НИТУ «МИСиС»

Институт ИТАСУ
Кафедра инженерной кибернетики
Направление подготовки: 01.03.04 Прикладная математика
Квалификация (степень): бакалавр
Группа: **БПМ-16-2**

ОТЧЕТ

ПО НАУЧНО-ИССЛЕДОВАТЕЛЬСКОЙ РАБОТЕ

на тему: «Система динамического выявления и мониторинга причин клиентских обращений по банковским продуктам»

VII семестр 2019 - 2020 у.г.

Студент _____/ _____/
подпись Фамилия И.О.

Руководитель НИР _____/ _____/
подпись Фамилия И.О.

Оценка: _____

Дата защиты: _____

Утвердил:

Председатель комиссии _____/ _____/
подпись Фамилия И.О.

Москва 2020

Оглавление

Введение

За последние десятилетия развитие технологий привело к тому, что клиентам стало удобно давать обратную связь о неполадках или ошибках по продуктам удаленно, пользуясь компьютером или смартфоном, в текстовой или речевой форме. Не являются исключением и банковские продукты, особенно много сообщений приходит из-за каких-либо ошибок или неполадок, следовательно, в банковской среде возникает потребность в решении задачи *поиска корневой причины* (англ *root cause analysis*), то есть поиска ответа на вопрос: «Из-за какой проблемы, связанной с использованием продукта, обратился клиент».

Развитие технологий, помимо прочего, привело к увеличению количества обращений. В данный момент в Сбербанк поступает от 8 до 10 тысяч подобных сообщений в день. Для того, чтобы качественно и своевременно обрабатывать такой массив информации, Сбербанк имеет большой штат сотрудников, который в ручном или полуавтоматическом режиме обрабатывает эти сообщения.

Поиск причины обращения клиента – это достаточно трудоемкая задача, требующая концентрации, так как клиент зачастую формирует сообщение на естественном языке, то есть, нет какой-либо общей структуры или схемы, описывающей запрос. Из вышесказанного естественным образом следует, что вероятность того, что сотрудник, выполняющий эту задачу, ошибется и неверно интерпретирует смысл обращения клиента, достаточно велика.

В настоящее время эта задача решается вручную или полуавтоматически. Следовательно, возникает потребность полной автоматизации процесса, то есть в системе, помогающей сотрудникам с поиском причин, так как это приведет к уменьшению расходов на штат и увеличит качество предоставления услуг.

Целью данной работы является создание алгоритмического и математического обеспечения, а также программного модуля, который на основе текста обращения, даты поступления, а также всех предыдущих обращений будет выявлять причину обращения клиента.

При реализации алгоритма используются различные математические и алгоритмические инструменты. Используется Word2vec – инструмент для анализа семантики естественных языков, основанный на дистрибутивной семантике, машинном обучении и векторном представлении слов, а также язык программирования Python 3.

1. Аналитический обзор литературы

1.1 Базовые методы и подходы обработки текста

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, т.е. 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]

Нормализация текста – приведение всех слов к нормальной (словарной) форме. Нужно для того, чтобы одно и то же слово, написанное в разных формах считалось алгоритмом как одно слово.

Расстояние Левенштейна – расстояние редактирования. Может использоваться как метрика близости значений слов, для задачи исправления ошибок в словах.

1.1.1 Регулярные выражения

Одним из недооцененных успехов в стандартизации в информатике были *регулярные выражения* (*Regular expressions*, *RE*). Этот язык используется во всех современных языках программирования, текстовых процессорах и инструментах текста, таких как инструменты Unix grep или Emacs. Говоря формально, регулярное выражение - это алгебраическое обо-

значение для некоторого множества строк. RE особенно полезны для поиска в тексте, когда имеется шаблон строк и текстовый корпус; в этом случае RE вернет все строки в тексте, подходящие под этот шаблон. Например, инструмент командной строки Unix grep принимает регулярное выражение и возвращает каждую строку документа, которая соответствует выражению.

Самый простой вид регулярных выражений – последовательность букв. Для того, чтобы найти *woodchuck*, нужно написать */woodchuck/*. Это выражение вернет все строки, имеющие *woodchuck* как подстроку.

RE зависят от регистра. Это значит, что шаблон */woodchuck/* не соответствует слову *Woodchuck*. Эту проблему можно решить с помощью *[* и *]*. Строка букв внутри квадратных скобок обозначает дизъюнкцию букв ("или"). Например, шаблон */[wW]/* соответствует как *w*, так и *W*. Регулярное выражение */[1234567890]/* соответствует любой цифре.

Можно использовать тире (-) для того, чтобы выделить интервал. Шаблон */[2-5]/* соответствует одному из символов *2, 3, 4, 5*.

Для того, чтобы обозначить символ, которого не может быть, используется "^". */[^a]/* соответствует любому символу кроме *a*. */[^A-Z]/* - не заглавная буква, */[^Ss]/* - ни *S*, ни *s*, но /a^b/ - буква *a*, после которой идет любой символ, кроме *b*.

Для того, чтобы обозначить опциональные символы, используется *"?"*, например, */colou?r/* соответствует как *color*, так и *colour*.

Для обозначения последовательности из нуля или больше одинаковых символов используют *"*"*, например, */a*/* соответствует *a, aa, aaa, aaaa* и т.д, при этом */[ab]*/* соответствует любой последовательности из символов *a* и *b*.

Для обозначения любого символа используется *"."*, например, */beg.n/* соответствует *begin, began, begun*.

Для выбора между строками используется *"*, например, шаблон */cat|dog/* соответствует двум строкам: *cat* и *dog*. Можно использовать *()*, чтобы выделить варьирующуюся часть текста, например */gupp(y|ies)/* соответствует *guppy* и *guppies*.

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

1.1.2 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

В зависимости от приложения алгоритмы *токенизации* могут 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*), выпущена Linguistic 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 размера 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.

1.1.3 Byte Pair Encoding

Одна из причин, почему полезно иметь подслова в качестве токенов - когда есть неизвестные слова. Это особенно актуальны для систем машинного обучения. Они часто узнают некоторые факты о словах в одном корпусе (учебном), а затем используют эти факты для принятия решений в тестовом корпусе. Таким образом, если 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 ис-

пользование другого вида 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 алгоритм подсчитывает количества пар букв, находит наиболее часто встречающуюся пару (A, B) и заменяет ее новым символом (AB). Алгоритм повторяется пока не произойдет 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.

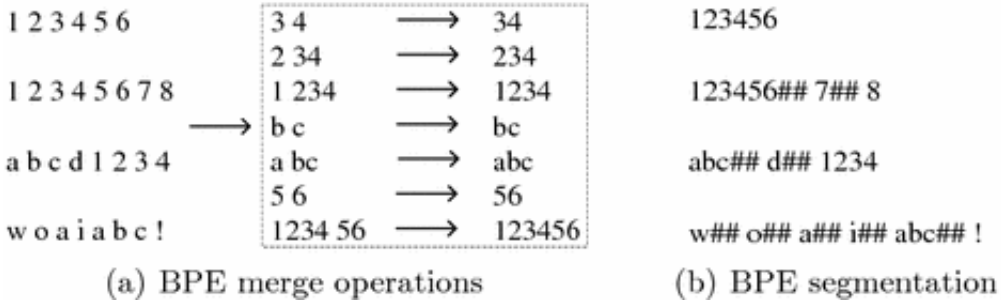

Рисунок 2 – пример выполнения алгоритма
BPE с 7 операциями слияния

1.1.4 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.

1.1.5 Нормализация слов, лемматизация, 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

1.1.6 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*). Это легче увидеть, посмотрев на Рисунок 3.

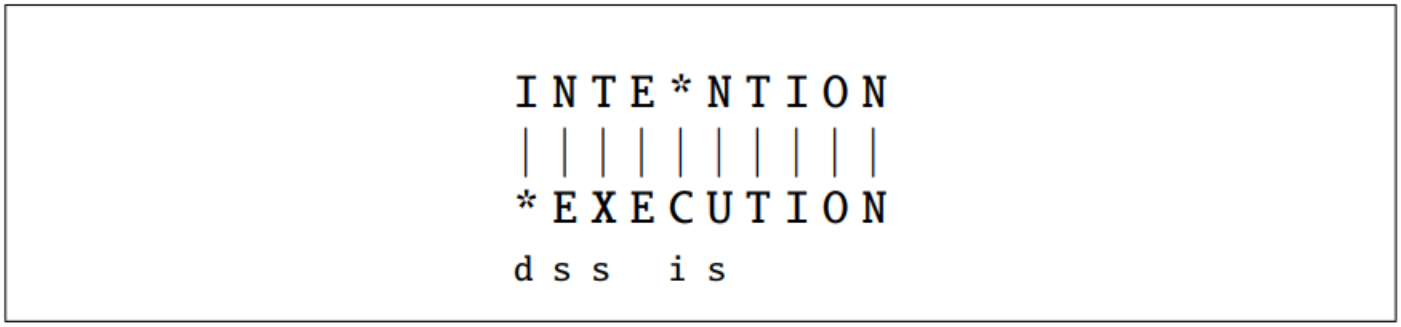

Рисунок 3 – Расстояние Левенштейна между словами "intention"и "execution".
Описание операций: d - удаление, s - замена, i - вставка.

1.2 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.

1.2.1 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:

$$P(the \mid its\ water\ is\ so\ transparent\ that) \tag{1.1}$$

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*":

$$P(the \mid its\ water\ is\ so\ transparent\ that) = \frac{C(its\ water\ is\ so\ transparent\ that\ the)}{C(its\ water\ is\ so\ transparent\ that)} \tag{1.2}$$

С огромным корпусом (например, Глобальная Сеть) можно рассчитать и посчитать вероятности с помощью формулы 1.1.

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 $P(X_i) = w_i$ за $P(w_i)$. Обозначим последовательность N слов как $w_1, w_2 \ldots, w_n$ и как w_1^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:

$$P(X_1, \ldots, X_n) = P(X_1)P(X_2|X_1)P(X_3|X_1^2)\ldots P(X_n|X_1^{n-1}) \tag{1.3}$$

Цепное правило показывает взаимосвязь между вычислением совместной вероятности последовательности и вычислением условной вероятности слова, при данных предыдущих словах. Формула 1.3 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. $P(X_n|X_1^{n-1})$. Но с помощью N-грам можно аппроксимировать историю по нескольким предыдущим словам.

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

Предположение, что вероятность слова зависит только от предыдущего слова называется Марковым предположение. Марковские модели [10] – 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:

$$P(w_n|w_1^{n-1}) \approx P(w_n|w_{n+1-N}^{n-1}) \tag{1.4}$$

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

$$P(w_1^n) \approx \prod_{k=1}^{n} P(w_k|w_{k-1}) \tag{1.5}$$

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.

$$P(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n)}{\sum_w C(w_{n-1}w)} \tag{1.6}$$

Формулу 1.6 можно упростить, заметив, что количество биграмм, начинающихся с w_{n-1} должно быть равно количество слов w_{n-1}.

$$P(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n)}{C(w_{n-1})} \tag{1.7}$$

Для общего случая MLE, оценка параметра n-граммы:

$$P(w_n|w_{n-N+1}^{n-1}) = \frac{C(w_{n-N+1}^{n-1}w_n)}{C(w_{n-N+1}^{n-1})} \tag{1.8}$$

Формула 1.8 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.

$$p_1 \times p_2 \times p_3 \times p_4 = exp(log(p_1) + log(p_2) + log(p_3) + log(p_4)) \tag{1.9}$$

1.2.2 Оценка языковых моделей

Лучший способ оценить производительность языковой модели - это непосредственно встроить её в систему и вычислить, насколько система 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*).

1.2.3 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 количеством слов. Для тестовой выборки $W = w_1 w_2 \dots w_N$:

$$PP(W) = P(w_1 w_2 \dots w_n)^{-\frac{1}{N}} \tag{1.10}$$

Можно использовать цепное правило, чтобы раскрыть вероятность W:

$$PP(W) = (\prod_{i=1}^{N} \frac{1}{P(w_i|w_1 \dots w_{i-1})})^{\frac{-1}{N}} \tag{1.11}$$

Если вычислять перплексию с помощью биграммной языковой модели:

$$PP(W) = (\prod_{i=1}^{N} \frac{1}{P(w_i|w_{i-1})})^{\frac{-1}{N}} \tag{1.12}$$

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

1.2.4 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.

1.2.5 Сглаживание

Самый простой способ - это просто добавить один ко всем количествам 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.

$$P(w_i) = \frac{c_i}{N} \tag{1.13}$$

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

$$P_{Laplace}(w_i) = \frac{c_i + 1}{N + V} \tag{1.14}$$

Сглаживание для вероятностей биграмм:

$$P^*_{Laplace}(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n) + 1}{\sum_w (C(w_{n-1}w) + 1)} = \frac{C(w_{n-1}w_n) + 1}{C(w_{n-1}) + V} \tag{1.15}$$

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

$$P^*_{Add-k}(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n) + k}{\sum_w (C(w_{n-1}w) + k)} = \frac{C(w_{n-1}w_n) + 1}{C(w_{n-1}) + kV} \tag{1.16}$$

1.2.6 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:

$$P^{'}(w_n|w_{n-2}w_{n-1}) = \lambda_1 P(w_n|w_{n-2}w_{n-1}) + \lambda_2 P(w_n|w_{n-2}) + \lambda_3 P(w_n) \tag{1.17}$$

$$\sum_i \lambda_i = 1 \tag{1.18}$$

Способ сложнее - давать каждой λ вес, зависящий от контекста. Если имеются точное количество для какой-то биграммы, можно сделать предположение, что число триграмм, основанных на этой биграмме будет заслуживать большего доверия, а значит можно дать соответствующей λ больший вес:

$$P^{'}(w_n|w_{n-2}w_{n-1}) = \lambda_1(w_{n-2}^{n-1})P(w_n|w_{n-2}w_{n-1}) + \lambda_2(w_{n-2}^{n-1})P(w_n|w_{n-2}) + \lambda_3(w_{n-2}^{n-1})P(w_n) \tag{1.19}$$

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

1.3 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. [11]

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.

1.3.1 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.

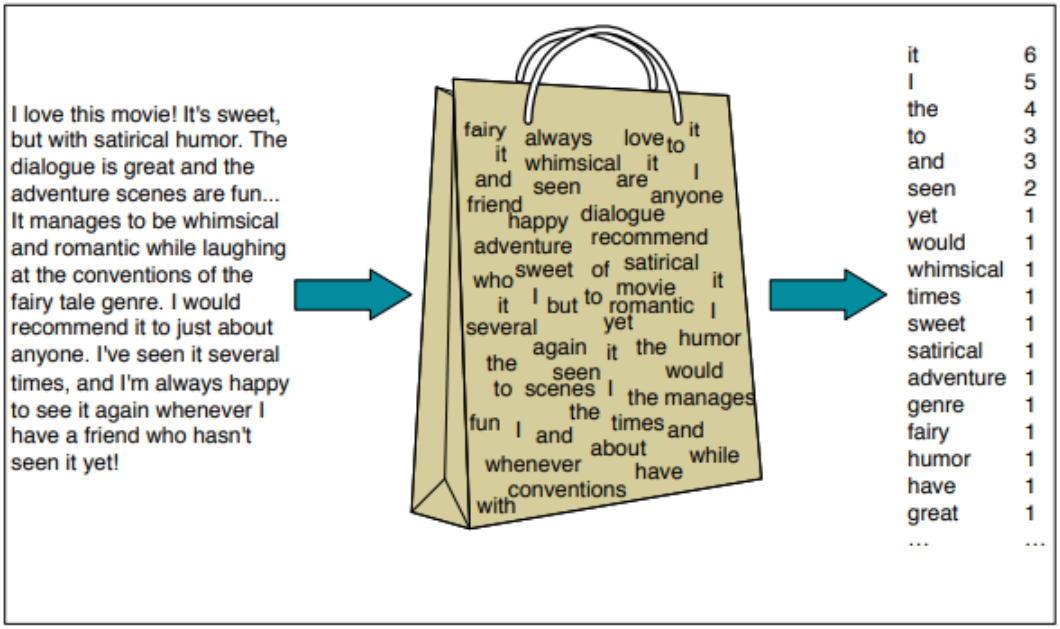

Рисунок 4 - Интуитивное описание классификатора на наивном Байесе, применённое к обзору фильма. Взаимное расположение слов игнорировано (взято предположение *мешка слов*).

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 формуле 1.20 нотация $\hat{}$означает предполагаемый корректный класс.

$$\hat{c} = \underset{c \in C}{argmax} P(c|d) \tag{1.20}$$

Эта идея *Байесовского вывода* была известна с момента публикации работы Байеса (1763) [12], и была впервые применена к классификации текста Мостеллером и Уоллесом (1964) [13]. 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:

$$P(x|y) = \frac{P(y|x)P(x)}{P(y)} \tag{1.21}$$

Используя формулу 1.20.

$$\hat{c} = \underset{c \in C}{argmax} P(c|d) = \underset{c \in C}{argmax} \frac{P(d|c)P(c)}{P(d)} \tag{1.22}$$

Знаменатель можно опустить.

$$\hat{c} = \underset{c \in C}{argmax} P(c|d) = \underset{c \in C}{argmax} P(d|c)P(c) \tag{1.23}$$

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 $P(d|c)$.

$$\hat{c} = \underset{c \in C}{argmax} P(c|d) = \underset{c \in C}{argmax} \overset{likelihood}{P(d|c)} \overset{prior}{P(c)} \tag{1.24}$$

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

$$\hat{c} = \underset{c \in C}{argmax} P(\overset{likelihood}{f_1, f_2, \ldots, f_n|c}) \overset{prior}{P(c)} \tag{1.25}$$

К сожалению, даже формула 1.25 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:

$$P(f_i, f_2, \ldots f_n|c) = P(f_1|c)P(f_2|c)\ldots P(f_n|c) \tag{1.26}$$

Результирующее уравнение для класса выбранного наивным Байесом таково:

$$C_{NB} = \underset{c \in C}{argmax} P(c) \prod_{f \in F} P(f|c) \tag{1.27}$$

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

$$C_{NB} = \underset{c \in C}{argmax} P(c) \prod_{i \in positions} P(w_i|c) \tag{1.28}$$

1.3.2 Наивный Байес как лингвистическая модель

Байесовские классификаторы могут использовать любые виды черт: словари, URL, e-mail адреса, черты сети, фразы и тому подобное. Но если, как в предыдущей секции, используются только черты отдельных слов и используются все слова в тексте, то наивный Байес имеет важную схожесть с моделированием речи. Конкретнее, модели на наивном Байесе могут быть рассмотрены как множество классо-специфичных юниграммных лингвистических моделей.

Поскольку правдоподобие черт из модели наивного Байеса присваивает вероятность для каждого слова $P(word|c)$, модель также присваивает вероятность целым предложениям.

$$P(s|c) = \prod_{i \in positions} P(w_i|c) \tag{1.29}$$

1.3.3 Метрики: точность, полнота, F-мера

Рассмотрим задачу обнаружения спама. Задача состоит в том, чтобы отнести каждое электронное письмо к одной из двух категорий: положительной (то есть, является спамом) или отрицательной (то есть, не является). Таким образом, для каждого письма требуется узнать, считает ли система его спамом или нет. Также требуется знать, является ли письмо спамом на самом деле, то есть определенные человеком ярлыки для каждого письма.Таким метки будут далее называться *золотыми ярлыками* (*gold labels*).

Для оценки любой системы распознавания требуется начать с постройки *таблицы сопряженности*.

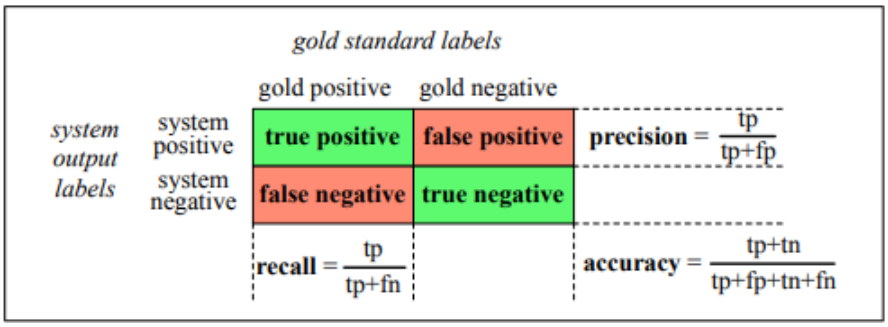

Рисунок 5 - таблица сопряженности

В правом нижнем углу таблицы уравнение для *accuracy*, которое показывает какую часть всех наблюдений система обозначила правильно. Хотя accuracy может показаться вполне естественной метрикой, она используется нечасто. Причина заключается в том, что accuracy работает плохо, если классы не сбалансированы. Другими словами, accuracy плоха, если требуется найти что-то редко встречающееся, или по крайней мере просто не сбалансировано по частоте встречи.

Именно поэтому вместо accuracy используются 2 метрики: *полнота* и *точность* (*precision* и *recall*).

Точность обозначает долю прецедентов которые система система обозначила как положительные и которые на самом деле являются положительными (то есть, например, обозначила как спам и письмо действительно им является).

$$Precision = \frac{true\ positives}{true\ positioves\ +\ false\ positives} \tag{1.30}$$

Полнота обозначает долю прецедентов, которые были корректно обозначены положительными (то есть среди всех писем обозначенных как спам доля тех, которые действительно им являлись).

$$Precision = \frac{true\ positives}{true\ positioves\ +\ false\ negatives} \tag{1.31}$$

Есть много способов определения единой метрики, которая сочетает в себе оба аспекта полноты и точности. Самая простая из этих комбинаций: *F-мера*.

$$F_\beta = \frac{(\beta^2 + 1)PR}{\beta^2 P + R} \tag{1.32}$$

Параметр β взвешивает важность полноты и точности, его задают в зависимости от задачи. Значения $\beta > 1$ отдают предпочтение точности, $\beta < 1$ - полноте. Когда $\beta = 1$, метрики равно сбалансированы; это самый частый случай использования, обозначается $F_{\beta=1}$ или F_1:

$$F_1 = \frac{2PR}{P+R} \tag{1.33}$$

Для случая, когда есть больше, чем 2 класса. В обработке языка часто встречается *мультиномиальная классификация*, в который классы - взаимоисключающие, каждый документ относится к одному классу. Для каждого класса строится свой бинарный классификатор, обучающийся на положительных результатах из класса c и отрицательных из всех остальных классов. После этого заданный документ d обрабатывается всеми классификаторами и выбирается ярлык из классификатора с наивысшим результатом. Рассмотрим *матрицу ошибок* для гипотетической категоризации писем по классам *важные*, *обычные*, *спам* (*urgent*, *normal*, *spam*).

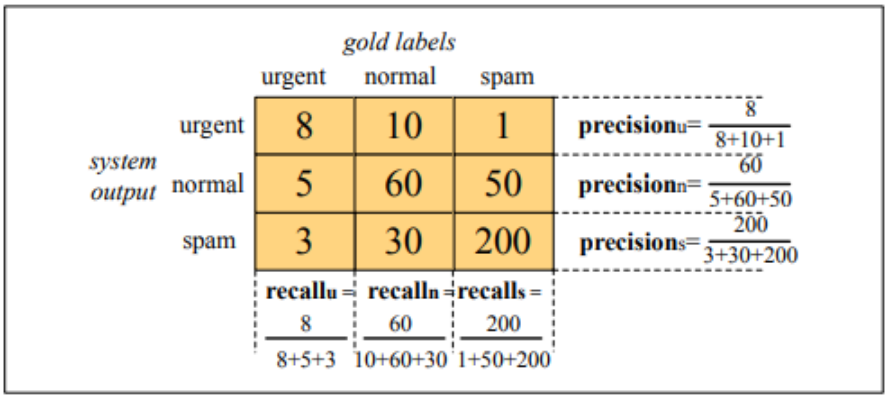

Рисунок 6 - матрица ошибок (confusion matrix)

Для того, чтобы выделить единую метрику для производительности системы в целом, можно воспользоваться *макроусреднением* (*macroaveraging*) или *микроусреднением* (*microaveraging*).

Макроусреднение - это вычисление производительности для каждого класса, а потом взятие среднего. *Микроусреднение* - это сбор всех решений в единую таблицу сопряженности и вычисление полноты и точности по ней. На следующем рисунке показана разница между этими двумя способами.

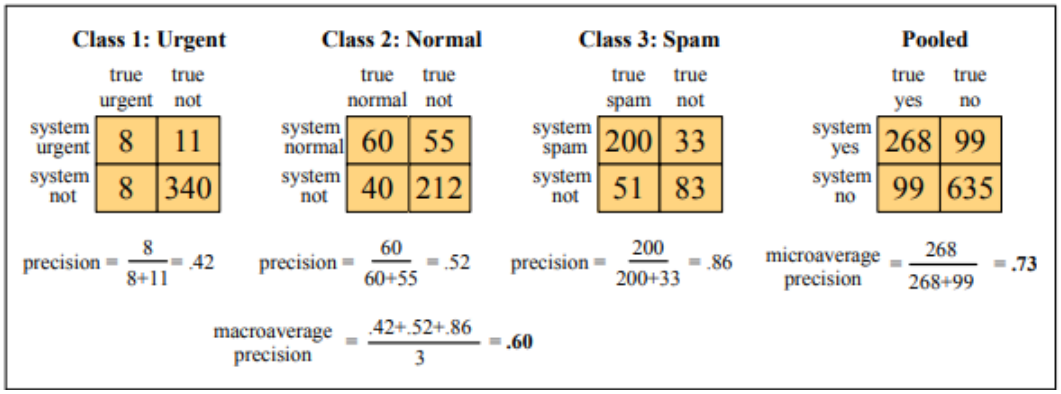

Рисунок 7 - макроусреднение и микроусреднение

1.3.4 Тестовые выборки и кросс-валидация

С фиксированными выборками возникает следующая проблема: так как для обучающей выборки нужно как можно больше прецедентов, тестовая или валидационная выборка могут оказаться недостаточно большими, чтобы быть репрезентативными. Лучшим вариантом было бы, если можно было бы использовать всю информацию а одновременно тестовой и обучающей выборке. Это позволяет сделать *кросс-валидация* [16] : случайным образом весь датасет разбивается на обучающую и тестовую выборки, классификатор обучается и проверяется на разбитых множествах. Эта процедура происходит 10 раз, средний результат среди 10 попыток дает оценку ошибок. Всё это называется *10-слойной кросс-валидацией.*

1.4 Логистическая регрессия

Логистическая регрессия - один из самых важных аналитических инструментов в социальных и естественных науках. В обработке естественного языка логистическая регрессия - базовый метод машинного обучения с учителем для классификации. Имеет очень близкое отношение к нейронным сетям [17]. Логистическая регрессия используется, чтобы классифицировать наблюдения в одни из 2 или более классов. Так как математика под двухклассовым случаем проще, сначала будет описана именно она. Затем будет описан случай мультиномиальной логистической регрессии для случая большего числа классов.

1.4.1 Генеративные и дискриминативные классификаторы

.

Самое важное отличие наивного Байеса от логистической регрессии состоит в том, что логистическая регрессия - *дискриминативный классификатор*, а наивный Байес - *генеративный.* Можно рассмотреть на визуальной метафоре: пусть задачей является отделить картинки кошек от собак.

Генеративный классификатор будет иметь цель понять как выглядит кошка и собака. Можно даже заставить модель «сгенерировать», то есть нарисовать собаку. При подаче тестового изображения система спрашивает, какая модель - кошки или собаки лучше подходит под изображение и, отталкиваясь от этого, выбирает ярлык.

Дискриминативная модель учится именно разделять классы. Возможно, все собаки носят ошейники, а все кошки - не носят. Если эта черта почти полностью разделяет классы, модели этого достаточно. Если спросить модель, что она знает о кошках, она ответит, что они не носят ошейников.

Генеративная модель, такая, как наивный Байес, использует термин *правдоподобие*, который выражает, как генерировать черты документа, если бы мы знали какого он класса.

Дискриминативная модель пытается непосредственно вычислить $P(c|d)$. Похоже, что она учится присваивать веса к чертам документа, которые непосредственно улучшают способность разделять между классами, даже если модель не может сгенерировать пример одного

из классов.

1.4.2 Компоненты вероятностного классификатора для машинного обучения

Как и наивный Байес, логистическая регрессия - это вероятностный классификатор, использующий *обучение с учителем*. Классификаторы машинного обучения требуют корпус из M пар ввода-вывода ($x^{(i)}$, $y^{(i)}$).

Система машинного обучения для классификации имеет 4 компоненты:

1 Репрезентация черт ввода. Для каждого наблюдения $x^{(i)}$ это вектор черт $[x_1, x_2, \ldots, x_n]$. Обозначение: черта i прецедента $x^{(j)}$ это $x_i^{(j)}$, иногда упрощенно x_i, иногда f_i, $f_i(x)$, или, для мультиклассовой классификации, $f_i(c, x)$.

2 Функция классификации, которая вычисляет $\hat{y}$, предполагаемый класс, как $p(y|x)$. В следующей секции будут введены инструменты *сигмоида* [18] и *софтмакс* для классификации.

3 Целевая функция для обучения, обычно включающая минимизацию ошибки на обучающей выборке.

4 Алгоритм для оптимизации целевой функции. Далее в работе будет представлен *алгоритм стохастического градиентного спуска.*

Логистическая регрессия имеет 2 фазы:

1 обучение: система учится (а именно веса w и b), используя стохастический градиентный спуск и функцию потерь *кросс-энтропии.*

2 проверка: по заданному прецеденту x вычисляется $P(y|x)$ и возвращается большая вероятность: $y = 1$ или $y = 0$.

1.4.3 Классификация: сигмоида

Целью бинарной логистической регрессии является обучение классификатора таким образом, чтобы он мог давать бинарное предсказание о классе нового прецедента. Здесь будет описана *сигмоида*, помогающая принять такое решение.

Рассмотрим единственный прецедент x, который будет обозначаться как вектор черт $[x_1, x_2, \ldots, x_n]$. Вывод классификатора может y может быть равен 1 (это означает, что прецедент является членом класса) или 0 (не является). Требуется узнать вероятность $P(y = 1|x)$ того, что прецедент является членом класса.

Логистическая регрессия добивается этого, обучая на выборке вектор весов и меру смещения. Каждый вес w_i - действительное число, ассоциировано с чертой x_i. Вес обозначает, насколько важна эта черта для классификации определенного класса, вес может быть положительным (значит, что ассоциировано) или отрицательным (не связано).

Чтобы сделать решение на тестовом случае, после того, как веса обучены, классификатор умножает каждый x_i на его вес w_i и прибавляет меру смещения b. Результирующее число z обозначает взвешенную сумму всех всидетельств класса.

$$z = (\sum_{i=1}^{n} w_i x_i) + b \tag{1.34}$$

Можно записать это уравнение короче, если воспользоваться скалярным произведением:

$$z = w \cdot x + b \tag{1.35}$$

В предыдущем уравнении ничего не останавливает z от того, чтобы перестать быть валидной вероятностью, то есть выйти из границ $[0, 1]$.

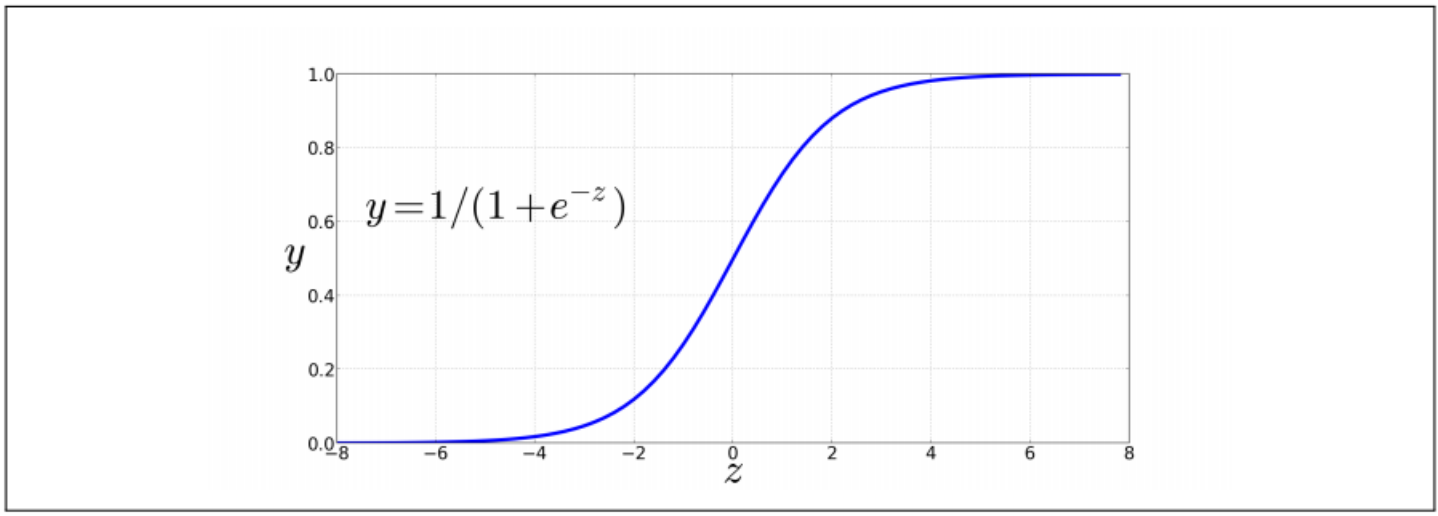

Рисунок 8 – сигмоида, на вход получает действительное число, на выходе - число из интервала $[0, 1]$, при этом практически линейна вокруг 0

Для того, чтобы получить вероятность, z пропускают через *сигмоиду* $\sigma(z)$. Сигмоиду также называют логистической функцией.

$$y = \sigma(z) = \frac{1}{1 + e^{-z}} \tag{1.36}$$

Сигмоида также дифференциируема, это очень полезное свойство.

Если применить сигмоида к сумме взвешанных черт, получится число в интервале $[0, 1]$. Для того, чтобы сделать её вероятностью, нужно просто сделать так, чтобы $p(y = 0) + p(y = 0) = 1$.

$$P(y = 1) = \sigma(w \cdot x + b) = \frac{1}{1 + e^{-(w \cdot x + b)}} \tag{1.37}$$

$$P(y = 0) = 1 - \sigma(w \cdot x + b) = \frac{e^{-(w \cdot x + b)}}{1 + e^{-(w \cdot x + b)}} \tag{1.38}$$

Теперь, когда есть алгоритм, который по x вычисляет вероятность $P(y = 1|x)$, остается вопрос, как принять решение о классе. Можно, например, положить, что x относится к классу, если $P(y = 1|x)$ больше, чем 0.5, и не относится в ином случае.

$$\hat{y} = \begin{cases} 1, \textit{ если } P(y=1|x) > 0.5 \\ 0, \textit{ иначе} \end{cases} \tag{1.39}$$

1.4.4 Обучение логистической регрессии

Для того, чтобы обучить логистическую регрессию, требуется, 2 компоненты.

Первая - метрика того, насколько близка оценка $\hat{y}$ к действительному классу. Вместо того, чтобы оценивать похожесть, обычно говорят об обратном: расстоянии между выводом системы и действительным классом. Это расстояние обычно называют . В следующей главе будет представлена функция потерь, которая часто используется для логистической регрессии и нейронных сетей, $-$.

Вторая компонента - это алгоритм оптимизации для итеративного обновления весов такого и минимизации функции потерь. Стандартный алгоритм для этого - *градиентный спуск*. В следующей главе будет рассмотрен *стохастический градиентный спуск*.

1.4.5 Функция потерь кросс-энтропии

Требуется функция потерь, которая выражает для прецедента x как близко вывод классификатора ($\hat{y} = \sigma(w \cdot x + b)$) находится к корректному выводу (y, который равен 0 или 1). Назовем её так:

$$L(\hat{y}, y) = \textit{Насколько } \hat{y} \textit{ отличается от настоящего } y \tag{1.40}$$

Это вычисляется с помощью функции потерь, которая считает, что корректный класс ярлыков более вероятен. Это называется *условная оценка максимальной правдоподобности*: выбираются такие параметры w, b, что максимизируется логарифмическая вероятность настоящего y в обучающей выборке. Результирующая функция потерь - *кросс-энропическая*.

Возьмем производную этой функции потерь, примененную к прецеденту x. Хотелось бы обучить веса таким образом, чтобы максимизировать вероятность $p(y|x)$. Поскольку есть только два дискретных выхода: 0 и 1, это распределение Бернулли, можно выразить вероятность $p(y|x)$, которую производит классификатор для одного прецедента как следующее уравнение:

$$P(y|x) = \hat{y}^y (1-\hat{y})^{1-y} \tag{1.41}$$

Теперь возьмем логарифм от каждой части:

$$logp(y|x) - log[\hat{y}^y (1-\hat{y})^{1-y}] = ylog\hat{y} + (1-y)log(1-\hat{y}) \tag{1.42}$$

Предыдущее уравнение описывает логарифмическую вероятность, которая должна быть максимизирована.

$$L_{CE}(\hat{y}, y) = -logp(y|x) = -[ylog\hat{y} + (1-y)log(1-\hat{y})] \tag{1.43}$$

Подставляя $\hat{y} = (\sigma(w \cdot x + b))$:

$$L_{CE}(w, b) = -[ylog\sigma(w \cdot x + b) + (1-y)log(1-\sigma(w \cdot x + b))] \tag{1.44}$$

1.4.6 Градиентный спуск

Цель градиентного спуска [19] в том, чтобы найти оптимальные веса: минимизировать функцию потерь.

$$\hat{\theta} = \underset{\theta}{argmin} \frac{1}{m} \sum_{i=1}^{m} L_{CE}(y^{(i)}, x^{(i)}; \theta) \tag{1.45}$$

Градиентный спуск - метод, который находит направление, в котором касательная функции растет быстрее всего и двигается в обратном направлении. Для логистической регрессии эта функция потерь выпукла.

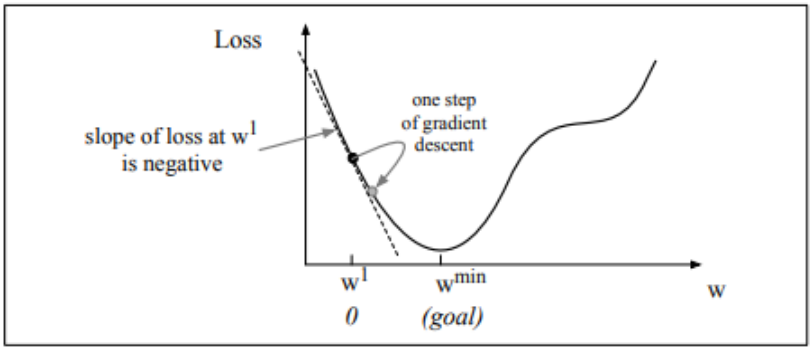

Рисунок 9 - шаг градиентного спуска, для одномерных векторов черт

По заданной случайной инициализации w каким-то значением w^1 и предполагая, что функция потерь имеет форму, как на Рисунке 9, требуется алгоритм, который сообщает, что на следующей итерации нужно сдвинуться влево или вправо, чтобы достичь минимума.

Алгоритм градиентного спуска отвечает на этот вопрос, находя градиент функции потерь в данной точке и двигаясь в направлении, противоположном найденному.

Величина движения в градиентном спуске - это наклон касательной $\frac{d}{dx}f(x; w)$, взвешенный скоростью обучения η. Большая скорость обучения означает, что нужно двигать w больше на каждой итерации. Изменение параметра - это скорость обучения, умноженная на градиент:

$$w^{t+1} = w^t - \eta \frac{d}{dx} f(x; w) \tag{1.46}$$

1.4.7 Градиент для логистической регрессии

Для того, чтобы обновить веса, нужно ввести определение градиента. Для функции потерь кросс-энтропии:

$$L_{CE}(w,b) = -[ylog\sigma(w \cdot x + b) + (1-y)log(1-\sigma(w \cdot x + b))] \tag{1.47}$$

Возьмем производную:

$$\frac{\partial L_{CE}(w,b)}{\partial w_j} = [\sigma(w \cdot x + b) - y]x_j \tag{1.48}$$

1.4.8 Алгоритм стохастического градиентного спуска

Стохастический градиентный спуск - это online-алгоритм минимизации функции потерь, который вычисляет градиент после каждого обучающего примера и корректирует *theta* (множество всех весов) [20].

функция Стохастический Градиентный Спуск $(L(), f(), x, y)$ **возвращает** θ
  # L - функция потерь
  # f - функция, параметризированная θ
  # x - вводы обучающей выборки
  # y - выводы обучающей выборки (ярлыки)
$\theta \leftarrow 0$
Пока требуется
  Для каждого прецедента $(x^{(i)}, y^{(i)})$ (в случайном порядке)
    1. Вычислить $\hat{y}^{(i)} = f(x^{(i)}; \theta)$
    2. Вычислить потерю $L(\hat{y}^{(i)}, y^{(i)})$
    3. $g \leftarrow \nabla_\theta L(f(x^{(i)}; \theta), y^{(i)})$
    4. $\theta \leftarrow \theta - \eta g$

Рисунок 10 - алгоритм стохастического градиентного спуска. Алгоритм может завершиться, когда сойдется, то есть, например, веса практически не изменяются.

Параметр η может быть скорректирован. Если он достаточно велик, то обучаемый будет делать слишком большие шаги, «перепрыгивая» минимум функции потерь, если слишком мал, то обучение будет занимаьт слишком много времени. Очень часто скорость обучения делают большой, затем постепенно её уменьшают. Таким образом, η становится функцией от номера итерации k.

1.4.9 Регуляризация

Есть проблема с обучением весов, которые заставляют модель идеально совпадать с обучающей выборкой. Если черта полностью определяет какой-то класс, потому что она встречается только у него, то её будет дан огромный вес. Веса черт будут пытаться идеально подстроиться под обучающую выборку, моделируя даже факторы шума, случайно скоррелировавшие с классом. Эта проблема называется *переобучением*. Хорошая модель должна

хорошо обобщать обучающую выборку к тестовой выборке. Переобученная модель будет иметь слабые показатели обобщения.

Для избежания переобучения, *регуляризация* $R(Q)$ добавляется к функции потерь.

$$\hat{\theta} = \underset{\theta}{argmax}\ logP(y^{(i)}|x^{(i)}) - \alpha R(\theta) \tag{1.49}$$

Регуляризация призвана «наказывать» модель за слишком большие веса. Таким образом, веса, которые соответствуют обучающей выборке идеально, но при этом используют веса с слишком большими значениями, будут наказаны больше, чем веса, которые соответствуют выборке чуть хуже, но зато имеют меньшие значения веса.

Есть два распространённых способа посчитать регуляризацию. *L2-регуляризация* - это квадратичная функция весов. Норма L2, $||\theta||_2$ - такая же, как *Евклидово расстояние* вектора θ от начала координат. Если θ состоит из n векторов, то:

$$R(\theta) = ||\theta||_2^2 = \sum_{j=1}^{n} \theta_j^2 \tag{1.50}$$

Тогда функция потерь становится такой:

$$\hat{\theta} = \underset{\theta}{argmax}\ logP(y^{(i)}|x^{(i)}) - \alpha \sum_{j=1}^{n} \theta_j^2 \tag{1.51}$$

L1-регуляризация - линейная функция от значений весов, названная так по L1 норме $||W||_1$, сумме модулей величин весов, также известна как *Манхэттенское расстояние*.

$$R(\theta) = ||\theta||_1 = \sum_{i=1}^{n} |\theta_j| \tag{1.52}$$

Подставленное в функцию потерь:

$$\hat{\theta} = \underset{\theta}{argmax}\ logP(y^{(i)}|x^{(i)}) - \alpha \sum_{j=1}^{n} |\theta_j| \tag{1.53}$$

L1 регуляризацию в статистике часто называют *lasso regression*, L2 - *ridge regression*. L2 предпочитает, чтобы в модели было много маленьких весов, L1 предпочитает разреженные решения с большими весами, но множеством нулевых весов.

1.5 Векторные представления и семантики

Роль контекста важна в представлении слов. Слова, которые появляются в похожих контекстах чаще имеют похожий смысл. Эту связь называют (*distributionalhypothesis*). В этой главе будут представлены векторные семантики, которые основываются на этой лингвистической гипотезе, обучаются представлениям значений слов, которые названы *эмбеддингами*, непосредственно из текстов.

1.5.1 Векторные семантики

Идея векторных семантик заключается в том, чтобы представить слово в виде точки на многоразмерном пространстве семантик. Векторы для представления обычно называют *эмбеддингами*.

Рисунок 11 – двумерная проекция эмбеддингов некоторых слов и фраз, показывающая слова с похожими значениями рядом в пространстве.

Можно заметить, что положительные и отрицательные слова расположены в разных частях рисунка. Векторные семантики очень полезны на практике тем, что они могут быть обучены автоматически из текста без каких-либо сложных обозначений или учителя.

Векторные семантики являются стандартным способом представления слов в NLP.

1.5.2 Слова и векторы

Слова или модели распределения значений обычно основаны на *матрице совмещения* (*co-occurrence matrix*).

Один из видов таких матриц - *матрица терминов-документов*. Каждая строка обозначает слово в словаре, каждый столбец - документ, откуда слово было взято. Далее имет пример для четырёх пьес Шекспира.

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	1	0	7	13
good	114	80	62	89
fool	36	58	1	4
wit	20	15	2	3

Рисунок 12 – матрица терминов-документов для четырёх слов и четырёх пьес Шекспира

Можно представлять вектор для документа как точка в $|V|$-мерном пространстве. Далее представлена проекция матрицы на двумерную плоскость.

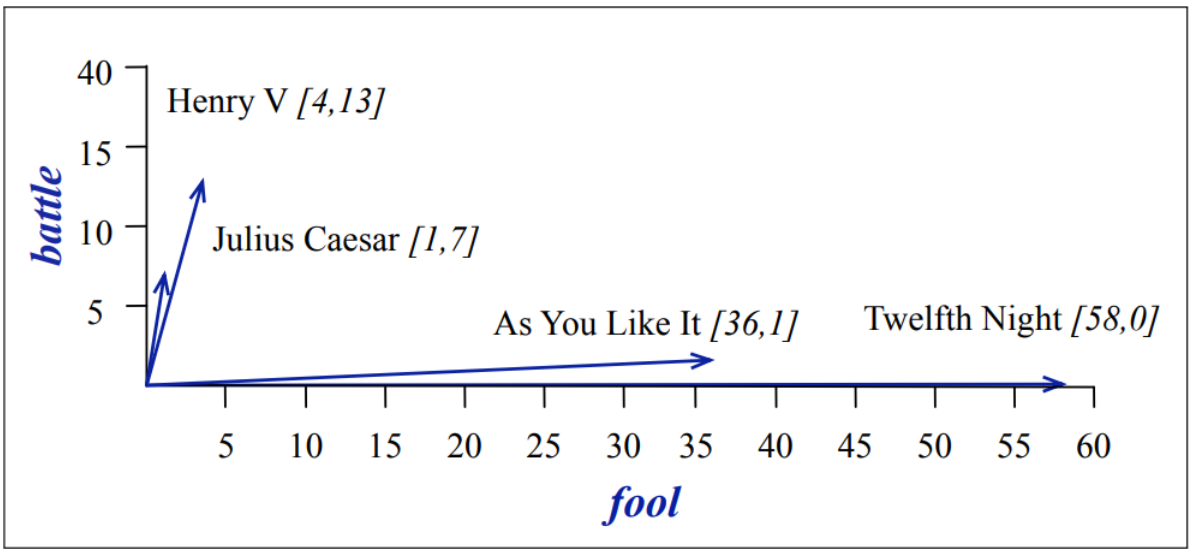

Рисунок 13 – визуализация с измерениями, соответствующими словам *battle* и *fool*

Матрица терминов-документов изначально была задуман для задачи информационного поиска, то есть, поиска документа d из множества D документов, который лучше всего подходит под запрос q.

1.5.3 Слова как векторы

Векторные семантики могут быть использованы для представления значения слов, ставя каждому слову в соответствие вектор.

Вместо матрицы терминов-документов используются *матрицы терминов-терминов*, чаще называемые *матрицами слов-слов* или *матрицами терминов-контекста* (*term-term matrix*, *word-word matrix*, *term-context matrix*). Это матрица размерности $|V| \times |V|$? каждая ячейка которой хранит количество раз слово в строке (целевое слово) и слово в столбце (контекст) встречались в каком-то контексте в одном корпусе.

1.5.4 Косинус для измерения расстояния между словами

Для того, чтобы оценить похожесть двух целевых слов, нужна функция от двух слов, показывающая, насколько они близки по значению. Самая часто встречающаяся метрика - косинус угла между векторами.

Косинус основан на скалярном произведении:

$$\textit{скалярное произведение}(v, w) = v \cdot w = \sum_{i=1}^{N} v_i w_i = v_1 w_1 + v_2 w_2 + \cdots + v_N w_N \tag{1.54}$$

В таком виде метрика имеет проблему: она предпочитает длинные вектора. Длина вектора определяется как:

$$|v| = \sqrt{\sum_{i=1}^{N} v_i^2} \tag{1.55}$$

Самый простой способ модифицировать - это взять *нормированное скалярное произведение.*

$$a \cdot b = |a||b|cos\theta \tag{1.56}$$

$$\frac{a \cdot b}{|a||b|} = cos\theta \tag{1.57}$$

Тогда расстояние между двумя словами может быть вычислено как:

$$cosine(v, w) = \frac{v \cdot w}{|v||w|} = \frac{\sum_{i=1}^{N} v_i w_i}{\sqrt{\sum_{i=1}^{N} v_i^2}\sqrt{\sum_{i=1}^{N} w_i^2}} \tag{1.58}$$

Иногда используют уже нормализованные векторы, то есть длины 1. Для нормализованных векторов косинус равен скалярному произведению.

1.5.5 TF-IDF: веса в терминах векторов

Частот в чистом виде недостаточно для того, чтобы оценить расстояние между словами. Одна из проблем заключается в том, что частоты сильно перекошены и недостаточно дискриминативны. Возникает небольшой парадокс. Слова, которые появляются рядом друг с другом важнее, чем слова, которые появляются однажды или дважды. При этом слова, которые возникают достаточно часто (такие как *хороший*, *он*) на самом деле неважны.

Алгоритм *tf-idf* - произведение двух множителей.

Первый - частота термина [14]: количество раз слово t встречается в документе d.

$$tf_{t,d} = count(t, d) \tag{1.59}$$

Можно также взять log_{10} он этого числа, так как если слово появляется в тексте 100 раз, это не значит, что оно в 100 раз более значимо для понимания документа.

$$tf_{t,d} = log_{10}(count(t, d) + 1) \tag{1.60}$$

Второй множитель используется для того, чтобы дать вес словам, которые появляются в некоторых документах, так как такие слова полезны для их отделения. *Частота документа* df_t слова t - это количество документов, в которых t появляется.

$$df_t = |D|, \forall d \in D : t \in d \tag{1.61}$$

Обратная частота документа $idf_t = N/df_i$, где N - количество документов, используется для того, чтобы выделить дискриминативные слова.

$$idf_t = \frac{N}{df_t} \tag{1.62}$$

Следуя той же самой логике, что и с tf_t, возьмем log_{10}:

$$idf_t = log_{10}(\frac{N}{df_t}) \tag{1.63}$$

Вес слова t в документе d - произведение $tf_{t,d}$ на idf_t

$$w_{t,d} = tf_{t,d} \times idf_t \tag{1.64}$$

1.5.6 Word2vec

Здесь слова будут представлены как *короткие* (длины 50-1000) и *плотные* (большинство значений - не нули) векторами, так как плотные векторы работают в любой задаче NLP намного лучше, чем разреженные. Здесь будут представлен метод для очень плотных, которких векторов, названный *skipgram with negative sampling*, иногда его называют *SGNS*. Алгоритм скипграмм - один из двух алгоритмов в пакете программного обеспечения, названном word2vec, поэтому сам алгоритм иногда называют также.

Идея алгоритма заключается в том, что вместо того, чтобы считать как часто каждое слово w встречается рядом с словом w_1, обучить бинарный классификатор, который будет отвечать, какова вероятность того, что w окажется рядом с w_1.

Революционной идеей является то, что можно использовать обрабатываемый текст как учителя для классификатора. Таким образом, не требуется собственноручно проставлять ярлыки.

Word2vec [15] хорош тем, что: во-первых, он упрощает задачу (бинарная классификация вместо предсказания слов), во-вторых, используется простая архитектура (обучается логистическая регрессия).

Суть скипграмм в следующем:

1 Считать, что слово и его «сосед» - положительные примеры.

2 Случайно выбрать другие слова из лексикона и считать их отрицательными примерами.

3 Обучить логистическую регрессию на разделение этих классов

4 Использовать веса регрессии как эмбеддинги.

2. Специальная часть

2.1 Содержательная постановка задачи

В настоящей научно-исследовательской работе ставится цель разработать систему выявления и мониторинга причин обращений клиентов банка на естественном языке, позволяющую получить причины на естественном языке. Под причиной в данной работе понимается часть исходного текста.

Для достижения поставленной задачи необходимо решить следующие задачи:

1 Провести обзор и критический анализ соответствующих математических моделей, методов и алгоритмов, выявить их преимущества и недостатки, оценить границы применимости.

2 Разработать математическое и алгоритмическое обеспечение системы, обеспечивающее корректную работу с русскоязычными текстами.

3 Разработать программное обеспечение, реализующее основную функциональность на основе разработанного математического и алгоритмического обеспечения.

4 Разработать обучающую, тестовую и валидационную выборки в виде корпуса текстов и известных дат их поступления.

5 Разработать методы оценки качества системы, получить эмпирические оценки качества работы на разработанной тестовой и валидационной выборках.

2.2 Математическая постановка задачи

Рассмотрим русскоязычный алфавит:

$$\Sigma = \{\varepsilon_1, \ldots, \varepsilon_{33}\} \tag{2.1}$$

Под словом длины n будем понимать упорядоченный набор (вектор) букв алфавита Σ:

$$w = (\varepsilon_{k_1}, \ldots, \varepsilon_{k_1}\varepsilon_{k_n}), k_1, \ldots, k_n \in \{1, 2, \ldots, 33\} \tag{2.2}$$

Множество русскоязычных слов обозначим за Σ^*. При этом всякое упорядоченное подмножество слов будем называть текстом на русском языке, состоящем из m слов.

$$W = \{w_1, \ldots, w_m\} \subset \Sigma^* \tag{2.3}$$

Обозначим время поступления обращения клиента за t.

Запрос клиента – кортеж из текста обращения и времени поступления:

$$q = (W, t) \tag{2.4}$$

Обозначим множество причин как $R = W_r$ – последовательность слов (эквивалентно тексту).

Множество обращений - упорядоченная по неубыванию времени поступления последовательность обращений клиентов:

$$Q = \{q_1, \ldots, q_n\} = \{(W_1, t_1), \ldots, (W_n, t_n)\}, t_i \leq t_{i+1}, \forall i \in [1, n-1] \tag{2.5}$$

На вход системы подается множество обращений.

Необходимо разработать оператор F, который по заданному запросу построит вектор причин:

$$F(q) = 2^{\Sigma^*} \times \Re \to 2^{\Sigma^*} \tag{2.6}$$

2.3 Функциональная схема решения

Структуру алгоритма нахождения корневых причин можно описать диаграммой, изображенной на рисунке 15

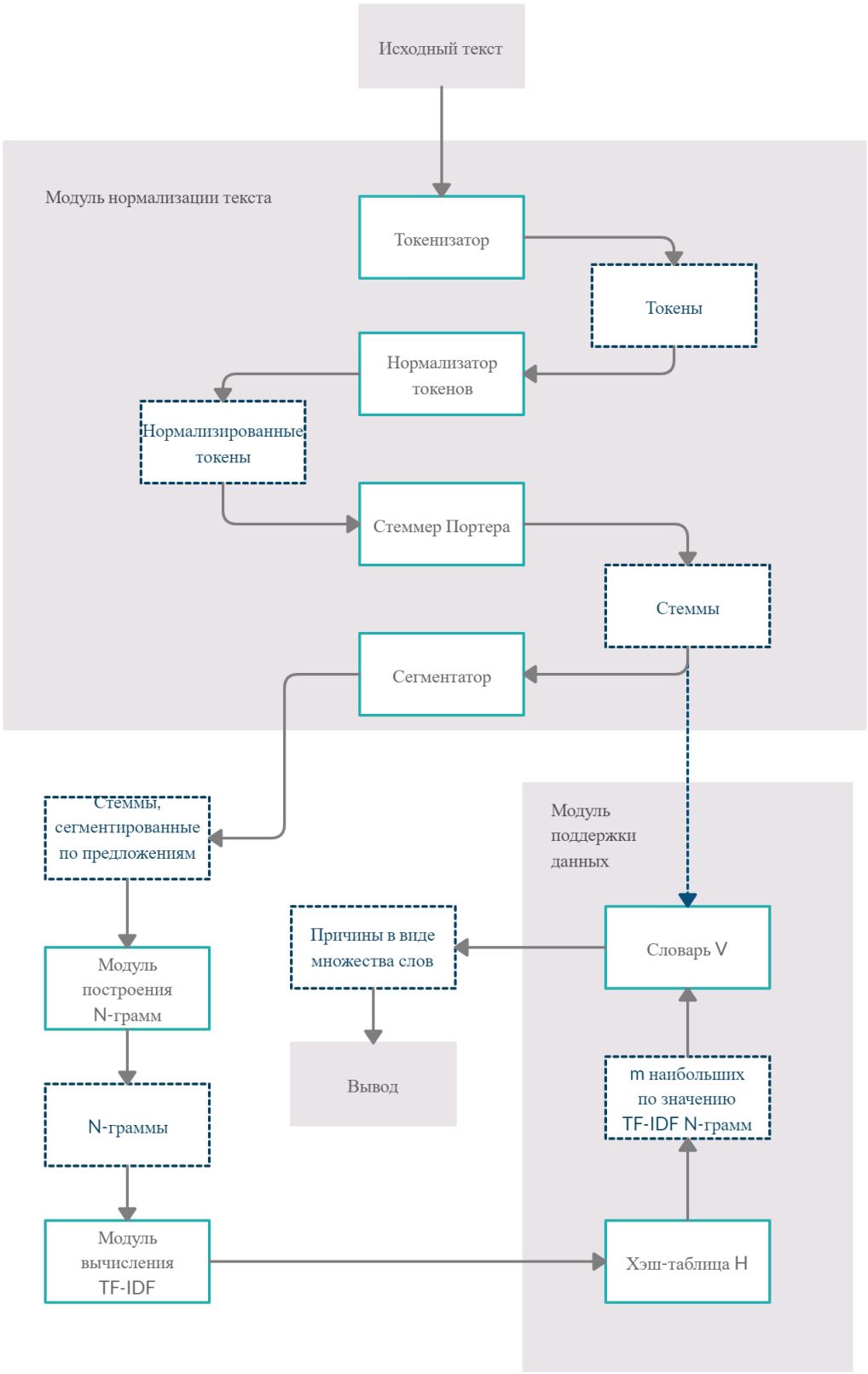

Рисунок 15 – Функциональная схема

2.4 Описание предметной области

2.5 Формат входных и выходных данных

В настоящей работе в качестве источника входной информации для нахождения корневых причин служат данные

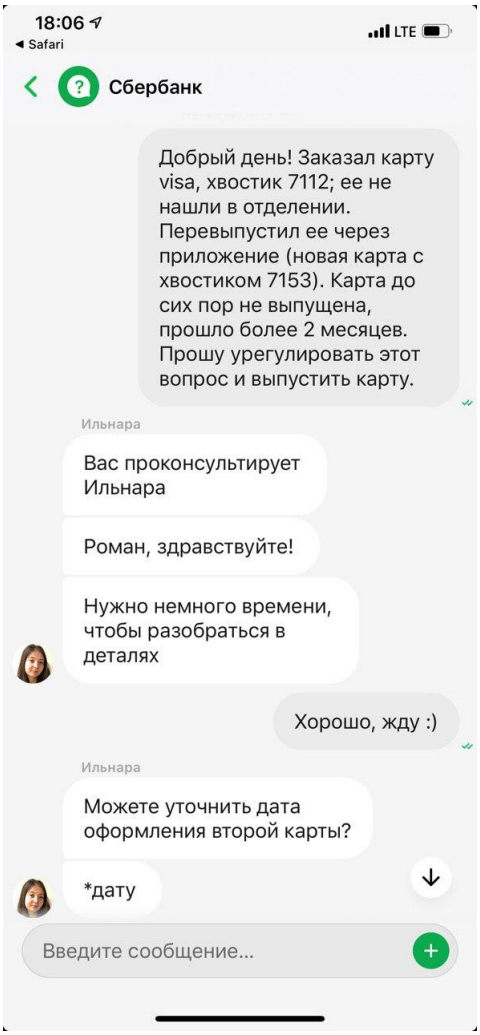

Рисунок 16 – Пример запроса клиента

2.6 Разработанное алгоритмическое обеспечение

2.6.1 Модуль нормализации текста

Модуль нужен для первичной обработки сообщения, то есть для перевода его в вид, удобный для обработки следующими модулями.

На вход модулю поступает необработанное сообщение клиента, в том виде, в каком клиент его написал. То есть на вход модулю подается текст $W = \{w_1, w_2, \ldots, w_m\}$

Модуль состоит из четырех частей:

1 Токенизатор

2 Нормализатор токенов

3 Стеммер Портера

4 Сегментатор

Токенизатор соединяет слова, которые только вместе составляют смысловую единицу. В основном, речь идет об наименованиях (например, «New York city»), датах и организациях.

Для этого используется стандарт токенизации *Penn Treebank* – он выделяет клитики (то есть несамостоятельные слова, например, предлоги), объединяет слова, написанные через дефис, выделяет знаки препинания. Для непосредственной токенизации используется *Python-based Natural Language Toolkit*, а точнее – его функция *nltk.regexp tokenize*. После этого применяется алгоритм *Byte Pair Encoding* для объединения наиболее часто встречающихся последовательностей токенов.

Если рассмотреть с математической точки зрения, то токенизатор - это функция $T(W)$, которая в исходном тексте W превращает k наиболее часто встречающиеся последовательности из двух слов в новое слово.

$$W' = F(W) \tag{2.7}$$

$$w'_{c_i} = w_j w_{j+1}, order(w_j w_{j+1}) \leq k \tag{2.8}$$

Далее токенизированный текст поступает в нормализатор токенов, где все токены приводятся к нормальной (словарной) форме. Сначала производится *Case folding* (*Сжатие регистра*), это требуется для того, чтобы одни и те же слова, написанные в разных регистрах (например, слово, с которого начинается предложение), не считалось системой разными словами.

После этого к токенам применяется *стеммер Портера*. Это детерминированный алгоритм, который последовательно «отсекает» от слова аффиксы. Этот алгоритм применяется для того, чтобы оставить от слов их стеммы, то есть корни, так как однокоренные слова очень часто близки по смыслу.

На данном этапе нужно сохранить из какого слова получилась та или иная стемма. Для этого в *блоке поддержки данных* существует словарь V. В этом случае под словарем подразумевается абстрактный тип данных. Он хранит пары «ключ-значение» и обеспечивает быстрый доступ к значениям по уникальным ключам. С математической точки зрения, словарь D представляет собой множество кортежей (k, v), k - ключ, v - значение.

$$D = \{(k_1, v_1), (k_2, v_2), \ldots, (k_n, v_n)\} \tag{2.9}$$

Наконец, получившиеся стеммы поступают в блок сегментирования предложений. Этот блок разбивает текст по знакам препинания окончания предложения, если таковые не являются частью какого-то слова.

2.6.2 Модуль построения N-грамм

Стеммы, разбитые по предложениям, являются входом для этого входа. В данном этапе строится множество Gr N-грамм, то есть последовательностей из N слов. В данной работе строятся 1, 2, 3 -граммы, то есть:

$$W = \{w_1, w_2, \ldots, w_m\} \tag{2.10}$$

$$Gr_1 := \bigcup_{i=1}^{m} w_i \tag{2.11}$$

$$Gr_2 := \bigcup_{i=1}^{m-1} (w_i, w_{i+1}) \tag{2.12}$$

$$Gr_3 := \bigcup_{i=1}^{m-2} (w_i, w_{i+1}, w_{i+2}) \tag{2.13}$$

$$Gr := Gr_1 \cup Gr_2 \cup Gr_3 \tag{2.14}$$

В формулах 2.10 - 2.14 есть небольшое упущение, связанное с тем, что на самом деле N-граммы не выходят из рамок предложения. Чтобы скорректировать формулы, нужно представить текст на входе как последовательность предложений S_i, каждое из которых является текстом, то есть последовательностью слов.

$$W = \bigcup_{i=0}^{s} S_i = \bigcup_{i=0}^{s} \{w_{i1}, w_{i2}, \ldots, w_{im_i}\} \tag{2.15}$$

В таком случае формулы 2.11 - 2.14 приобретают такой вид:

$$Gr_1 := \bigcup_{i=1}^{n} \bigcup_{j=1}^{m_i} w_j \tag{2.16}$$

$$Gr_2 := \bigcup_{i=1}^{n} \bigcup_{j=1}^{m_i-1} (w_i, w_{j+1}) \tag{2.17}$$

$$Gr_3 := \bigcup_{i=1}^{n} \bigcup_{j=1}^{m_i-2} (w_j, w_{j+1}, w_{j+2}) \tag{2.18}$$

$$Gr := Gr_1 \cup Gr_2 \cup Gr_3 \tag{2.19}$$

Gr и является выходом этого модуля.

2.7 модуль вычисления TF-IDF

Входом этого модуля является множество N-грамм Gr. Для каждого элемента этого множества вычисляется его *TF-IDF*. Эта метрика состоит из 2 множеителей: *term frequency* и *inverse document frequency*. Метрика устроена так, что она отдает предпочтение терминам, которые часто встречаются в данном документе и редко встречаются в других документах. Для более объективного результата оба множителя сглаживаются и логарифмируются, то есть:

$$tfidf(w) = \frac{log(tf + 1)}{log(df + 1)} \tag{2.20}$$

Этот модуль также используется хэш-таблицу H для вычисления.

Заключение

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. - Т. 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. 219223. 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. Кельберт М. Я., Сухов Ю. М. Вероятность и статистика в примерах и задачах. Т. ??: Марковские цепи как отправная точка теории случайных процессов и их приложения. ? М.: МЦНМО, 2010. ? 295 с. ? ISBN 978-5-94057-252-7.

11. Domingos, Pedro & Michael Pazzani (1997) ?On the optimality of the simple Bayesian classifier under zero-one loss?. Machine Learning, 29:103-137. (also online at CiteSeer: [1])

12. 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

13. 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.

14. Luhn, H. P. (1957). A statistical approach to the mechanized encoding and searching of literary information. IBM Journal of Research and Development, 1(4), 309?317

15. Mikolov T., Chen K., Corrado G., Dean J. Efficient Estimation of Word Representations in Vector Space // In Proceedings of Workshop at ICLR. ? 2013

16. Geisser, Seymour (1993). Predictive Inference. New York, NY: Chapman and Hall. ISBN 978-0-412-03471-8.

17. Tolles, Juliana; Meurer, William J (2016). "Logistic Regression Relating Patient Characteristics to Outcomes". JAMA. 316 (5): 533?4. doi:10.1001/jama.2016.7653. ISSN 0098-7484. OCLC 6823603312. PMID 27483067.

18. Mitchell, Tom M. Machine Learning. ? WCB?McGraw?Hill, 1997. ? ISBN 0-07-042807-7.

19. Акулич И. Л. Математическое программирование в примерах и задачах. ? М.: Высшая школа, 1986. ? С. 298-310.

20. Taddy, Matt (2019). "Stochastic Gradient Descent". Business Data Science: Combining Machine Learning and Economics to Optimize, Automate, and Accelerate Business Decisions. New York: McGraw-Hill. pp. 303?307. ISBN 978-1-260-45277-8.