая модель была раньше рассмотрена другими авторами, в данной работе мы предъявляем конкретный алгоритм для эффективной генерации кандидатов исправлений и создания признаков. Также мы предложим конкретные признаки, которые можно использовать именно для опечаток типа сегментации. Разработанный нами подход был реализован и протестирован, было получено значительное улучшение качества, по сравнению со стандартным подходом.

Структура данной работы такова: вначале сформулируем задачу исправления опечаток типа сегментации. Затем проведем обзор релевантных работ. В следующей секции опишем классический подход к исправлению ошибок сегментации и какой подход предлагается в данной работе. После этого речь пойдёт о проведенных на предложеной модели экспериментах, и в заключение предложим возможные пути развития.

2 Постановка задачи

Сначала опишем общую постановку задачи исправления опечаток в поисковом запросе.

Имеется некоторая фраза (query), которую пользователь вводит в поисковую систему. В этой фразе может содержаться опечатка. При этом пользователь имел в виду эталон — правильное написание запроса (reference). Наша задача состоит в том, чтобы предложить корректное исправление опечатки в запросе (correction), которое может совпадать с оригинальным запросом, если мы считем, что в нем не было ошибок.

Для того чтобы подобрать валидное исправление, в исходной постановке задачи, предложенной Дамерау [11], необходимо найти вариант, который будет максимально близок к запрашиваемому слову в смысле какой-либо строковой метрики $mindist(Q, C)$

Однако такой подход не учитывает, что хочется найти наиболее вероятное исправление, так как самое близкое слово может оказаться совсем не тем, что задумал ползователь. Поэтому переформулируем проблему: $dist(Q, C) < \delta$ and $P(c) \to max$. Таким образом, мы максимизируем вероятность исправления, зафиксировав допустимое расстояния между запросом и исправлением. Для нахождения оценки вероятности кандидатов в исправления используется метод максимального правдоподобия. Таким образом мы полуаем баланс между близостью исправления к оригиналу и его большой вероятностью.

Тем не менее, здесь используется только априорная вероятность исправления, которая не связана с тем, что мы хотели исправить конкретно Q. Это приводит нас к модели канала с шумом. Итак, будем искать исправление для которого мы имеем максимум апостериорной вероятности

$$C = \underset{c}{argmax} P(C|Q) \tag{3}$$

Если переписать это с помощью формулы Байеса, получим:

$$C = \underset{c}{argmax} \frac{P(Q|C)P(C)}{P(Q)} = \underset{c}{argmax} P(Q|C)P(C) \tag{4}$$

Множители в этой формуле означают:

P(C) - языковая модель, моделирует вероятность появления в языке фразы C

P(Q|C) - модель канала с шумом, моделирует вероятность того, что пользователь, задумав ввести Q, вводит C

Таким образом получаем формулу, на основе которой можно оценивать кандидатов в исправления. На практике обычно используют не вероятности, а «веса». Чтобы

перейти к весам, воспользуемся формулой

$$w = -log(P) \tag{5}$$

Это гораздо удобнее, потому что произведение вероятностей превращается в сумму. С весами также удобно работать и по другим причинам. Они достаточно наглядно интерпретируются: наилучшее исправление будет иметь минимальный вес; $w(Q|C)$ можно считать штрафом за опечатку. В дальнейшем, когда мы будем применять графическую модель, веса слов превратятся в веса ребер и таким образом нужно будет решать обычную задачу поиска минимального пути в графе

Во вступлении мы кратко описали классификацию опечаток, которую мы используем, упомянув, что получившиеся классы далеко не равнозначны. В самом деле, в то время как ошибок орфографии 66,7% от всего количества опечаток, мы имеем 13.7% ошибок сегментации, 9.1% раскладки клавиатуры, 2.1% ошибки транслитерации и, наконец, 8.4% занимают смешанные и прочие опечатки. [2] Видно, что исправление ошибок орфографии выглядит более перспективным. Например, если пересчитать число опечаток типа сегментации относительно всех поисковых запросов в целом, получим, что они составляют только 1.6%. Поэтому кажется, что даже если мы научимся идеально исправлять все ошибки склейки-разрезания, это не принесет много пользы.

Действительно, согласно проведенному нами эксперименту, если измерить полноту при идеальноv исправлении ошибок типа сегментации, получаем прирост полноты в 5%. В то же время, идеальное исправление ошибок орографии дает прирост полноты в 19%, что почти в 4 раза больше и уже выглядит значительным улучшением.

Несмотря на это, существуют вполне определенные причины, почему необходимо заниматься именно опечатками типа сегментации. Во-первых, они занимают второе место по частоте встречаемости, и хотя они сильно проигрывают орфографическим ошибкам, если мы не будем их учитывать, то тоже значительно ухудшим качество. Во-вторых, ошибки орфографии уже очень хорошо изучены. Над ними очень много работали и выработали оптимальные подходы к их исправлению. В настоящий момент существуют алгоритмы машинного обучения, которые с достаточно высоким качеством справляются с этими опечатками. Также придумано много признаков для обучения алгоритмов. Исправлению этого вида опечаток посвящены многочисленные статьи [1, 4–6] С другой стороны, опечатки типа сегментации достаточно плохо изучены. Это связано с их спецификой, потому что к ним невозможно применить те подходы, которые используются для исправления других типов опечаток. Главное отличие состоит в том, что здесь разбиение запроса на слова может меняться, и поэтому мы уже не можем говорить о каких-то конкретных признаках слова, так как даже количество слов в запросе и исправлении может отличаться.

Поэтому так важно заняться изучением именно опечаток типа склейки-разрезания,

так как такие задачи, как оптимальный выбор кандидатов для исправления и создание признаков для машинного обучения до сих пор эффективно не решены.

3 Обзор литературы

Исправление опечаток является очень приоритетной задачей и ей посвящено много научных работ. В этом разделе мы приведем краткий обзор основных подходов, которые были использованы для решения этой задачи. Будем рассматривать исследования в их исторической последовательности.

Одними из самых первых и простых были подходы, основанные на словарных методах [11]. В упомянутой статье ошибками считаются слова, которых нет в словаре, а чтобы их исправить, необходимо сделать одну из операций: вставка, удаление или замена символа. Подобный процесс достаточно часто встречается в литературе и также описан, например, в [4]. Это достаточно интуитивный подход, его подкрепляет что, по статистике приведенной [1] 80% опечаток могут быть исправлены путем только одной из этих операций. Последующие работы были основаны на этом подходе, вводя также некоторые улучшения. Следующим таким улушением было рассматривать дистанцию редактирования (edit distance) больше чем из одной операции [12]. Эта идея была развита дальше, когда было предложено не присваивать одинаковые веса всем вариантам редактирования, а индивидуально подходить к каждой такой операции [4]. Таким образом, мы подходим к вероятностной модели генерации слова с опечаткой и, в частности, модели канала с шумом.

Алгоритм исправления опечаток, который предлагается в большинстве статей состоит из двух этапов: генерация кандидатов для исправлений и их последующее переранжирование для поиска лучшего исправления.

В основном на первом этапе используется модель канала с шумом для выбора исправления с наибольшей вероятностью по языковой модели. Этому вопросу посвящены следующие работы [4,13] Впервые такой поход был предложен в [4]. Эта статья описывает программу, в которой канал с шумом используется не для генерации исправлений, а для выбора одного наилучшего. На первом этапе авторы генерируют исправления путем подбора слов из данных с помощью операций, описанных Дамерау плюс инверсия двух символов. Интересно, что в этом исследовании используются 4 матрицы переходных вероятностей моделирующие канал с шумом (отдельно для случаев вставки, удаления, замены и инверсии). Однако, нельзя сказать, что авторы добились очень хороших результатов с использованием описанной ими программы исправления опечаток, потому что их исследование было проведено на очень ограниченном корпусе.

Две интересных статьи [5,6] описывают как можно пользоваться тем же самым каналом с шумом, но обучаясь не на размеченных парах опечатка-эталон (query-reference), а непосредственно на самих логах запросов к поисковой системе.

В работе [4] подробно описывается алгоритм максимизации правдоподобия, с помощью которого автроы находят исправление с наибольшей вероятностью. Для того, чтобы обучить свою модель, они создают таблицу частых замен, которая заполняется на основе вероятности какой-либо операции редактирования. В статье была

приведена такая таблица, из которой авторы сделали вывод, что распределение самых частотных замен символов совпадает с тем, которое можно получить на основе размеченных пар опечаток-эталонов.

Статья [5] в свою очередь относится к следующему подходу к выбору наилучшего исправления — использованию машинного обучения для ранжирования кандидатов на основе признаков. В своей статье авторы [5] делают упор на то, что их подход не зависит от данных размеченных вручную. Они используют информацию непосредственно из инернета чтобы построить языковую модель а также модель ошибок. Такое решение проблемы хорошо, потому что в его основе нет каких-то фиксированных списков правильных слов, обучение происходит на осное статистики. В качестве текста для обучения классификаторов обнаружения опечаток и автозамен используются новости, так как авторы предполагают, что там содержится мало опечаток. Важным итогом использования такой модели является то, что она не зависит от какого-то конкретного языка. В данной работе очень подробно и качественно описано, как можно создать модель ошибок используя только данные из интернета. Таким образом получаются те же самые пары опечатка-эталон, только полученные без помощи ручной аннотации. Также новизна в решении авторов заключается в том, что они вводят штраф для языковой модели, который отражает уровень доверия к ней. Ранжирование кандидатов состоит из 3х этапов: вычисление исправления с наибольшим весом по языковой модели; определение, содержалась ли опечатка в исходном запросе и, наконец, предсказание, действительно ли стоит заменять запрос на наилучший предложенный вариант. Для того, чтобы тренировать свои классификаторы, авторы [5] используют опечатки полученые искуственно путем порчи новостных текстов которые предполагаются грамотными. В результате экспериментов было выявленно, что применение языковой модели улучшает качество исправлений и в целом разработанная в этой статье система работает лучше, чем предыдущие подходы. При этом улушение показано на многих разных языках, в том числе достаточно сложных, таких как русский и арабский.

Наибольший интерес представляет статья [3], в которой описан алгоритм, сходный с предлагаемым в нашей работе. Авторы также разделяют весь процесс на создание кандидатов и переранжирование на основе большого числа признаков (в статье предлагается 96). Однако, это как раз и является недостатком, данного исследования: в то время как авторы описывают, какие признаки они использовали, они не дают никакого конкретного описания, как можно их вычислить. Поэтому остается неясным, как применить их алгоритм для исправления опечаток сегментации. Также не понятно, на чем основана классификация признаков, которая приводится в статье и может ли каждый выделенный вид применяться для ошибок склейки-разрезания. Тем не менее, в исследовании [3] была поделана большая работа, было проведено много экспериментов, которые доказали эффективность данного подхода. Это является еще одним доказательством состоятельности нашего метода, который схож с

методом из [3]. Однако мы предлагаем конкретный способ вычисления признаков, которые действительно применимы не только конкретно к опечаткам сегментации, но и легко могут быть обобщены на опечатки любых типов.

Из анализа литературы видно, что с начала изучения вопроса исправления опечаток в поисковых запросах, эта сфера достаточно хорошо развилась. Были испробованы разные техники, и в итоге ученые выработали эффективные и надежные подходы, которые сейчас используются в поисковых системах. Несмотря на такой прогресс, совсем мало литературы касается именно ошибок сегментации. Поэтому наше исследование несет научный интерес и новизну.

4 Описание алгоритма на основе ранжирования

4.1 Классическая модель

В классической модели, которая применяется для исправления опечаток сегментации используется граф возможных разбиений на слова. Каждому пути - разбиению - присваевается длина, которая получается из вероятности в статистической языковой модели. Ответом является разбиение, представляющее собой кратчайший путь. Оно находится с помощью алгоритма поиска в графе. Далее опишем все этапы этого подхода подробнее.

4.1.1 Построение графа

В данном разделе приведем описание алгоритма построения графа возможных разбиений.

Итак, у нас есть поисковый запрос Q. Удаляем из него пунктуацию и пробелы, получая таким образом простую последовательность букв.

Далее разделяем на отдельные буквы и составляем из них граф, каждая вершина которого - буква. Ребра соединяют последовательные буквы. Также в этот граф необходимо добавить вершины, означающие начало и конец слова (обозначим их \$ и $\wedge$).

На следующем этапе создаем новые вершины, состоящие из двух последовательных букв. То есть, если у нас было n исходных букв, в графе будет n+2 однобуквенных вершины (включая символы начала и конца слова), а также n-1 двухбуквенная вершина.

Продолжаем этот процесс последовательно, пока не получим вершину, в которой записана последовательность букв длины n. Финальный граф можно увидель на рисунке 1

Таким образом мы получаем граф в котором каждый путь из начальной вершины в конечную - какое-то разбиение запроса на слова. Эта модель часто используется в задачах обработки естественного языка [7]

Следующим этапом будет задать веса ребрам согласно вероятности по статистической языковой модели. В нашем случае мы пользуемся биграммной вероятностью по языковой модели. Расскажем как найти "вес"($-log(P)$) в общем случае, используя вероятность последовательности слов на основе статистической языковой модели.

4.1.2 Языковая модель

Статистическая языковая модель описывает распределение вероятностей на словах входного текста.

Пусть дана последовательность слов $w_1, \ldots, w_n$. Ищем её вероятность по следующей формуле:

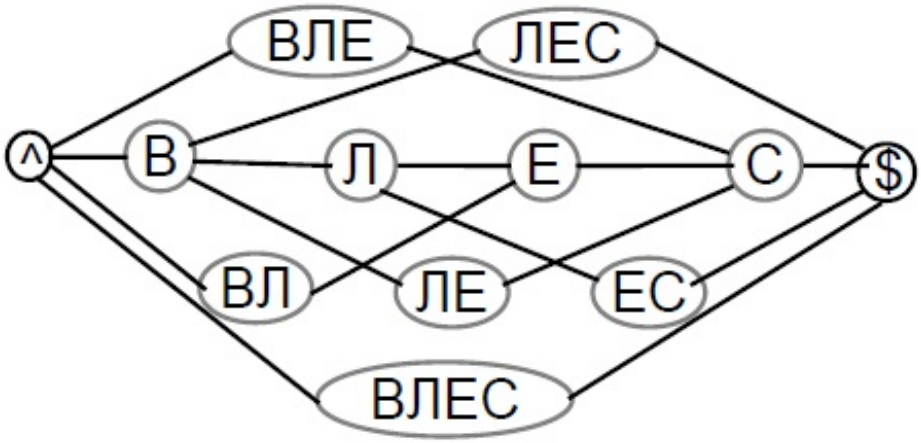

Рис. 1: Построенный граф разбиений

$$P(w_1, ...w_n) = P(w_1) \cdot P(w_2|w_1) \cdot \dots \cdot P(w_n|w_{n-1}, \dots, w_1) \tag{6}$$

Теперь воспользуемся Марковским предположением, согласно которому, каждое слово зависит только от ограниченного количества предыдущих:

$$P(w_i|w_{i-1}, w_{i-2}, \dots, w_1) = P(w_i|w_{i-1}, w_{i-2}, \dots, w_{i-k}) \tag{7}$$

Тогда получаем упрощенную формулу для вычисления n-граммной вероятности:

$$P(w_1, ...w_n) = P(w_1) \cdot P(w_2|w_1) \cdot \dots \cdot P(w_{n-1}|w_{n-2}, \dots, w_{n-1-k}) \cdot P(w_n|w_{n-1}, \dots, w_{n-k}) \tag{8}$$

В нашем случае с биграммами имеем, что слово зависит только от одного предыдущего:

$$P(w_1, ...w_n) = P(w_1) \cdot P(w_2|w_1) \cdot P(w_3|w_2) \dots P(w_n|w_{n-1}) \tag{9}$$

Вероятности в формулах оценим методом максимального правдоподобия, посчитав, как часто встречаются эти последовательности слов в данных:

$$P(w_i|w_{i-1}) = \frac{count(w_{i-1}, w_i)}{count(w_{i-1})} \tag{10}$$

В данном случае получены только вероятности. Чтобы получить "веса"ребер, которые будет более удобно использовать для дальнейшей разработки алгоритма необходимо их преобразовать по формуле

$$w(s) = -logP(s) \tag{11}$$

Таким образом, мы получили взвешенный граф всех возможных разбиений исходного запроса на слова. Для того, чтобы найти оптимальное разбиение, необходимо воспользоваться алгоритмом поиска минимального пути в графе. На рисунке 2 нарисован итоговый взвешеный граф.

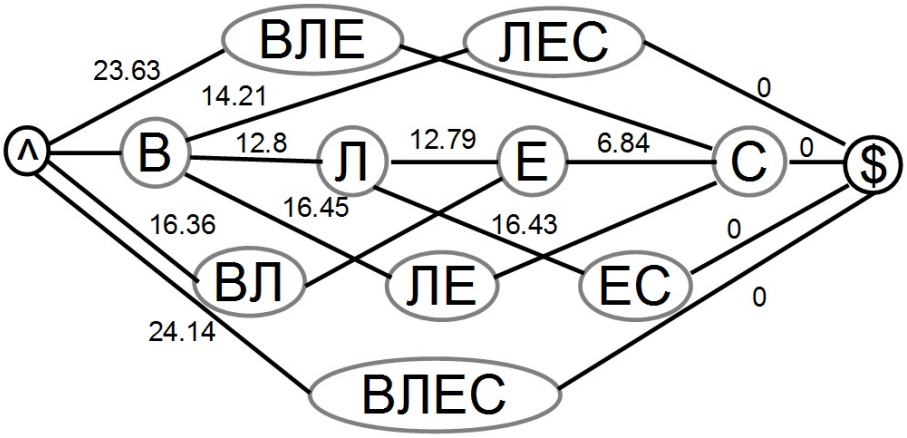

Рис. 2: Взвешенный граф

4.1.3 Алгоритм Дейкстры

В качестве алгоритма поиска кратчайшего пути в графе можно пользоваться различными алгоритмами, например Витерби, $A^*, \ldots$. Мы использовали алгоритм Дейкстры. Этот алгоритм хорош тем, что он легко модифицируем и достаточно быстро работает. Также возможно встроить в него отсечение кандидатов, чтобы добиться большей эффектвности по времени, что очень важно при работе с таким большим количеством данных, как в поисковых логах.

Алгоритм Дейкстры работает следующим образом:

Будем хранить очередь с путями, отсортированными по их длине
Шаг 0. Кладем в очередь начальную вершину
Шаг 1. Достаем из очереди путь с минимальным весом и пытаемся продолжить его всеми возможными вариантами (добавляем ребра, смежные с последней вершиной в пути)
Получившиеся последовательности добавляем обратно в очередь
Повторяем **Шаг 1** пока не попадем в конечную вершину.

Однако такой алгоритм до сих пор не является оптимальным. Для примера рассмотрим как он будет работать на следующем графе (рисунок 3)

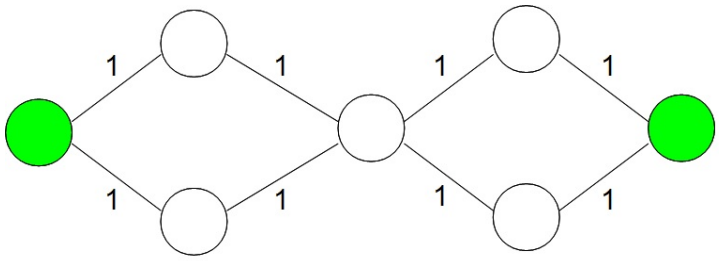

Рис. 3: Пример графа, для которого проход неоптимален

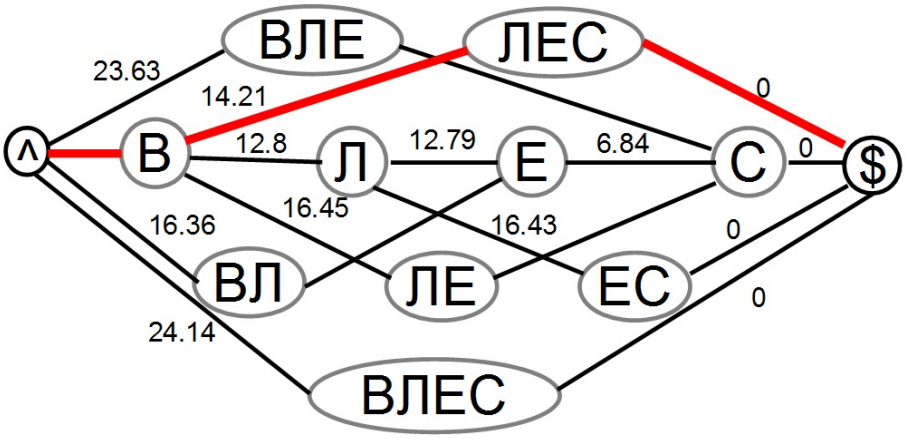

Рис. 4: Кратчайший путь, выбранный алгоритмом Дейкстры

Очевидно, что до того, как мы попадем в конечную вершину, мы обойдем все возможные пути. Хочется как-то откинуть варианты, которые заведомо нам не подходят.

Для этого создадим в каждой вершине логический индикатор, который будет указывать была ли уже посещена эта вершина. Очевидно, что когда мы в первый раз попадаем в вершину, мы попадаем в нее по кратчайшему пути, поэтому второй раз в нее ведет путь, не оптимальнее уже найденного.

Этот метод является очень эффективным для отсечения кандидатов, особенно это касается кандидатов на правильное исправление опечатки. Действительно, если опечатка достаточно нелепая, то эталон будет намного превсходить по вероятности появления и исходный запрос, и всех прочих кандидатов. Таким образом, нам не придется генерировать много ложных путей: мы сразу пройдем по правильному.

Таким образом на **Шаге 1** необходимо ввести дополнительную проверку, что конец ребра, которым мы хотим продлить текущий путь еще не был посещен.

Чтобы восстановить кратчайший путь достаточно в вершине хранить не логические значения, а номер предыдущей вершины в кратчайшем пути до текущей. Таким образом после завершения алгоритма мы просто последовательно пройдем по номерам предыдущих вершин.

Время работы алгоритма $O((V+E)log(V+E))$. Также плюсом алгоритма Дейкстры является то, что он по времени работы не зависит от порядка модели (униграммная, биграммная, n-граммная...)

Уже на основе такого алгоритма можно реализовать эффективное исправление опечаток, однако, классический подход на этом не останавливается. Для улучшения качества исправлений было предложено использоать в графе n-граммную языковую модель. На рисунке 4 показан выбранный алгоритмом Дейкстры путь, который соответствует правильному разбиению для нашего примера.

4.1.4 N-граммы

Для того, чтобы лучше учитывать контекст исправляемого слова, создадим еще один граф на основе предыдущего. Далее для упрощения будем вести повествование в терминах биграммной модели.

Итак, предположим что на этапе построения графа у нас получился такой граф.

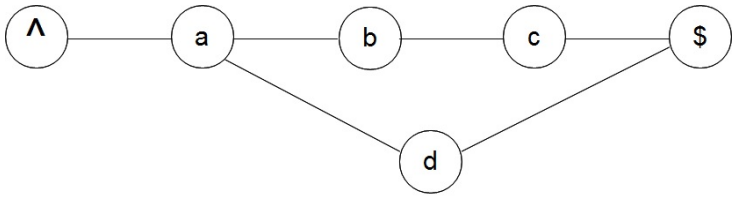

Рис. 5

В новом графе для каждой вершины мы запишем ее с учетом предыдущей. Поэтому получается следующий граф:

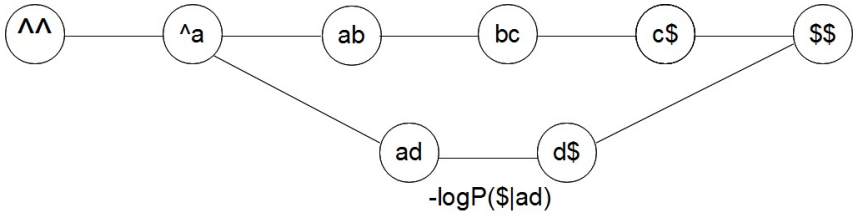

Рис. 6

Формально, в каждой вершине будет записана последовательность $w_i w_{i-1}$ Тогда на ребре из вершины $w_i w_{i-1}$ в вершину $w_{i+1} w_i$ мы будем ставить вес $-logP(w_{i+1}|w_i w_{i-1})$ На иллюстрации поставлен только вес для пути из вершины ad в $d\$$

Далее будем применять алгоритм поиска кратчайшего пути в графе уже к получившемуся n-граммному графу

4.2 Предлагаемый подход

Подход на основе вероятностой языковой модели один из самых популярных, ему посвящено много статей и он дает очень хорошие результаты. Более того, эта вероятность имеет интуитивную интерпретацию, так как каждому слову предписывается какая-то "значимость".

Однако эта модель обладает главным недостатком - при оценке кандидата мы пользуемся только его вероятностью, но существует еще немало признаков, которыми можно воспользоваться. Например, имеет смысл использовать такие признаки, как наличие данного слова в словарях или количество возможных контекстов. В рамках задачи исправления опечаток сегментации хочется оценивать длину слова, чтобы выявлять слова, в которых пробел будет более вероятен.

Также, чтобы убедиться, что данное расширение необходимо, мы измерили полноту для 1 лучшего исправления и 30. Оказалось, что полнота для 1 варианта - 74%, а для 30 вариантов - 91%. Это значит, что необходимое исправление находится совсем близко к первому месту и имеет большую вероятность, но по какой-то причине меньшую, чем первый вариант. Принимая это во внимание, возникает желание добавить других признаков, с помощью которых желаемое исправление будет выигрывать у прочих.

Поэтому в данной работе предлагается обобщить классический подход и добавить к нему преранжирование возможных исправлений слова на основе большого числа признаков.

Задача ранжирования заключается в следующем: на вход ранкеру дается вектор признаков $\boldsymbol{f}$, принадлежащий паре запрос-исправление. На выходе мы ожидаем увидеть вещественное число y, которое будет индикатором, насколько нам подходит данное исправление.

4.2.1 Общее описание

Итоговый алгоритм переранжирования включает в себя 2 стадии:

1. создание графа наилучших разбиений, по которому генерируются 30 лучших кандидатов и
2. переранжирование кандидатов на основе различных признаков

Сначала приведем общую схему алгоритма ранжирования, а в следующих секциях остановимся на каждом этапе более подробно.

- Генерация гипотез
- Вычисление признаков
- Машинное обучение
- Переранжирование кандидатов

запрос	Гипотезы
точь вточь како ни	точь в точь как они точь в точь какони точь вточь как они точь в точь как он и точьв точь как они точьвточь как они точь в точь как о ни точь в точькак они точь в точь ка кони точь в точь какон и точь в точь како ни ... ...

Рис. 7: Пример генерации гипотез

4.2.2 Генерация гипотез

Создание кандидатов в данной работе мы проводим на основе графа всевозможных разбиений запроса. Подробная схема построения его была приведена в пункте .

После построения графа, необходимо модифицировать алгоритм поиска кратчайшего пути в графе, чтобы находить лучшие n исправлений. Вспомним, что в для ограничения пространства поиска в каждой вершине стоял логический указатель, показывающий, была ли посещена данная вершина. Теперь будем хранить счетчик, который определяет, сколько раз данная вершина была посещена. Для того, чтобы сгенерировать необходимое количество гипотез, введем ограничение, при котором при количестве путей в вершину $\geq n$ эта вершина 'закрывается'- вход в неё будет запрещен. Тем самым мы гарантируем, что на каждом этапе у нас будет не больше n возможных вариантов разбиения.

Таким образом мы получаем необходимое число кандидатов. Однако это не единственный способ создания гипотез. В статьях [3], [8] используется алгоритм A*, также может применяться алгоритм Ахо-Корасик [9]. При этом в качестве структуры данных [8], [9] используют тернарное дерево поиска. Мы предпочитаем этому алгоритм Дейкстры, потому что он прост и достаточно быстро работает даже при большом пространстве поиска.

4.2.3 Вычисление признаков

Постановка проблемы

Одной из основных и самых сложных компонент нашего решения является генератор гипотез. Как уже отмечалось выше, существует много решений вычисляющих признаки для отдельных слов. Но такие решения неприменимы для нашей задачи, так как, строго говоря, здесь нельзя говорить о конкретных словах - их границы в рамках одного запроса могут варьироваться. Поэтому получаем, что количество слов в запросе и исправлении, а также среди различных кандидатов в исправление, может быть разным. Как видно, непонятно, как применить классические признаки слов для такой задачи.

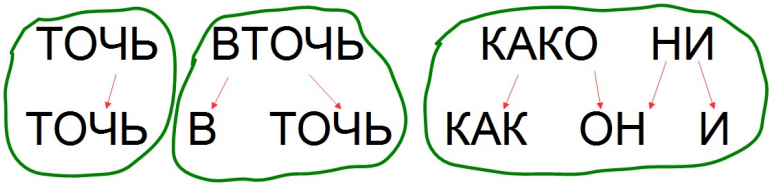

Рис. 8: Пример выравнивания на фрагменты

Фрагменты

В то время как непонятно, каким образом построить соответствие слов в запросе и исправлении, можно попробовать решить эту задачу для последовательности подряд идущих слов. Иными словами, мы хотим построить такое разбиение пары запрос-исправление, чтобы границы каждой группы проходили строго по пробелам и внутри каждой группы были одинаковые буквы. Правда таких развиений может быть несколько, поэтому еще добавим условие, что слов внутри каждой группы должно быть минимальное количество.

Назовем эту структуру фрагментом. Её вполне можно назвать естественным аналогом для обычных слов для решения задачи исправления сегментации. Фрагмент задает более глобальное разбиение текста, нежели простые слова и зависит от обеих частей пары запрос-исправление. Также ясно, что в тривиальном случае (когда исправление не было сделано) каждый фрагмент соответствует отдельному слову. Пример выравнивания фрагмента можно видеть на рисунке 8.

Также укажем, как можно получить данную структуру. Алгоритм её поиска достаточно простой: зафиксируем первое слово в запросе и пусть оно содержит n_1 букв. Далее будем жадно брать слова из исправления по порядку, пока количество букв n_2 не станет больше или равно n_1. Если стало равно - мы получили готовый фрагмент, зануляем все переменные и начинаем процесс снова. Если стало больше - берем следующее слово из запроса (или даже несколько - главное чтобы сделать $n_1 > n_2$), увеличиваем счетчик n_1 и снова пытаемся жадно добрать слова из исправления, чтобы количества букв совпали. Такой процесс назовём выравниванием. Отсюда видно, что вполне может оказаться так, что вся пара запрос-исправление будет состоять из одного фрагмента или наоборот, всякий раз будем получать точное выравнивание слово в слово.

Решение проблемы

Теперь можно использовать фрагменты для решения проблемы о несоответствии разбиений запроса и исправления на слова. Для этого для каждого слова вычислим признаки обычным способом. А далее возьмем от них какие-то аггрегированные значения внутри каждого фрагмента. Такими значениями могут быть: сумма по фрагменту, среднее, минимум, максимум (напомним, что признак - это численное значение, поэтому такие операции вполне применимы).

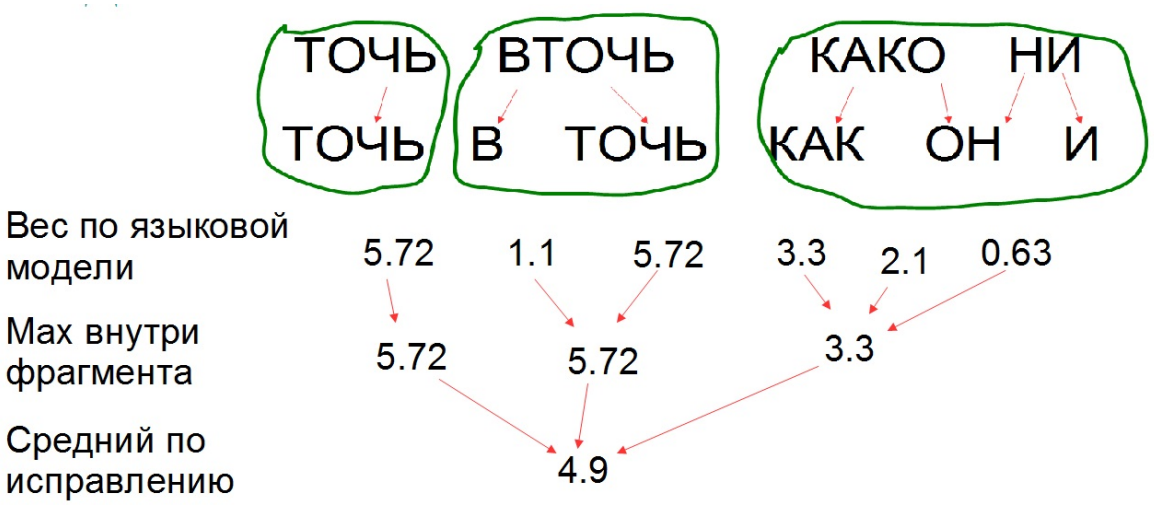

Рис. 9: Вычисление комбинаций признаков по фрагменту

Тем не менее пока нельзя говорить о признаках на уровне пар запрос-исправление, потому что для одного и того же запроса, но разных исправлений мы получили разные фрагменты. Они отличаются как своим составом, так и количеством. Но в этом случае воспользуемся той же техникой - повторно возьмем аггрегированные функции, только уже по фрагменту.

На выходе - одно число для каждой пары, которое будет являться её признаком. Пример можно увидеть на рисунке 9

Возможные словарные признаки

В данном разделе приведем некоторые признаки, которые могут быть использованы на уровне отдельных слов. Одна из классификаций их указана в [3], также они хорошо описаны в [14]. Укажем сначала признаки, которые мы использовали в нашем алгоритме.

Вероятность по языковой модели. Также, как и в работе [3] мы используем вероятность по статистической языковой модели не только для создания гипотез исправления, но и для переранжирования. Причина в том, что этот признак имеет большую значимость, подтверждением чему служит то, что многие алгоритмы, из рассмотренных в обзоре литературы, работают на основе только одной только языковой модели.

При этом, в отличие от этапа генерации графа разбиений, возможно применить не только биграммную вероятность, но и униграммную и трёхграммную. Здесь может возникнуть вопрос - что делать, если при использовании биграммной и трехграммной модели нам не хватит контекста (слово от которого мы ищем вероятность находится в начале фразы). Если такое произойдет, будем использовать контекст меньшего порядка, пока необходимое число предыдущих слов не наберется.

В такой постановке кажется, что наряду с триграммами не имеет смысла использовать n-граммы меньшего порядка. Однако использование их позволяет усилить

вектор признаков, дать алгоритму больше наводок для правильного обучения.

Еще раз заметим, что если использовать такие комбинации как сумма вероятностей по языковой модели, то мы получаем вероятность встретиться всей фразы. Тким образом данный признак имеет наглядную интерпретацию

Длина слова Такой признак имеет большую значимость для задачи исправления слитного-раздельного написания. Если, например, длина одного слова слишком большая, вполне вероятно, что это слово необходимо еще разбить на несколько. Или если в запросе есть много коротких слов, возможно что там много ненужных пробелов. Таким образом имеют интерпретируемый смысл любые аггрегированные значения от этого признака.

Кроме этих двух пунктов мы использовали некоторые признаки на уровне предложения, например, количество слов и подобные. Далее опишем другие признаки, которые тоже можно использовать. Для этого воспользуемся классификацией, приведенной в [3]

- **Различия в форме написания.** Являются логическими значениями, которые указывают, есть ли различие между формами написания слова в запросе и в исправлениями. Такими различиями могут быть: присутствие апострофа ("Кот д'Азюр"), различие в регистре символов ("Вконтакте"и "в контакте") и так далее.

- **Проверка имени собственного.** Значение этого признака может быть индикатором того, что оригинальный запрос - имя, название организации, сайт и прочее ("Лев "Мвидео"). Это полезный индикатор, потому что имена собственные в основном не подчиняются общим правилам грамматики.

- **Присутствие запроса или исправления в словарных источниках.** Этот признак указывает на наличие слова в словаре. Возможно использовать различные словари (морфологический, географических названий) и разные языки словарей. Также к этому классу можно отнести и присутствие слова в Википедии (тоже на разных языках).

- **Признаки связанные с частотой.** Это могут быть признаки частоты упоминания в различных источниках, таких как Википедия, определенные базы данных или просто частота упоминания на веб-страницах. Сюда же можно отнести частоту встречаемости слова с большой буквы и другие частотные признаки.

- **Буквенная языковая модель.** Помимо оценки вероятности слова, можно аналогично ввести буквенную языковую модель и оценивать вероятность с помощью неё. Тогда вероятность слова будет складываться из суммы вероятностей каждой буквы.

В этой секции мы описали, какими признаками на уровне слова можно воспользоваться, чтобы в дальнейшем использовать их для создания комбинаций внутри

Рис. 10: Пример для обучения matrixnet

фрагментов и между фрагментами. Можно обратить внимание, что в данном списке очень мало признаков, которые основываются на оценках одновременно запроса и исправления. Это также связано со спецификой задачи сегментации, поскольку нельзя ввести такие понятия, как, например, расстояние Левенштейна, поскольку оно неприменимо к фрагментам. А выравнивание происходит как раз по фрагментам, а не по отдельным словам. Именно эта проблема и останавливала развитие исправления опечаток слитно-раздельного написания, поскольку нельзя просто переложить на них многие признаки, используемые для ошибок орфографии.

4.2.4 Машинное обучение

На данном этапе необходимо решить задачу обучения ранжированию. Это вид машинного обучения с учителем, когда на вход дается обучающая выборка и по ней необходимо построить ранжирующую модель. Результатом обучения будет являться способность алгоритма предсказывать задание порядка на новых неизвестных данных. На вход обычно подается перечень признаков, на основе которых алгоритм и обучается и строит какую-то формулу, в которой присутствуют значения этих признаков.

Существует несколько алгоритмов, которые позволяют работать с обучением ранжированию. Например, для решения сходной с нашей задачи в статье [3] используются нейронные сети.

В своей работе мы используем алгоритм Matrixnet [15]. В его основе лежат решающие деревья (Gradient Boosting Oblivious Decision Trees) Главная сильная сторона Matrixnet - то, что он очень устойчив к переобучению. Matrixnet автоматически отбирает признаки, которые будут состоятельны. Поэтому можно не беспокоиться за баланс размеров обучающей выборки и размерности пространства признаков. Так как в нашем алгоритме потенциально может использоваться очень большое количество признаков (из-за всевозможных комбинаций базовых признаков) это является оптимальным решением для использования.

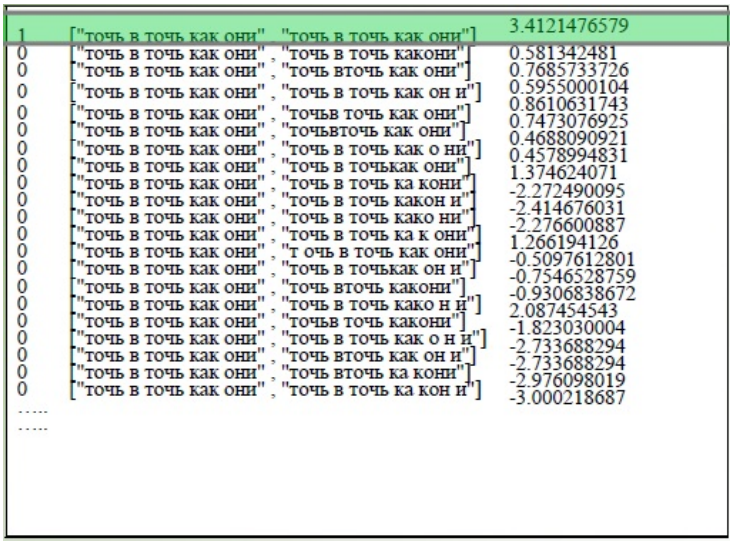

Рис. 11: Выбор кандидата с наибольшей оценкой matrixnet

Для работы с matrixnet в ходе эксперимента необходимо подготовить обучающий набор, который состоит из вариантов исправления запроса. В каждой его строке содержится информация о какой-либо паре запрос-исправление: список признаков, а также индикатор, совпадает ли исправление с эталоном (является ли данный ответ правильным). На выходе каждой строке с признаками Matrixnet приписывает какой-то вес, с помощью которого он оценивает релевантность данного варианта.

4.2.5 Переранжирование кандидатов

При использовании машинного обучения необходимо учитывать специфику опечаток типа сегментации. А именно, как уже было сказано ранее, опечатки составляют 10% от всех поисковых запросов, среди них ошибок сегментации - 16%. Получаем, что всего в наших наборах данных, которые являются настоящими логами запросов только лишь 1.6% с ошибкой, а остальные исправлять не надо.

Это значит, что наш алгоритм не исправил бы ни одной опечатки, потому что из-за их малой примеси, он не смог бы их заметить вообще. Отталкиваясь от предоставленных данных, алгоритм бы обучился, что правильно, когда запрос соответствует исправлению и просто бы выдавал каждый раз исходную фразу.

Необходимо устранить этот дисбаланс, для этого сделаем правильно исправленные опечатки "ценнее". Для этого теперь "вес"правильного ответа будет зависеть от того, была ли опечатка в запросе. Если опечатка была, ставим "1". Если опечатки не было - поставим какой-то другой, меньший вес $w \geq 1$.

Заметим также, что мы получили есще один параметр, который можно подбирать для оптимальной работы алгоритма, опираясь на оценку качества выбранными метриками.

5 Детали реализации решения

В данном разделе будут приведена реализация описанного выше подхода на практике. Основными языками программирования, на которых выполнялась работа были C++ и Python. На C++ были реализованы технически и логически самые сложные части: генератор гипотез и вычислитель признаков. Такие задачи, как обработка входных данных, выбор гипотезы с наибольшей оценкой Matrixnet и прочие небольшие задания были написаны с помощью Python.

5.1 Описание структуры модулей

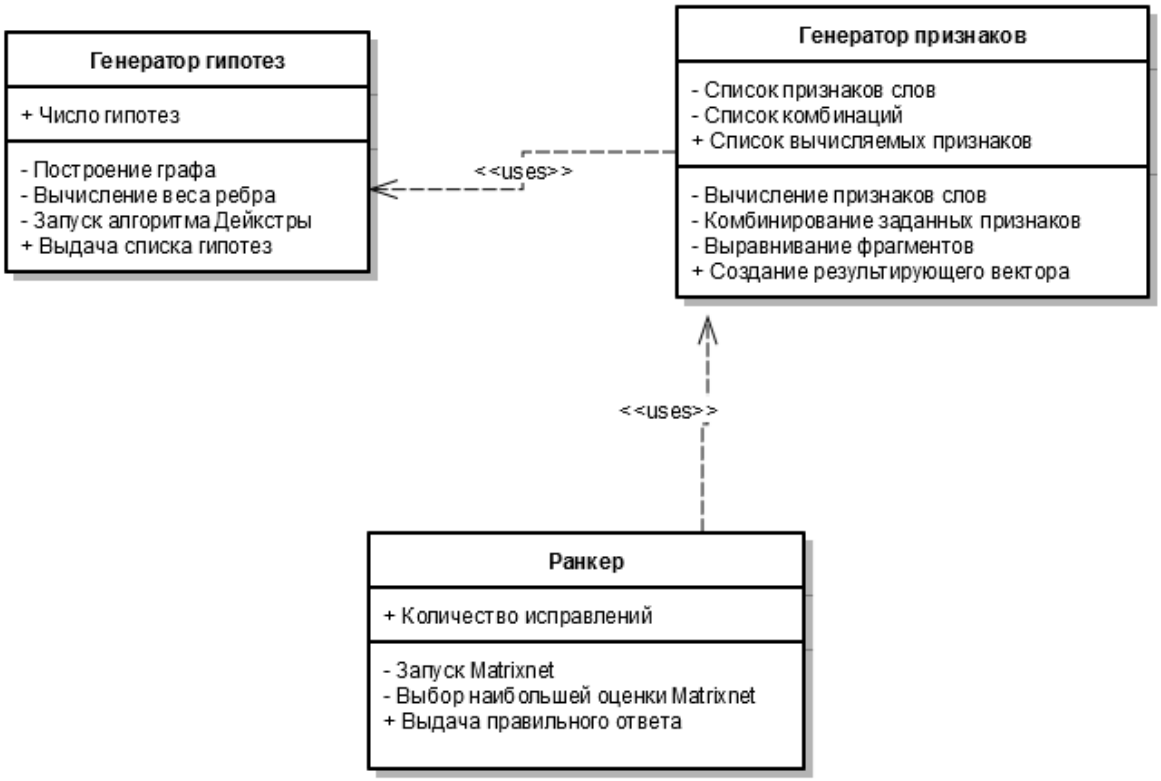

Рис. 12: Диаграмма классов

Приведем описание модулей, с помощью которых работет наше решение. Диаграмма классов приведена на рисунке 12. Видно, что решение имеет три основных компоненты: генератор гипотез, генератор признаков и ранкер.

На вход генератору гипотез подается исходный запрос. Также необходимо задать число желаемых вариатов разбиений. В своих экспериментах мы использовали 30. Генератор должен выполнять следующие методы: создавать граф разбиений, вычислять веса ребер которые получаются на основе вероятности по статистической

языковой модели и находить путь с наименьшим весом с помощью модифицированного алгоритма Дейкстры.

Другим компонентом является генератор признаков. Он зависит от генератора гипотез, так как именно оттуда получаются пары запрос-исправление. Генератор гипотез получает на вход такую пару. Далее он должен выдать результирующий вектор признаков для этой пары. Для этого он должен реализовывать методы для вычисления признаков отдельных слов и для комбинаций данных признаков по фрагментам и предложению. Эти признаки и комбинации задаются списками доступных операций. Однако не обязательно вычислять все возможные комбинации признаков, поэтому также в нем хранится перечень определенных признаков, которые мы хотим посчтать. Также генератор признаков должен уметь делать выравнивание на фрагменты, так как на вход ему подаются пары и для вычисления комбинаций по фрагментам надо разметить их границы.

Наконец, ранкер принимает на вход сформированный обучающий набор и запускает на нем Matrixnet. После того, как отработал Matrixnet, он должен уметь выбрать строку с наибольшей оценкой от Matrixnet и выдать то исправление, которое соответствует этой строке.

5.2 Описание рабочего процесса

Еще раз опишем в данной части как происходит весь процесс исправления опечаток с помощью ранжирования.

На первом этапе производится ввод данных: берутся наборы для обучения и тестирования, которые состоят из пар вида опечатка-эталон. Далее эти данные обрабатываются - устраняются орфографические опечатки, производится детокенизация и так далее. После этого из этих данных берутся оригинальные запросы и направляются на вход алгоритму генерации гипотез.

На этапе генерации гипотез сначала строится граф разбиений, затем для каждого ребра считается биграмная вероятность по языковой модели, преобразуется по формуле $-logP$ и таким образом получается вес ребра. После этого можно запустить поиск кратчайших путей в графе.

Когда сгенерировано все заданное количество гипотез, нужно посчитать признаки для каждой пары запро-исправление. Для этого сначала каждую пару надо выравнять, чтобы получить фрагменты. Затем генератор признаков вычисляет признаки для каждого слова, после этого создаются их комбинации согласно заданным признакам внутри фрагментов и между ними.

После этого необходимо сформировать образец для работы Matrixnet, который состоит из вектора признаков, самой пары запрос-исправление, а также вектора правильных ответов, составляемом на основе данных эталонов. Далее происходит работа Matrixnet, который приписывает оценки каждой паре. После этого необходимо выбрать исправление с наибольшей оценкой. Это и будет ответом нашего алгоритма.

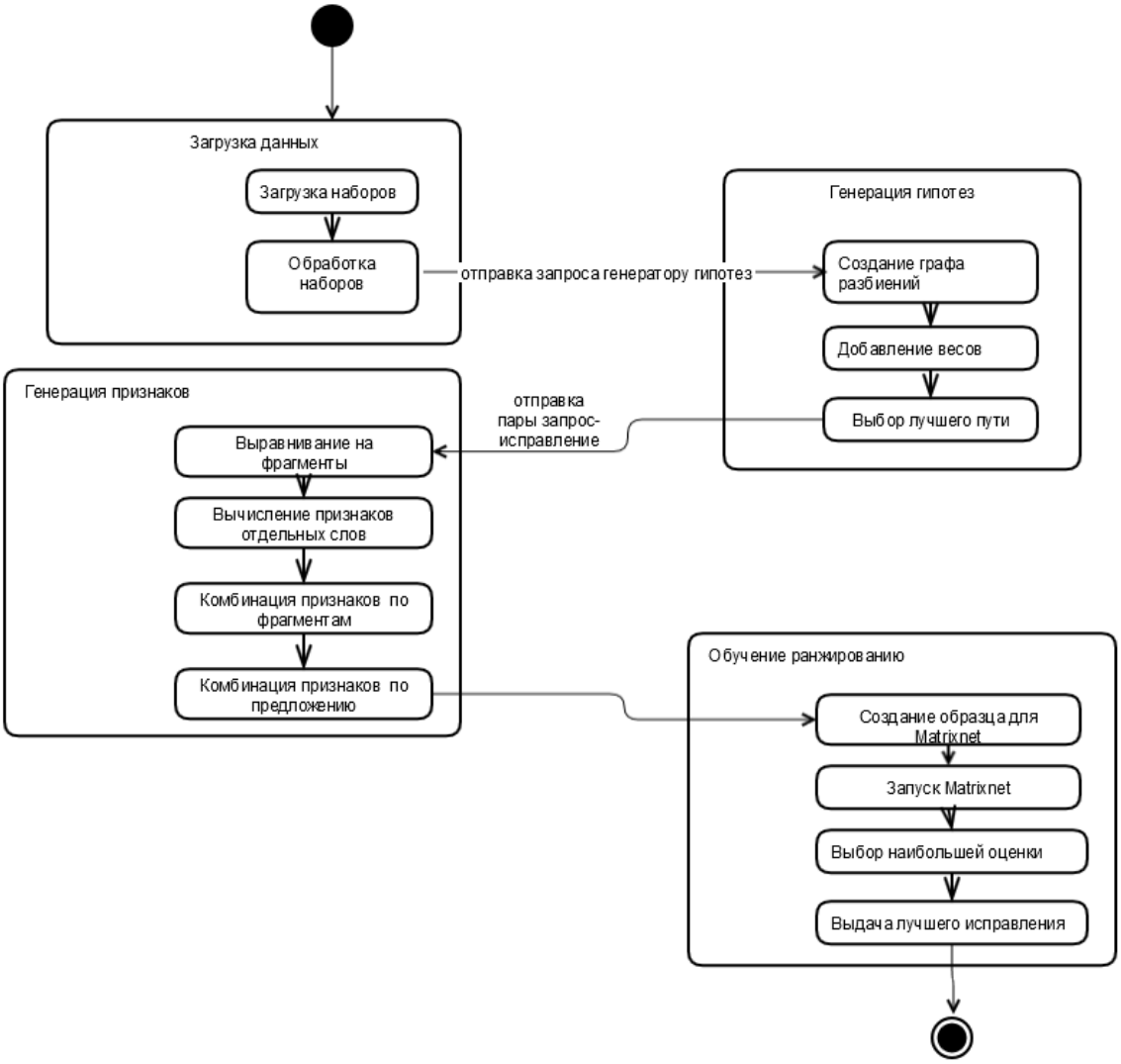

Рис. 13: Диаграмма процесса

6 Эксперименты

6.1 Метрики качества

Для того, чтобы оценить качество работы предложенного алгоритма, сначала введем метрики качества, которыми мы будем пользоваться.

Для упрощения повествования приведем общую классификацию исправления опечаток

- **Хорошее исправление (good)** - в запросе была ошибка, и алгоритм правильно ее исправил
- **Без изменений (nop)** - если запрос был правильный, и алгоритм его не исправил (в качестве ответа выдал исходный запрос)
- **Плохое исправление (bad)**- в запросе была опечатка, алгоритм исправил её, но не правильно (ответ алгоритма не совпал с эталоном)
- **Ложное исправление (false)** - в запросе не было опечатки, но алгоритм решил его исправить
- **Нет исправления (nosug)** - в запросе была опечатка, но алгоритм её не исправил никак (выдал исходный запрос, хотя должен был исправить)

Точность (precision) определяется как отношение правильно исправленных запросов с опеаткой ко всем исправленным

$$precision = \frac{good}{good + bad + false} \tag{12}$$

Точность показывает, насколько "аккуратно"работает алгоритм. К примеру, если мы не исправляем запрос, если точно не уверены, что исправление верно, то точность будет высокой. Если мы правильно исправляем все запросы с опечатками, но при этом исправляем запросы и без опечаток - выдаем какой-то ложнополоительный ответ - то точность будет низкой, несмотря на то, что запросы с опечаткой исправлены.

Полнота (recall) определяется как отношение числа правильно исправленных запросов к числу запросов с опечаткой в тестовом наборе

$$recall = \frac{good}{good + bad + nosug} \tag{13}$$

Полнота показывает, какая доля запросов с опечаткой исправляется. К примеру, у алгоритма, который исправляет запрос, только если он абсолютно уверен в правильности исправления, будет низкая полнота. Причина в том, что многие запросы с ошибками он не исправит, потому что не будет уверен в ответе. Если низкая точность

(если исправлять как запросы с опечатками, так и без опечаток) будет высокая полнота, так как если исправлять всё подряд, велика вероятность исправить все запросы с опечатками.

Видим, что баланс между точностью и полнотой алгоритма - это компромисс. Нам хочется одновременно улучшать обе эти метрики. Поэтому для оценки этих параметров одновременно будем строить парето-фронт для поиска парето-оптимального решения.

Оптимальность по Парето — такое состояние системы, при котором значение каждого частного показателя, характеризующего систему, не может быть улучшено без ухудшения других.

6.2 Обработка данных

6.2.1 Наборы данных

В качестве выборок, на которых мы проводим обучение и тестирование алгоритма, были использованы реальные поисковые запросы, размеченные вручную ассессорами. Таким образом, единицей данных была пара "запрос - эталон". Всего данные охватывали запросы с 2011 по 2013 годы. Наборы имели разную сложность. А именно, были использованы "обычные"запросы - случайно выбранные из потока и "сложные- это запросы, по которым ничего не найдено или они редко встречаются (например, "талидамид")

6.2.2 Подготовка данных

Для того, чтобы подготовить данные непосредственно для задачи исправления сегментации, были проведены следующие операции:

Исправление орфографических ошибок Так как данная работа фокусируется только на исправлении опечаток типа сегментации, а в описаных выше наборах встречаются все виды ошибок, необходимо исправить все, кроме склейки-разрезания. Однако, если мы заменим запросы с орфографическими ошибками на эталоны, мы получим "идеальный"набор, полностью без ошибок, за исключением сегментации. Когда необходимо будет применять его на реальных данных, в составе программы проверки орфографии, эти ошибки уже исправлены не будут. Получится, что мы обучились на нерелевантных данных.

Поэтому имеет смысл не исправлять ошибки орфографии на эталон, а, наоборот, "испортить"эталон подобно опечатке, чтобы создать видимость реальных данных.

Детокенизация Следующий этап обработки - детокенизация. Пунктуация, такая как дефисы, апострофы и прочее, мешает при обучении алгоритма, потому что мы начинаем рассматривать этот знак пунктуации как часть слова, а она больше по-

хожа на пробельный символ. Для этого необходимо применить алгоритм детокенизации. Для этого создадим временную сторку, в которой будет содержаться запрос без пунктуационных знаков вообще. Далее будем строить граф уже по этой строке. Когда мы получим какое-то исправление, токенизируем его. Снова применим побуквенное выравнивание с исходным запросом, и на местах пунктуации добавим его в исправление.

6.3 Проведение экспериментов

6.3.1 Подход, взятый за основу для сравнения

В качестве алгоритма, с которым мы сравним производительность описанного подхода, был взят метод на основе только вероятности по статистической языковой модели. Этот алгоритм подробно описан выше. То есть, когда получается кратчайший путь в буквенном графе, это и будет являться ответом алгоритма.

Также нам необходимо указать параметры, которые надо варьировать, чтобы подбирать оптимальный баланс полноты-точности. Этим параметром будем использовать разность в весе по языковой модели запроса и полученного варианта. Под этим можно найти такую интуицию: если исправление не очень сильно выигрывает в весе по языковой модели, то можно сказать, что они равнозначны в пределах случайной погрешности. А если мы имеем такие равнозначные варианты, необходимо выбрать оригинальный запрос, потому что вводивший его пользователь действительно мог хотеть написать именно так. Еще один аргумент в пользу выбора запроса в этом случае это то, что бывают запросы которые можно правильно разбить несколькими способами (например, "они рис ели "он ирис ел и"). В таком случае эти два варианта будут иметь хорошие веса.

Таким образом, установим допустимую разницу δ и будем допускать исправление, только если разность между весом для запроса и исправления превышает эту величину. При изменении значения δ естественным образом меняется соотношение полнота-точность.

В строгом виде это можно наблюдать на следующем уравнении:

$$correction^* = \begin{cases} correction_{baseline} & w(correction_{baseline}) - w(query) > \delta \\ query & w(correction_{baseline}) - w(query) < \delta \end{cases}$$

6.3.2 Параметры алгоритма

Для предложенного алгоритма тоже надо указать параметры, подлежащие изменению. В данном случае имеется несколько возможностей:

Изменение параметров matrixnet При работе над обучением ранжированию мы таке имеем параметры, изменение которых может повлиять на качество. В данном

исследовании мы меняли такие параметры, как:

- количество итераций Matrixnet и
- весовой коэффициент, который приписывается кажому дереву.

Подробнее про эти коэффициенты можно прочесть в [15]

Изменение веса, предписываемого неисправленному правильному запросу Как было указано выше, очень важно при машинном обучении сделать различие между ошибками, которые удалось правильно исправить и случаями, когда достаточно указать, что ошибки в запросе не было. Для выравнивания этого дисбаланса было предложено ставить некий вес $w \geq 1$ для второго случая. Если изменять значение этого веса, также будем получать различное соотношение оценок качества. Это можно объяснить тем, что чем больше мы ставим вес, тем меньше нам хочется исправлять опечатки и тем больше хочется оставить запрос без изменений.

6.3.3 Проведение эксперимента

После того, как были получены готовые наборы, мы провели измрения указанных выше метрик для двух подходов: классического и предлагаемого. Общая схема эксперимента выглядит следующим образом:

1. Загрузка и обработка обучающих наборов
2. Создание графа разбиений и выбор по нему лучшего исправления (прогон классического подхода)
3. Измерение качества классического подхода
4. Генерация 30 лучших вариантов разбиений
5. Вычисление признаков
6. Ранжирование с помощью matrixnet
7. Выбор варианта с наибольшим ответом matrixnet
8. Измерение качества предлагаемого подхода

Для того, чтобы получать точки с различными значениями полноты и точности в пунктах 2) и 6) мы меняли параметры алгоритмов. Как было уже сказано, в пункте 2) изменялся порог при котором получившееся исправление допускается, в пункте 6) менялись следующие параметры:

- для matrixnet количество итераций и коэффициент бинаризации,
- а также вес неисправленного правильного запроса

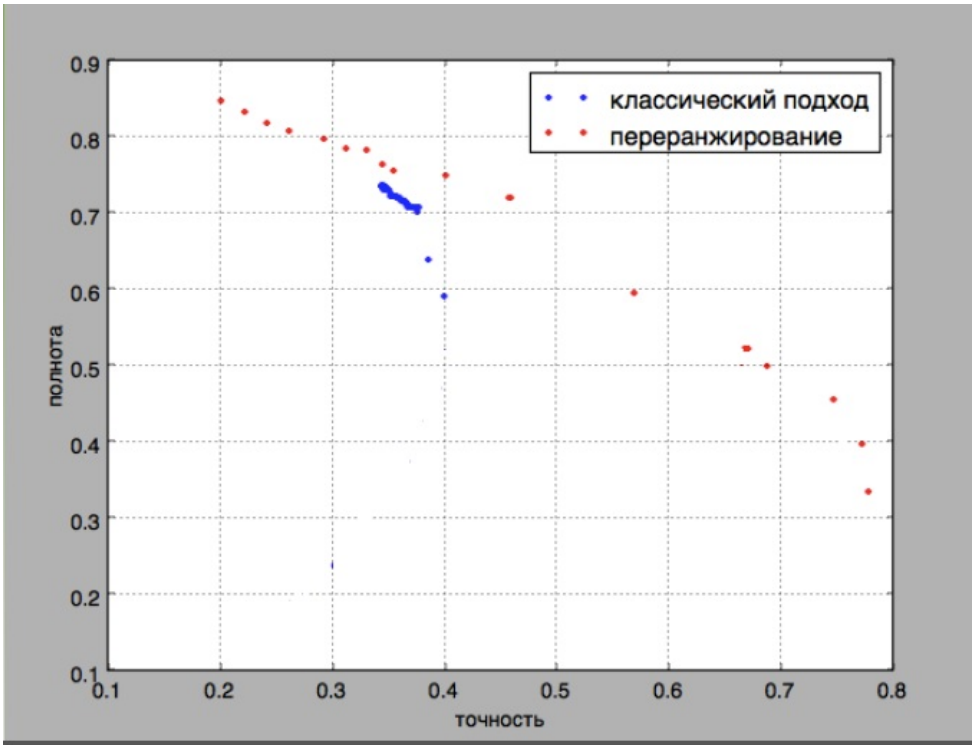

Рис. 14: Парето-фронт полнота-точность

В последнем случае для нахождения оптимального набора параметров, варьировался один из них, при зафиксированных других. В итоге мы получили набор точек, координаты которых - значения точности и полноты. Также мы обладаем более наглядной информацией - непосредственно сами варианты исправлений для обоих алгоритмов, поэтому можно даже в ручную оценить правильность исправления каждого подхода. При измерении полноты и точности в пунктах 3) и 9) для сравнения использовались эталоны из оригинальных наборов данных.

6.4 Результаты

В данном разделе мы продемонстрируем результаты описанного выше эксперимента. Этот эксперимент позволяет оценить работу качественно - посмотрев на то, что исправления действительно имеют смысл - и количественно - получив конкретные значения полноты и точности.

Из таблицы на рисунке 14 видно, что если зафиксировать одну из метрик качества, при проведении экспериментов вторая у предлагаемого подхода выше, чем у классического. Во второй строке этой таблицы мы фиксируем точность и получем прирост по полноте, в третьей наоборот - прирост по точности при фиксированной полноте.

На графике можно видеть парето-фронт полнота-точность (рисунок 15). Видно, что если построить огибающую, она для предлагаемого подхода будет лежать выше, а значит он лучше работает.

Чтобы осветить качественный анализ, предоставим небольшую выборку тех пра-

	Полнота	Точность
Языковая модель	74%	34%
Переранжирование	77%	34%
Переранжирование	74%	39%

Рис. 15: Сравнительная таблица методов

вильных исравлений улучшенного алгоритма, с которыми старый подход не смог справиться (рисунок 16)

Таким образом экспериментальное исследование доказала состоятельность выбранного подхода.

	Классический подход	Переранжировние
райодиннадвоих скачать	райодиннадвоих скачать	рай один на двоих скачать
подключение nowifi	подключение nowifi	подключение по wifi
контрагент	контр агент	контрагент
интернет радиоудлинитель	интернет радио удлинитель	интернет радиоудлинитель
рельеф спортзал	рельеф спорт зал	рельеф спортзал

Рис. 16: Исправления, которых удалось добиться в предлагаемом подходе

7 Заключение

В данной работе решалась проблема улучшения качества исправления опечаток слитного-раздельного написания в поисковых запросах. Необходимость такого исследования очень остра, потому что имеет много полезных для пользователя последствий. В частности, это экономит его время и помогает добиться лучшего соответствия поисковой выдачи целям поиска.

До этого эта проблема также изучалась, но лишь в немногочисленных работах и не так детально. Предлагаемые решения отличались невысокой эффективностью и не учитывали специфику запросов с опечаткой сегментации.

В данном исследовании предлагается альтернативный подход, который состоит из двух этапов: генерация кандидатов в исправление и выбор лучшего кандидата на основе ранжирования. При этом используется большое количество признаков пары запрос-исправление, что отличает данную работу от предыдущих.

В результате был получен значительный прирост качества, по сравнению с классическим решением. Созданный алгоритм обладает большей полнотой и точностью. Поэтому его можно использовать для работы в реальных условиях поисковой системы.

Также возможно предложить пути дальнейшего развития данной работы. Для того чтобы лучше натренировать алгоритм ранжирования, можно увеличить количество обучающих наборов. Также возможно ввести еще признаки и их различные комбинации. Еще одной задачей может быть обработка частных случаев, которые присущи поисковым запросам. Это включает в себя работу с цифрами, URL, собственными именами и прочими словами, которые специфичны для запросов к поисковым системам.

Список литературы

[1] Martins B., Silva M. J. Spelling correction for search engine queries //Advances in Natural Language Processing. – Springer Berlin Heidelberg, 2004. – С. 372-383.

[2] Cucerzan S., Brill E. Spelling Correction as an Iterative Process that Exploits the Collective Knowledge of Web Users //EMNLP. – 2004. – Т. 4. – С. 293-300.

[3] Gao J. et al. A large scale ranker-based system for search query spelling correction //Proceedings of the 23rd International Conference on Computational Linguistics. – Association for Computational Linguistics, 2010. – С. 358-366.

[4] Kernighan M. D., Church K. W., Gale W. A. A spelling correction program based on a noisy channel model //Proceedings of the 13th conference on Computational linguistics-Volume 2. – Association for Computational Linguistics, 1990. – С. 205-210.

[5] Ahmad F., Kondrak G. Learning a spelling error model from search query logs //Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing. – Association for Computational Linguistics, 2005. – С. 955-962.

[6] Whitelaw C. et al. Using the web for language independent spellchecking and autocorrection //Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing: Volume 2-Volume 2. – Association for Computational Linguistics, 2009. – С. 890-899.

[7] Russell S., Norvig P. Artificial intelligence: a modern approach. – 1995.

[8] Duan H., Hsu B. J. P. Online spelling correction for query completion //Proceedings of the 20th international conference on World wide web. – ACM, 2011. – С. 117-126.

[9] Christymol Augustne, Thomas J. An Efficient Model for String Transformation //International Journal for Research and Applied Science and Engineering Technology, 2015

[10] Devlin J. et al. Fast and robust neural network joint models for statistical machine translation //Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics. – 2014. – Т. 1. – С. 1370-1380.

[11] Damerau F. J. A technique for computer detection and correction of spelling errors //Communications of the ACM. – 1964. – Т. 7. – №. 3. – С. 171-176.

[12] Wagner R. A., Fischer M. J. The string-to-string correction problem //Journal of the ACM (JACM). – 1974. – Т. 21. – №. 1. – С. 168-173.

[13] Brill E., Moore R. C. An improved error model for noisy channel spelling correction //Proceedings of the 38th Annual Meeting on Association for Computational Linguistics. – Association for Computational Linguistics, 2000. – С. 286-293.

[14] 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. – Т. 1. – С. 556-567.

[15] A.Gulin. Matrixnet. Technical report, http://www.ashmanov.com/arc/searchconf2010/08gulin-searchconf2010.ppt, 2010