

САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Направление: 01.03.02 Прикладная математика и информатика

ООП: Прикладная математика, фундаментальная информатика и программирование

Кафедра технологии программирования

ОТЧЕТ О НАУЧНО-ИССЛЕДОВАТЕЛЬСКОЙ РАБОТЕ

Тема задания: Исследование алгоритмов лемматизации

Выполнил: Смирнов Алексей Артёмович, группа 21.Б02-пу

Руководитель научно-

исследовательской работы: Блеканов Иван Станиславович, кандидат

технических наук, доцент, заведующий кафедрой технологии программирования

САНКТ-ПЕТЕРБУРГ

2023

Содержание

1 Введение

Данная работа посвящена изучению технологии Natural Language Processing(NLP) - одной из важнейших отраслей машинного обучения, дающую возможность обработки и создания информации на естественном языке. Мы рассмотрим лишь часть NLP, называемую лемматизацией, получим численную оценку нескольких популярных алгоритмов на реальных данных, что позволит рассмотреть данную задачу со всех сторон.

Лемматизация - процесс приведения словоформы к лемме, то есть к её нормальной форме. В русском языке при построении текста используются различные формы слова в зависимости от контекста, но в большинстве случаев идею текста можно понять заменив все слова на их нормальные формы, тем самым можно облегчить задачу компьютеру, уменьшив количество вариаций входных данных. Например все формы: 'ездил', 'ездила', 'ездили' после лемматизации становятся равными 'ездить'. Конечно, хоть лемматизация упрощает работу с текстом, после ее выполнения мы теряем часть информации заложенной изначально, но для некоторых задач, например таких как классификация и кластеризация документов, необязательно знать всю информацию заложенную в нем, достаточно различать лишь те признаки, которые позволяют определять объекты одного класса.

1.1 Актуальность работы

Исследование различных алгоритмов лемматизации может помочь наиболее эффективно подбирать соответствующие методы в зависимости от конкретных задач на основе знания их преимуществ и недостатков, а также понимать наиболее важные проблемы и их решения в данной области. Технология NLP широко применяется в браузерах, например, при обработке запроса пользователя, извлекая ключевые идеи текста запроса, а также при индексировании вебстраниц, исправляя ошибки написания документов. Усовершенствование технологии NLP может привести к переходу к Natural Language Understanding(NLU), сделав возможным обработку не самого языка, а образов, которые закладывает человек при написании текста. Также ныне реализованные языковые модели, способные общаться с человеком, поддерживая разговоры на определенную тему, показывают практическую применимость рассматриваемых методов в области машинного обучения.

1.2 Цель и задачи работы

Цель данной работы в описании и практической оценке популярных алгоритмов лемматизации для русского языка, сравнении их преимуществ и недостатков, а также в рассмотрении возможностей их усовершенствования с примерами фреймворков, использующихся

на практике.

2 Различные метрики символьных последовательностей

Основная идея ввода различных метрических характеристик состоит в том, что мы хотим представить слово как элемент некоторого метрического пространства. Если мы будем знать все нормальные формы всех слов, то есть иметь словарь языка, то, в предположении, что словоформа не сильно отличается от своей нормальной формы по введенной метрике, мы сможем найти ближайший элемент из словаря, который и будет являться леммой.

Стоит отметить, что в современных алгоритмах данные метрики используются лишь как один из этапов приведения слова к лемме. Рассмотрим несколько примеров таких метрик.

2.1 Расстояние Левенштейна

Расстояние Левенштейна - метрика, определяющая расстояние между символьными последовательностями по минимальному количеству односимвольных операций: **вставок**, **удалений** и **замены**, необходимых для приведения одной последовательности к другой.

Для определения минимального количества операций определенных выше, необходимо вычислить значение $D(m, n)$, где m - длина слова W_1 , n - длина слова W_2 . Значение $D(m, n)$ можно вычислить по следующей рекуррентной формуле:

$$D(i, j) = \begin{cases} \max(i, j), & \min(i, j) = 0 \\ \min(& \text{иначе} \\ & D(i, j - 1) + 1, \\ & D(i - 1, j) + 1, \\ & D(i - 1, j - 1) + (W_1[i] \neq W_2[j]) \\ &) \end{cases} \quad (2.1)$$

$$(W_1[i] \neq W_2[j]) = \begin{cases} 1, & \text{если } i\text{-тый символ } W_1 \text{ не равен } j\text{-тому символу } W_2 \\ 0 & \text{иначе} \end{cases}$$

Из вида формулы (??) можно получить оценку сложности по времени и памяти для алгоритма, который принимал бы на вход слово длины n и по известному словарю лемм языка, с количеством слов - N , выдавал ближайшую лемму по расстоянию Левенштейна,

которую и будем считать нормальной формой данного слова. Значение оценки по времени есть $O(Nna)$, где a - средняя длина слова в словаре, по памяти - $O(na)$, которую можно сократить до $O(\min\{n, a\})$, так как нас не интересует способ приведения строк к равенству.

Сразу можно увидеть серьезный идейный недостаток данной метрики для задачи лемматизации, а именно то, что расстояние между короткими несовпадающими словами может быть довольно мало, из-за чего результат лемматизации отдельных коротких слов может быть сильно ошибочным, искажающим суть текста.

2.2 Расстояние Джаро

Расстояние Джаро - метрика, определяющая расстояние между символьными последовательностями по их схожести. В отличие от расстояния Левенштейна, чем больше значение метрики, тем более строки похожи друг на друга, так как метрика основана на совпадающих символах.

Расстояние Джаро d_j между двумя словами W_1 и W_2 определяется по следующей формуле:

$$d_j = \begin{cases} 0, & m = 0 \\ \frac{1}{3} \left(\frac{m}{n_1} + \frac{m}{n_2} + \frac{m-t}{m} \right) & \text{иначе} \end{cases}, \quad (2.2)$$

где n_1, n_2 - длины слов W_1 и W_2 , m - количество совпадающих символов, расстояние между местами которых не превосходит $\lfloor \max\{n_1, n_2\} / 2 \rfloor - 1$, t - количество транспозиций, которое равно числу неупорядоченных пар совпадающих символов, чьи номера мест в соответствующих словах различны.

Из формулы (??) получим оценки алгоритма описанного выше, где вместо расстояния Левенштейна будем использовать расстояние Джаро. Сложность по времени сопоставима с использованием расстояния Левенштейна, которая равна $O(Nna)$. Сложность по памяти равна $O(1)$.

Аналогично расстоянию Левенштейна, данная метрика может плохо справляться со словами малой длины, так как из формулы видно, что ценность совпадающего символа, для данной метрики, зависит от длин слов, из-за чего короткие слова могут иметь довольно большое значение d_j .

Также один из недостатков расстояния Джаро состоит в том, что слова состоящие из один и тех же символов на разных местах также могут иметь довольно большое значение $d_j \leq 2/3$, так как высокое значение t штрафует не вычитанием из уже имеющегося, а отсутствием прибавки, что также может негативно сказаться на нашей метрической оценке.

2.3 Расстояние Джаро-Винклера

Расстояние Джаро-Винклера - псевдометрика, являющаяся обобщением рассмотренного ранее расстояния Джаро, основанная на добавке в виде префиксного бонуса для префиксовоспадающих символьных последовательностей. Расстояние Джаро-Винклера d_{jw} между двумя словами W_1 и W_2 определяется по следующей формуле:

$$d_{jw} = \begin{cases} d_j, & d_j < b \\ d_j + lp(1 - d_j) & d_j \geq b \end{cases}, \quad (2.3)$$

где l - длина общего префикса слов W_1 и W_2 , p - коэффициент масштабирования, b - порог усиления, произведение $lp \leq 1$. Преимущество расстояния Джаро-Винклера над предыдущими метриками в том, что оно является трех-параметрическим семейством, что позволяет подбирать соответствующие параметры из практических нужд. Например в стандартной вариации принимают: $p = 0.1$, $l = \min\{l, 4\}$, $b = 0.7$.

Из формулы (??) очевидны оценки сложности по времени - $O(Nna)$, и по памяти - $O(1)$.

Префиксная добавка $lp(1 - d_j)$ направлена на устранение недостатков расстояния Джаро, описанных выше, так как, благодаря подсчету значения длины общего префикса l , короткие совпадающие слова получают большой бонус к значению метрики из-за того, что совпадающий префикс может быть сравним с длиной всего слова. Проблема слов из совпадающих символов на одинаковых местах также частично решается параметром l , так как эти слова не получают префиксную добавку, хотя она, также как и параметр количества транспозиций t , не вычитает величину из уже имеющегося значения при малом l , а лишь не дает прибавку, что не защищает от потенциально высокого значения метрики в $2/3$.

Недостаток расстояния Джаро-Винклера состоит в том, что неправильно подобранный параметр b могут ухудшить результаты по сравнению с расстоянием Джаро на определенных вариантах входных данных, так как большую добавку могут получить неверные варианты нормальной формы, например если нормальная форма слова имеет отличный префикс от тестируемой формы.

2.4 Результаты практического тестирования метрик

Тестирование приведенных выше расстояний символьных последовательностей будем проводить на словах без контекста, взятых из размеченного датасета [1] состоящим из 2000 предложений различной тематики. Оценивать работу алгоритма ламматизации будем на отношении правильных ответов к общему числу запросов, а также рассмотрим их

соотношение по длине слов и частям речи. В качестве словаря нормальных форм используем датасет [1] состоящий из 51733 уникальных слов. В силу приведенных выше оценок времени работы алгоритма для всех метрик - $O(Nna)$, тестирование будет проводиться на случайно выбранной тысячи слов, причем гарантируется, что указанная в размеченном датасете нормальная форма присутствует в словаре, а само слово не является стоп-словом.

Далее примем: L - расстояние Левенштейна, J - расстояние Джаро, $JW(\alpha, \beta, \gamma)$ - расстояние Джаро-Винклера с параметрами $l = \min\{l, \alpha\}$, $p = \beta$, $b = \gamma$. В итоге получены следующие результаты:

J	L	$JW(4, 0.2, 0.7)$	$JW(2, 0.4, 0.7)$	$JW(1, 0.9, 0.7)$
0.595	0.750	0.762	0.764	0.783

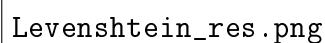
Рассмотрим подробнее структуру полученных данных:

Jaro_res.png

Результаты тестирования расстояния Джаро (J)

JaroВГWinkler_res.png

Результаты тестирования расстояния Джаро-Винклера ($JW(2, 0.4, 0.7)$)

The image is a placeholder for a file named 'Levenshtein_res.png'. It is represented by a large, empty rectangular box with a thin black border.

Результаты тестирования расстояния Левенштейна (L)

Из приведенных выше данных можно увидеть схожую структуру распределения верных ответов. Заметно, что точность для слов длины 3 у всех метрик примерно на одном уровне и выше среднего значения, это может быть связано с тем, что слова такой длины являются довольно уникальными в том, что нормальные формы различных слов, скорее всего, содержат совершенно разные символы из-за малости длины, что делает их легко отличимыми для всех метрик.

Далее, на словах длины 4-7, виден спад точности на всех графиках ниже или на уровне средней точности, этот спад связан с недостатком всех метрик, который упоминался ранее, а именно то, что схожие слова малой длины получают лучшее значение метрики, чем схожие длинные слова, особенно данная проблема видна при использовании расстояния

Джаро, где ,вплоть до слов длины 8, значение точности метрики не увеличивается.

Далее, на словах длины 8-12, виден подъем значения точности, так как, начиная со слов средней длины, недостатки метрик, из-за которых проседала точность коротких слов, уже не оказывают столь сильного влияния, поэтому на данном промежутке длины слов все метрики показывают результат выше или на уровне средней точности.

И наконец, на словах длины 13-16, есть некоторые различия в поведении метрик, например, при использовании расстояния Левенштейна точность оценки снижается сильно ниже среднего, это может быть связано с тем, что количество минимальных операций приведения формы слова к нормальной форме, на основе которого и измеряется расстояние Левенштейна, увеличивается по мере роста длины слова, из-за чего увеличивается вероятность спутать иную нормальную форму с нужной. При использовании расстояния Джаро или расстояния Джаро-Винклера значение точности на уровне среднего, кроме слов длины 14, что, скорее всего, является статистической погрешностью, из-за малой выборки.

При рассмотрении графиков распределения верных ответов от части речи видна очень схожая структура. Значение верных ответов сильно ниже при лемматизации глаголов и прилагательных, это может быть связано с тем, что у этих частей речи наиболее разнообразны формы слова, что и влияет на качество ответов.

3 Представление словарей в памяти

Одной из проблем алгоритмов лемматизации является представление словарей в памяти компьютера. Понятно, что эффективное представление должно занимать не слишком много места, а главное чтобы для них существовали алгоритмы позволяющие уменьшать количество потенциальных нормальных форм, используя информацию из полученной формы слова. Рассмотрим несколько таких видов структур.

3.1 Префиксное дерево

Префиксное дерево представляет собой структуру, с топологической точки зрения являющуюся направленным деревом. Данное дерево имеет единственный корень. Каждый узел дерева может иметь неограниченное количество потомков. Каждый узел хранит список уникальных для узла ключей и потомков, соответствующих каждому ключу, а также флаг. Иногда узел хранит также дополнительное значение.

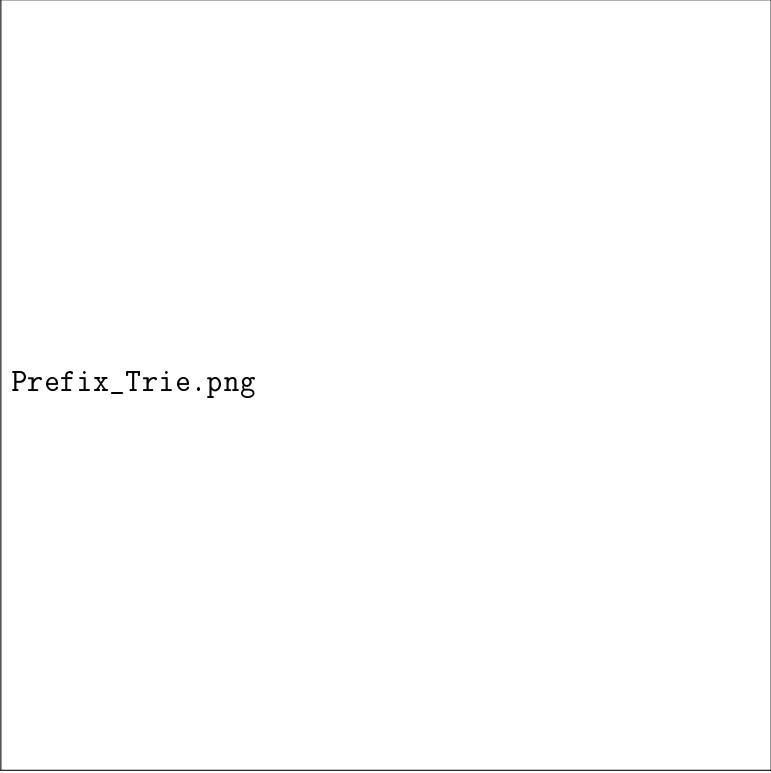
При заполнении дерева каждому слову соответствует путь начинающийся из корня и заканчивающийся в узле с истинным флагом, ключами между узлами в данной цепочке выступают символы слова. Если слова имеют одинаковый префикс, то порожденная ими цепочка будет иметь совпадающие узлы, за счет которых и экономится память.

Очевидно, что по памяти данная структура не превосходит простое хранение всех элементов словаря, а при большом количестве слов может значительно сократить затраты. Также один из плюсов префиксного дерева это быстрая проверка существования слова в словаре за $O(n)$, где n - длина слова.

3.2 Детерминированный ациклический конечный автомат

Детерминированный конечный автомат (DFA) - набор из 5 элементов: $M = (Q, \Sigma, \delta, q_0, F)$, где Q - конечное множество состояний, Σ - конечное множество возможных входных символов (алфавит), $\delta : Q \times \Sigma \rightarrow Q$ - функция перехода, $q_0 \in Q$ - начальное состояние, $F \subset Q$ - множество допускающих состояний.

Изначально автомат находится в стартовом состоянии q_0 . Автомат считывает входные символы $r_1, r_2, \dots, r_n \in \Sigma$, переходя в состояние $q_1, q_2, \dots, q_n \in Q$, где $q_i = \delta(q_{i-1}, r_i)$,



Prefix_Trie.png

Пример префиксного дерева. Красным помечены узлы с истинным флагом

$\forall i = \overline{1, n}$. Процесс продолжается до тех пор, пока не будет достигнут конец входного слова.

Будем говорить, что автомат M допускает слово $r_1 r_2 \dots r_n$, если для каждого символа определена $\delta(q_{i-1}, r_i)$ и $q_n \in F$. Множество всех слов, которые допускает автомат M будем называть языком автомата $L(M)$.

Теперь не сложно заметить, что префиксное дерево является детерминированным ациклическим конечным автоматом (DAFSA), где Q - множество узлов, Σ - символы алфавита, δ - ребра дерева, q_0 - корневой узел, F - узлы с истинным флагом, а так как префиксное дерево топологически является деревом, то оно не имеет циклов. Если к DAFSA применить алгоритм минимизации конечных автоматов, который уменьшает количество возможных состояний Q и при этом множество $L(M)$ оставляет неизменным, можно добиться еще большей экономии памяти.

Стоит отметить, что проверка допустимости слова, то есть проверка существования слова в словаре, также, как и в префиксных деревьях выполняется за $O(n)$, но при этом теперь невозможно хранить дополнительную информацию в узлах, из-за чего всю вспомогательную информацию нужно хранить в отдельных структурах.




Пример минимизации префиксного дерева

3.3 Пример использования префиксного дерева в алгоритме лемматизации

В качестве примера использования префиксного дерева возьмем алгоритм описанный в статье [2]. Сначала создаем префиксное дерево из всех словарных слов, далее, чтобы определить нормальную форму слова мы проверяем все возможные префиксы слова следующим образом:

1. Если префикс оказался допустимым словом, то возвращаем его же.
2. Если префикс оказался недопустимым, то начинаем искать ближайщее допустимое слово в дереве методом обхода в ширину начиная из последней вершины, до которой дошел алгоритм, при этом, ограничивая глубину поиска половиной длины входного префикса.
3. Если при обходе в ширину не найдено ни одного допустимого слова, то возвращаем самое последнее допустимое слово встреченное при работе алгоритма (если такового нет, то возвращаем пустую строку), иначе возвращаем первое найденное допустимое слово.

Если расстояние Левенштейна между всеми результатами обработки префиксов и исходным словом больше длины исходного слова, то возвращаем исходное слово, иначе возвращаем результат с минимальным расстоянием.



Prefix_diag.png

Блок-схема обработки префикса слова в префиксном дереве

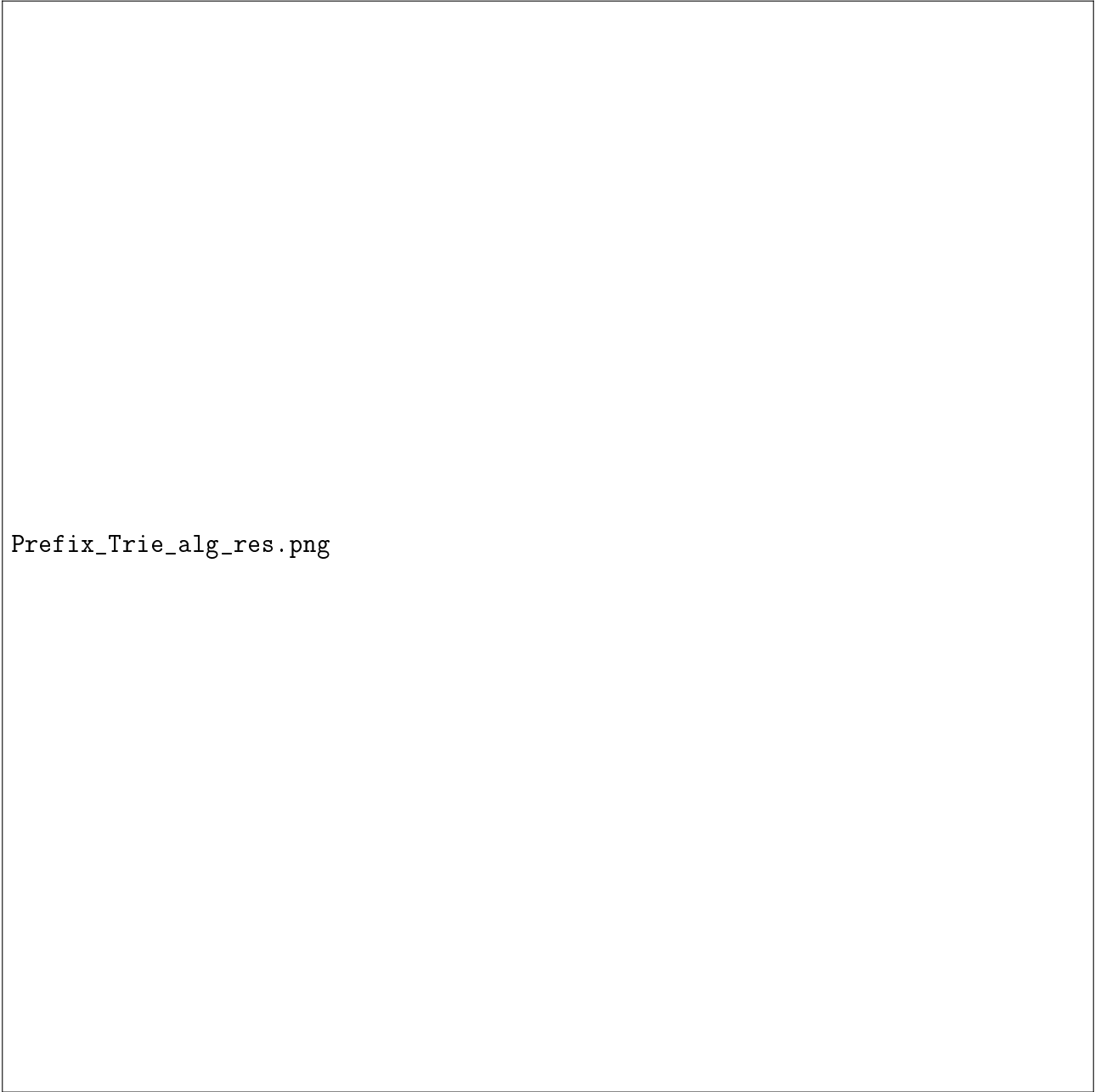
В итоге временная сложность данного алгоритма составит $O(n^3)$, где n - длина исходного слова, что намного лучше временных оценок для алгоритмов, использующих только метрики. Если учитывать, что длина слова в русском языке ограничена, то можно считать, что временная сложность составляет $O(1)$, и, в отличие от рассмотренных ранее алгоритмов, она не зависит от количества слов в словаре.

Протестируем данный алгоритм на всем датасете [1], то есть на 14786 размеченных словах без контекста. Гарантируется, что слово не является стоп-словом. В качестве словаря используем аналогичный датасет [1] из 51733 уникальных слов.

Полученная точность лемматизации составила 0.790, что выше, чем ранее полученные результаты для метрик. Также намного меньшая оценка временной сложности делает данный алгоритм более применимым на практике. Рассмотрим структуру практического тестирования:

Как видим, результаты в зависимости от длины слова довольно слабо разбросаны вокруг средней точности, кроме слов длины 2, 17, 18, 19. Это может быть связано с тем, что слова длиной 17-19 являются довольно редкими в русском языке, из-за чего найти их нормальную форму довольно просто, а высокая точность для слов длины 2 может быть связана с тем, что не так много слов такой длины не являются стоп-словами, из-за чего найти их нормальную форму также не сложно в силу их уникальности.

При рассмотрении графика результатов в зависимости от части речи видно, что доля верных ответов сильно ниже для прилагательных, глаголов и детерминативов, что объясняется тем, что у данных частей речи больше всего вариантов форм слова, из-за чего



Prefix_Trie_alg_res.png

Результаты тестирования алгоритма с использованием префиксного дерева

усложняется процесс лемматизации.

4 Современные алгоритмы лемматизации

В данной главе рассмотрим устройство современных алгоритмов лемматизации, отметим отличительные черты в сравнении между собой и приведенными выше алгоритмами.

4.1 Библиотека `rumorphy2`

Библиотека `rumorphy2` [3] является морфологическим анализатором написанным на языке Python. Она построена на базе словаря OpenCorpora, состоящего из более чем 400 тысяч лексем и 5 миллионов отдельных слов. Данный словарь был обработан следующим образом:

1. Извлечение парадигмы из лексем.
2. Преобразование информации в цифры, если это возможно.
3. Кодировка словаря в формат структуры DAFSA.

Парадигмой называется образец для склонения или спряжения; правила, согласно которым можно получить все формы слов в лексеме для неизменяемой части слова(стеми).

Из обработки 400 тысяч лексем было выделено порядка трех тысяч уникальных парадигм. Далее все парадигмы нумеруются и упаковываются в массив.

Создается структура данных DAFSA, в которую помещаются не просто слова, а строки вида:

< слово >< разделитель >< номер парадигмы >< номер формы в парадигме > .

Данный способ хранения данных позволяет экономить память и упрощать алгоритмы, например, для получения всех вариантов разбора слова достаточно найти все ключи начинающиеся с

< слово >< разделитель > .

Итого имеется 4 массива для хранения суффиксов, префиксов, тегов(грамматической информации) и массив индексов парадигм, на которые затрачивается не более 5 Мб памяти. Созданная структура DAFSA для всех слов занимает примерно 7 Мб памяти.

Алгоритм разбора слова по словарю следующий:

1. Извлечение всех ключей из DAFSA, начинающихся на < слово >< разделитель >.

2. По информации о номере парадигмы и номере слова в парадигме восстанавливаем нормальную форму слова и грамматическую информацию из массивов.

Алгоритм разбора несловарного слова следующий:

1. Отсечение известных и неизвестных префиксов. Если префикс известный, то он отсекается, оставшаяся часть разбирается, после чего склеивается обратно с префиксом. Если не найдено известного префикса, то `rumorphy2` считает первую букву префиксом, потом первые 2, и так далее до 5 букв, проверяя наличие остатка в словаре.
2. Предсказание по суффиксу. Библиотека `rumorphy2` собирает статистику по всем окончаниям до 5 символов. Данная статистика очищается от редких и непродуктивных парадигм и результат кодируется в структуре DAFSA в виде:

< конец слова >< разделитель >< продуктивность >

< номер парадигмы >< номер формы в парадигме > .

Происходит поиск наибольшего суффикса имеющегося в базе данных, на аналогии с которым может проводится разбор неизвестного слова.

3. Сортировка результатов разбора из пункта 2(результаты разборов из пункта 1 в данной версии не сортируются). Полученные результаты сортируются по продуктивности, то есть номера парадигм упорядочены по частоте, с которой эти номера парадигм соответствуют данному окончанию для данной части речи без учета частотности по корпусу, то есть просто используется частотность из словаря.

В итоге библиотека `rumorphy2` занимает около 15 Мб памяти и имеет скорость разбора от 10 до 100 тысяч слов в секунду, точность разбора несловарных слов порядка 90%.

4.2 Алгоритм MyStem

Алгоритм MyStem [4] является морфологическим анализатором компании Yandex, использующийся для обработки запросов пользователей и индексации вебстраниц.

В отличие от библиотеки `rumorphy2`, в качестве структуры хранения словарей используются префиксные деревья, хранящие инвертированные неизменяемые части и суффиксы.

Алгоритм разбора слова по словарю следующий:

1. Обработываем слово справа налево, извлекая из префиксного дерева суффиксов все возможные суффиксы и ставим в соответствие каждому суффиксу его стему.

2. Начинаем обрабатывать все пары стемов и суффиксов, начиная с наибольшего суффикса.
3. Если суффикс словоизменяющей модели, идентификатор которой получен из префиксного дерева стем, совпадает со значением соответствующего суффикса стемы, то мы определили парадигму слова и можем получить всю информацию о слове.

Алгоритм разбора несловарного слова следующий:

1. Обрабатываем слово аналогично словарному случаю, при этом для каждой стемы запоминаем все ветвящиеся узлы при обработке.
2. После обработки всех пар стем-суффикс начинаем поиск из ветвящихся узлов префиксного дерева стем, начиная с самого глубокого, находя все возможные словоизменяющие модели, для которых совпадает соответствующий суффикс.
3. Оцениваем вероятность каждой выбранной модели с помощью машинного обучения в виде наивного байесовского классификатора, в основе которого лежит предположение, что события 'встретить стему слова' и 'встретить суффикс слова' являются независимыми:

$$\begin{aligned}
 P(scheme | word) &= \frac{P(word | scheme)P(scheme)}{P(word)} = \\
 &= \frac{P(stem | scheme)P(suffix | scheme)P(scheme)}{P(word)},
 \end{aligned}$$

используя вероятности из данных проекта Национальный корпус русского языка [5].

4. Наконец, все варианты сортируются по соответствующей вероятности и выбирается один или несколько вариантов разбора.

Также в алгоритме MyStem имеется возможность разбора неизвестного слова по контексту с помощью сверточной нейронной сети MatrixNet, имеющую архитектуру Feature Pyramid Net(FPN).

В итоге алгоритм MyStem занимает около 20 Мб памяти и имеет скорость разбора порядка 100 тысяч слов в секунду, оценка точности разбора несловарных слов без контекста порядка 95% и с контекстом 98%.

5 Заключение

5.1 Результаты работы

В данной работе мы исследовали различные алгоритмы использующиеся в технологии NLP при лемматизации текста. Протестировали и попытались объяснить результаты различных метрик символьных последовательностей, описав их недостатки и убедившись в них на практике. Рассмотрели различные способы хранения словарей для лемматизации в памяти и увидели их практическое применение в современных алгоритмах лемматизации, при сравнении алгоритмов в библиотеке `rumpy2` и алгоритма `MyStem`. В дальнейшем планируется исследовать способы контекстного разбора несловарных слов, полностью разобраться с алгоритмом `MyStem`, изучив работу нейронной сети `MatrixNet`, и перейти к следующим этапам технологии NLP.

6 Список использованных источников

[1] Используемые датасеты для практического тестирования алгоритмов:

<https://github.com/Dx-by-Dy/Python-programs/tree/main/NLP/Datasets>

[2] Md. Kowsher, Anik Tahabilder, Md Murad Hossain Sarker, Md. Zahidul Islam Sanjid, Nusrat Jahan Prottaha | Lemmatization Algorithm Development for Bangla Natural Language Processing:

<https://ieeexplore.ieee.org/document/9306652>

[3] Внутреннее устройство библиотеки `rumpy2`:

<https://pymorphy2.readthedocs.io/en/stable/internals/dict.html>

[4] Принципы работы `MyStem`:

<https://download.yandex.ru/company/iseg-las-vegas.pdf>

[5] Национальный корпус русского языка:

<https://ruscorpora.ru>