Оглавление

Глава 1

Аннотация

бла-бла-бла

Глава 2

Аналитический обзор литературы

2.1 Базовые методы и подходы

Текстовый корпус – большой и структурированный набор текстов (в настоящее время обычно хранится и обрабатывается в электронном виде). В корпусной лингвистике они используются для статистического анализа и проверки гипотез, проверки происшествий или языковых правил на определенной языковой территории.

Регулярные выражения (англ. regular expressions) - формальный язык поиска и осуществления манипуляций с подстроками в тексте, основанный на использовании метасимволов (символов-джокеров, англ. wildcard characters). Для поиска используется строка-образец (англ. pattern, по-русски её часто называют «шаблоном», «маской»), состоящая из символов и метасимволов и задающая правило поиска. Для манипуляций с текстом дополнительно задаётся строка замены, которая также может содержать в себе специальные символы. [1]

Парсер (англ. parser; от parse – анализ, разбор), или синтаксический анализатор, – часть программы, преобразующей входные данные (как правило, текст) в структурированный формат. Парсер выполняет синтаксический анализ текста.

Стемминг - это процесс нахождения основы слова для заданного исходного слова. [2]

Лемматизация - процесс приведения словоформы к лемме - её нормальной (словарной) форме. [3]

Клитика - это часть слова, которая не может стоять сама по себе и может появиться в тексте только когда оно прикреплено к другому слову. [4]

2.1.1 Регулярные выражения

Одним из невосполнимых успехов в стандартизации в информатике стали регулярные выражения (RE), язык для уточнения строк текстового поиска. Этот язык используется во всех языках программирования, текстовых процессорах и инструментах обработки текста, таких как инструменты Unix grep или Emacs. Формально регулярное выражение - это алгебраическое обозначение для характеристики набора строк. Они особенно полезны для поиска в текстах, когда есть шаблон для поиска и корпус текста для поиска. Функция поиска по регулярному выражению будет искать через корпус, возвращая все тексты, которые соответствуют шаблону. Корпус может быть отдельным документом или коллекцией. Например, инструмент командной строки Unix grep принимает регулярное выражение и возвращает каждую строку входного документа, которая соответствует выражению. Поиск может быть спроектирован так, чтобы возвращать каждое совпадение в строке, если их больше одного или только самое первое совпадение. Регулярные выражения бывают разных видов. Далее будут описываться описывать расширенные регулярные выражения; различные парсеры регулярных выражений могут распознавать только их подмножества, или трактуют некоторые

выражения немного по-другому. Использование онлайн-тестера регулярных выражений - это удобный способ проверить свои выражения и изучить эти варианты.

Character	Description	Example
[]	A set of characters	"[a-m]"
\	Signals a special sequence (can also be used to escape special characters)	"\d"
.	Any character (except newline character)	"he..o"
^	Starts with	"^hello"
$	Ends with	"world$"
*	Zero or more occurrences	"aix*"
+	One or more occurrences	"aix+"
{}	Exactly the specified number of occurrences	"al{2}"
|	Either or	"falls|stays"
()	Capture and group	

Рисунок 1 – регулярные выражения в языке программирования Python

2.1.2 Нормализация текста

Перед обработкой текста практически на любом естественном языке текст должен быть нормализован. Как минимум три задачи входят в любой процесс нормализации:

1. Токенизация (сегментирование) слов
2. Нормализация форматов слов
3. Сегментирование предложений

Токенизатор также может быть использован для расширения клитикальных сокращений. В зависимости от приложения алгоритмы токенизации могут также токенизировать выражения из нескольких слов, такие как «New York» или «Rock 'n' Roll», как один токен, для чего требуется некоторый словарь выражений из нескольких слов. Таким образом, токенизация тесно связана с обнаружением именованных объектов, задачей обнаружения имен, дат и организации.

Один широко используемый стандарт токенизации известен как стандарт токенизации Penn Treebank, используемый для проанализированных корпусов (treebanks), выпущенных Linuistic Data Consortium (LDC), источником многих полезных наборов данных. Этот стандарт выделяет клитики, объединяет слова, написанные через дефисы, и выделяет все знаки препинания. На практике, поскольку токенизация должна выполняться перед любой другой языковой обработкой, она должна быть очень быстрой. Поэтому стандартный метод токенизации – использовать детерминированные алгоритмы, основанные на регулярных выражениях, скомпилированных в очень эффективные конечные автоматы.

Токенизация в таких языках, как письменный китайский, японский и тайский, намного сложнее, так как они не используют пробелы для обозначения потенциальных границ слов. В китайском языке, например, слова состоят из символов (называемых ханзи). Каждый символ обычно представляет одну единицу значения (называется морфема) и произносится как один слог. Длина слов в среднем составляет около 2,4 символов. Но решить, что считать словом на китайском, сложно.

Существует третий вариант токенизации текста. Вместо того, чтобы определять токены как слова или в виде символов (как на китайском), мы можем использовать данные для автоматического определения, какого размер токены должны быть. Возможно, иногда нам могут потребоваться токены – разделенные пробелами слова; в других случаях полезно иметь токены размера больше, чем слова (например, New York Times), а иногда меньше, чем слова (например, морфема не-) Морфема - это наименьшая смысловая единица языка.

2.1.3 Byte Pair Encoding

Причина, по которой полезно иметь токены подслов - когда обращаешься с неизвестными словами. Неизвестные слова особенно актуальны для систем машинного обучения. Они часто узнают некоторые факты о словах в одном корпусе (учебный корпус), а затем используют эти факты для принятия решений слов в тестовом корпусе. Таким образом, если наш учебный корпус содержит, скажем, слова «низкий» и «нижайший», но не «ниже», но слово «ниже» появляется в тестовом корпусе, система не будет знать, что с этим делать. Решением этой проблемы является использование своего рода токенизации, в которой большинство токенов являются словами, но некоторые токены являются частыми морфемами или другими подсловами, такими как «-ший», так что не появившееся в учебном корпусе слово может быть представлено путем объединения частей.

Самый простой такой алгоритм - это кодирование байтовой пары, или BPE (Byte Pair Encoging, также известный как Digram Coding [5]).

Алгоритм начинается с набора символов, равного набору букв в алфавите. Каждое слово представлено в виде последовательности символов плюс специальный символ конца слова. На каждом шаге алгоритма мы подсчитываем количество пар букв, находим наиболее часто встречающуюся пару (A, B) и заменяем ее новым символом (AB). Мы продолжаем считать и объединять, пока не сделали k слияний; k является параметром алгоритма. Результирующий набор символов будет состоять из исходного набора букв плюс k новых символов. Конечно, в реальных алгоритмах BPE выполняется со многими тысячами слияний на очень больших входных словарях. В результате большинство слов будет представлено как один символ и только очень редкие слова (и неизвестные слова) должны быть представлены по частям.

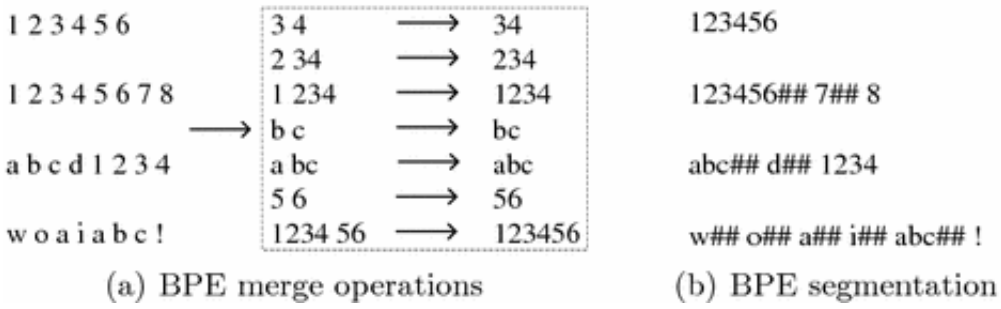

Рисунок 2 – пример выполнения алгоритма
BPE с 7 операциями слияния

2.1.4 WordPiece

Есть несколько альтернатив алгоритму BPE. Как и BPE, алгоритм WordPiece начинается с некоторой простой токенизации (например, по пробелам), а затем разбивает получившиеся грубые лексемы на токены подслов. Модель WordPiece отличается от BPE тем, что специальные маркеры границ слова появляется в начале слова, а не в конце, и в том, как он объединяет пары. Вместо того, чтобы объединять наиболее часто встречающиеся пары, WordPiece объединяет те пары, которые максимизируют схожесть текста с выбранной языковой моделью на обучающей выборке. Тогда каждое слово токенизируется с использованием жадного алгоритма с самым длинным соответствием префиксу. Это отличается от алгоритма декодирования, который был введен для BPE, выполняющий слияния на тестовом тексте в том же порядке они были извлечены из учебного набора. Жадное декодирование с самым длинным соответствием префиксу иногда называют максимальным соответствием или максимальным паросочетанием.

2.1.5 Нормализация слов, леммализация, стемминг

Нормализация слов - это задание слов / токенов в стандартном формате, выбор одной нормальной формы для слов с несколькими формами.

Сжатие регистра - переводит все слова в тексте в нижний регистр, это очень полезно для обобщения во многих задачах, таких как поиск информации или распознавание речь. Для анализа тональности текста и других задач классификации текста, извлечения информации и машинного перевода, напротив, регистр может оказаться весьма полезными и сжатие вообще не применяется.

Лемматизация определяет, имеют ли два слова имеют одинаковый корень, несмотря на их поверхностные различия. Например, слова *am*, *are* и *is* имеют общую лемму *be*; Самые сложные методы лемматизации предполагают полный морфологический анализ слова. (Морфология - наука о том, как слова строятся из более мелких значащих единиц, называемых морфемами)

Алгоритмы лемматизации могут быть достаточно сложными. По этой причине иногда используются более простые, но грубые методы, который в основном заключаются в обрезке аффиксов окончаний слова. Этот наивный вариант морфологического анализа называется *стемминг*. Одним из наиболее распространенных алгоритмов стемминга является стеммирование Портера (1980). [6]

Стеммер Портера, примененный к следующему параграфу:

This was not the map we found in Billy Bones's chest, but an accurate copy, complete in all things-names and heights and soundings-with the single exception of the red crosses and the written notes.

производит такой стеммированный вывод:

Thi wa not the map we found in Billi Bone s chest but an accur copi complet in all thing name and height and sound with the singl except of the red cross and the written note

2.1.6 Расстояние Левенштейна.

Расстояние Левенштейна [7] или Minimum edit distance между двумя строками определяется как минимальное количество операций редактирования (такие операции, как: вставка, удаление, замена) необходимых для того, чтобы преобразовать одну строку в другую. Разрыв между *intention* и *execution*, например, равен 5 (удалить i, заменить e на n, заменить x на t, вставить c, заменить u на n). Намного легче увидеть это, посмотрев на Рисунок 3.

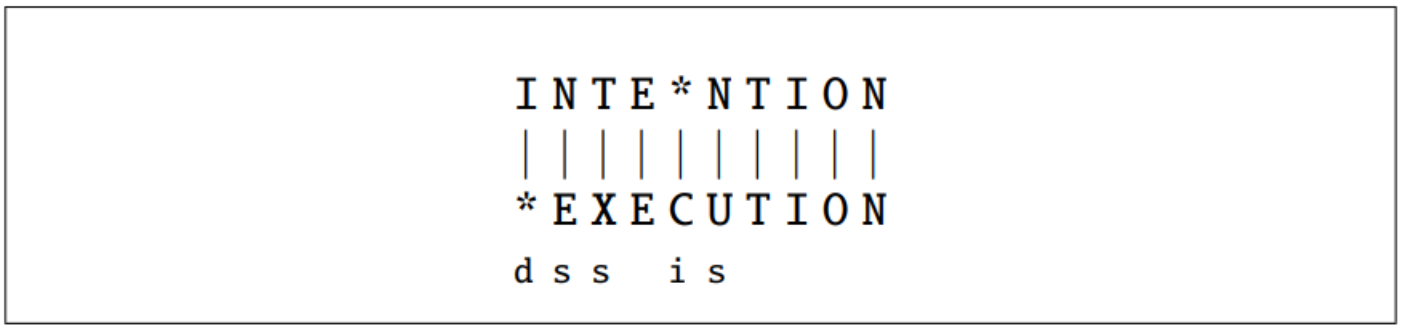

Рисунок 3 – Расстояние Левенштейна между словами "intention"и "execution".
Описание операций: d - удаление, s - замена, i - вставка.

2.2 Языковые модели основанные на N-граммах.

Модели, которые сопоставляют последовательностям слов вероятности называются *Language Models* (*LM*, *Языковые Модели*).

N-грамма - это последовательность из N слов. 2-грамму называют *биграммой* (*bigram*), 3-грамму называют *триграммой* (*trigram*). В области обработки естественного языка N-граммы используется в основном для предугадывания на основе вероятностных моделей. N-граммная модель рассчитывает вероятность последнего слова N-граммы, если известны все предыдущие. При использовании этого подхода для моделирования языка предполагается, что появление каждого слова зависит только от предыдущих слов. [8]

Другим применением N-грамм является выявление плагиата. Если разделить текст на несколько небольших фрагментов, представленных N-граммами, их легко сравнить друг с другом и таким образом получить степень сходства анализируемых документов[9]. N-граммы часто успешно используются для категоризации текста и языка. Кроме того, их можно использовать для создания функций, которые позволяют получать знания из текстовых данных. Используя N-граммы, можно эффективно найти кандидатов, чтобы заменить слова с ошибками правописания.

2.2.1 N-граммы.

Рассмотрим задачу вычисления *P(w|h)* - вероятности, что следующим словом является *w*, при данной истории *h*. Пусть история h: *its water is so transparent that*, нужно узнать вероятность того, что следующим словом является *the*, то есть, нужно найти:

$$P(\textit{the} \mid \textit{its water is so transparent that})$$

Один из способов оценить эту вероятность - по относительным частотам: можно взять очень большой корпус, подсчитать, сколько раз встречается *its water is so transparent that*, подсчитать, сколько раз за этим последует *the*. Тогда это было бы ответом на вопрос "Мы видели историю h некоторое число раз, сколько раз следующее слово было *the*":

$$P(\textit{the} \mid \textit{its water is so transparent that}) = \frac{C(\textit{its water is so transparent that the})}{C(\textit{its water is so transparent that})}$$

С огромным корпусом (например, Глобальная Сеть) можно рассчитать и посчитать вероятности с помощью этой формулы.

Метод расчета вероятностей таким методом работает в многих случаях, но оказывается, что даже Глобальная Сеть недостаточно велика, чтобы дать хорошие оценки в большинстве случаев. Это происходит потому, что новые предложения создаются все время.

Похожим образом, если нужно узнать вероятность появления какой-то последовательности из n слов, нужно посчитать, сколько раз она появляется вреди всем последовательностей из n слов. На это требуется огромная вычислительная сила.

Есть способ умнее: обозначим вероятность того, что конкретное слово X_i примет значение w_i или $\mathrm{P}(X_i = w_i)$ за $\mathrm{P}(w_i)$. Обозначим последовательность N слов как $\mathrm{w}_1, \mathrm{w}_2, \ldots, \mathrm{w}_\mathrm{n}$ и как $\mathrm{w}_1{}^\mathrm{n}$. Обозначим совместную вероятность того, что каждое слова в последовательности имеет какое-то значение $P(X = w_1,\ Y = w_2,\ \ldots,\ W = w_n)$ за $P(w_1,\ w_2,\ \ldots,\ w_n)$. Теперь, чтобы вычислить вероятность целой последовательности $P(w_1,\ w_2,\ \ldots,\ w_n)$ можно воспользоваться цепным правилом вероятностей:

$$P(X_1, \ldots, X_\mathrm{n}) = P(X_1)P(X_2|X_1)P(X_3|X_1{}^2)\ldots P(X_\mathrm{n}|X_1{}^{\mathrm{n-1}})$$

Цепное правило показывает взаимосвязь между вычислением совместной вероятности последовательности и вычислением условной вероятности слова, при данных предыдущих словах.

Предыдущая формула предполагает, что можно оценить совместную вероятность целой последовательности слов умножая между собой условные вероятности. Но похоже, что цепное правило не сильно-то и помогает, так как неизвестен способ вычисления точной вероятности слова после длинной последовательности предыдущих слов. $P(X_n \mid X_1^{n-1})$. Но с помощью N-грам можно аппроксимировать историю по нескольким предыдущим словам.

Биграмма, например, аппроксимирует вероятность слова по 1 предыдущему слову, то есть $P(w_n \mid w_1^{n-1}) \approx P(w_n \mid w_{n-1})$.

Предположение, что вероятность слова зависит только от предыдущего слова называется Марковым предположение. Марковские модели - это класс вероятностных моделей, чтобы предполагают, что можно предсказать вероятность какого-то будущего события не смотря слишком далеко назад. Можно обобщить биграмму (которая смотрит только на одно слово в прошлое) до триграммы (которая смотрит на 2 слова в прошлое), а значит, по индукции, до N-граммы (которая смотрит на n-1 слово в прошлое).

Таким образом, общее уравнение для этой N-граммной аппроксимации условной вероятности следующего слова в предложении таково:

$$P(w_n \mid w_1^{n-1}) \approx P(w_n \mid w_{n+1-N}^{n-1})$$

При заданном биграмном предположении для вероятности отдельного слова, можно вычислить вероятность полной последовательности слов:

$$P(w_1^n) \approx \prod_{k=1}^{n} P(w_k \mid w_{k-1})$$

Для того, чтобы оценить вероятности этих биграм или n-грам можно использовать *метод наибольшего правдоподобия* (*maximum likelihood estimation*, *MLE*).

Мы получаем оценки MLE для параметров N-граммы подсчитав их в корпусе и нормализовав их таким образом, чтобы они лежали между 0 и 1.

Например, чтобы вычислить конкретную биграммную вероятность слова y при предыдущем слова x, нужно вычислить количество биграмм $C(xy)$ и нормализовать по сумма всех биграмм, у которых первое слово x.

$$P(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n)}{\sum_w C(w_{n-1}w)}$$

Эту формулу можно упростить, заметив, что количество биграмм, начинающихся с w_{n-1} должно быть равно количество слов w_{n-1}.

$$P(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n)}{C(w_{n-1})}$$

Для общего случая MLE, оценка параметра n-граммы:

$$P(w_n|w_{n-N+1}^{n-1}) = \frac{C(w_{n-N+1}^{n-1}w_n)}{C(w_{n-N+1}^{n-1})}$$

Эта формула оценивает вероятность N-граммы деля увиденную частоту конкретной последовательности на увиденную частоту префикса. Это отношение называется *относительной частотой*.

В практике применяются триграммы, 4-граммы или даже 5-граммы, в зависимости от объема обучающей выборки.

Также, чаще всего используются *логарифмические вероятности*, так как перемножение большого числа вероятностей может дать слишком маленькое число, которое может "сломать"разрядную сетку вещественного числа в цифровом представлении. При использовании логарифмов умножение заменяется на сложение, следовательно, число не будет слишком мало.

$$p_1 \times p_2 \times p_3 \times p_4 = exp(log(p_1) + log(p_2) + log(p_3) + log(p_4))$$

2.2.2 Оценка языковых моделей

Лучший способ оценить производительность языковой модели - это встроить её в приложение и вычислить, насколько приложение улучшается. Такой подход называется *внешней оценкой*. Внешняя оценка - это единственный способ узнать, помогает ли в решении задачи конкретное улучшение какой-то компоненты. Таким образом, для распознавания речи можно сравнить производительность двух языковых моделей, запустив дважды распознаватель и посмотреть, какой запуск дает более точный перевод.

К сожалению, запуск больших NLP систем от начала до конца - это очень дорого. Вместо этого хорошо бы иметь метрику, которая может дать быструю оценку потенциального улучшения в языковой модели. Метрика *внутренней оценки* - это одна из мер качества модели, вне зависимости от приложения.

Для внутренней оценки языковой модели требуется *тестовая выборка*. Как и с многими другими статистическими моделями, вероятности N-граммовой модели выходят из корпуса, на котором модель обучалась, которая называется *обучающей выборкой*. После этого можно оценить качество модели на основе её производительности на каких-то еще неиспользованных данных, которые и называются *тестовой выборкой*.

Если задан какой-то корпус текста, он делится на обучающую и тестовую выборки, после этого обучаются на обучающей и проверяются на тестовой. Потом происходит сравнение двух обученных моделей: насколько хорошо они удовлетворяют тестовой выборке.

Очень важно не допустить того, чтобы информация из тестовой выборки встречалась в обучающей. Если это происходит, попавшему туда предложению ошибочно ставится большая вероятность. Это называют *обучением на тестовой выборке*.

Иногда тестовая выборка используется настолько часто, что модель явным образом подгоняется под её характеристики. Для того, чтобы этого избежать, используют свежие данные, не использованные ни в тестовой, ни в обучающей выборке. Эти данные называют *валидационной выборкой* (*development test set*, *devset*).

2.2.3 Перплексия.

На практике, вероятности не используются для метрики оценки языковой модели. Вместо этого используется *перплексия* (иногда называют PP). Перплексия языковой модели на тестовой выборке - это обратная величина к вероятности на тестовой выборке, нормализованная количкством слов. Для тестовой выборки $W = w_1 w_2 \dots w_N$:

$$PP(W) = P(w_1 w_2 \dots w_n)^{-\frac{1}{N}}$$

Можно использовать цепное правило, чтобы раскрыть вероятность W:

$$PP(W) = (\prod_{i=1}^{N} \frac{1}{P(w_i|w_1 \dots w_{i-1})})^{\frac{-1}{N}}$$

Если вычислять перплексию с помощью биграммной языковой модели:

$$PP(W) = (\prod_{i=1}^{N} \frac{1}{P(w_i|w_{i-1})})^{\frac{-1}{N}}$$

Можно представить перплекцию по-другому, *как взвешенное среднее количество сыновей вершины графа*. Количество сыновей - это количество различных слов, которые могут последовать за текущим.

2.2.4 Неизвестные слова.

Неизвестные слова не могут появиться в тестовой выборке, если используется *закрытый словарь*, который содержит все слова в лексиконе. В других случаях придется обрабатывать слова, никогда ранее не встречавшиеся, назовем такие слова *неизвестными* или *словами вне словаря* (*out of vocabulary*, OOV). Процент неизвестных слов, которые попадаются в тестовой выборке, называют *мерой неизвестности* (OOV *rate*). Система с *открытым словарем* - в которой неизвестные слова добавляются как псевдо-слово <UNK>.

Есть несколько способов обучить вероятности слов <UNK>. Один из них - это свести задачу назад к закрытому словарю, добавив слово <UNK> в словарь на стадии нормализации текста, после этого обращаясь к нему как к любому другому слову.

2.2.5 Сглаживание.

Самый простой способ - это просто добавить один по всем счетам биграмм перед тем, как производить нормализацию их в вероятности. Этот алгоритм называется *сглаживанием Лапласа*. Оно недостаточно хорошо себя показывает, чтобы использовать его в современных моделях на N-граммах, но дает множество полезных идей, которые используются в других алгоритмах сжатия, также является практичным алгоритмам для таких задач, как *классификация текста*.

Оценка вероятности появления слова w_i - это его количество $_i$, нормализованное общим числом слов N.

$$P(w_i) = \frac{c_i}{N}$$

Сглаживание Лапласа добавляет 1 ко всем счетам. Всего есть V уникальных слов в словаре, к каждому добавляется 1, значит знаменатель увеличивается на V.

$$P_{Laplace}(w_i) = \frac{c_i + 1}{N + V}$$

Сглаживание для вероятностей биграмм:

$$P^*_{Laplace}(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n) + 1}{\sum_w (C(w_{n-1}w) + 1)} = \frac{C(w_{n-1}w_n) + 1}{C(w_{n-1}) + V}$$

Альтернативный способ сглаживания - прибавлять ко всем счетам не 1, а k. Этот алгоритм называется *«прибавь k»-сглаживание* (*add-k smoothing*).

$$P^*_{Add-k}(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n) + k}{\sum_w (C(w_{n-1}w) + k)} = \frac{C(w_{n-1}w_n) + 1}{C(w_{n-1}) + kV}$$

2.2.6 Интерполяция.

Иногда, когда требуется вычислить $P(w_n|w_{n-1})$, но нет примера триграммы $w_{n-2}w_{n-1}w_n$, можно оценить вероятность используя биграммную вероятность $P(w_n|w_{n-1})$. Похожим образом, если нет примеров биграммы $P(w_n|w_{n-1})$, можно воспользоваться $P(w_n)$.

Примером такого использования является *интерполяция*, она вычисляет оценку вероятности взвешивая и комбинируя триграмму, биграмму и юниграмму.

В простой линейной интерполяции оценки комбинируются линейно:

$$P^{'}(w_n|w_{n-2}w_{n-1}) = \lambda_1 P(w_n|w_{n-2}w_{n-1}) + \lambda_2 P(w_n|w_{n-2}) + \lambda_3 P(w_n)$$

$$\sum_i \lambda_i = 1$$

Способ сложнее - давать каждой λ вес, зависящий от контекста. Если имеются точные счеты для какой-то биграммы, можно сделать предположение, что счеты для триграмм, основанные на этой биграмме будут заслуживать большего доверия, а значит можно дать соответствующей λ больший вес:

$$P'(w_n|w_{n-2}w_{n-1}) = \lambda_1(w_{n-2}^{n-1})P(w_n|w_{n-2}w_{n-1}) + \lambda_2(w_{n-2}^{n-1})P(w_n|w_{n-2}) + \lambda_3(w_{n-2}^{n-1})P(w_n)$$

Для того, чтобы вычислить все λ_i, пользуются *сохранённым* (*held-out*) корпусом - дополнительным обучающей выборкой, которую используют для настройки параметров системы, таких, как λ.

2.3 Подходы и методы, основанные на наивном Байесе.

Наивный Байес зачастую применяется к задаче *категоризации текста* - присваиванию категории к целым текстам или документам, например, к *анализу тональности текста*, то есть выделению, например, положительного или отрицательного настроя автора к какому-либо объекту.

Обнаружение спама - еще одно приложение. Задача бинарной классификации, состоящая в том, чтобы дать электронному письму оценку, является ли оно спамом.

Цель классификации состоит в том, чтобы произвести простые наблюдения, выделить какие-либо полезные черты и классифицировать наблюдения в какой-то дискретный класс.

2.3.1 Наивный Байес.

Интуитивное описание показано на рисунке 4. Текст представляется как *мешок слов*, то есть неупорядоченное множество слов, их взамное расположение проигнорировано, сохранены только частоты появления слова в тексте.

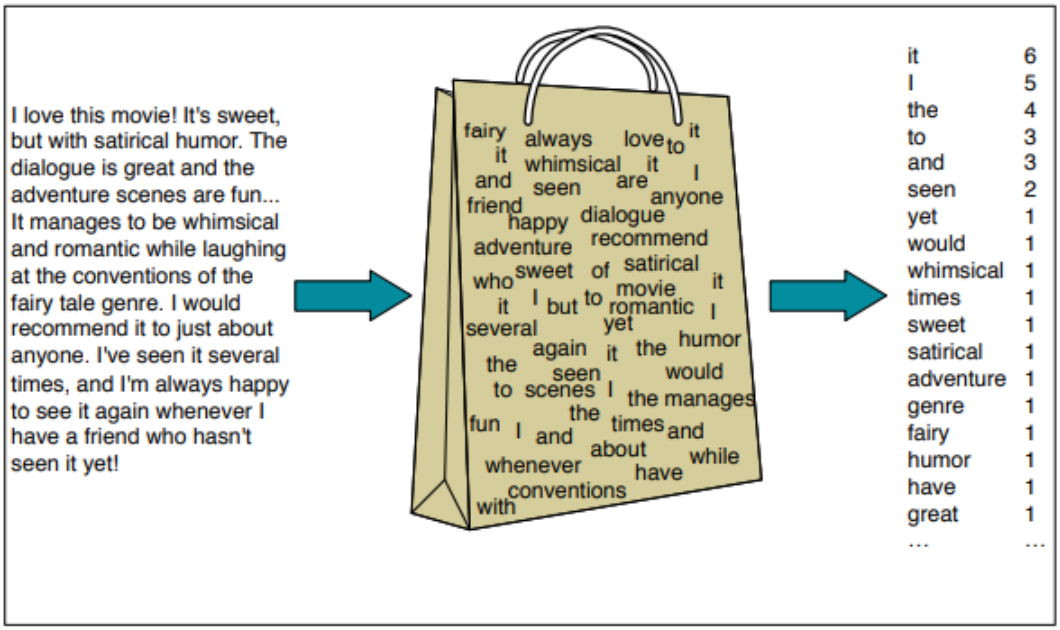

Рисунок 4 - Интуитивное описание классификатора на наивном Байесе, применённое к обзору фильма. Взаимное расположение слов игнорировано (взято предположение *мешка слов*).

Наивный Байес - это вероятностный классификатор, означающий, что для документа d из всех классов $c \in C$ классификатор возвращает класс $\hat{c}$ который имеет максимальную вероятность встречи в тексте. В следующей формуле нотация $\hat{}$означает предполагаемый корректный класс.

$$\hat{c} = \underset{c \in C}{argmax} P(c|d)$$

Эта идея *Байесовского вывода* была известна с момента публикации работы Байеса (1763)[10], и была впервые применена к классификации текста Мостеллером и Уоллесом (1964)[11]. Идея Байесовского классификатора - использовать правило Байеса и трансформировать предыдущее уравнение в другие вероятности, имеющие полезные свойства. Правило Байеса представлено в следующем уравнении. Оно дает способ разбить условную вероятность $P(x|y)$ в три другие вероятности:

$$P(x|y) = \frac{P(y|x)P(x)}{P(y)}$$

Используя предыдущую формулу.

$$\hat{c} = \underset{c \in C}{argmax} P(c|d) = \underset{c \in C}{argmax} \frac{P(d|c)P(c)}{P(d)}$$

Знаменатель можно опустить.

$$\hat{c} = \underset{c \in C}{argmax} P(c|d) = \underset{c \in C}{argmax} P(d|c)P(c)$$

Таким образом вычисляется самый вероятный класс $\hat{c}$ по заданному документу d с помощью класса который имеет наибольшее произведение двух вероятностей: априорная вероятность класса $P(c)$ и функция правдоподобия документа $P(d|c)$.

$$\hat{c} = \underset{c \in C}{argmax} P(c|d) = \underset{c \in C}{argmax} \overset{likelihood}{P(d|c)} \overset{prior}{P(c)}$$

Не умаляя общности, можно представить документ d как набор черт $f_1, f_2, \ldots, f_n$.

$$\hat{c} = \underset{c \in C}{argmax} P(\overset{likelihood}{f_1, f_2, \ldots, f_n|c}) \overset{prior}{P(c)}$$

К сожалению, даже эта формула тяжела для вычисления: без какиих-либо упрощающих предположений оценка вероятность любой возможной комбинации черт потребует огромного числа параметров и невозможно большую обучающую выборку. Поэтому Наивный Байес делает два упрощающих предположения.

Первое - это *мешок слов* (*bag of words*): предположение, что позиция слова не имеет значения. Например, слово *любовь* будет иметь один и тот же эффект вне зависимости от того, чтоит оно на первом, двадцатом или последнем месте в тексте.

Второе часто называют *предположением наивного Байеса*: это предположение условной независимости того, что вероятности $P(f_i|c)$ независимы при данном классе c и могут быть «наивно» перемножены таким способом:

$$P(f_i, f_2, \ldots f_n|c) = P(f_1|c)P(f_2|c) \ldots P(f_n|c)$$

Результирующее уравнение для класса выбранного наивным Байесом таково:

$$C_{NB} = \underset{c \in C}{argmax} P(c) \prod_{f \in F} P(f|c)$$

Для того, чтобы применить наивного Байеса к тексту, нужно учесть позиции, просто проверяя индексом каждую позицию слова в документе:

positions $\leftarrow$ все позиции слов в тестовом документе

$$C_{NB} = \underset{c \in C}{argmax} P(c) \prod_{i \in positions} P(w_i|c)$$

Литература

1. Фридл, Дж. Регулярные выражения = Mastering Regular Expressions. - СПб.: «Питер» , 2001. - 352 с. - (Библиотека программиста). - ISBN 5-318-00056-8.

2. Lovins, Julie Beth. Development of a Stemming Algorithm // Mechanical Translation and Computational Linguistics. - 1968. - T. 11.

3. Маннинг К., Рагхаван П., Шютце Х. Введение в информационный поиск. - Вильямс, 2011. - 512 с. - ISBN 978-5-8459-1623-5.

4. Crystal, David. A First Dictionary of Linguistics and Phonetics. Boulder, CO: Westview, 1980. Print.

5. Ian H. Witten, Alistair Moffat, and Timothy C. Bell. Managing Gigabytes. New York: Van Nostrand Reinhold, 1994. ISBN 978-0-442-01863-4.

6. P. Willett. The Porter stemming algorithm: then and now (англ.) // Program: Electronic Library and Information Systems. 2006. Vol. 40, iss. 3. P. 219?223. ? ISSN 0033-0337.

7. В. И. Левенштейн. Двоичные коды с исправлением выпадений, вставок и замещений символов. Доклады Академий Наук СССР, 1965. 163.4:845-848.

8. Jurafsky, D. and Martin, J.H. Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition. Pearson Prentice Hall, 2009. ? 988 p. ? ISBN 9780131873216.

9. Proceedings of the ITAT 2008, Information Technologies Applications and Theory, Hrebienok, Slovakia, pp. 23-26, September 2008. ISBN 978-80-969184-8-5

10. Bayes, T. (1763). An Essay Toward Solving a Problem in the Doctrine of Chances, Vol. 53. Reprinted in Facsimiles of Two Papers by Bayes, Hafner Publishing, 1963

11. Mosteller, F. and Wallace, D. L. (1964). Inference and Disputed Authorship: The Federalist. SpringerVerlag. A second edition appeared in 1984 as Applied Bayesian and Classical Inference.