Санкт-Петербургский Государственный Университет

Факультет Прикладной Математики - Процессов Управления

Кафедра Математической Теории Игр и Статистических Решений

***НИКИТИН Максим Дмитриевич***

**Выпускная квалификационная работа**

# Машинное обучение в задачах автоматической обработки текстов

Направление 01.03.02 «Прикладная математика и информатика

Основная образовательная программа СВ.5005.2015 «Прикладная математика, фундаментальная информатика и программирование»

Прикладная математика и информатика

*Научный руководитель:*

Ассистент, кафедра математической теории игр

и статистических решений

Староверова К. Ю.

Санкт-Петербург

2019

Оглавление

# Введение

Анализ тональности текста является одной из главных задач обработки естественного языка (Natural Language Processing) и обработки текста в частности. Её суть подчеркивает главную идею NLP: научить компьютер «понимать» естественный язык.

Многие компании заинтересованы в автоматическом определении тональности текста. Это помогает понимать отношение к продукту большого количества людей. Понимать, какие продукты и услуги востребованы на рынке в данный момент. При должном уровне автоматизации это занимает очень короткое время. Подходы к анализу тональности всё время эволюционируют и улучшаются, в том числе за счет более совершенных методов векторизации текста.

Векторизация текста – ключевой этап в любой задаче анализа текста. Она по сути и является переводом естественного языка на «машинный». Простые методы векторизации, такие как Bag of Words учитывают количество вхождения слова в текст. В них размерность вектора соответствует количеству уникальных слов в анализируемом массиве текстовых документов и исчисляется десятками и сотнями тысяч. Современные методы векторизации, такие как word2vec стремятся построить вектора меньшей размерности, которые заключают в себе семантическую и грамматическую информацию о соответствующем слове. Этот процесс требует большой вычислительной мощности, поскольку для построения каждого вектора учитываются всевозможные контексты применения слова, но конечный результат использовать проще и быстрее за счет меньшей размерности итогового вектора: несколько сотен измерений в word2vec против сотен тысяч в Bag of Words.

## Постановка задачи

Цель работы – исследование методов векторной репрезентации текста. Необходимо рассмотреть реализовать и сравнить между собой модели Bag of Words, TF-IDF и Word2Vec на примере задачи анализа тональности текста (сентимент-анализа).

Нужно обеспечить качество классификации в рамках выбранного алгоритма логистической регрессии. Оценить преимущества и недостатки каждого метода векторизации, выбрать наиболее подходящий для задачи такого рода.

В качестве платформы для проведения эксперимента выбран облачный ресурс “google colab”(https://colab.research.google.com/).

## Обзор литературы

Задачи компьютерной обработки естественного языка начали ставиться с самого времени появления вычислительных машин в 50-е годы. В первую очередь это были задачи машинного перевода. В 80-х годах машинное обучение переживает бурное развитие, а вместе с ним и обработка естественного языка. Ставятся новые задачи: определение частей речи, грамматический вывод и другие. С появлением Интернета задач становится еще больше: вопросно-ответные системы, спам-фильтры, распознавание именованных сущностей и много других, в том числе сентимент-анализ, описанный в данной работе.

Технологии в поле обработки естественного языка также развиваются очень бурно. В *Applied Natural Language Processing with Python* [1] описываются различные методы машинной обработки текста, в том числе Bag of Words, TF-IDF и Word2Vec, которые были рассмотрены в этой работе, и их приложения к некоторым из вышеупомянутых задач.

Обширный инструментарий для обработки естественного языка поставляет глубокое обучение (Deep Learning), в частности, повсеместно используются нейронные сети.

В книге *Neural Networks and Deep Learning - A Textbook* [2] детально и обширно рассматриваются нейронные сети.  Информация из нее была использована для определения нейронной сети, алгоритма ее работы и получения формул.

Проведение тонального анализа текста предполагает классификацию текстов по признаку эмоциональной окраски. Для этого существует много методов, в данной работе был выбран метод логистической регрессии, развернуто

описанный в книге A*pplied Logistic Regression* [3] с теоретической и практической точки зрения.

Одной из возможных проблем при классификации является несбалансированность классов: ситуация, когда объектов одного класса значительно меньше, чем объектов другого. Это приводит к тому, что классификатор пренебрежительно относится к классам-меньшинствам, предпочитая называть более многочисленные классы.

Одном из методов борьбы с этой проблемой является oversampling. Информация по популярному методу oversampling взята из статьи *SMOTE: Synthetic Minority Over-sampling Technique* [4]. В статье описан принцип работы алгоритма оверсэмплинга SMOTE и случаи, когда его следует применять. Также в ней присутствует описание метрик качества классификации. В этой работе приводится сравнение качества классификации с применением SMOTE и без него.

Наиболее интересным и современным способом векторного представления слов, рассмотренных в данной работе является метод Word2Vec, два алгоритма которого были представлены в 2013 году в работе *Efficient Estimation of Word Representations in Vector Space* [5]. В статье вводятся две архитектуры для вычисления векторных репрезентаций: Continuous Bag Of Words и SkipGram, приведены семантические интерпретации результирующего векторного пространства. Грамматические закономерности векторов из пространства Word2Vec, описаны в работе *Text comparison using word vector representations and dimensionality reduction* [6]. Одной из таких особенностей, например, является наличие вектора, при сдвиге на который многие глаголы прошедшего времени переходят в соответствующие глаголы настоящего времени.

Также были изучены видеозаписи курса лекций Стэндфордского Университета «Natural Language Processing with Deep Learning» (читают Chris Manning и Richard Socher) [8]. В них, кроме векторных репрезентаций излагаются другие инструменты компьютерной лингвистики, такие как, например, парсинг зависимостей.

## Описание и обработка данных

В качестве данных для исследования были извлечены отзывы с сайта ozon.ru на различные товары в количестве 769251. Текстовые обзоры являются короткими текстами порядка 10-50 слов.

Например: *«Один из наиболее заметных трудов в области теоретической антропологии и этнографии. Язык изложения доступен и качество перевода достойно замечательной серии "Этнографическая библиотека".»*

Каждый отзыв снабжен оценкой от 1 до 5, которая, по мнению автора отзыва, характеризует качество товара.

Вышеупомянутый товар получает оценку «5».

Таблица 1 описывает, сколько товаров соответствует каждой оценке.

| *Оценка* | *Количество отзывов* |
|---|---|
| «5» | 511671 |
| «4» | 110665 |
| «3» | 59226 |
| «2» | 55327 |
| «1» | 32362 |

Таблица 1

Поскольку мы не будем заниматься предсказанием оценки отзыва, а только лишь его эмоциональной окраски, преобразуем корпус. Пусть оценки «5» и «4» соответствуют Положительным отзывам, а «2» и «1» - отрицательным. От отзывов с оценкой «3» избавимся, как от нейтральных. Далее разделим выборку из отзывов на тренировочную и валидационную выборки в соотношении 85:15. Результат представлен в таблице 2:

| *Оценка* | *Количество отзывов в тренировочной выборке* | *Количество отзывов в валидационной выборке* |
|---|---|---|
| Положительная | 528985 | 93351 |
| Отрицательная | 74536 | 13153 |

Таблица 2

Разделение выборки на тренировочную и валидационную проводится при помощи функций, встроенных в пакет scikit-learn:

```
X_train, X_test, y_train, y_test = train_test_split(x_no3, y_no3,
                                                    stratify=y_no3,
                                                    test_size=0.15)
```

Заметим, что товаров с положительной оценкой гораздо больше, чем с отрицательной, это отрицательно скажется на качестве работы классификатора, но можно предпринять несколько путей решения этой проблемы.

Знаки препинания и заглавные буквы увеличат общее количество уникальных слов, что отрицательно скажется как на качестве работы алгоритма, так и на его скорости, так что преобразуем все прописные буквы в строчные и избавимся от знаков препинания за исключением восклицательных и вопросительных знаков, смайликов «:)» и «:)», в которых может содержаться ценная информация об эмоциональной окраске. После данной обработки вышеприведенный текст будет выглядеть так: *«один из наиболее заметных трудов в области теоретической антропологии и этнографии язык изложения доступен и качество перевода достойно замечательной серии этнографическая библиотека»*

В программной реализации на python обработка текста была проведена при помощи регулярных выражений.

```
train_list = []
for review in X_train:
  ms = re.findall(r'[а-я ]|:\)|\!|\?|:\(',review)
  ns = ' '
  for letter in ms:
    if re.match(r'[а-я!?()]',ns[-1])and re.match(r'[^а-я]',letter):
      ns+=' '
    ns+=letter
  ns = re.sub(r' +', ' ', ns)
  if ns[0] == ' ': ns = ns[1:]
  if len(ns)>0:
    if ns[-1] == ' ': ns = ns[:-1]
  train_list.append(ns.split())
```

При этом было уделено особое внимание тому, чтобы вопросительные и восклицательные знаки и смайлы оказались отделены от слов, для того чтобы в дальнейшем преобразовать их в самостоятельные токены – группы символов, которые будут рассматриваться наравне со словами.

Также из текста были удалены стоп-слова. Стоп-слова - это наиболее распространенные слова, которые не несут нужной эмоциональной информации. В основном это союзы, вводные слова, частицы. В данной работе сделано исключение для частиц «не», «без», «кроме» и некоторых других. Они модифицируют смысл последующих слов, что учитывается при векторизации с помощью word2vec.

Последним этапом предварительной обработки текста является стемминг. При стемминге слово приводится в начальную форму, что позволяет уменьшить количество уникальных слов в документе, объединяя однокоренные слова в одно. В результате чего дальнейшая работа алгоритма происходит быстрее. При этом информация об эмоциональной окраске слова не теряется.

Удаление стоп-слов и стемминг на python:

```
train_stemmed = []
for review in train_list:
  a = []
  for word in review:
    if a not in stops:
      c = stemmer.stem(word)
      a.append(c)
  train_stemmed.append(a)
```

# 2. Частотные векторные репрезентации

## 2.1 Модель Bag of Words

Рассмотрим простейшую модель – мешок слов (bag of words) [1]. В ней каждому тексту сопоставляется частотный вектор, который имеет размерность рассматриваемого словаря. В нем на i-м месте стоит количество вхождений i-го слова в данный документ.

Пример:

Рассмотрим два текста:

(1) «Боб идет в кино с Алисой.»

(2) «Боб гуляет с Вики. Вики друг Боба.»

В данном примере, не делая различий между прописными и строчными буквами, словарь будет состоять из слов:

{«алисой», «боб», «боба», «в», «вики», «гуляет», «друг», «идет», «кино», «с»}

В таком случае соответствующие этом предложениям BOW-векторы будут выглядеть следующим образом:

(1) [ 1, 1, 0, 1, 0, 0, 0, 1, 1, 1]

(2) [ 0, 1, 1, 0, 2, 1, 1, 0, 0, 1]

В корпусе отзывов на товары с ozon найдено 557194 уникальных слов. После запланированной обработки, их количество сокращено до 459786.

Программная реализация векторизации проводится на python при помощи функций из класса scikit-learn. Также для того чтобы смайлы и знаки препинания корректно выделялись в отдельные токены необходимо написать свой токенайзер.

```
def my_tokenizer(doc):
    return(doc.split(' '))

cv = CountVectorizer(tokenizer=my_tokenizer)
cv.fit(train_stemmed_s)
bow_train = cv.transform(train_stemmed_s)
bow_test = cv.transform(test_stemmed_s)
```

## 2.2 Модель TF-IDF

TF-IDF (от англ. TF — term frequency, IDF — inverse document frequency) — статистическая мера, используемая для оценки важности слова в контексте документа, являющегося частью коллекции документов или корпуса. Вес некоторого слова пропорционален частоте употребления этого слова в документе и обратно пропорционален частоте употребления слова во всех документах коллекции. [1]

(1) $$tfidf(t,d,D) = tf(t,d) \times idf(t,D)$$

Где t – слово, d(t) – документ, в контексте которого считается мера tf-idf, D – весь корпус документов.

(2) $$tf(t,d) = \frac{n_t}{\sum_k n_k}$$

$n_t$ – частота вхождений слова t в документ d, $\sum_k n_k$ – общее количество слов в документе d.

(3) $$idf(t,D) = log\frac{|D|}{|\{d_i \in D \mid t \in d_i\}|}$$

$|D|$ - число документов в корпусе

$|\{d_i \in D \mid t \in d_i\}|$ - число документов, в которых встречается слово t.

Программная реализация:

```
tf_s = TfidfVectorizer(tokenizer=my_tokenizer)
tf_s.fit(train_stemmed_s)
bow_train = tf_s.transform(train_stemmed_s)
bow_test = tf_s.transform(test_stemmed_s)
```

# 3. Векторные репрезентации, учитывающие контекст

## 3.1Нейронные сети

В третьем рассмотренном методе векторизации текста – SkipGram – используются искусственные нейронные сети. В этом параграфе будет дано определение этого объекта [2].

Искусственная нейронная сеть – векторная функция векторного аргумента. Может быть изображена в виде графа. Вершины графа называются нейронами, дуги между нейронами – коэффициентами или весами. Для наглядности на рисунке (рис.1) полносвязная нейронная сеть с пятью входами $x_i$, четырьмя выходами $y_i$ и одним скрытым слоем $a_i$, состоящим из четырех нейронов.

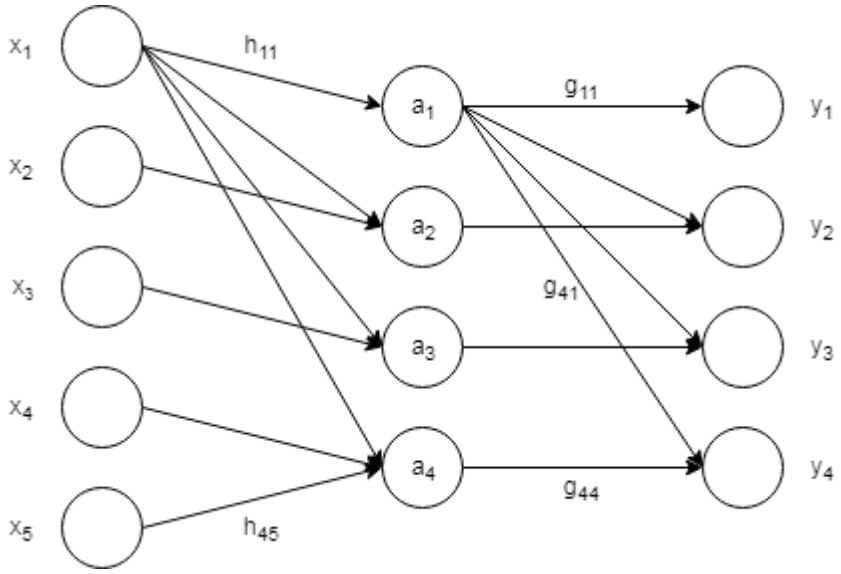



*Рис. 1*

Ради меньшей загроможденности на рисунке изображены не все связи. На самом деле каждый нейрон предыдущего слоя кроме одного соединен с каждым нейроном следующего. В каждом слое кроме выходного имеется нейрон (на диаграмме последний), значение которого не вычисляется по предыдущему слою, а полагается константой (bias neuron).

Входами и выходами нейронной сети служат действительные векторы. В скрытых нейронах вычисляются их значения $a_i$, также называемые активациями:

(4) $a_i = \varphi\left(\sum_j h_{ij} \times x_j\right)$

Суммирование ведется по всем нейронам предыдущего слоя. Коэффициенты $h_{ji}$ – вещественные числа. $\varphi$ – веществennозначная функция вещественного аргумента, называемая функцией активации. Для разных топологий нейронной сети и целей исследования выбираются разные функции активации, часто используется логистическая функция $\sigma(t) = \frac{1}{1+e^{-t}}$.

Эти значения затем ипользуются для вычисления значений нейронов следующего слоя. Или, как в примере с нейронной сетью с одним скрытым слоем, значений выходного слоя.

(5) $y_i = \varphi\left(\sum_j g_{ji} \times a_j\right)$

Значения коэффициентов нейронной сети определяются в процессе, называемом обучением нейронной сети. При этом решается задача максимизации функционала качества или минимизации функции потерь. Для этого необходима обучающая выборка – набор из входных данных $x^0$ и соответствующих им выходных данных $y^0$.

Вводится функция потерь $C(x^0, y^0, h)$, где $h$ - множество весов сети, характеризующая насколько отличается результат, вычисленный нейронной сетью на входных данных $x^0$ от предполагаемого «правильного» результата $y^0$. Функция потерь выбирается исходя из смысла решаемой задачи.

Целью обучения нейронной сети является минимизация функции потерь на обучающей выборке. Поскольку входные и выходные данные фиксированы, сделать это можно лишь путем изменения весов $h$. Обычно для этого используется метод градиентного спуска.

(6) $$h^{(l)} = h^{(l-1)} - \alpha \frac{\partial C(x^0, y^0, h)}{\partial h} \Big|_{h = h^{(l-1)}}$$

α – множитель градиентного спуска. Процесс повторяется до тех пор, пока не будет выполнен критерий остановки. В частности, критерием остановки может быть условие того, что модуль очередного приращения станет меньше некоторого наперед заданного значения.

В заключение заметим, что веса между любыми двумя слоями можно представить в виде матрицы $H = \{h_{ij}\}$, а активации соответствующих слоев объединить в вектора-столбцы: $x = \{x_j\}$, $a = \{a_i\}$.

Тогда активации $a$ вычисляются по формуле:

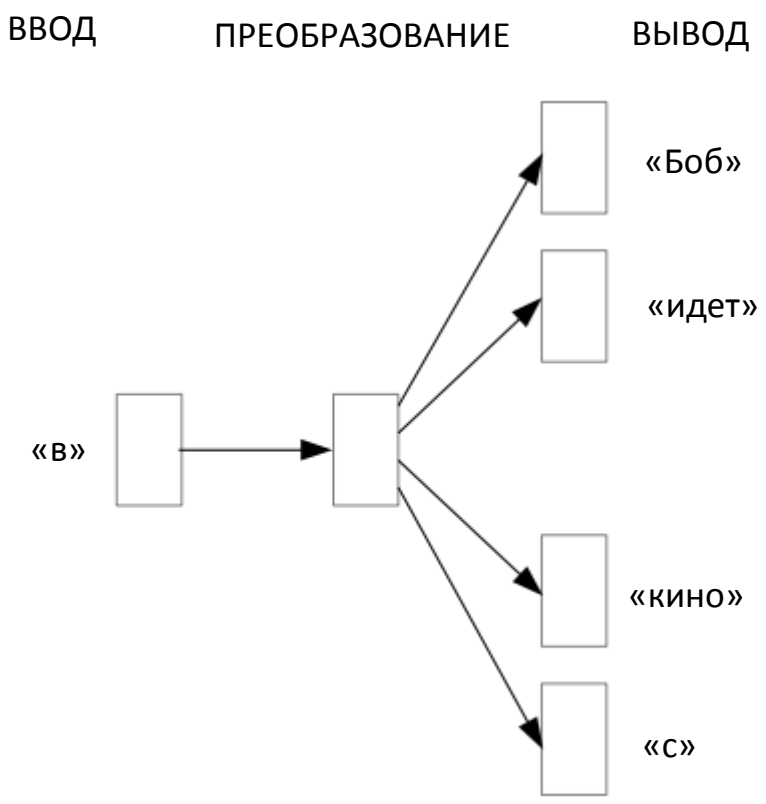
(7) $$a = \varphi(H \times x),$$

где функция активации вычисляется поэлементно.

## 3.2 Модель Word2Vec

Модель word2vec кардинально отличается от двух векторных репрезентаций, рассмотренных ранее. В ней каждому уникальному слову исходного корпуса сопоставляется вектор низкой размерности – несколько сотен против десятков или даже сотен тысяч в BoW и TF-IDF. Идеей метода является построение векторов так, чтобы часто встречающиеся в корпусе слова соответствовали близко расположенным векторам.

На самом деле под названием word2vec объединяются две похожих между собой модели, предложенные в 2013 году Томасом Миколовым и его командой из Google [5][8]: SkipGram и Continuous Bag Of Words (CBOW).




SkipGram

*Рис. 2*

В обеих моделях конечные вектора получаются путем обучения на корпусе слов. В модели SkipGram целью обучения ставится предсказание по каждому слову из корпуса его соседей (рис 2). Так в предложении:

«Боб идет в кино с Алисой.»

На очередной итерации обучения центральным словом – словом, по которому требуется предсказать его соседей – выбирается слово «в». Тогда нейронной сети, ответственной за обучение модели подается вектор слова «в» на вход и вектора слов «Боб», «идет», «кино», «с» как ожидаемый выход для данного входа.

Continuous Bag Of Words работает похожим образом, но в отличие от SkipGram нейронная сеть предсказывает не соседей слова по центральному слову, а центральное слово по его соседям (рис.3). Так, в нашем примере, на вход подаются вектора слов «Боб», «идет», «кино» и «с», а вектор слова «в» является ожидаемым результатом.

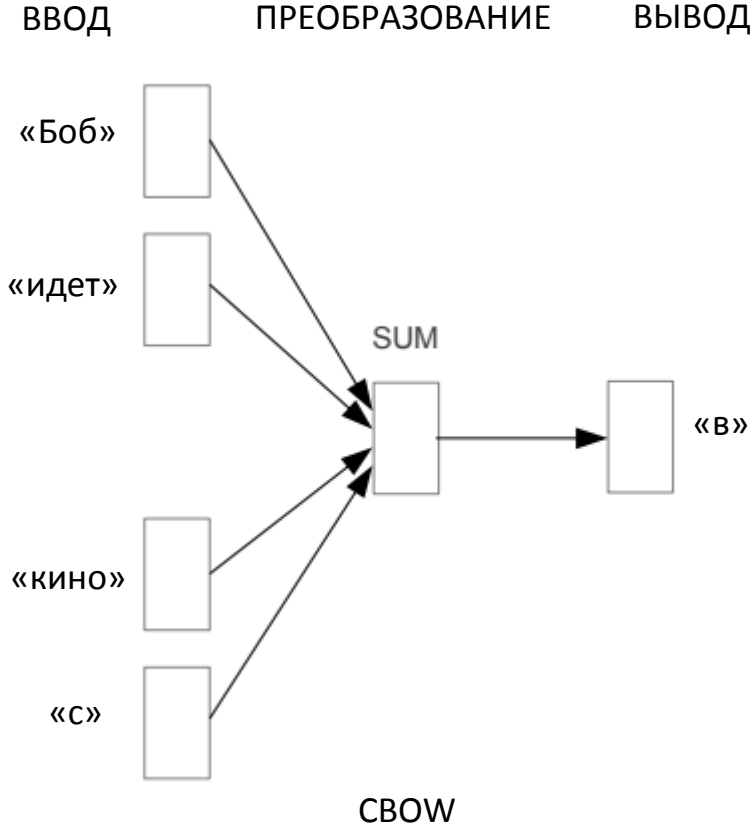



*Рис. 3*

Обе модели используются повсеместно, при этом CBOW работает немного быстрее, а SkipGram показывает слегка лучшие результаты при маленьких корпусах. [5]

Для более подробного рассмотрения выбран SkipGram.

### 3.3 SkipGram

Цель алгоритма SkipGram заключается в том, чтобы построить векторные репрезентации слов так, чтобы в итоговом пространстве близкие семантически и грамматически [6] слова лежали близко, а очень разные – далеко. При этом в качестве метрики для оценки близости берется косинус угла между векторами. Пусть d – размерность итогового пространства.

$$\cos(u, v) = \frac{u \cdot v}{\|u\| \times \|v\|}$$

Таким образом, длины векторов не имеют значения.

Эту задачу решает следующий алгоритм.

Для каждого слова t в корпусе размером T, которое будем в дальнейшем называть центральным, рассматривается «окно» радиуса m. Это m слов до центрального слова и m слов после. Введем полносвязную нейронную сеть, состоящую из одного входного слоя, одного выходного слоя и одного скрытого слоя между ними. На вход будет подаваться $v_t$ – репрезентация центрального слова t, представленная в виде унитарного кода – вектора размерности V с нулями на всех позициях, кроме той, которая соответствует данному слову t в ней стоит единица. V – количество уникальных слов в корпусе. На выходе получаются «распределения» вероятности встретить каждое уникальное слово в корпусе для всех позиций в «окне». Таким образом, размерность входного слоя – V, размерность выходного слоя – 2×m×V.

Введем целевую функцию:

(8) $$J'(\theta) = \prod_{t=1}^{T} \prod_{\substack{-m \le j \le m \\ j \neq 0}} p(w_{t+j}|w_t;\theta)$$

Она отражает вероятность встретить слово $w_{t+j}$ по соседству $w_t$ для всех слов в корпусе. Эту функцию нужно максимизировать.

Для удобства будем рассматривать функцию

(9) $$J(\theta) = -\frac{1}{T} ln\big(J'(\theta)\big) = -\frac{1}{T} \sum_{t=0}^{T} \sum_{\substack{-m \le j \le m \\ j \neq 0}} ln(\, p(w_{t+j}|w_t;\theta))$$

Поскольку было выполнено монотонно убывающее преобразование, функцию $J(\theta)$ необходимо минимизировать.

Параметр $\theta$ – векторные репрезентации слов в корпусе. По этому параметру будет проводиться оптимизация.

Вероятности будем рассчитывать по следующей формуле:

(10) $$p(o|c) = \frac{e^{{u_o}^T v_c}}{\sum_{w=1}^{v} e^{{u_w}^T v_c}}$$

Здесь $o$ – индекс рассматриваемого слова из «окна», которое требуется предсказать, $u_o$ – его векторная репрезентация, c – индекс центрального слова, $v_c$ – его векторная репрезентация. Суммирование в знаменателе ведется по всем уникальным словам в корпусе.

Функция

(11) $$\sigma(t)_i = \frac{e^{t_i}}{\sum_{j=1}^{n} e^{t_j}}, i = \overline{1,n}, \boldsymbol{t} = (t_1, \dots, t_n) \in \mathbb{R}^n$$

Называется softmax и является обобщением логистической функции на векторный случай.

Таким образом, в этой формуле в числителе вычисляется экспонента от скалярного произведения векторов слов $o$ и $c$, которое показывает близость этих векторов, а в знаменателе сумма по всем таким экспонентам. В результате $p(o|c)$ принимает значения в интервале (0,1) и тем больше, чем ближе вектора $o$ и $c$.

Для понимания процесса обучения рассмотрим топологию вышеупомянутой нейронной сети более подробно.

На вход подается вектор $w_t$ размерности V.

Второй слой представляет из себя вектор-репрезентацию слова $w_t - v_c$ и имеет размерность d.

Таким образом, матрица весов $W$ размерности d×V осуществляет переход от входного слова к его векторной репрезентации. Можно заметить, что при умножении $W \times w_t$ получается вектор, являющийся столбцом матрицы $W$, соответствующим входному слову. То есть матрица $W$ состоит из векторных репрезентаций слов в модели SkipGram и при её оптимизации будут найдены искомые вектора.

Выходной слой представляет собой 2×m векторов размерности V. Рассмотрим один из них. Он представляет собой функцию softmax от вектора из скалярных произведений векторной репрезентации центрального слова с каждой из векторных репрезентаций: $softmax({u_o}^T \cdot v_c), o = \overline{1,d}$. *Softmax* в данном случае играет роль функции активации выходного слоя. Заметим, что матрица весов между скрытым и выходным слоем размерности V×d имеет каждой строчкой векторную репрезентацию слова $u_o$ эта репрезентация отличается от репрезентации центрального слова $v_c$, и тоже является параметром нейронной сети, а поэтому оптимизируется в соответствии с функцией потерь $J(\theta)$.

Для реализации на python использовался пакет genism.

```
w2v_model = Word2Vec(ya_data_ready,\
                            workers=num_workers,\
                            size=num_features,\
                            min_count=min_word_count,\
                            window=context,
                            sample=downsampling)
```

# 4 Тональный анализ текста как задача классификации

## 4.1 Логистическая регрессия

Следующий этап после векторизации – построение классификатора. В данном случае в качестве признаков выступают вектора, полученные в результате одного из методов векторизации текста, обозначенные $x_i$, в качестве ответов – метки классов: «0» – отрицательный отзыв, «1» – положительный, обозначенные $y_i$.

Для решения задачи классификации выбран метод логистической регрессии [2], так как он быстро работает, хорошо подходит для классификации векторов большой размерности, а коэффициенты, полученные в результате обучения имеют хорошую интерпретацию.

При методе логистической регрессии в пространстве признаков строится разделяющая гиперплоскость. Классификатор будет присваивать объектам по одну сторону от нее метку «1», а по другую – метку «0».

$w$ – вектор весов для разделяющей гиперплоскости.

(12) $$h_w(x) = \frac{1}{1+e^{-(w^T x+w^0)}}$$

$h_w$ – целевая функция. Цель задачи классификации – подобрать вектор весов таким образом, чтобы значение целевой функции соответствовало вероятности принадлежности объекта $x$ классу «1».

Для этого введем функцию потерь

(13) $$Cost(h_w(x), y) = \sum_{i=1}^{m} [y_i \log\big(h_w(x_i)\big) + (1 - y_i)\log(1 - h_w(x_i))]$$

Суммирование производится по всем парам $(x_i, y_i)$.

Оптимизация производится с помощью метода градиентного спуска.

Программная реализация линейной регрессии осуществлена на python при помощи функций класса LogisticRegression пакета scikit-learn.

```
lr = LogisticRegression(C=0.05)
lr.fit(bow_train, y_train)
```

Поскольку Word2Vec, в отличие от остальных двух методов векторизации не создает векторную репрезентацию документа для классификации автоматически, нужно определить ее. В качестве признака для классификации документа для Word2Vec возьмем вектор – среднее арифметическое векторов всех слов в этом документе.

## 4.2 Метрики качества классификации

Оценку качества классификации может выражаться несколькими метриками. Наиболее используемые из них: доля верных ответов (accuracy), точность (precision), полнота (recall) и F1- мера – среднее гармоническое точности и полноты. [4]

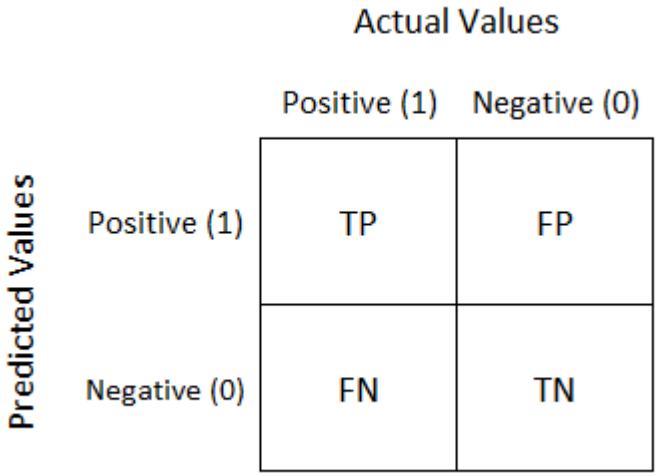



*Рис. 4*

На рис. 4 изображены всевозможные ошибки при классификации. Рассмотрим ситуации, которые могут возникнуть при присваивании объекту метки класса «1». TP (true positive) и TN (true negative) – это правильно определенные метки «1» и «0» соответственно. FP (false positive) – присвоение классификатором объекту класса «1» при том, что на самом деле объект принадлежит классу «0». FN (false negative) – присвоение классификатором класса «0» объекту, который в действительности принадлежит классу «1». В таблицу, приведенную выше, называемую матрицей ошибок, заносится результат работы алгоритма. Таким образом, под TP будем также иметь в виду количество ситуаций вида TP и т. д.

Введем метрики качества классификации.

(14) $accuracy = \frac{TP+TN}{TP+TN+FP+FN}$

Accuracy является самой простой метрикой и представляет собой долю правильно классифицированных объектов.

(15) $precision = \frac{TP}{TP+FP}$

Precision определяется тем, насколько много классификатор совершил ошибок, при которых объект класса «0» распознался как объект класса «1». Иначе говоря, метрика особо важна, когда необходимо получить как можно более «чистый» класс «1».

(16) $recall = \frac{TP}{TP+FN}$

Метрика recall определяется тем, насколько много классификатор совершил ошибок, при которых объект класса «1» распознался как объект класса «0». Иначе говоря, метрика особо важна, когда необходимо получить как можно более «полный» класс «1».

(17) $F1 = 2 \times \frac{precision \times recall}{precision+recall}$

Метрика F1 является средним гармоническим метрик precision и recall, ее значение быстро устремляется к нулю при низком значении хотя бы одной из составных метрик, что позволяет часто использовать только её.

Метрики качества определяются для отложенной выборки, на которой не производится обучение модели, чтобы не допустить утечки данных. Значения метрик получены при помощи функции classification_report из пакета scikit-learn.

```
y_true, y_pred = y_test, lr.predict(X_test)
print(classification_report(y_true, y_pred))
print(delta_per(y_true, y_pred))
```

Кроме стандартных метрик рассмотрена разность в пропорции класса «0» к классу «1» между тестовой выборкой и результатом работы классификатора, обозначенная Δ%. Её значение должно быть близко к нулю если при тональном анализе важно получить общую картину. Оно будет положительным если классификатор предсказал класс «0» больше чем следует и отрицательным если он предсказал класс «1» больше чем следует. Это значение вычисляется функцией:

```
def delta_per(y_true, y_pred):
  one = 0.0
  zero = 0.0
  for i in y_true:
    if i == 0: zero +=1
    else: one+=1
  ones = 0.0
  zeros = 0.0
  for i in y_pred:
    if i == 0: zeros +=1
    else: ones+=1
  return(zeros/ones-zero/one)
```

Еще одной метрикой используемой для оценки качества классификации является ROC AUC – площадь под ROC-кривой [9]. ROC-кривая – это зависимость между долей объектов, верно отнесенных к положительному классу (*чувствительностью*) и долей объектов, неверно отнесенных к положительному классу (*специфичностью*) при всевозможных порогах решающего правила (рис.5). Значение ROC AUC варьируется от 0 до 1. Чем оно ближе к единице, тем лучше классификатор справляется со своей задачей. Значение ROC AUC около 0.5 соответствует классификатору, который не лучше случайного угадывания метки класса.

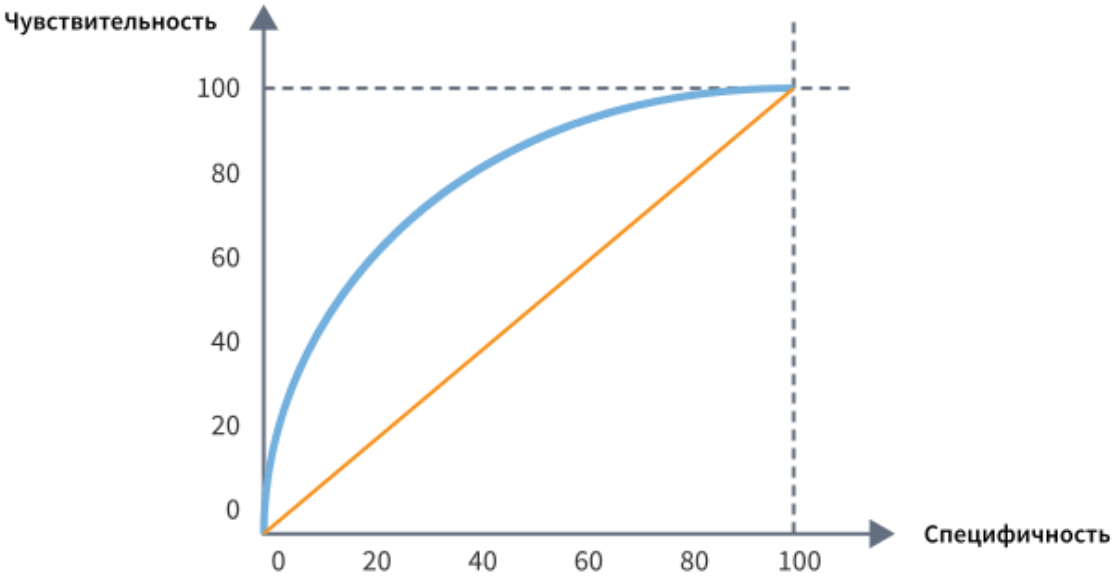

Чувствительность
100
80
60
40
20
0
0
20
40
60
80
100
Специфичность

*Рис. 5*

### 4.3 Методы работы с несбалансированными классами

Несбалансированные классы представляют проблему для точности работы классификатора. Поскольку при обучении, классификатор пытается минимизировать функцию потерь, рассматривая каждый пример из обучающей выборки наравне с другими, по факту, он старается максимизировать accuracy. Но поскольку в выборке 622336 объектов первого класса и 87689 второго, классификатор может получить accuracy = 0.88 просто относя все объекты к первому классу. Чтобы уровнять классы, применяется алгоритм SMOTE [4], который генерирует искусственные объекты недостающего класса, похожие на уже существующие в выборке. Обучим классификатор на выборке, состоящей уже из 622336 объектов как первого, так и второго класса. Такой метод борьбы с несбалансированными классами в международной литературе называют oversampling.

## Результаты работы алгоритма

Результаты классификации для частотных векторных репрезентаций:

Bag of Words

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 0.78 | 0.49 | 0.60 | 13153 |
| 1 | 0.93 | 0.98 | 0.96 | 93351 |

Δ% = -0.0572
roc_auc = 0.9348

TF-IDF

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 0.84 | 0.30 | 0.44 | 13153 |
| 1 | 0.91 | 0.99 | 0.95 | 93351 |

Δ% = -0.0954
roc_auc = 0.9185

Bag of Words + SMOTE

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 0.47 | 0.73 | 0.57 | 13153 |
| 1 | 0.96 | 0.88 | 0.92 | 93351 |

Δ% = 0.0944
roc_auc = 0.9252

TF-IDF + SMOTE

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 0.45 | 0.78 | 0.57 | 13153 |
| 1 | 0.96 | 0.87 | 0.91 | 93351 |

Δ% = 0.1305
roc_auc = 0.9086

Классификация с использованием частотных векторных репрезентаций показывает удовлетворительный результат. При необходимости более полного выявления отрицательных отзывов можно применить SMOTE,

однако при этом возрастает ошибка в пропорции между отзывами, классифицированными как положительные и отрицательные.

Кроме того, извлечены коэффициенты модели логистической регрессии. Большие значения означают что наличие соответствующего слова в документе «сигнализирует» о вероятной принадлежности документа к классу «1», а маленькие – к классу «0».

| Слово | Коэффициент в модели логистической регрессии |
|---|---|
| отличн | 2.0885 |
| довол | 1.8456 |
| пожалеет | 1.7836 |
| супер | 1.6837 |
| классн | 1.4445 |
| ... | ... |
| :) | 1.1042 |
| ! | 0.1916 |
| ? | -0.3672 |
| :( | -1.0903 |
| ... | ... |
| бесполезн | -1.7838 |
| разочарован | -1.8154 |
| выброшен | -1.9380 |
| ужасн | -1.9438 |
| отвратительн | -2.1611 |

Как видно, специальные символы действительно имеют немаленькие коэффициенты, однако и не самые большие. Самые большие же по модулю коэффициенты имеют слова с экспрессивной окраской.

Были рассмотрены две модели Word2Vec. Одна была обучена на полном тексте Википедии за 2014 год, тренировочном корпусе слов для перевода, составленным компанией Yandex, состоящем из 1 000 000 коротких фраз [10], а также тренировочной части корпуса отзывов с ozon.ru. Другая модель была обучна на коллекции книг на русском языке общим обхемом 12.9 млрд. токенов [11].

Для локально обученной модели были реализованы два варианта классификации: логистическая регрессия, обученная на всей тренировочной части датасета и на ста тысячах отзывов.

Результаты локально обученной модели Word2Vec:

Word2Vec (100k):

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 0.67 | 0.31 | 0.43 | 13140 |
| 1 | 0.91 | 0.98 | 0.94 | 93279 |

Δ% = -0.0798
roc_auc = 0.8946

Word2Vec (full):

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 0.67 | 0.32 | 0.43 | 13140 |
| 1 | 0.91 | 0.98 | 0.94 | 93279 |

Δ% = -0.0784
roc_auc = 0.8937

Word2Vec (100k) + SMOTE:

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 0.37 | 0.82 | 0.51 | 3607 |
| 1 | 0.97 | 0.81 | 0.88 | 26373 |

Δ% = 0.2247
roc_auc = 0.8940

Результаты также удовлетворительны. Причем меньший объем тренировочного корпуса не влияет на качество классификации. Это важно, так как не имеется возможности провести эксперимент с предобученной моделью Word2Vec на полном датасете: нахождение центральных векторов

заняло бы около шестидесяти часов.  Эксперимент с предобученной моделью проведен только для 100 000 документов в обучающей выборке и 30 000 в валидационной.

Результаты предобученной модели Worf2Vec.

Pre-trained Word2Vec:

```
   precision    recall  f1-score   support

0       0.73      0.01      0.01      3609
1       0.88      1.00      0.94     26377

Δ% = -0.1358

roc_auc = 0.8535
```

Pre-trained Word2Vec+SMOTE:

```
   precision    recall  f1-score   support

0       0.30      0.83      0.45      3609
1       0.97      0.74      0.84     26377

Δ% = 0.3549

roc_auc =  0.8587
```

Качество классификации несколько ниже чем у локально обученной модели, причем без использования SMOTE негативные отзывы распознаются с очень плохой полнотой.

# Заключение

Был проведен эксперимент с тремя различными векторными репрезентациями текста на примере задачи сентимент-анализа.

Качество классификации между моделями Bag of Words и TF-IDF различается незначительно и является достаточно высоким. Скорость работы обоих методов высокая: около минуты без SMOTE. Со SMOTE – порядка 20 минут для Bag of Words и 40 минут для TF-IDF.

Качество классификации с использованием Word2Vec уступает частотным моделям и является менее эффективным по времени – 2 часа для локальной модели и 9 часов для предобученной.

Поскольку качество классификации не упало при сокращении тренировочного датасета в шесть раз, можно предположить что более сложный классификатор, такой как SVM, Random Forest или классификатор на основе нейронной сети, например LSTM, справится лучше.

Также стоит отметить, что google colab является неоптимальной платформой для проведения подобных экспериментов, так как длительные вычисления часто самопроизвольно прекращались.

В итоге, для оптимизации качества классификации и временных издержек для данной структуры программы следует выбирать модель Bag Of Words.

## Список литературы

[1] Taweh Beysolow II Applied Natural Language Processing with Python. - San Francisco, California, USA: Springer Science+Business Media New York, 2018.

[2] Charu C. Aggarwal Neural Networks and Deep Learning - A Textbook. - Cham, Switzerland: Springer Nature Switzerland AG, 2018.

[3] David W. Hosmer, Stanley Lemeshow Applied Logistic Regression. - 2 изд. - Danvers MA: John Wiley & Sons, Inc, 2000.

[4] Nitesh V. Chawla, Kevin W. Bowyer, Lawrence O. Hall, W. Philip Kegelmeyer SMOTE: Synthetic Minority Over-sampling Technique // Journal of Artificial Intelligence Research. - 2002. - №16.

[5] Tomas Mikolov, Kai Chen, Greg Corrado, Jeffrey Dean. Efficient Estimation of Word Representations in Vector Space. // In Proceedings of Workshop at International Conference on Learning Representations (ICLP) – 2013, http://arxiv.org/abs/1301.3781

[6] Hendrik Heuer. Text comparison using word vector representations and dimensionality reduction // arXiv:1607.00534 [cs.CL]

https://arxiv.org/abs/1607.00534

[7] Chris Manning, Richard Socher. Natural Language Processing with Deep Learning. // запись курса лекций Стэндфордского университета по программе CS224d. – 2017,

https://www.youtube.com/playlist?list=PL3FW7Lu3i5Jsnh1rnUwq_TcylNr7EkRe6

[8] Tomas Mikolov, Kai Chen, Ilya Sutskever, Greg Corrado, Jeffrey Dean. Distributed Representations of Words and Phrases and their Compositionality // arXiv:1310.4546 [cs.CL]– 2013,

https://arxiv.org/abs/1310.4546

[9] Tilmann Gneiting, Peter Vogel. Receiver Operating Characteristic (ROC) Curves// arXiv:1809.04808 [stat.ME]

https://arxiv.org/abs/1809.04808

[10] https://translate.yandex.ru/corpus?lang=en

[11] https://nlpub.ru/Russian_Distributional_Thesaurus#.D0.92.D0.B5.D0.BA.D1.82.D0.BE.D1.80.D0.B0_.D1.81.D0.BB.D0.BE.D0.B2_.28word_embeddings.29

# Приложение 1

Исходный код программы можно найти по адресу

https://colab.research.google.com/drive/1fb9KrUz2uNJysdjdfH0Ejdi_A70QMLLk

https://colab.research.google.com/drive/1_Wcg47tzCtQ13sgdxZT_iN7Ec8Mj1iaE

Корпус отзывов можно запросить по почте: nikitin.maxim.94@gmail.com