Национальный исследовательский университет
Высшая школа экономики

Факультет компьютерных наук

Программа Прикладная математика и информатика

Курсовая работа по теме Использование строковых структур для анализа текстов

Студент: А. А. Урусов

Москва, 2016

1. Введение

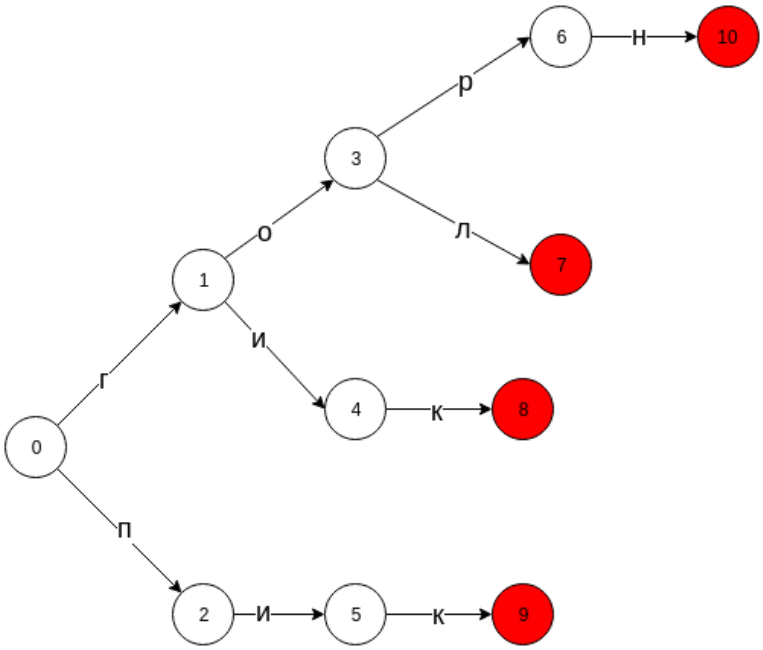
1. Постановка задачи

Дан набор текстов, необходимо разбить их на k кластеров, где k задается экзогенно. При этом

2. Нахождение попарной похожести текстов

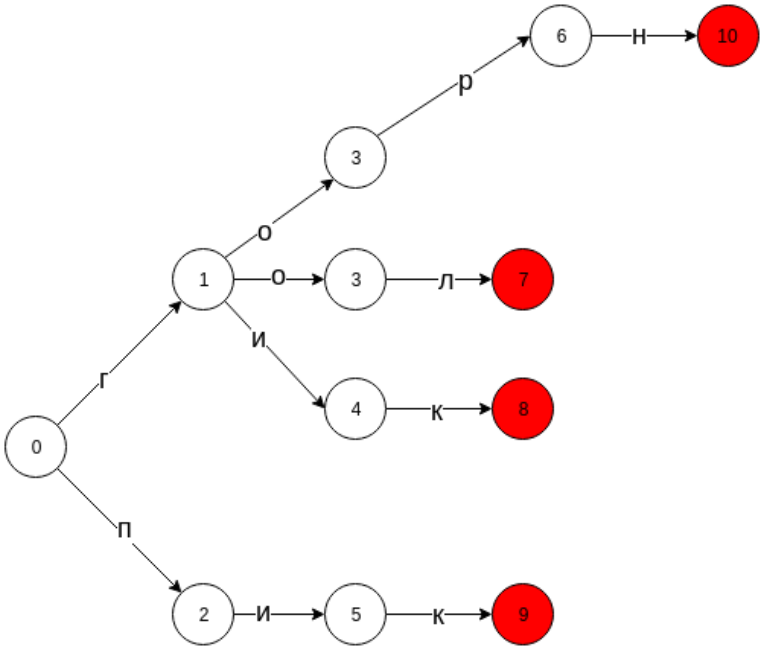
1. Понятие об аннотированном суффиксном дереве

Чтобы определить аннотированное суффиксное дерево, определим сначала более общее понятие *бора*. Итак, *бор* - это структура данных для хранения набора строк, представляющая собой корневой дерево, каждому ребру которого соответствует некоторый символ. При этом некоторые вершины помечены как *терминальные*. Каждой вершине ставится в соответствие строка как последовательность символов, написанных на ребрах между корнем и этой вершиной. Говорят, что бор *принимает* строку s, если есть такая терминальная вершина, которой соответствует строка s.
Приведем пример. Пусть у нас есть набор строк, например, A = {"гол", "горн", "гик", "пик"}. Для него бор будет выглядеть следующим образом:



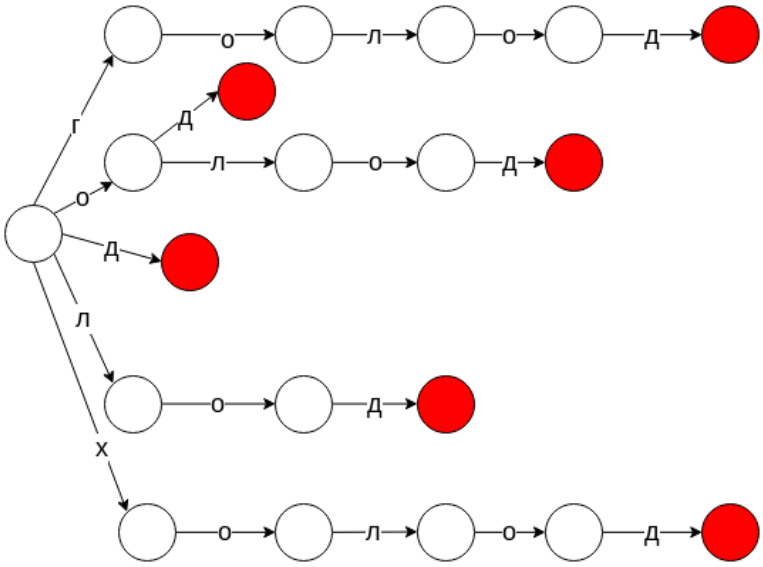
Здесь для удобства все вершины пронумерованы, а терминальные отмечены красным цветом.

Тогда, например, вершине 10 соответствует строка "горн", поскольку последовательность ребер от корня до нее выглядит как $0 \to 1, 1 \to 3, 3 \to 6, 6 \to 10$, и ей соответствует последовательность символов "горн".
Заметим, что в текущем определении есть некоторая неоднозначность. Действительно, например, следующее дерево также удовлетворяет всем условиям:

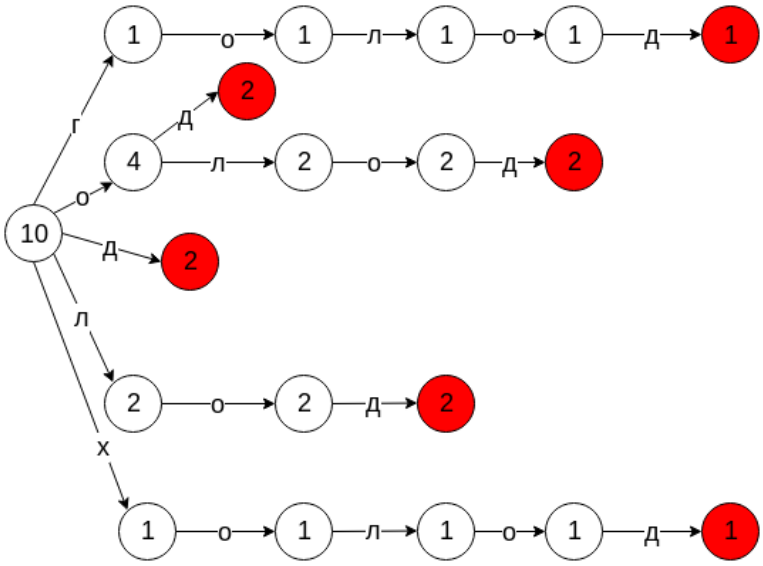


Чтобы избежать этого, потребуем дополнительно, чтобы для каждой вершины v и символа c было не более одного ребра из v в какого-нибудь из ее потомков, соответствующего символу c. Кроме того, потребуем, чтобы все листья были терминальными вершинами, то есть соответствовали какой-нибудь строке из множества (иначе дерево может быть сколь угодно большим). Тогда для любого множества строк бор определяется однозначно. Действительно, пусть есть 2 различных бора для одного и того же множества строк. Найдем ребро, которое есть в одном, но нету во втором, причем если таких несколько, то выберем то, которое ближе всех к корню. Пусть это ребро $v \to to$, ему соответствует символ c, а вершине v - строка s. Поскольку каждый лист - терминальная вершина, какой-то потомок to - терминальная вершина, значит, в множестве строк есть какая-то строка S с префиксом $s + c$. С другой стороны, если мы пойдем от корня по строке s во втором дереве (то есть сначала пойдем по ребру, которому соответствует символ s_0, потом - s_1, и так далее), то мы, очевидно, придем в ту же вершину v. Однако, мы не сможем пойти дальше по символу c по предположению, значит, второй бор не может принимать строку с префиксом $s + c$. Таким образом, мы пришли к противоречию, и тем самым доказали, что бор по множеству строк определяется однозначно. Кроме того, доказательство показывает способ построения этого бора. Пусть изначально бор пустой, то есть состоит из 1й вершины -

корня. Будем последовательно добавлять в него строки следующим образом: будем идти по имеющемуся дереву в соответствии с очередной строкой s, и если в какой-то момент не находим нужного ребра, то просто создаем новое ребро, ведущее из текущей вершины в новую и соответствующее нужному нам символу. В конце помечаем вершину, в которой оказались, терминальной. Очевидно, этот алгоритм не может добавить 2 ребра с одним и тем же символом из одной вершины, потому что мы создаем ребро, только если его еще нет, и каждый лист будет терминальным, потому что каждая вершина соответствует некоторому префиксу какой-то из строк множества, значит, у каждой вершины есть терминальный потомок.
Пусть теперь у нас есть некоторый текст, который мы будем рассматривать как некоторый набор слов. *Суффиксным деревом* для этого текста назовем бор, который принимает суффиксы всех слов и только их.
Снова приведем пример. Пусть есть, например, текст, состоящий из 2х слов "голод" и "холод". Тогда суффиксное дерево должно принимать все суффиксы этих двух слов, то есть строки "голод", "олод", "лод", "од", "д", "холод". Построим бор на этих строках по выше приведенному алгоритму:



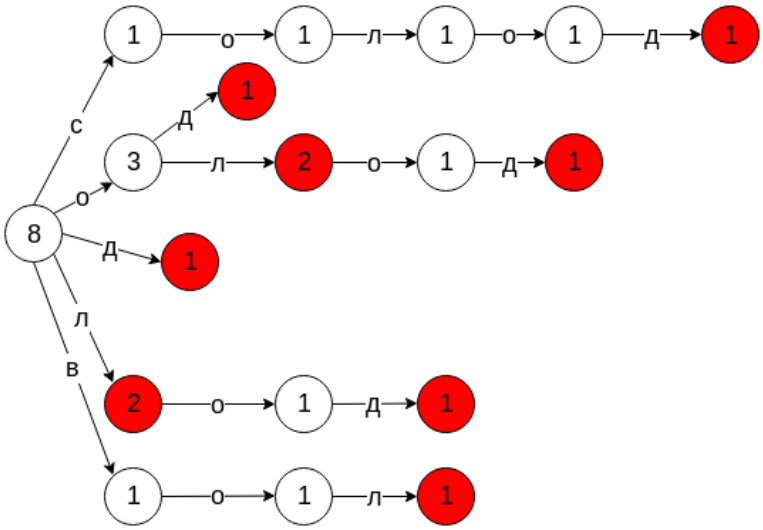
Теперь введем понятие *аннотированного суффиксного дерева*. Аннотированное суффиксное дерево для набора строк - это суффиксное дерево, в котором для каждой вершины дополнительно хранится целое число (частота) - количество строк из набора, префиксом которых является строка, соответствующая этой вершине. Заметим, что это не всегда совпадает с количеством терминальных вершин в поддереве, поскольку в наборе каждая из строк может встречаться несколько раз.
Например, для суффиксного дерева, которое было построено ранее, аннотированное суффиксное дерево будет выглядеть следующим образом:



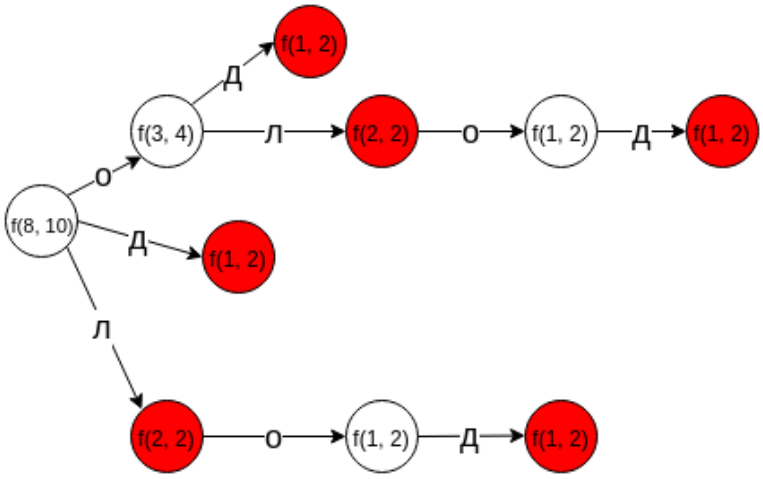
Так, в этом примере суффиксы "олод" , "лод" , "од" , "д" учитывыются по 2 раза.
Отметим, алгоритм, приведенный выше для построения суффиксного дерева легко модифицировать для подсчета частот. Действительно, при добавлении очередного суффикса в дерево значения частот в вершинах пути от корня к вершине, отвечающей за этот суффикс, увеличиваются на 1, а в остальных - не изменяются. Поэтому можно просто добавлять 1 к частоте каждой вершины, которую проходим. У только что созданных вершин это значение должно быть равным 0, поскольку мы еще ни разу не проходили эту вершину.

2. Нахождение величины похожести текстов на основе суффиксных деревьев

Пусть есть 2 текста A, B. Построим по ним аннотированные суффиксные деревья S_A, S_B. Далее найдем для этих суффиксных деревьев *общее поддерево* S, то есть такое подмножество вершин, которые есть в обоих деревьях. При этом в качестве частот в S будем использовать $f(a, b)$, где a, b - частоты вершины в S_A, S_B, а f - некоторая функция. Теперь остается оценить это дерево. В качестве оценки используем величину $\sum\limits_v \frac{frequency(v)}{frequency(p(v))}$, где $frequency(v)$ - частота вершины, а $p(v)$ - предок вершины v. Таким образом, мы суммируем по всем вершинам их условную вероятность появления в тексте, то есть вероятность перехода в вершину v при условии, что мы уже дошли до $p(v)$.
Рассмотрим, например, 2 текста A = {"голод" , "холод"}, B = {"солод" , "вол"}. Для A мы уже строили дерево выше, для B оно выглядит следующим образом:



Тогда общее поддерево имеет следующий вид:



Пусть $f(a,b) = \frac{a+b}{2}$. Тогда оценка похожести равна:

$$\frac{f(3,4)}{f(8,10)} + \frac{f(1,2)}{f(8,10)} + \frac{f(2,2)}{f(8,10)} + \frac{f(1,2)}{f(3,4)} + \frac{f(2,2)}{f(3,4)} + \frac{f(1,2)}{f(2,2)} + \frac{f(1,2)}{f(2,2)} + \frac{f(1,2)}{f(1,2)} + \frac{f(1,2)}{f(1,2)} =$$

$$\frac{3+4}{8+10} + \frac{1+2}{8+10} + \frac{2+2}{8+10} + \frac{1+2}{3+4} + \frac{2+2}{3+4} + \frac{1+2}{2+2} + \frac{1+2}{2+2} + \frac{1+2}{1+2} + \frac{1+2}{1+2} = 5.27778$$

3. Параметризация

Как можно заметить, данный алгоритм нахождения похожести принимает один дополнительный параметр, а именно, функцию $f(a,b)$. В данной работе были использованы следующие функции:

- $f(a,b) = \frac{a+b}{2}$
- $f(a,b) = min(a,b)$
- $f(a,b) = max(a,b)$
- $f(a,b) = \sqrt{ab}$

3. Кластеризация

1. Оценка эффективности кластеризации

Чтобы иметь возможность сравнивать различные алгоритмы кластеризации, нужно привести способ оценивания их эффективности. В данной работе для тестирования использовалось подмножество популярной коллекции Reuters-21578, а именно, были выбраны все тексты из нее, которые принадлежат ровно одной из списка категорий {acq, crude, earn, grain, interest, money-fx, ship, trade}. Таким образом, выделялось 8 кластеров. Для каждого из тестируемых алгоритмов было и для каждого варианта функции из списка выше находится кластеризация, то есть разбиение множества документов на несколько групп, обозначенных метками. Кроме того, была также использована 5я функция, считающаяся стандартной для подобных задач. Она вводится с целью сравнить предлагаемый алгоритм с уже существующими. Далее, с целью вычисления эффективности решения для каждого из найденных кластеров находится самая часто встречающаяся категория, которая и полагается основной для данного кластера, и затем считается доля ошибок, то есть количество документов, категория которых не совпадает с категорией их кластера, деленное на общее количество документов.
Теперь рассмотрим алгоритмы, использованные для кластеризации, а именно, следующие:

- Модификация k-means
- K-medoids
- Spectral clustering

Отметим, что в первых 2х алгоритмах требуется не похожесть, а, наоборот, расстояние между документами, то есть некоторая величина, которая тем больше, чем меньше документы похожи. В качестве такой величины удобно использовать значение $\frac{1}{similarity}$.

2. Модификация k-means

Классический k-means заключается в следующем. Пусть есть сколько-то точек в некотором линейном пространстве. Сначала выбираем случайно k точек - центроиды наших кластеров. Далее итеративно выполняем следующее:

- Пересчитываем кластеры, а именно, для каждой точки множества находим ближайший центроид.
- Пересчитываем центроиды, то есть для каждого из полученных кластеров находим новый центроид как центр масс кластера.

Однако, проблема заключается в том, что у нас нет самих координат точек, а только расстояния между ними (расстояния считаются как $\frac{1}{similarity}$). В связи с этим модифицируем алгоритм следующим образом: будем выбирать изначальные центроиды как элементы нашего множества, а при пересчете будем выбирать тот элемент множества, от которого сумма расстояний до точек кластера минимальна. Таким образом, нам не требуется знать координаты точек, а только попарные расстояния между ними.
Ниже приведены результаты работы этого алгоритма для каждого способа нахождения похожести текстов при 60 итерациях.

- Для $f(a, b) = \frac{a+b}{2}$ результат равен 0.523162.
- Для $f(a, b) = min(a, b)$ результат равен 0.469608.
- Для $f(a, b) = max(a, b)$ результат равен 0.516789.
- Для $f(a, b) = \sqrt{(}ab)$ результат равен 0.514338.
- Для стандартного способа результат равен 0.655392.

3. K-medoids

Идея алгоритма похожа на k-means, но отличается способом пересчета центроидов. Изначально мы инициализируем кластеризацию, выбрав k случайных центроидов и отнеся каждый из остальных элементов в кластер к ближайшему центроиду. Далее, на каждой итерации для каждого документа пробуем поменять его с центром его кластера. Для полученной конфигурации пересчитываем кластеры (т.е. для каждого документа выбираем ближайший центр) и вычисляем некоторую величину, которая показывает, насколько хороша полученная кластеризация. В данном случае удобно использовать сумму расстояний от каждого документа до его центра. Таким образом, чем меньше эта величина, тем лучше кластеризация. Итак, для каждой из таких

замен мы посчитали, насколько улучшится (и улучшится ли вообще) кластеризация. Теперь просто выберем ту, которая дает самое большое улучшение, и применим.
Количество итераций в этом алгоритме, в отличие от k-means, не выбирается фиксированным, а алгоритм выполняется до тех пор, пока у нас получается найти улучшающую замену. При выбранных ограничениях (8000 элементов) на практике требуется всего несколько десятков итераций, поэтому алгоритм достаточно быстро завершает работу. Ниже приведены результаты работы этого алгоритма так же, как это было сделано в предыдущем пункте.

- Для $f(a,b) = \frac{a+b}{2}$ результат равен 0.545833.
- Для $f(a,b) = min(a,b)$ результат равен 0.484681.
- Для $f(a,b) = max(a,b)$ результат равен 0.575.
- Для $f(a,b) = \sqrt(ab)$ результат равен 0.546569.
- Для стандартного способа результат равен 0.648284.

4. Spectral clustering

Этот алгоритм был выбран как самый подходящий из модуля scikit-learn для python. Его суть состоит в нахождении координат точек в k-мерном пространстве по матрице похожести, и после этого применяется k-means к полученным точкам. Ниже приведены результаты работы этого алгоритма.

- Для $f(a,b) = \frac{a+b}{2}$ результат равен 0.6867647058823529.
- Для $f(a,b) = min(a,b)$ результат равен 0.6938725490196078.
- Для $f(a,b) = max(a,b)$ результат равен 0.6843137254901961.
- Для $f(a,b) = \sqrt(ab)$ результат равен 0.6922794117647059.
- Для стандартного способа результат равен 0.6931372549019608.

4. Выводы

На основе проделанного исследования можно сделать несколько выводов.
Во-первых, доля ошибок примерно одинаковая для каждой из четырех функций, которые выбирались как параметры при нахождении похожести текстов. Это не значит, что можно выбирать совершенно любую функцию от двух переменных: все приведенные примеры удовлетворяют условию $min(a,b) \leq f(a,b) \leq max(a,b)$, что логично.

Однако, можно сделать вывод, что выбор функции не является существенным, и поэтому в большинстве случаев можно использовать какую-нибудь достаточно простую, например, среднее арифметическое.
Во-вторых, результаты говорят, что в первых двух алгоритмах стандартный подход к нахождению похожести значительно более эффективен, однако в Spectral clustering они различаются лишь незначительно, что означает возможность применения этого подхода в реальных задачах. Тем не менее, в текущей реализации это не слишком хорошая идея, поскольку генерация таблицы занимает достаточно много времени, соответственно, для использования этого алгоритма его необходимо значительно оптимизировать.