Министерство образования и науки Российской Федерации
МОСКОВСКИЙ ФИЗИКО-ТЕХНИЧЕСКИЙ ИНСТИТУТ
(Государственный Университет)
ФАКУЛЬТЕТ ИННОВАЦИЙ И ВЫСОКИХ ТЕХНОЛОГИЙ
КАФЕДРА КОМПЬЮТЕРНОЙ ЛИНГВИСТИКИ

Дедупликация корпусной выдачи и поиск чужой речи

Выпускная квалификационная работа бакалавра

студента 196 группы

Федюнина Валерия Викторовича

Направление подготовки 01.03.02 «Прикладная математика и информатика»

Заведующий кафедрой В.П. Селегей

Научный руководитель Н.Ю. Копылов

Студент В.В. Федюнин

г. Долгопрудный

2015

1.Введение.

Интернет в наше время является огромной базой данных. И количество текстов можно считать статистически значимым для выводов о природе языка. Но отсутствие естественной структурированности этого объема текстов порождает огромное количество проблем. Среди них возникновение дубликатов, в том числе и нечетких.

Дубликаты необходимо удалять как минимум потому, что из корпуса, содержащего много дубликатов, нельзя выводить какие-либо статистические данные о языке. А также, дубликаты увеличивают количество машинного времени, необходимого для обработки данных и увеличивают итоговый размер корпуса. Также, дубликат не является авторским текстом, потому к документу, являющемуся дубликатом, нельзя отнести какие-либо атрибуты автора (пол, возраст, время написания, регион и т.д.).

Причинами для возникновения дубликатов могут являться ошибки в ходе сбора данных, структура сайта с которого собираются данные, авторы текстов, копирующие материал с других источников (заимствование или кража контента) и прочие. Для поиска нечетких дубликатов уже разработаны алгоритмы, достигающие неплохих результатов при относительно небольших сложностях.

Но большой размер корпуса, хоть и позволяет получать статистические данные о языке, порождает в свою очередь несколько проблем. Первая из них заключается в том, что помимо просто сбора, при построении корпуса необходимо провести некоторую обработку данных, в особенности дедупликацию. Но при слишком большом количестве данных эта задача может оказаться невыполнимой с имеющимися машинными мощностями.

В данной работе предлагается решение, которое не уменьшает размер корпуса, но уменьшает требуемые затраты на получение дедуплицированных результатов. Решение заключается в том, чтобы проводить дедупликацию

корпусной выдачи после каждого запроса. У такой дедупликации, помимо недостатка в виде отсутствия уменьшения размера корпуса, есть некоторые преимущества. Например, такое решение предоставляет пользователю самому выбирать, проводить ли дедупликацию выдачи, а также самому настраивать ее параметры.

Вторая проблема заключается в том, что помимо статистических данных некоторым пользователям хотелось бы получать различные способы употребления словоформы. Это может оказаться полезным, например, самые частые словоупотребления можно указывать в примерах к словарям. Но при работе с большими корпусами количество употреблений некоторой словоформы может достигать нескольких сотен тысяч (а для наиболее частых словоформ эта цифра может быть еще на порядок больше). И выборку такого размера человеку не под силу обработать. Из этого вытекает необходимость разработки автоматических методов уменьшения размера выдачи, предполагающая меньшие требования к исходному корпусу. Примерами таких требований могут быть отсутствие дубликатов (то есть предварительно проведенная дедупликация корпуса) или наличие морфологической, синтаксической или семантической разметки корпуса, наличие лемматизации и прочее. И желательно разработать алгоритм уменьшения размера выдачи, для работы которого необходимо как можно меньшее количество требований к корпусу, так как для выполнения каждого из вышеперечисленных требований необходимы некоторые (порой даже очень значительные) затраты труда и вычислительных мощностей.

Решение данной проблемы в этой работе предоставляться не будет. Но алгоритм, предложенный в данной работе, можно немного модифицировать, и в итоге получить решение для данной проблемы.

Также данная работа затрагивает проблему выделения чужой речи. Чужую речь необходимо выделять как минимум по двум причинам.

Первая заключается в том, что если в корпусе хранятся и используются какие-либо атрибуты авторов, то они не могут распространяться на чужую речь. Тут сразу встает вопрос: а кому из авторов стоит причислять некоторый, повторяющийся у нескольких авторов, отрывок документа? Автоматически найти ответ на него довольно затруднительно, так как даже если мы располагаем данными о дате появления документа, нельзя гарантировать, что более ранняя версия документа не позаимствовала этот фрагмент из другого источника, по каким-то причинам не попавшего в наш корпус. Таким источником вполне может оказаться другой сайт или еще что-либо.

Вторая причина заключается в том, что чужая речь нарушает дистрибуцию словоформ по корпусу. Например, для регионализмов. Так как автор, употребивший некоторый регионализм в составе цитаты другого автора, может не иметь к ареалу данного регионализма никакого отношения. Также полезно бы убрать чужую речь для подсчета использования некоторой словоформы, или для сравнения частот использований различных словоформ, так чужая речь вносит искажения в дистрибуцию словоформ.

2.Определения.

Все документы корпуса разбиваются на токены. Токен – самая базовая единица корпуса. *Токен* - это последовательность символов, не содержащая пробелов, табуляций и переносов строк. Также следует отметить, что в корпусах, на которых проводились исследования, пунктуаторы и различные разделители всегда выносятся в отдельный токен, даже если в оригинальном документе они не были отделены от слов пробелами.

Под *корпусной выдачей* в данной работе будет иметься в виду результат запроса к корпусу. Другими словами, набор корпусных позиций, удовлетворяющих условию запроса. Каждый элемент выдачи называется *сниппетом*. Сниппет содержит в себе информацию о позициях в корпусе и находящихся в них словоформах, удовлетворяющих данному запросу. В данной работе эти словоформы будут называться *центральной частью сниппета.* Для приведенного в данной работе алгоритма необходимо, чтобы помимо словоформ, удовлетворяющих запросам, сниппет содержал словоформы, ближайшие к центральной части сниппета. Словоформы, содержащиеся в корпусных позициях, находящихся прямо перед центральной частью сниппета называются *левым контекстом*. Аналогично вводится определение *правого контекста*, как словоформ, которые следуют сразу за центральной частью сниппета.

Для приведенного в работе алгоритма необходимо, чтобы сниппеты содержали левый и правый контексты (для тех случаев, где они определены). Все цифры указаны для корпуса, у которого максимальный контекст ограничивается документом. То есть, если центральная часть сниппета находится в начале некоторого документа в составе корпуса, то левый контекст этого сниппета в выдаче будет пустым. Аналогично, правый контекст будет пустым, если центральная часть сниппета приходится в аккурат на конец

документа. В остальных случаях, контекст ограничивается некоторой константой.

Оценки алгоритмов для случаев, когда контекст ограничен рамками предложения, абзаца или корпуса в данной работе приведены не будут.

3.Формальная постановка задачи дедупликации корпусной выдачи.

3.1. Дубликаты.

Назовем два документа в корпусе *дубликатами*, если они являются полностью совпадающими последовательностями токенов. Аналогично определяется понятие *дубликата* для сниппетов.

3.2. Нечеткие дубликаты.

Два документа в составе корпуса являются *нечеткими дубликатами* если один из них представляет собой немного искаженную копию другого.

Например, из-за ошибки в сборе данных с web-сайта в текст комментария иногда попадала дата его написания или ник автора. В таком случае два комментария содержащие одинаковый текст, но написанные в разное время или разными авторами имеют в корпусе немного различное представление.

Но если сниппеты имеют достаточно длинный контекст для того, чтобы повторяющиеся части некоторого документа были различимы, то можно вводить аналогичное определение для сниппетов. И в таком случае, можно даже допустить сравнение между алгоритмом поиска нечетких дубликатов среди документов и алгоритмами, осуществляющими дедупликацию среди сниппетов, так как при достаточно широкомо контексте сниппетов эти задачи должны приходить примерно к одному и тому же результату.

Также в нашем случае на корпусах имеется проблема с унификацией знаков препинания и разделителей (унификация дефисов, использование нескольких разных пробелов, унификация кавычек), потому было принято решение игнорировать токены, являющиеся пунктуаторами.

Пример нечетких дубликатов:

«Minna Что нужно для поделки : цельная яичная скорлупа от двух или трех»

«Что нужно для поделки : цельная яичная скорлупа от двух или трех»

Приведенные выше сниппеты являются нечеткими дубликатами. Голубым выделена часть, удовлетворяющая запросу. Красным – совпадающие токены. У второго сниппета в левом контексте на один токен меньше в связи с тем, что там находится граница документа.

Важно понимать, что невозможно дать строгое формальное определение нечетких дубликатов, чтобы оно удовлетворяло всем требованиям, в силу высокой вариативности задач и разного происхождения документов.

Например, рассмотрим корпус, составленный на основе данных из социальных сетей. И допустим, что на нем планируется исследование регионализмов. Как известно, в социальных сетях широко распространены репосты. Это копирования какого-то поста себе, с возможностью добавить небольшой комментарий. При сборе данных, такие репосты вполне могли попасть одним документом, который содержит в себе авторский комментарий и содержание оригинального документа, который также представлен в корпусе. В таком случае, если размер комментария мал, по сравнению с размером поста, эти документы стоит обозначить как дубликаты, и тогда можно потерять несколько регионализмов. Учитывая распространенность репостов, можно предположить, что мы можем потерять некоторую долю редких вхождений. Поэтому, удаление репостов может оказаться негативное влияние на исследование регионализмов. Но зато окажет положительное влияние на объем корпуса и статистику распространения слов, содержащихся в оригинальных документах, которые были скопированы в репостах.

3.3. Задача дедупликации.

Процесс выделения дубликатов в корпусной выдаче называется *пост-индексной дедупликацией.*

Задача: произвести пост-индексную дедупликацию корпусной выдачи. Этот алгоритм должен выделять цепочки дублей, т.е. множества, в которых любые 2 элемента являются дубликатами.

При реализации также надо будет решить, что делать с цепочками дублей. Так как для таких случаев крайне трудно установить авторство. Поэтому для корпусов, в которых хранятся атрибуты автора и для которых крайне важна актуальность данных атрибутов, может оказаться полезной опция удаления всех элементов цепочки (вместо того, чтобы оставить один элемент в качестве образца). В других корпусах кому-то может стать интересна частота повторов различных элементов. Потому реализация к данной работе будет включать настраиваемую пост-дедубликацию, в которой пользователь сможет выбирать любой из изложенных выше вариантов.

4.Проводимые исследования данной задачи.

По данной теме в русскоязычном сегменте является значимой работа Зеленкова Ю. Г. и Сегаловича И. В. [1], где помимо описания алгоритмов приводятся сравнительные результаты наборе полных web-сайтов РОМИП. Особенного внимания в данной заслуживает алгоритм «3+5», который получает неплохие результаты, несмотря на ориентированность на экономию требуемых для работы памяти и машинного времени.

Также много информации по данной проблеме собрано в работе [2]. Там содержатся краткие описания и анализ различных алгоритмов для поиска нечетких дубликатов. Но опять же, задача рассматривается в поиске дубликатов на этапе сборки корпуса, а не в выдаче.

4.1. Алгоритм «3+5».

В каждом документе фиксируются 3 самых длинных предложения и 5 самых длинных слов. Если в документ менее 3 предложений, то фиксируются все предложения. Затем в общий для всего корпуса файл (далее он называется сигнатурным) записываются три сигнатуры:

<ss1, len, id, num, ss1, ss2, ss3, ws1…5>

<ss2, len, id, num, ss1, ss2, ss3, ws1…5>

<ss3, len, id, num, ss1, ss2, ss3, ws1…5>

Где ss1…3 это сигнатуры трех самых длинных предложений документа, ws1…5 – сигнатуры пяти самых длинных слов документа, len – длина документа в словах, num – количество предложений в документе, id – некоторый идентификатор документа.

Далее сигнатурный файл сортируется. Притом порядок сравнения таков: при совпадении сигнатуры предложения (первое поле) сортируется по длине. А при совпадении по длине – по id. Дальше сигнатуры разбиваются на цепочки. Цепочкой в данном алгоритме называется последовательность сигнатур в отсортированном, притом у всех сигнатур в пределах цепочки

совпадают первые поля, и для любых двух соседних сигнатур соотношение длин документов не превышают определенного порога. Этот порог определяется минимальным коэффициентом сходства документов. А коэффициент сходства — это параметр, определяемый пользователем.

Затем составляется файл цепочек. Он сортируется, из него удаляются дубли и цепочки, которые целиком входят в другие, более длинные.

Поскольку внутри цепочки элементы упорядочены по возрастанию длин, для каждого элемента существует некоторая окрестность рассматриваемых документов, определяемая тем же пороговым соотношением.

Далее просматривается файл цепочки, и для каждого элемента цепочки ищутся дубликаты по следующим правилам:

- ищем только в пределах окрестности рассматриваемых документов

- сравниваем пары, только если отношение числа предложений в них не превышает определенного порога и из пяти сигнатур слов совпадает не менее двух (вне зависимости от порядка).

- два документа считаем дубликатами, если у них либо совпали сигнатуры самого длинного предложения <ss1>, либо из трех сигнатур предложений совпадают две (вне зависимости от порядка).

5.Описание алгоритма дедупликации.

В основе идеи алгоритма лежит структура данных бор, где в качестве строки будет рассматриваться контекст сниппета, а в качестве символа перехода будет рассматриваться токен.

Построим направленный граф, каждая вершина которого соответствует некоторому значению контекста.

Значение контекста будем обозначать «левый ; правый», например, для сниппета "пьет за чужой счет" в выдаче по запросу «чужой» соответствующими вершинами могут быть таким контексты: « ; » (пустой), « ; счет», «за ; », «за ; счет», «пьет за ; », «пьет за ; счет». Также в вершине графа будет храниться список соответствующих ей сниппетов в выдаче. Ребро из вершины x в y означает, что существует некоторый сниппет, соответствующий вершине x, такой что, если увеличить размер контекста, охватываемого в вершине x на 1 вправо или влево, то получим контекст, соответствующий вершине y.

Рис. 1. Схема узла графа.

Также в вершине полезно хранить список сниппетов, завершившихся в ней. Сниппет завершается в вершине x, если контекст вершины полностью охватывает контекст сниппета.

Алгоритм построения**:**

1. Инициализируем "корень" графа. Корнем в данном случае называется вершина, соответствующая пустому контексту и всем сниппетам в выдаче.

2. Запускаем относительно всех вершин в графе следующий алгоритм:

2.1 Для каждого сниппета, соответствующего данной вершине пытаемся генерируем контекст на 1 шире данного (либо влево, либо вправо).

2.2 Если в графе уже имеется вершина с новым контекстом - добавить этот сниппет в список сниппетов, соответствующих данной вершине и добавить ребро из текущей вершины в вершину с новым контекстом

Пример выдачи

в романе степной	волк	писавшего о	0
в 1927 году , когда степной	волк	был опубликован	1
работа не	волк	, в лес не уйдет	2
потому что не	волк	я по крови своей	3
Работа не	волк	в лес не сбежит	4

Рис. 2. Пример корпусной выдачи.

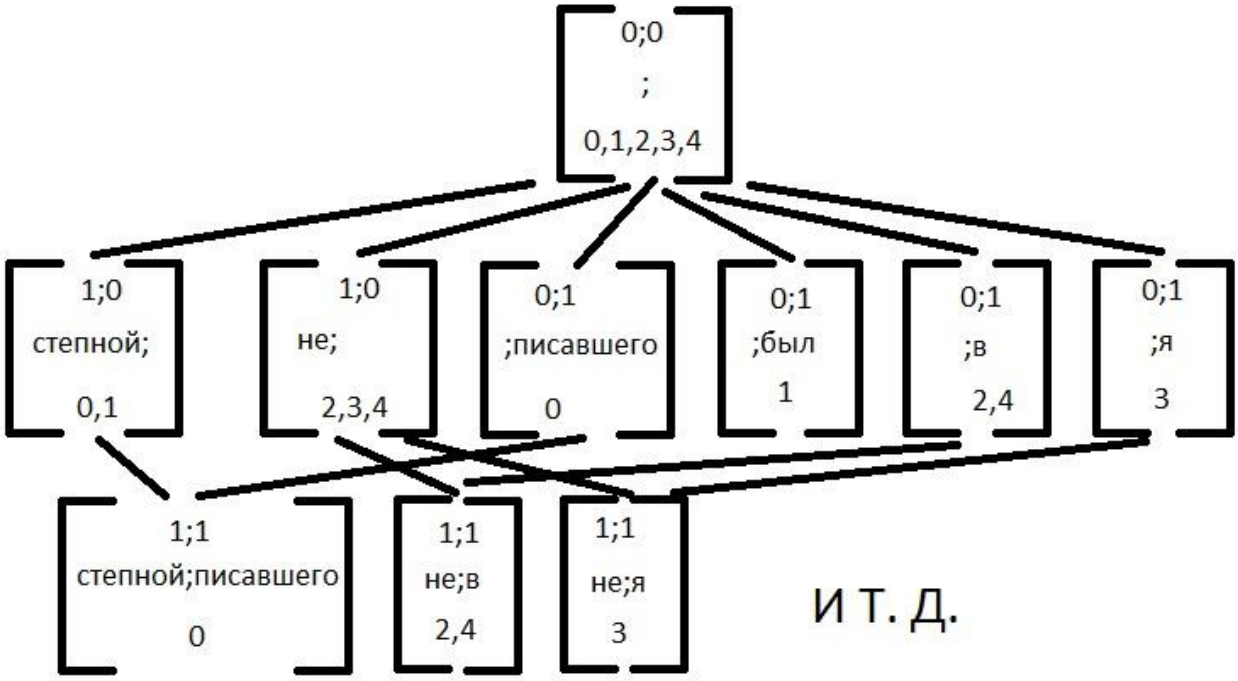

Рис. 3. Часть графа, построенного по выдаче, изображенной на Рис. 2.

2.3 Если в графе нет вершины с новым контекстом, то создать новую вершину и проделать пункт 2.2

Назовем уровнем в графе набор вершин, суммарная длина контекстов которых равна некоторой константе. Например, для 0 это корень. А для 1 - все дети корня.

Можно заметить, что в данном графе ребра всегда проведены с уровня x на уровень $x + 1$. Потому, если хранить граф в виде массива и при создании новой вершины добавлять ее в конец этого массива, то в итоге в массиве граф будет храниться в порядке топологической сортировки (все ребра будут направлены «вдоль» массива). Также, из этого свойства следует, что данный граф всегда будет ацикличен.

На уровне k количество вершин не превосходит kN где N - количество сниппетов в выборке.

Назовем высотой графа количество уровней в нем и обозначим ее за h. В таком случае количество вершин в графе можно ограничить числом $\sum_{\{k=1\}}^{h} kN = \frac{h(h+1)}{2} N \leq h^2 N$.

Сложность алгоритма построения в таком случае рассчитаем следующим образом.

На уровне k каждая вершина имеет суммарную длину левого и правого контекстов равную k. Зафиксируем некоторый сниппет. У него есть всего $k + 1$ контекстов, имеющих суммарную длину k («$k; 0$», «$k - 1; 1$», … , «$0; k$»). Итого получаем утверждение, что каждый сниппет на уровне k встречается среди сниппетов, соответствующих вершинам не более чем $k + 1$ раз.

Ответ на пункт 2.2 можно находить за $O(\log(Nk))$. Вставка тоже может выполняться за $O(\log(Nk))$. Итоговая сложность для построения уровня k равна $O(kN \log(Nk))$. В таком случае сложность построения графа можно ограничить сверху значением $O(h^2 N \log(Nh))$.

Также построение графа можно соптимизировать, если ввести такое правило: если данной вершине соответствует всего один сниппет, то построение в этой вершине можно не проводить, так как в результате построения получится лишь поддерево, в котором все вершины будут соответствовать этому сниппету. И никакой полезной информации это поддерево не несет, потому нет смысла его строить.

Очевидный недостаток данного алгоритма: он не обнаруживает нечеткие дубли, если в них вставлен предлог/слово и т.д. Проблемы изменения форм слова в цитатах решаются использованием в контекстах начальной формы вместо словоформ.

Чтобы граф был способен обнаруживать нечеткие дубли, где один отличается от другого вставкой или удалением слова, введем такую модификацию: при построении будем генерировать вершины используя не только прилегающие токены контекста, но и через несколько токенов (игнорируя пропущенные). Допустим, помимо соседних токенов, также будем использовать токен, находящийся через 1. Для вывода количества вершин на уровне k графа с игнорированием токенов вычислим соотношение между количеством вершин на уровне k графа с игнорированием одного токена, и графа без игнорирования токенов.

На первом уровне только корень.

На втором уровне вершин стало не более чем в 2 раза больше, по сравнению с предыдущим вариантом построения дерева.

На третьем - не более чем в 4 раза больше, чем на втором уровне текущего графа, и не более чем в 4 раза больше, чем в предыдущем варианте построения.

Продолжая рассуждения (применяя математическую индукцию) мы получаем, что на k уровне графа не более чем в 2^k раз больше вершин, чем на k уровне у предыдущего алгоритма построения. Потому получаем сложность построение уровня k равной $O(2^k kN \log(2^k kN)) = O(2^k k^2 N \log(Nk))$. Сложность построения графа $O(2^h hN \log(2^h Nh))$. Притом, если разрешить игнорировать до 2 или 3 токенов, то 2^h заменится на 3^h или 4^h.

Так как подобная сложность слишком велика, есть способ оптимизировать данный алгоритм: разрешим в пределах одного сниппета пропускать не более s токенов.

В таком случае кол-во вершин можно ограничить "выбором" пропущенных позиций, т.е. N^s (а при $s = 0$ получаем самую первую

реализацию, без пропусков токенов). Количество вершин на уровне k можно ограничить $(Nk)^{s+1}$, а сложность построения графа: $O(h(Nh)^{s+1}(s+1)\log(Nh))$.

Также, важно заметить, что ребра всегда направлены на уровень ниже, а значит, все рассуждения об ацикличности и топологической сортировки остаются в силе.

Можно отметить, что в такой реализации мы не различаем ребра, образованные взятием ближайшего токена, от ребра, образованного взятием токена, находящегося на некотором расстоянии от края контекста, охватываемого данной вершиной. За счет этого сниппеты, содержащие контекст, различающиеся лишь в s токенах окажутся в одной вершине графа. Но для этого необходимо хранить в вершине графа ширину используемого контекста для каждого сниппета в отдельности. Например, в графе с игнорированием не более одного токена

Как находить из этого графа дубликаты.

Назовем вершину *листом*, если у нее нет исходящих ребер. Такое может быть по двум причинам:

1. Достигнута максимальная высота графа

2. У всех сниппетов этой вершины не определены дальнейшие контексты (т.к. в самом корпусе могут встречаться документы с очень маленькой длиной). И в таком случае окружающий контекст некоторого токена ограничен размером документа.

Назовем сниппет *завершившимся* в вершине графа, если в графе нет вершин, соответствующих расширенному контексту данного сниппета. То есть при построении по каким-то причинам не было построено вершины, которая

получается из данной расширением контекста этого сниппета. Причина этому либо достижение в данной вершине максимальной высоты графа, либо в том, что контекст данной вершины охватывает весь имеющий у сниппета контекст.

Сниппеты, завершившиеся в одной и той же вершине в этом графе являются дубликатами.

В итоге получили алгоритм с двумя параметрами: максимальной высотой графа и максимальным количеством игнорируемых токенов в сниппете.

6.Полученные результаты.

Для получения результатов проводилась дедупликация запросов к различным корпусам проекта ГИКРЯ (Генерального Интернет-Корпуса Русского Языка). К сожалению, из-за некоторой нечеткости в определении нечетких дубликатов и отсутствия предварительно размеченных корпусных выдач, оценка результата производилась вручную.

Данные оценки основаны на дедупликации 30 выдач размером от 70 до 1300 сниппетов.

Лучший результат был замечен на следующих параметрах графа:

- высота: 25

- количество игнорируемых токенов: 1.

Итоговые результаты:

- полнота: 0.96

- точность: 0.97

- F-мера: 0.96

Для сравнения также приведу результаты из работы [1]:

Алгоритм	Полнота	Точность	F-мера
«3+5»	0.96	0.95	0.95
Long Sent	0.84	0.80	0.82
TF	0.60	0.94	0.73
Opt Freq	0.59	0.94	0.73
TF *RIDF	0.59	0.95	0.73

Heavy Sent	0.62	0.86	0.72
TF * IDF	0.54	0.96	0.69
Lex Rand	0.50	0.97	0.66
Descr Words	0.44	0.77	0.56
Log Shingles	0.39	0.97	0.56
Megashingles	0.36	0.91	0.51
MD5	0.23	1.00	0.38

7. Постановка задачи поиска чужой речи.

Привести формальное определение чужой речи крайне затруднительно. Понятно, что это та речь, которая не принадлежит автору. Более формально, это та речь, к которой не относятся атрибуты автора текста. И, конечно, такие участки в документах в корпусе хотелось бы уметь выделять. Но при попытке более точного определения возникает много вопросов.

Например, куда отнести пословицы и поговорки? С одной стороны, большинство из них были придуманы в народе. И вроде бы автор конкретного документа лично пословицу не составлял. Да и слова в поговорках имеют несколько иное значение, отличное от словарных значений. Потому с одной стороны было бы полезно обозначить их за чужую речь. С другой стороны, если посмотреть на пословицу как на нечто неделимое, то можно сказать, что автор просто использовал данный элемент языка в своем тексте. Также, некоторые пословицы распространены лишь в определенных регионах, потому они могут нести полезную информацию для региона автора. Или наоборот, имея информацию об авторах, можно построить статистику по употреблениям пословиц относительно различных атрибутов автора.

Не лучше дела состоят с фразами из песен, фильмов, литературных произведений. С одной стороны, авторы данных фраз вполне конкретны. И употребление этих фраз любым другим автором является чужой речью. С другой стороны, некоторые знаменитые цитаты из фильмов уже стали, фактически, крылатыми фразами, и их можно воспринимать как единое целое. И опять, возможность исследование употребления таких выражений.

И это только часть примеров. А теперь прибавим ко всем этим проблемам тот факт, что подобные фразы нередко употребляются с некоторыми искажениями. Некоторые из них вызваны тем, что, скорее всего,

автор писал по памяти. Например, вот две цитаты из корпуса ГИКРЯ Живой Журнал:

1. Как там, у Игоря Талькова: Господа демократы, Вы знали примеры, Когда Ваши коллеги Развязали террор, истребили цвет нации Мечом Робеспьера – И Париж до сих пор отмывает позор.
2. Ну а нас все через … Господа демократы, вы знали примеры, Когда ваши коллеги учинили террор : Истребили цвет нации мечом Робеспьера, И Париж по сей день отмывает позор.

Очевидно, что это цитаты из одной песни. Но один из авторов допустил небольшие ошибки при воспроизведении текста. В данном случае вопрос, являются ли эти отрывки чужой речью особых трудностей не вызывает. Но притом желательно, чтобы данные отрывки находились, и алгоритм помечал, что они похожи. И границы отрывка определялись границами цитирования песни, несмотря на некоторые вариации.

Но бывают ситуации, когда не ясно, нужно ли считать цитату чужой речью. Например, некоторое переделывание названия известного фильма. Приведу примеры из того же корпуса: «Как я перестал бояться и полюбил Ктулху», «Как я перестал бояться и полюбил редактора». В таких случаях не ясно, кто придумал это пародийное название, так как такое мог высказать и автор комментария. Также не ясно, считать это чужой речью с вариативностью, ведь за основу взято название известного фильма, или не считать это чужой речью, так как эту вариативность мог дописать автор, вкладывая в него совершенно иной смысл.

И так как корпуса разрабатываются для лингвистов, и у разных лингвистов могут быть различные точки зрения на перечисленные выше проблемы, в данной работе не будет приведено строгое определение чужой речи. И целью дальнейших рассмотрений будет разработка алгоритма, выделяющего в выборке все устойчивые коллокаций.

Затем предоставить пользователю данные о группах сниппетов, содержащих некоторую устойчивую коллокацию. А какое-либо разделение устойчивых коллокаций на виды, например, на чужую и авторскую речь, мы оставим на выбор пользователя.

8.Проводимые исследования задачи поиска чужой речи.

Так как в данной работе будет рассматриваться поиск устойчивых коллокаций, то нельзя не упомянуть работу Саломатиной Ю. Г. и В. Д. Гусева [2] в которой приводится крайне важный тезис о том, как по частоте можно выделить устойчивые коллокации.

В рамках данной работы также рассматривались работы по поиску плагиаризмов. Например, в работе [3] рассматривался метод, на основе анализа n-грамм слов, входящих в разные статьи. В этой работе рассматривалась задача нахождения в новостной статье фрагментов чужого текста, в их случае, другой статьи, написанной ранее и взятой за основу. В качестве базы оригинальных статей, была взята база статей Press Association. А в качестве набора статей, в которых ищется чужая речь, взяты статьи из девяти газет, которые имеют право использовать статьи Press Association. Но сам метод выделения всех n-грамм, и попарного сравнения элементов слишком долгий, чтобы его можно было проводить каждый раз после выполнения запросов.

Также были изучены работы на тему автоматического поиска цитат. Например, в работе [4] описан алгоритм автоматического выделения цитат и их авторов из текста новостных статей на различных языках, включая русский. Основной прием заключается в том, чтобы находить текст, окруженный в кавычки, а также проверять около такого текста наличие глаголов, свойственных прямой речи и именованных сущностей (персон, организаций). Но данный анализ не подходит для выделения чужой речи в текстах из интернета, так как они менее формальны, по сравнению с новостными статьями. И чужая речь в интернете не всегда выделена кавычками с указанием автора. Особенно, когда это умышленное искажения документа с целью кражи контента и обмана поисковых систем (так называемый рерайт).

9.Описание алгоритма.

9.1.Идея алгоритма.

Для поиска устойчивых сочетаний мы будем использовать тезис, сформулированный в работе [2]. Он заключается в следующем.

Если рассмотреть количество вхождений в корпус некоторой устойчивой коллокации, и той же устойчивой коллокации, но без первого или последнего токена, то можно заметить, что эти цифры не сильно различаются. Но в то же время, контекст, в котором употребляется некоторая устойчивая коллокация будет сильно варьироваться.

То есть, если зафиксировать некоторую устойчивую коллокацию и еще один токен перед ней или после нее, то количество вхождений этой цепочки токенов будет сильно отличаться от количества вхождений устойчивой коллокации без дополнительных токенов. Таким образом на основе частот употреблений цепочек токенов можно выделить среди них устойчивые коллокации.

Разумеется, появляется вопрос, что данная теория верна на полном корпусе. А в данной работе рассматривается выделение лишь в корпусной выдаче, которая является очень малой частью корпуса. Но корпусная выдача на некоторый запрос, это и есть все вхождения токенов, удовлетворяющих условиям запроса. Потому из корпусной выдачи можно делать какие-либо выводы и анализировать статистику для токенов, непосредственно удовлетворяющих запросу. Единственно, что для этого необходимо, это чтобы элементы выдачи содержали достаточно информации о контексте, окружающем вхождение. Так как слишком короткий контекст может не содержать целиком устойчивую коллокацию, и в таком случае ее невозможно будет выделить никакими алгоритмами.

9.2. Модификация графа.

Для начала необходимо ввести понятия левых и правых ребер. Назовем ребро в графе *левым*, если переход вызван расширением контекста влево. Аналогично, введем понятие *правого* ребра.

9.2.1. Стабильные ребра.

Затем необходимо ввести понятие стабильного ребра. Зафиксируем некоторую вершину u. Левый переход из вершины u в вершину v называется стабильным, если соотношение количества сниппетов, соответствующих вершине v к суммарному количеству сниппетов соответствующим всем вершинам, в которые есть левые переходы из вершины u, превышает некоторый порог, называемый *порогом стабильности* (например, равный 0.7). Аналогично вводится стабильность для правых переходов.

Введем определение стабильности справа: после некоторого набора токенов (контекста, зафиксированного в начальной вершине стабильного ребра) чаще всего следует токен, который был добавлен в контекст вершины, в которую входит стабильное ребро.

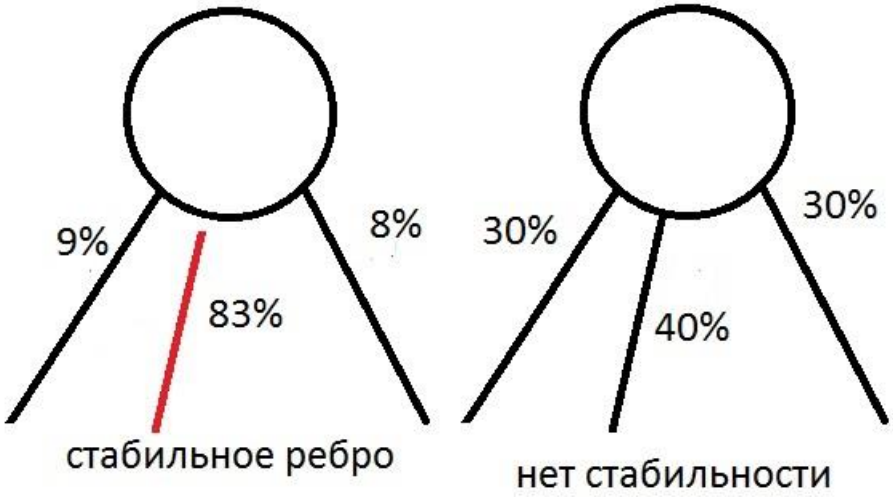

Рис. 4. Примеры с наличием и отсутствием стабильности, для порога 0.8.

9.2.2. Флаги идиомы.

Пусть у каждой вершины хранятся 2 флага идиомы: левый и правый. Если в вершину входит стабильное левое ребро, из вершины не выходит никаких стабильных ребер, то поднимаем левый флаг идиомы. Если в вершину входит стабильное правое ребро и из вершины не выходит никаких стабильных ребер и вершина не является листом, то поднимаем правый флаг.

Эти флаги означают, что у этой вершины контекст похож на конец некоторой устойчивой коллокации. То есть, этот контекст ведет себя как устойчивая коллокация, а какое-либо его расширение – уже нет.

9.2.3. Алгоритм разметки флагов стабильности и идиом.

Разметку рекомендуется проводить от листьев к корню. Упомянутая выше топологическая сортировка является таким порядком, и гарантирует завершение алгоритма.

В каждой вершине заводится 4 флага: стабильность слева и справа и флаг идиомы слева и справа.

В процессе обхода в некоторой выбранной вершине u сначала просматриваются все левые ребра. Из вершин, в которые ведут левые ребра из u, выделяется вершина с наибольшей частотой. Если соотношение частоты этой вершине к суммарной частоте всех вершин, в которые ведут левые ребра, превышает определенный порог стабильности, то считаем ребро стабильным. В таком случае у вершины u, в которой мы сейчас находимся поднимаем флаг стабильности слева.

Если стабильное ребро нашлось, то затем обращаем внимание на вершину, в которую это ребро направлено. В этой вершине флаги стабильности уже обозначены, так как обход осуществлялся от больших уровней к меньшим,

а ребра всегда направлены от меньших уровней к большим. Про эту вершину нам известно, что в нее входит левое стабильное ребро. Если из этой вершины не выходит стабильных ребер, то есть если не поднят ни один из флагов стабильности, то мы поднимаем флаг левого конца идиомы.

Затем возвращаемся к вершине, у которой проверяли левые ребра, и проделываем аналогичную работу с правыми ребрами, поднимая флаги правой стабильности и правой идиомы.

Сложность алгоритма можно оценить следующим образом. Каждое ребро просматривается лишь один раз, когда мы выделяем максимальное из левых или правых ребер некоторой вершины. Потому необходимо оценить лишь количество ребер в графе.

А количество ребер в графе можно оценить таким способом. Максимальное количество ребер в графе достигается в таком случае, если при построении каждая попытка расширить контекст сниппета завершается созданием новой вершины и ребра в него. Поэтому максимальное количество ребер в графе можно ограничить сверху оценкой максимального количества вершин, выведенной ранее и равной $O(h * (Nh)^{s+1})$, где s это максимальное количество игнорируемых токенов в сниппете, h - высота графа и N – размер выдачи.

9.2.4. Нахождение результата в графе.

Затем проводится еще один обход графа для поиска устойчивых словосочетаний. В данном случае порядок обхода уже не имеет значения.

В данном обходе для каждой вершины проверяем выполнение одного из трех условий:

- подняты оба флага идиом

- поднят флаг левой идиомы, а длина правого контекста в данной вершине равна 0

- поднят флаг правой идиомы, а длина левого контекста в данной вершине равна 0.

Если одно из трех условий выполняется, то контекст данной вершины считаем устойчивым словосочетанием. Группа сниппетов, имеющих общий контекст, который является устойчивым словосочетанием называется *кластером*. Важно, что один элемент может принадлежать нескольким кластерам одновременно.

В итоге получаем алгоритм с 3 параметрами: высотой графа, порогом стабильности и максимальным количеством игнорируемых токенов. Алгоритм принимает на вход набор сниппетов и выдает набор кластеров.

10. Проблема кластеров и ее решение.

Проблема выданных алгоритмом кластеров заключается в том, что если игнорирование токенов было включено, то помимо кластера с устойчивым словосочетанием *x* будет выделено много кластеров, контекст которых является *x* с пропущенными токенами.

Например, для запроса «волк» и пословицы «работа не волк в лес не убежит» будут выделены кластеры, соответствующие таким контекстам:

- «работа не волк в лес не убежит»
- «работа не волк в лес убежит»
- «работа не волк в не убежит»
- И т.д.

Притом списки сниппетов в этих кластерах будут почти совпадать. В целом, ответ будет точным, но пользователю будет труднее такие кластеры обрабатывать.

Для решения этой проблемы необходимо ввести определение похожести кластеров.

Так как кластеры, в нашем понимании, это множества сниппетов, введем такую функцию похожести кластеров:$Sim(X,Y) = \frac{(|X \cap Y|)}{|X \cup Y|}$.

И после выделения кластеров запускается следующий алгоритм: для каждого кластера составляется список похожих с ним, и затем производится слияние всех этих кластеров в один. Похожими в данном алгоритме называются те кластеры, похожесть которых превосходит определенный порог. Сложность данного алгоритма $O(L^2N)$, где L – это количество кластеров, а N – это количество элементов в выдаче.

Если в данном алгоритме задать порог, равный 1, то получается алгоритм, который просто удаляет все одинаковые кластеры.

Проведенные эксперименты показали, что оптимальное значение порога равно примерно 0.85-0.875.

У данного алгоритма есть очевидные недостатки. Например, его результат зависит от того, в каком порядке будут подаваться на вход исходные множества. Но более аккуратный подсчет будет занимать больше времени. Например, если попытаться взять за основу алгоритм построения иерархической кластеризации, то сложность может возрасти до $O(L^3N)$, что недопустимо. Потому что при большой выдаче и параметрах алгоритмах, в которых игнорируется 3-4 токена, исходное количество множеств, которые необходимо слить, может достигать нескольких десятков тысяч.

Также, важно понимать, что это просто некоторый способ почистить данные, полученные на выходе, а не некая приоритетная задача, требующая строгого точного решения. В связи этим в рамках данной работы было решено использовать данный алгоритм слияния кластеров, несмотря на все имеющиеся у него недостатки.

11. Результаты.

В результате предложенный алгоритм должен выделить все возможные вхождения чужой речи в выборке (далее *кандидаты*). А задача выделения непосредственно чужой речи в данной работе не рассматривается, в силу причин отсутствия строгого определения.

Оценки выводились вручную. Оценивалась точность и полнота поиска кандидатов в чужую речь.

(Высота графа, игнор. токенов, порог стабильности)	Полнота	Точность
(20, 2, 0.9)	0.964	0.393
(25, 4, 0.7)	0.959	0.256
(30, 5, 0.8)	0.983	0.267
(30, 5, 0.8) + длина не менее 5 токенов	0.827	0.755

Также важно понимать, что в данной задаче больший приоритет отводится полноте. Это связано с тем, что задачу выделения чужой речи мы оставляем пользователю. Значения точности, которая не является целевым показателем выявления устойчивых коллокаций, являются авторской характеристикой концентрации идиом, пословиц, поговорок, басен, песен по отношению к фрагментам письменной речи, расположение словоформ в которых обусловлено наиболее типичным линейным порядком вводных слов, последовательностями цепочек связующих частей частей речи: союзов, предлогов и наиболее часто употребляемых с ними слов.

12. Вывод.

В рамках данной работы был предложен алгоритм поиска дублей в корпусной выдаче и оценено его качество (полнота, точность, F-мера = 0.96, 0.97, 0.96).

Также был разработан инструментарий для исследований на корпусе на тему чужой речи, а также оценено его качество (полнота = 0.983).

13. Заключение.

Дальнейшие исследования в области автоматического обнаружения чужой речи потребуют либо явно ввести какое-то определения, осознавая все его недостатки, либо оставить некоторые грани, которые определяются лингвистами. Оба случая наверняка потребуют привлечения различных словарей и семантической разметки. А во втором случае потребуются еще и усилия лингвистов.

14. Список используемой литературы.

[1]. Ю. Г. Зеленков, И. В. Сегалович «*Сравнительный анализ методов определения нечетких дубликатов для Web-документов*»

[2]. Ян Помикалек «*Removing Boilerplate and Duplicate Content from Web Corpora»*

[3]. В. Д. Гусев, Н. В. Саломатина «*Алгоритм выявления устойчивых словосочетаний с учетом их вариативности (морфологической и комбинаторной»* //труды международной конференции Диалог'2004

[4]. Alberto Barrón-Cedeño and Paolo Rosso «*On Automatic Plagiarism Detection Based on n-Grams Comparison*»