|  |
| --- |
|  |
| МИНОБРНАУКИ РОССИИ  Федеральное государственное бюджетное образовательное учреждение  высшего образования  **«МИРЭА – Российский технологический университет»**  **РТУ МИРЭА**  Институт информационных технологий  Кафедра прикладной математики |

**РАБОТА ДОПУЩЕНА К ЗАЩИТЕ**

Заведующий

кафедрой\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Р.И. Дзержинский

«\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_ 2021  г.

**ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА**

по направлению подготовки бакалавров 09.03.01 Информатика и вычислительная техника

На тему: Анализ тональности текста новостей

Обучающийся *Бакуркин Кирилл Сергеевич*

*Подпись Фамилия, имя, отчество*

|  |  |  |
| --- | --- | --- |
| шифр | 17И0077 |  |
| группа | ИВБО-08-17 |  |

|  |  |  |  |
| --- | --- | --- | --- |
| **Руководитель работы** | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  *подпись* | *к.т.н., доцент* | *Скляр А.Я* |
|  |  |  |  |
| **Консультант** | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  *подпись* | *к.э.н., доцент* | *Григоренко О.В* |
|  |  |  |  |

Москва 2021 г.

|  |
| --- |
|  |
| МИНОБРНАУКИ РОССИИ |
| Федеральное государственное бюджетное образовательное учреждение  высшего образования  **«МИРЭА – Российский технологический университет»**  **РТУ МИРЭА** |
| Институт информационных технологий |
| Кафедра прикладной математики |

|  |  |  |
| --- | --- | --- |
| СОГЛАСОВАНО |  | УТВЕРЖДАЮ |
| Заведующий  кафедрой \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  *подпись* |  | Директор  института \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  *подпись* |
| Дзержинский Роман Игоревич |  | Зуев Андрей Сергеевич |

### 

### ЗАДАНИЕ

на выполнение выпускной квалификационной работы бакалавра

|  |  |  |
| --- | --- | --- |
| Обучающийся | Бакуркин Кирилл Сергеевич | |
|  | *Фамилия Имя Отчество* | |
| Шифр | 17И0077 | |
| Направление подготовки | 09.03.01 | Информатика и вычислительная техника |
|  | *индекс направления* | *наименование направления* |
| Группа | ИВБО-08-17 | |

**1. Тема выпускной квалификационной работы**: Анализ тональности текста новостей

**2. Цель и задачи выпускной квалификационной работы**

**Цель работы:** обучение модели на данных новостей для предсказания тональности новостных текстов

**Задачи работы:**

1. Поиск и разбор данных новостей
2. Семантический анализ структуры новостей
3. Анализ математических моделей для естественного языка
4. Написание программ и приведение экспериментов

**3. Этапы выпускной квалификационной работы**

|  |  |  |  |
| --- | --- | --- | --- |
| №  этапа | Содержание этапа выпускной квалификационной работы | Результат выполнения этапа ВКР | Срок выполнения |
| 1 | Ознакомление с предметной областью и изучение литературы по заданной теме |  | 27.04.2021 |
| 2 | Анализ предметной области и методов обработки текстов на естественном языке |  | 29.04.2021 |
| 3 | Предобработка корпуса данных |  | 30.04.2021 |
| 4 | Использование методов классификации текста новостей |  | 01.04.2021 |
| 5 | Получение результата тональности методами оценки классификаторов |  | 05.05.2021 |
| 6  6.1  6.2 | Экономический раздел  Организация и планирование работ по теме  Расчет стоимости проведение работ по теме |  | 09.05.2021 |
| 7 | Введение, заключение, список источников, приложения |  | 15.05.2021 |
| 8 | Презентация |  | 26.05.2021 |
| 9 | Нормоконтроль |  | 27.05.2021 |

**4. Перечень разрабатываемых документов и графических материалов:** печатная и электронная версии выпускной квалификационной работы бакалавра, презентационный материал с основными результатами выпускной квалификационной работы бакалавра ̶ модели бизнес-процессов в нотации BPMN 2.0 или IDEF0, схемы БД (модель данных на логическом и физическом уровне), макеты экранных форм, формы выходных документов, дерево функций, сценарий диалога, структурная схема, графический материал по исследованию предметной области

**5. Руководитель и консультанты выпускной квалификационной работы**

|  |  |  |  |
| --- | --- | --- | --- |
| Функциональные обязанности | Должность в Университете | Фамилия, имя, отчество | Подпись |
| Руководитель ВКР | к.т.н., доцент | Скляр А.Я. |  |
| Консультант по экономическому разделу | к.э.н., доцент | Григоренко О.В. |  |

|  |  |
| --- | --- |
| Задание выдал | Задание принял к исполнению |
| Руководитель ВКР:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | Обучающийся:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
| *подпис*ь | *подпис*ь |
|  |  |

# **АННОТАЦИЯ**

Выпускная квалификационная работа представлена в объеме 44 страниц (без учета приложения), состоит из 3 разделов, 18 рисунков, 8 источников информации, 4 таблиц и 35 формул.

Ключевые слова: тональность, классификация, машинное обучение.

В данной выпускной квалификационной работе производится обучение модели для определения тональности текста новостей. Объектом анализа являются данные статей новостных источников.

Основная цель работы – разработка автоматизированной системы для определения тональности текста новостей. Реализация выполнена средствами python.

В ходе выполнения работы проведен анализ различных типов математических и лингвистических моделей для обработки естественного языка.

В данной работе описывается подход к оценке эмоциональной окрашенности естественно-языковых текстов на основе данных новостных источников. Предложен метод автоматической оценки мнения СМИ с помощью анализа тональности новостей. Разработан исследовательский прототип программной системы, производящей анализ тональности естественно-языкового текста на русском языке. Выполнено тестирование, которое показало положительные результаты работы предложенных моделей в этой работе. Качество классификации показало результат 72,4% для модели TF-IDF и обучения с логистической регрессией, а также более 90% для модели с применением наивного байесовского классификатора.

Полученные результаты могут быть использованы в бизнес-задачах. Например, оценки отзывов в СМИ о запущенном компаниями продукте.

ОГЛАВЛЕНИЕ

[ВВЕДЕНИЕ 6](#_Toc74514959)

[1 Исследовательский раздел 7](#_Toc74514960)

[1.1 Предметная область 7](#_Toc74514961)

[1.2 Разновидности классификации для анализа текста 7](#_Toc74514962)

[1.3 Способы выявления признаков из текстовых данных 8](#_Toc74514963)

[1.3.1 Метод выявления признаков TF-IDF 8](#_Toc74514964)

[1.3.2 Наивный байесовских классификатор 10](#_Toc74514965)

[1.3.3 Метод опорных векторов 11](#_Toc74514966)

[1.3.4 Логистическая регрессия 14](#_Toc74514967)

[1.4 Методы оценки машинного обучения 17](#_Toc74514968)

[1.4.1 Матрица ошибок 17](#_Toc74514969)

[1.4.2 F1-мера 19](#_Toc74514970)

[1.4.3 Простые оценки 20](#_Toc74514971)

[2 Специальный раздел 21](#_Toc74514972)

[2.1 Инструменты разработки 21](#_Toc74514973)

[2.2 Методология 21](#_Toc74514974)

[2.3 Данные 22](#_Toc74514975)

[2.3.1 Чтение и обработка формата данных 23](#_Toc74514976)

[2.3.2 Токенизация 23](#_Toc74514977)

[2.3.3 Выделение N-грамм из выборки 24](#_Toc74514978)

[2.4 Обработка текста 26](#_Toc74514979)

[2.5 Обучение классификатора 27](#_Toc74514980)

[2.5.1 Применение TF-IDF с логистической регрессией 27](#_Toc74514981)

[2.5.2 Применение Байесовского классификатора 28](#_Toc74514982)

[2.6 Оценка классификатора 29](#_Toc74514983)

[3 Экономический раздел 33](#_Toc74514984)

[3.1 Планирование и организация работы по теме ВКР 33](#_Toc74514985)

[3.1.1 Расчёт стоимости проведения работ 36](#_Toc74514986)

[ЗАКЛЮЧЕНИЕ 43](#_Toc74514987)

[СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ И РЕСУРСОВ 44](#_Toc74514988)

[ПРИЛОЖЕНИЕ 1 45](#_Toc74514989)

**ВВЕДЕНИЕ**

С развитием голосовых интерфейсов и чат-ботов, обработка текстов на естественном языке стала одной из самых важных технологий искусственного интеллекта. Но полное понимание и воспроизведение смысла языка — чрезвычайно сложная задача, так как человеческий язык имеет ряд особенностей. Например, человеческий язык — специально сконструированная система передачи смысла сказанного или написанного. Это не просто экзогенный сигнал, а осознанная передача информации. Решение подобных проблем будет означать создание более удобной формы взаимодействия компьютера и человека.

Традиция изучения вербального проявления эмоций – лингвистическая эмотиология, являющаяся достаточно сформировавшейся традицией в лингвистике, в последние годы получила широкое распространение и развитие вследствие значительного роста текстового материала за счет популяризации Интернета и социальных сетей. С технологической стороны анализ тональности представляет собой «процедуру идентификации и извлечения из текста эмоционально и оценочно окрашенного отношения говорящего или пишущего к предмету высказывания».

**1 Исследовательский раздел**

## **1.1 Предметная область**

Решение задач области обработки текста предполагает понимание множество разных терминов и аббревиатур.

Модель машинного обучения – это приложение искусственного интеллекта (ИИ), которое дает возможность автоматически учиться и совершенствоваться на основе собственного опыта без явного участия человека

Обработка текстов на естественном языке (англ. Natural Language Processing, NLP) – смежное направление искусственного интеллекта и математической лингвистики. Фокусом для изучения являются проблемы интеллектуального анализа и синтеза естественного языка.

Анализ тональности текста (англ. Sentiment analysis) – задача компьютерной лингвистики, заключающаяся в определении эмоциональной окраски (тональности) текста и, в частности, в выявлении эмоциональной оценки авторов по отношению к объектам, описываемым в тексте. С точки зрения пользователя, люди могут публиковать собственный контент через различные социальные сети, такие как форумы, микроблоги или сайты социальных сетей в Интернете, а также различные источники СМИ. С точки зрения исследователя, многие сайты социальных сетей выпускают свои интерфейсы прикладного программирования (API), побуждая исследователей и разработчиков собирать и анализировать данные. Следовательно, задача анализа тональности имеет прочную основу при поддержке массивных онлайн-данных, которые предоставляются сообществом разработчиков.

## **1.2 Разновидности классификации для анализа текста**

Анализ тональности текста можно рассматривать как процесс классификации для выявления отличительных особенностей синтаксиса для исследуемого корпуса данных. Для работы с текстом существуют три основных уровня классификации: уровень документа, уровень предложения,уровень особенностей слов. Уровень документа – метод, который стремиться классифицировать документ из корпуса данных, который в свою очередь состоит из набора документов. Уровень предложения – метод классифицирует предложения из одного определенного документа. Для задач анализа текста изначально определяется тип предложения: объективный или субъективный. Если предложение субъективное, то для него выявление признаков более характерно, чем у объективного предложения. Однако принципиальной разницы между классификациями на уровне документов и предложений нет, потому что предложения – это просто короткие документы. Поэтому наиболее распространенный в использовании уровень классификации – это уровень аспектов. Он классифицирует конкретные аспекты конкретной сущности в тексте, чаще всего это слова, но также встречаются эмодзи и другие символы, которые выражают эмоции человека в тексте.

## **1.3 Способы выявления признаков из текстовых данных**

Для решения задач обработки естественного языка существует два основных подхода к классификации данных: контролируемый и неконтролируемый. В контролируемой классификации, классификатор обучается на размеченных данных, которые аналогичны тестовым примерам. Этот подход используется для классификации, когда у нас есть данные обучения, и он может решить проблему зависимости предметной области, а также при необходимости сократить данные обучения. Неконтролируемой классификация или еще часто называют «обучение без учителя» - подход, который основывается на внутренних различиях единиц данных. Например, слов или символов в слове.

### **1.3.1 Метод выявления признаков TF-IDF**

TF-IDF – это показатель, который используется для оценки важности слова в документе.

TF (частота слов) характеризует отношение числа вхождений конкретного слова к общему набору слов в документе. Чем выше TF, тем весомее конкретное слово в рамках документа.  Таким образом, оценивается важность слова t {\displaystyle t\_{i}}в пределах отдельного документа:

где {\displaystyle n\_{t}}n есть число вхождений слова {\displaystyle t}t в документ,

знаменатель – общее число слов в данном документе.

IDF (обратная частота документа) характеризует инверсию частотности, с которой конкретное слово используется в тексте. С помощью этой метрики можно снизить важность слов — например, союзов или предлогов:

где |D| ­­– число документов в коллекции,

­­– число документов из коллекции {\displaystyle D}D, в которых встречается {\displaystyle t}слово.

Таким образом, мера TF-IDF является произведением двух сомножителей:{\displaystyle \operatorname {tf-idf} (t,d,D)=\operatorname {tf} (t,d)\times \operatorname {idf} (t,D)}

Метрика TF-IDF используется в алгоритмах поисковых систем для проверки текстов. В частности, данный показатель включен в алгоритм проверки релевантности контента пользовательскому поисковому запросу во многих автоматизированных системах. Также этот показатель применяется для определения степени сходства нескольких документов.

Для того чтобы определить, захватывает ли модель TF-IDF необходимые нам классы, можем визуализировать метод и посмотреть, насколько хорошо разделены эти классы. Чаще всего словари из полученных весов являются очень большими, поэтому для корректного вывода векторного представления воспользуемся методом главных компонент (PCA). Проекция метода TF-IDF на рисунке 1.1 дает разделение слов на релевантные (например, имеют ли тексты отношение к катастрофам или не имеют) и нерелевантные слова в тестовом тексте.

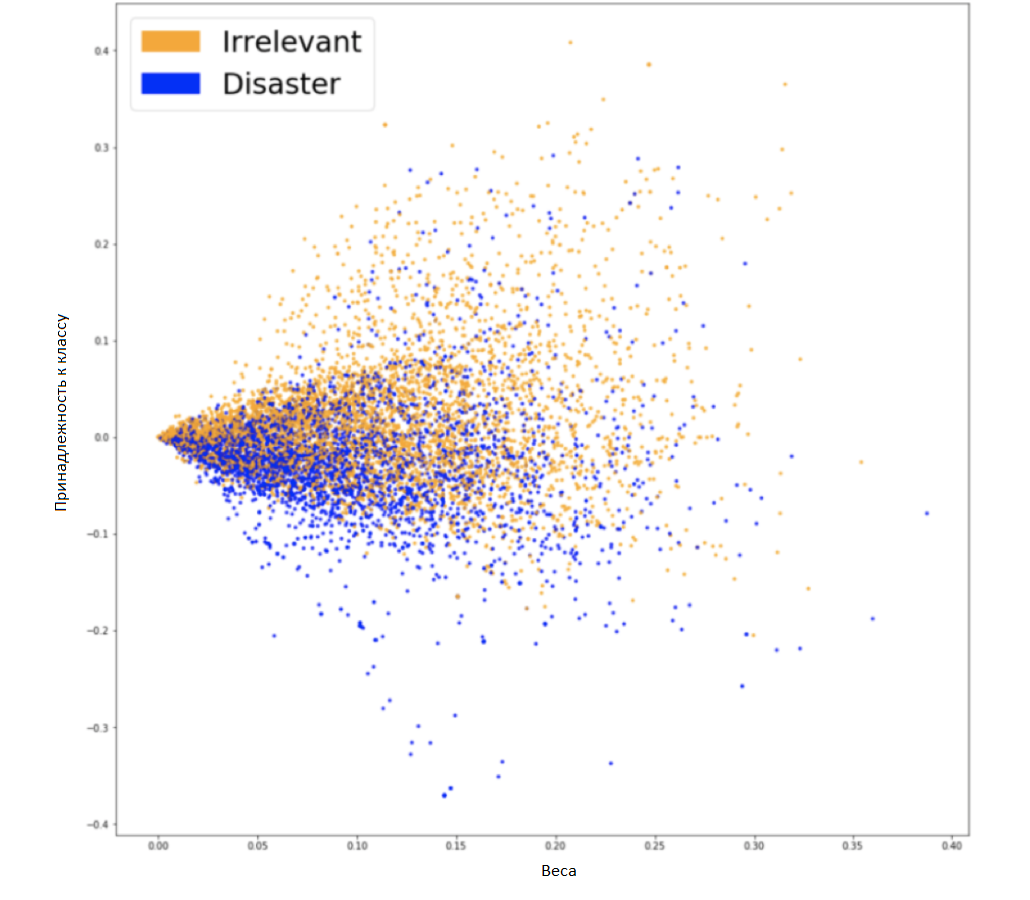


Рисунок 1.1 ­– Визуализация векторного представления с применением TF-IDF

Данное разделение свидетельствует о том, что при обучении машинной модели классификатор с большей вероятности может показать необходимый результат. Если бы мы на рисунке видели меньшее количество одного из цветов, то это значило бы, что метод не эффективен для текущих данных. Чаще всего для анализа используется тренировочная выборка, т.е применяется контролируемый подход к обучению.

### **1.3.2 Наивный байесовских классификатор**

Наивный байесовский классификатор ­– это вероятностная модель, основанная на предположении о независимости характеристик для классификации входных данных. Алгоритм чаще всего используется для классификации текстов, где используется тренировочная выборка.

В основе алгоритма лежит теорема Байеса для прогнозирования вероятности того, что данный набор функций принадлежит определенному классу. Теорема Байеса выглядит следующим образом:

где  – априорная вероятность гипотезы *c*,

 – вероятность гипотезы c при наступлении события *d* (апостериорная вероятность),

 – вероятность наступления события *d* при истинности гипотезы *c*,

– полная вероятность наступления события *d*.

Наивная байесовская модель классификации вычисляет апостериорную вероятность класса на основе распределения слов в документе. «Наивность» метода заключается в предположении, что вероятности попадания слов в определенный класс независимы друг от друга.

### **1.3.3 Метод опорных векторов**

Метод опорных векторов– это метод классификации как линейных, так и нелинейных данных. Если данные линейно разделимы, то метод ищет линейную оптимальную разделяющую гиперплоскость (линейное ядро), которая является границей решения, которая отделяет данные одного класса от другого.

Главный принцип метода опорных векторов ­– определить линейные разделители в пространстве поиска, которые могут лучше всего разделить различные классы. Математически разделяющую гиперплоскость можно записать так:

где *W* – вектор весов,

*X* – обучающий кортеж,

*b* – скаляр.

Чтобы оптимизировать гиперплоскость, задача трансформируется в минимизацию *W*, которая в конечном итоге вычисляется как:

где – числовые параметры выборки,

– метки, основанные на опорных векторах.

В итоге, алгоритм работы метода опорных устроен таким образом, что он ищет точки на графике, которые расположены непосредственно к линии разделения ближе всего. Эти точки называются опорными векторами. Затем, алгоритм вычисляет расстояние между опорными векторами и разделяющей плоскостью. Это расстояние называется зазором. Основная цель алгоритма – максимизировать расстояние зазора. Лучшей гиперплоскостью считается такая гиперплоскость, для которой этот зазор является максимально большим.

Разделяющая гиперплоскость определяется в пространстве уравнением при заданных и – множество принадлежит пространству меньшей размерности , где *n –* размерность пространства.

На рисунке 1.2 представлены три гиперплоскости *A*, *B* и *C*, разделяющие множества классов, которые выделены в виде кругов и звезд на плоскости. Гиперплоскость *A* обеспечивает наилучшее разделение между классами, поскольку нормальное расстояние любой из точек данных является наибольшим. Также гиперплоскость *A* разделяет на два полупространства и , что является разделением на классы.

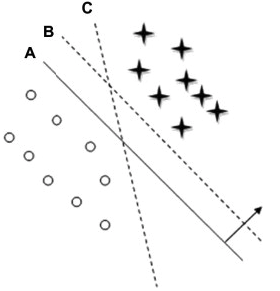


Рисунок 1.2 ­– Использование метода опорных векторов для линейной классификации

Если данные линейно-неразделимы, метод опорных векторов использует нелинейное отображение для преобразования данных в более высокое измерение. На рисунке 1.3 представлен пример нелинейного отображения точек.

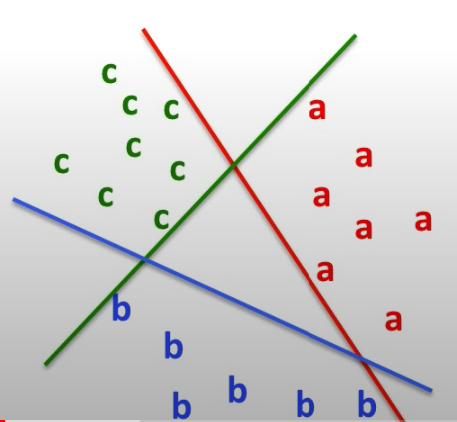


Рисунок 1.3 – Использование метода опорных векторов для нелинейной классификации

Основная проблема метода опорных векторов заключается в том, что при добавлении дополнительных измерений размер пространства функций увеличивается, что влияет на время обучения и занимает значительно больше ресурсов вычислительного устройства.

### **1.3.4 Логистическая регрессия**

Логистическая регрессия – это алгоритм классификации, используемый для отнесения наблюдений к дискретному набору классов. Логистическая регрессия преобразует свои выходные данные с помощью логистической сигмоидной функции, чтобы сопоставить предсказанные значения с вероятностными значениями. Функция отображает любое значение в диапазоне от нуля до единицы. В машинном обучении она используется для сопоставления прогнозов с вероятностями. Общий вид формулы логистической регрессии и ее график:

где x – входные параметры.

На рисунке 1.4 представлен график логистической функции.

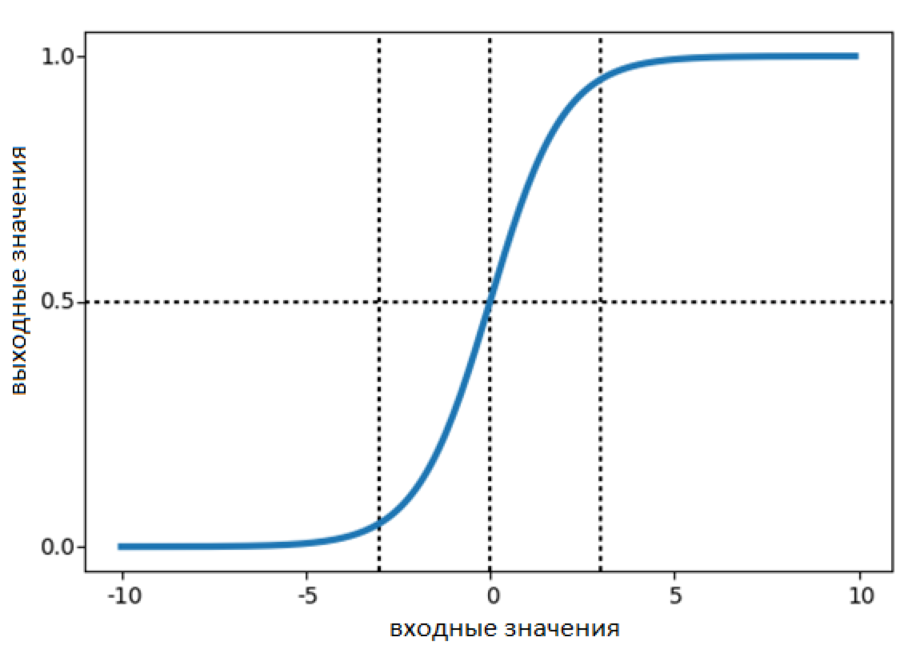


Рисунок 1.4 – График логистической функции

Представление гипотезы логистической регрессии для бинарной классификации:

Для небинарной классификации:

где – вектор параметров,

*x –* вектор признаков TF\_IDF,

k – количество классов.

Классификатор предоставит нам набор выходных данных или классов на основе вероятности, после этого нужно передавать входные данные через функцию прогнозирования и вернем оценку вероятности от 0 до 1. Для повышения точности модели используется функция стоимости, которая представляет собой оптимизацию, т.е создается функция, а затем минимизируется, чтобы модель получилась с минимальной ошибкой. Функция стоимости логистической регрессии имеет вид:

На рисунке 1.5 приведен график зависимости функции стоимости от гипотезы при y=1.

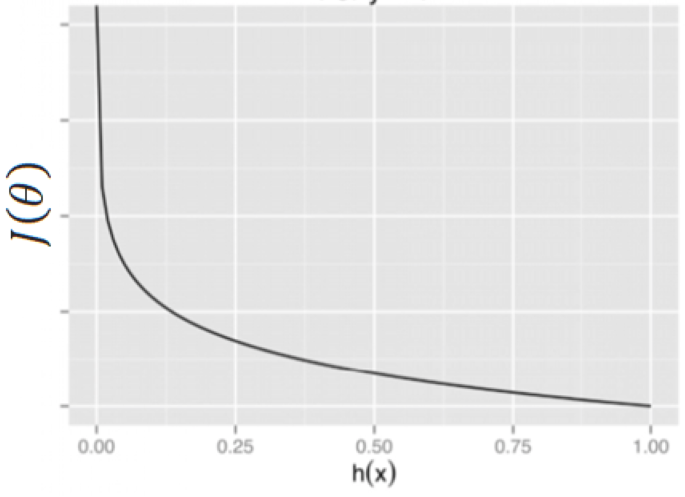
.

Рисунок 1.5 – Функция стоимости при y=1

На рисунке 1.6 приведен график зависимости функции стоимости от гипотезы при y=0.

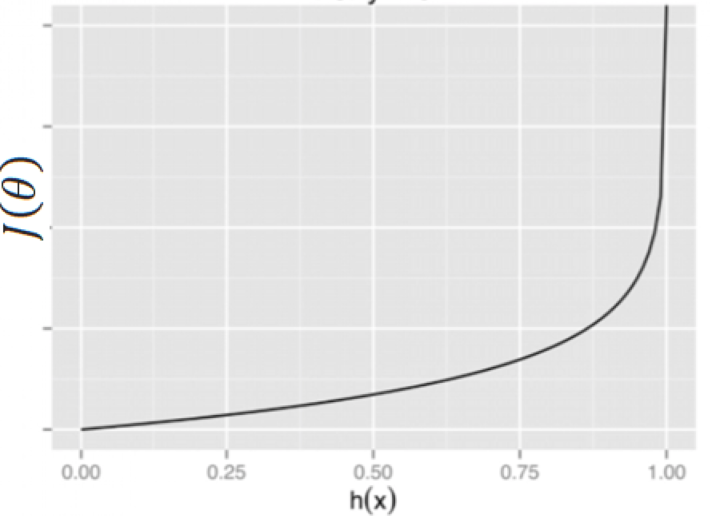


Рисунок 1.6 – Функция стоимости при y=0

Если объединить обе функции, то итоговая формула будет иметь вид:

Для минимизации функции стоимости используется градиентный спуск. Градиентный спуск применяется к каждому параметру :

## **1.4 Методы оценки машинного обучения**

Когда модель построена для задачи классификации, почти всегда возникает необходимость оценить точность этой модели, чаще всего оценивается количество правильных прогнозов из всех сделанных прогнозов. Это называется оценкой классификации, которая используется для проверки надежности классификатора. Одной точности классификатора недостаточно. Оценка помогает разработчикам решить, достаточно хороша модель для поставленной задачи.

### **1.4.1 Матрица ошибок**

Матрица ошибок – это матрица, которая часто используется для описания эффективности модели классификации (или классификатора) на наборе тестовых данных, для которых известны истинные значения. Матрица имеет размер N на N, где N – это количество классов. Например, для задачи двоичной классификации таблица имеет две строки и два столбца. Каждая ячейка содержит количество прогнозов, сделанных классификатором, которые попадают в эту ячейку. Классифицируемый документ из тестовой выборки инкрементируется в число, стоящее на пересечении строки класса, который вернул классификатор и столбца класса, к которому действительно относится документ. Имея такую матрицу точность и полнота для каждого класса рассчитывается очень просто. Точность равняется отношению соответствующего диагонального элемента матрицы и суммы всей строки класса. Полнота – отношению диагонального элемента матрицы и суммы всего столбца класса. На рисунке 1.7 представлена матрица ошибок.

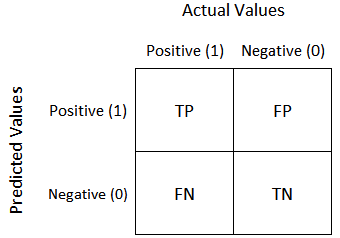


Рисунок 1.7 – Матрица ошибок для бинарной классификации

В матрице содержится информация сколько раз система приняла верное и сколько раз неверное решение по документам заданного класса. А именно:

* TP — истино-положительное решение;
* TN — истино-отрицательное решение;
* FP — ложно-положительное решение;
* FN — ложно-отрицательное решение.

Тогда, точность и полнота определяются следующим образом:

Точность – это количество истинных положительных результатов, разделенное на количество истинных положительных и ложных положительных результатов. Точность можно рассматривать как меру точности классификаторов. Низкая точность также может указывать на большое количество ложных-положительных предсказаний классификатора.

Полнота – это количество истинных положительных результатов, разделенное на количество истинных положительных результатов и количество ложных отрицательных результатов. Полноту можно рассматривать как меру полноты классификаторов. Низкий уровень отзыва указывает на множество ложно-отрицательных результатов.

### **1.4.2 F1-мера**

Для общей оценки качества классификатора часто используют F1 меру – среднее гармоническое между точностью и полнотой. Оценка F1 отражает баланс между точностью и отзывчивостью:

Данная формула придает одинаковый вес точности и полноте, поэтому F-мера будет падать одинаково при уменьшении и точности и полноты. На рисунке 1.8 приведена сбалансированная F-мера в зависимости от точности и полноты.

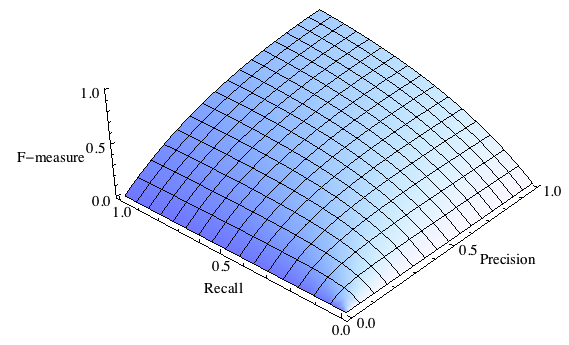


Рисунок 1.8 – Сбалансированная F-мера

### **1.4.3 Простые оценки**

Простые оценки никак не учитывают изначальное распределение классов в выборке, но эффективны для предварительной оценки классификатора. Также часто они являются частью алгоритмов оценки, например F1-меры.

Accuracy (точность) – показывает долю правильных классификаций. Несмотря на очевидность и простоту является одной из самых малоинформативных оценок классификаторов:

Specificity (специфичность) ­­– показывает долю верных срабатываний классификатора к общему числу объектов за пределами класса. Проще говоря, показывает то, насколько часто классификатор правильно не относит объекты к классу:

**2 Специальный раздел**

## **2.1 Инструменты разработки**

Для решения задач в области анализа данных чаще всего используется язык программирования Python, а также сопутствующие ему инструменты и библиотеки. Python – это очень общий язык программирования с множеством встроенных библиотек, который отлично справляется с управлением данными, сетевым программированием и базами данных. Одним из инструментов этого языка программирования является Jupyter notebook – это интерактивная вычислительная среда, в которой можно комбинировать выполнение кода, форматированный текст, математику и графики. Реализуемое программное обеспечение с использованием данной технологии имеет формат ipynb, который позволяет удобно взаимодействовать с кодовой базой, графиками и другими элементами программы. Для решения задачи анализа тональности текста будут использованы программные пакеты python, такие как: deeppavlov, scikit-learn, samsung it academy stepik-dl-nlp, nltk, pandas.

## **2.2 Методология**

Методология, используемая для анализа тональности новостных статей в этой статье, основана на подходе, с применением обучающей выборки.

Процесс обработки данных представлен на рисунке 2.1.

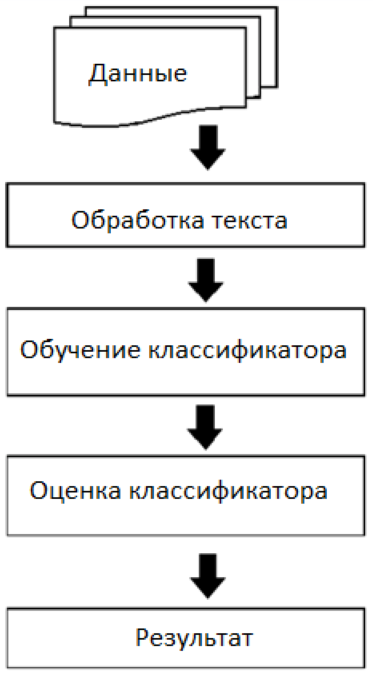


Рисунок 2.1 – Методология анализа тональности текста новостей

## **2.3 Данные**

Данные, используемые в качестве тренировочной выборки, представляют собой тексты различных новостных источников СМИ на русском языке. Исходный формат файла – JSON, то есть структура данных имеет ключ и значение. Ключ «text» содержит в значении текст новости; «id» содержит уникальный идентификатор этой новости; «sentiment» содержит метку тональности новости, которая может принимать значения трех видов эмоционального окраса: негативный, нейтральный и позитивный. На рисунке 2.2 приведен пример, который показывает структуру файла данных в формате JSON.

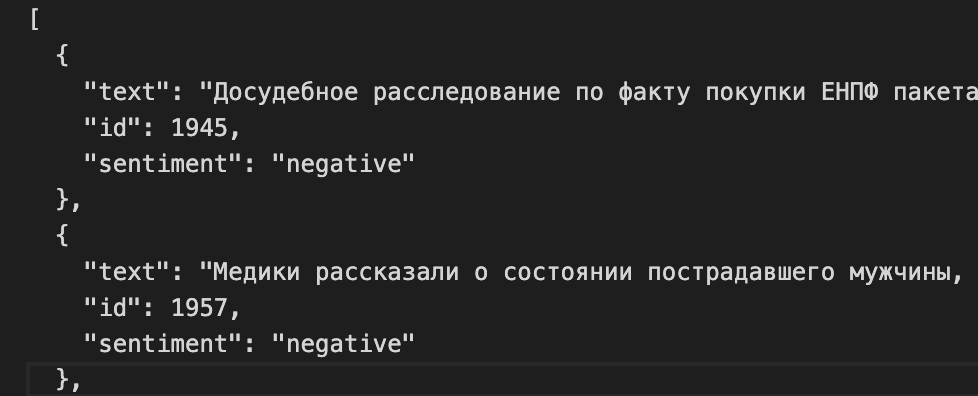


Рисунок 2.2 – Данные в формате JSON

### **2.3.1 Чтение и обработка формата данных**

Первый этап взаимодействия с тренировочной выборкой – это обработка и приведение данных в удобный для взаимодействия вид. Для этого данные нужно считать из файла и преобразовать в формат DataFrame. DataFrame – это структура данных, которая содержит помеченные оси, т.е строки и столбцы, а арифметические операции выравниваются по меткам строк и столбцов. На рисунке 2.3 приведены данные в формате DataFrame.

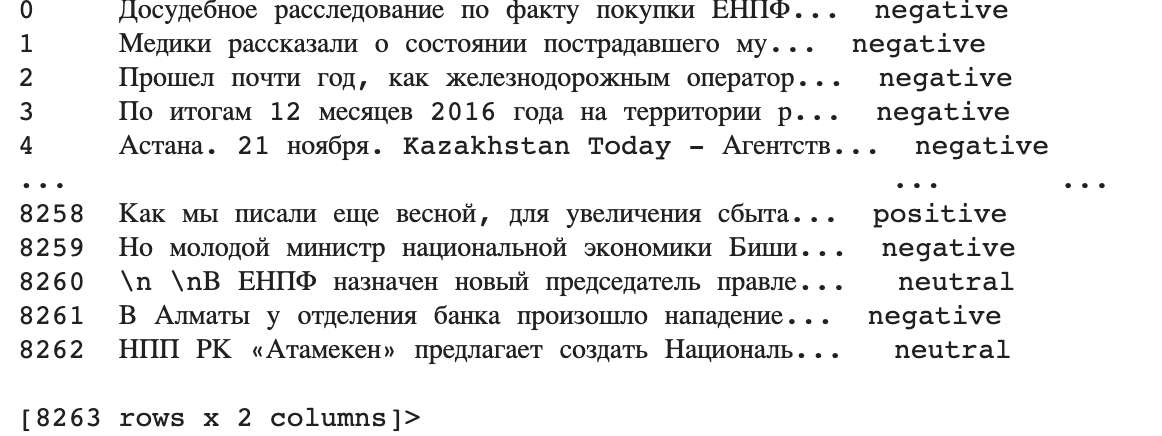


Рисунок 2.3 – Данные в формате DataFrame

### **2.3.2 Токенизация**

Токенизация – процесс разделения всего текста на небольшие части, также известные как токены, которые затем могут быть использованы для различных преобразований перед обучением модели решения задачи. Процесс токенизации текста помогает программе воспринимать текст как поток слов или токенов, а не букв или байтового представления символов.

В программном пакете nltk содержится метод для токенизации данных, который преобразовывает коллекцию текстовых документов в вектор количества токенов. В итоге, получается векторное представление текстовых данных. Пример работы метода токенизации представлен далее.

До токенизации:

«МИРЭА лучший ВУЗ, все знают».

После токенизации:

[‘мирэа’, 'лучший ', ‘вуз', ',', 'все','знают’].

### **2.3.3 Выделение N-грамм из выборки**

Для применения N-грамм требуется преобразовать данные и распределить их частоту появления в тексте. Выделим униграммы и биграммы. Для задачи обработки текста этого достаточно, триграммы и выше не дадут ожидаемую точность работы модели. Распределение относительных частот слов биграмм и униграмм приведены на рисунке 2.4 и рисунке 2.5 соответственно.

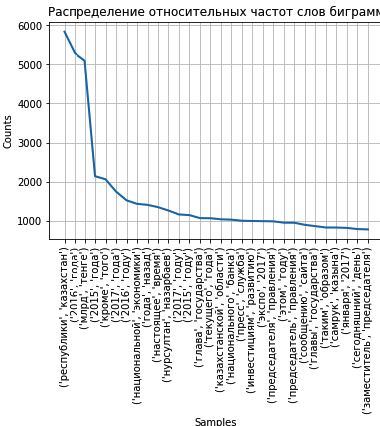


Рисунок 2.4 – Распределение относительных частот биграмм



Рисунок 2.5 – Распределение относительных частот слов униграмм

## **2.4 Обработка текста**

Предварительная обработка – это необходимый шаг для очистки текста (уменьшения шума текста) и уменьшения несогласованности в нем, чтобы эти очищенные данные можно было более эффективно использовать в задачах интеллектуального анализа текста.

Первый этап обработки – удаление лишних атрибутов текста, таких как эмодзи. Выполнено это с применением регулярных выражений и таблицы юникодов.

Второй этап – это разделение слипшихся слов. Это нужно для того, чтобы учитывать в обучении орфографические ошибки и опечатки, т.к такие случаи могут иметь значительный вес в классификации данных. Для того чтобы разделение слов было возможно используется частотный корпус русского языка, который находится в открытом доступе на сайте национального корпуса русского языка. При обработке будем проходить по словам в корпусе. Если находим слово, которое не существует в словаре, будем делить его на части полным перебором. При каждом разделении вычисляем вероятность того, что данное разделение возможно, основываясь на частотных вероятностях полученных частей слова. В итоге, выбираем разделение наиболее вероятное. Для ускорения работы программы используется кэширование – это высокоскоростной уровень хранения, на котором требуемый набор данных храниться в оперативной памяти вычислительной машины. Доступ к данным на этом уровне кэша осуществляется значительно быстрее, чем к основному месту их хранения. С помощью кэширования становится возможным эффективное повторное использование ранее полученных или вычисленных данных.

Третий этап – это лемматизация и стемминг. Лемматизация – процесс преобразования слова в его базовую форму. Разница между стеммингом и лемматизацией заключается в том, что лемматизация учитывает контекст и преобразует слово в его значимую базовую форму, тогда как стемминг просто удаляет последние несколько символов, выделяя корень слова и отбрасывая суффиксы и окончания. Как итог, слово преобразуется в базовую форму. Пример: кошками → кошка.

Данный алгоритм нужен для сокращения словаря наших данных. Алгоритм лемматизации реализован в библиотеке python – DeepPavlov.

Завершающий этап – это удаление стоп-слов. Слова, которые не повлияют на обучение, а лишь будут тратить больше ресурсов при обучении. Чаще всего такие слова являются предлогами и союзами, также другие атрибуты, которые часто встречаются во всех русскоязычных текстах.

## **2.5 Обучение классификатора**

### **2.5.1 Применение TF-IDF с логистической регрессией**

После предварительной обработки текста будем использовать статистический метод TF-IDF. Согласно этой методике слова, которые часто встречаются в документе, считаются важными, и этим словам придается вес. Чем больше вес – тем больше важность слова, которая используется при обучении классификатора. С помощью TF-IDF важные слова или термины в документе были идентифицированы, и им был присвоен вес в соответствии с появлением различных слов в новостной статье.

Для обучения используется логистическая регрессия для мультиклассификации, т.к в текущей работе присутствует три класса тональности: позитивный, негативный и нейтральный. Данная регрессия в программном коде использует «softmax» функцию –  это обобщение логистической функции для многомерного случая. Функция преобразует вектор *Z* р азмерности *K* {\displaystyle K} в вектор {\displaystyle \sigma } той же размерности, где каждая координата  {\displaystyle \sigma \_{i}} полученного вектора представлена вещественным числом в интервале [0,1] и сумма координат равна 1. Координаты {\displaystyle \sigma \_{i}}  {\displaystyle \sigma \_{i}}вычисляются следующим образом:

{\displaystyle \sigma \_{i}}

где K = 3, т.к показывает количество наших классов,

Z – вектор весов, полученные в результате вычисления TF-IDF.

Функции расчетов логистической регрессии предоставлены в программном пакете sklearn. Выполненные настройки и применение пакетных функций представлены в разделе приложения. [[1]](#_ПРИЛОЖЕНИЕ)

### **2.5.2 Применение Байесовского классификатора**

Как правило, информация о настроении передается прилагательными или, более конкретно, определенными комбинациями прилагательных с другими частями речи. Эта информация может быть получена путем добавления таких функций, как последовательные пары слов биграммы, но также учтем и униграммы. На полученные n-граммы применим байесовский классификатор для получения тональности новостей.

Новость d представлена в виде вектора атрибутов:

где w – это элемент униграмм или биграмм.

Вычисление нетривиально, поэтому вводится предположение, что все значения характеристик независимы с учетом метки :

На основе этого уравнения можно построить максимальный апостериорный классификатор, путем поиска оптимальной метки, которая максимизирует апостериорный классификатор :

где — предсказанный класс,

= 1 для каждого класса *с*.

где – общее количество обучающих новостей,

– количество новостных текстов в обучающей выборке с классом

где – количество встречаемого слова в обучающей выборке с классом ;

V – количество слов или длина списка в python всех элементов униграмм или биграмм:

## **2.6 Оценка классификатора**

Для того чтобы удостовериться насколько правильно прошло обучение классификатора требуется воспользоваться методами оценки. Наиболее оптимальной оценкой для модели TF-IDF c логистической регрессией является простой метод оценки точности:

На рисунке 2.6 представлен вывод программы и результаты по другим метрикам, также общий результат.

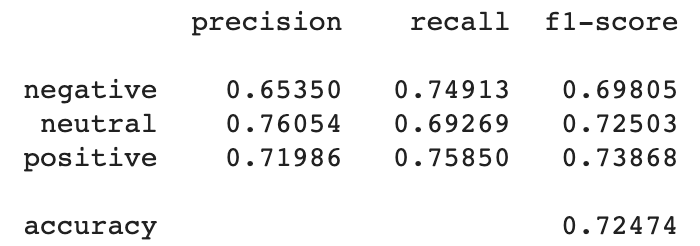


Рисунок 2.6 – Вывод работы программы с TF-IDF

Полученный результат является приемлемым в области обработки естественного языка. Предположительно, при большей выборке результат окажется лучше.

Для наивного байесовского классификатора с применением униграмм и биграмм следует применить более сложный метод оценки – F1-мера, которая вычисляет среднее гармоническое точности и полноты. Также для более точного анализа используем матрицу ошибок. Расчеты произведены с помощью программного пакета python sklearn. Всего классифицировано новостей: 3171

Результат точности f-1 меры байесовского классификатора с униграммами:

Более подробный алгоритм расчета приведен в программном коде и выполнен с применением библиотек python. В результате получаем матрицы ошибок для двух случаев. На рисунке 2.7 приведена матрица для униграмм. На рисунке 2.8 приведена матрица для биграмм.

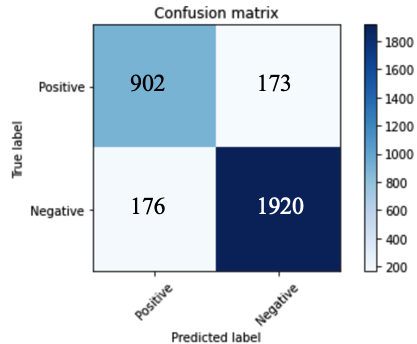


Рисунок 2.7 – Матрица ошибок с применением униграмм

Результат точности f1-меры байесовского классификатора с биграммами:

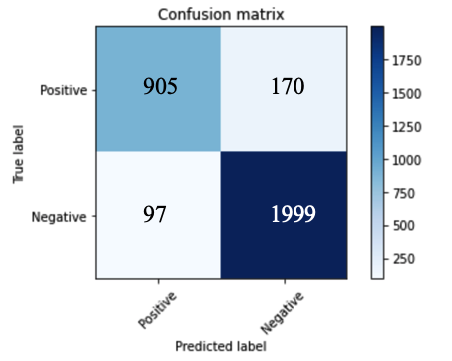


Рисунок 2.8 – Матрица ошибок с применением биграмм

Используемые алгоритмы дают лучше результат, но это также увеличивает время, необходимое для обработки и обучения, чем обучение предыдущего классификатора.

Классификаторы обучены до приемлемой точности и выбор между ними исходит уже из задачи и возможностей вычислительного устройства. Модели можно применять для анализа тональности текста новостей, а также интегрировать в другие автоматизированные системы.

**3 Экономический раздел**

**3.1 Планирование и организация работы по теме ВКР**

В разработке алгоритмов решения задачи обработки естественного языка и построению моделей машинного обучения для анализа тональности текста задействовано три человека:

* руководитель выпускной квалификационной работы – ответственен за грамотность в постановке задачи, осуществляет контроль за некоторыми этапами работы, вносит требующиеся поправки и даёт оценку выполненной работе в целом;
* консультант по экономической части ВКР – отвечает за консультирование экономической части выпускной квалификационной работы;
* разработчик – исполняет все поставленные задач, в том числе проводит тестирование готового продукта и подготавливает документацию проекта.

Состав задействованных в работе участников отображен на рисунке 3.1.

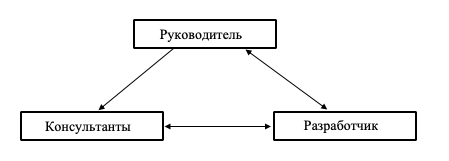


Рисунок 3.1 – Состав задействованных в работе участников

На разработку и реализацию выделено 60 рабочих дней. В таблице 3.1 представлены все этапы разработки.

Таблица 3.1 – Этапы разработки

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **№** | **Название этапа** | **Исполнитель** | **Трудоемкость,**  **чел/дни** | **Продолжительность работ, дни** |
| 1 | 2 | 3 | 4 | 5 |
| 1 | Разработка и утверждение технического задания | Руководитель | 5 | 5 |
| Разработчик | 5 |
| 2 | Техническое предложение | Руководитель | 3 | 3 |
| Консультант | 2 |
| Разработчик | 3 |
| 3 | Эскизный проект: |  |  | 12 |
| 3.1 | Оценка исходных данных и требований | Разработчик | 3 |
| 3.2 | Постановка задачи | Консультант | 2 |
| 3.3 | Общее описание алгоритма | Руководитель | 3 |
| Разработчик | 7 |
| 4 | Технический проект: |  |  | 8 |
| 4.1 | Определение способа представления входных и выходных данных | Руководитель | 2 |
| Разработчик | 3 |

Продолжение таблицы 3.1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 |
| 4.2 | Разработка структуры программы и модулей | Руководитель | 2 | 8 |
| Консультант | 2 |
| Разработчик | 5 |
| 5 | Рабочий проект: |  |  |  |
| 5.1 | Программирование и отладка программы | Разработчик | 15 |
| 5.2 | Испытание программы | Разработчик | 5 | 32 |
| 5.3 | Коррекция программы по результатам испытаний | Разработчик | 5 |
| 5.4 | Составление  технической документации на программный продукт | Консультант | 1 |  |
|  |
| Разработчик | 2 |
| 5.5 | Сдача и внедрение готового продукта | Руководитель | 1 |
| Консультант | 1 |
| Разработчик | 5 |
| **Итого** | | | 82 | 60 |

Календарный график исполнения работ представлен на рисунке 3.2.

Из рисунка наглядно видно, что весь срок разработки составляет 60 дней.

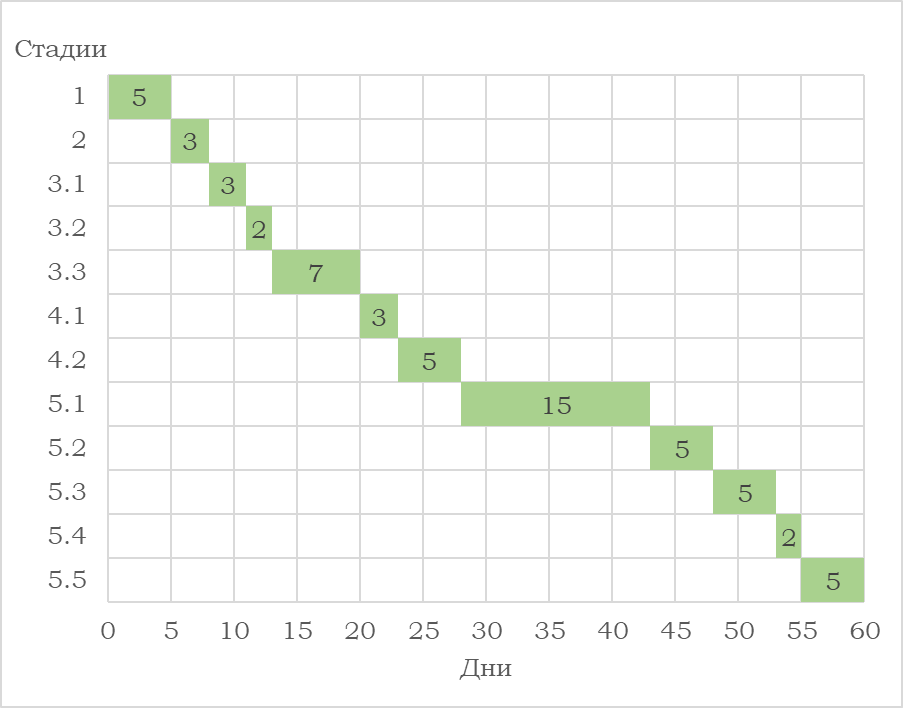


Рисунок 3.2 – График исполнения работ

**3.1.1 Расчёт стоимости проведения работ**

В выпускной квалификационной работе объем затрат на НИР (научно-исследовательские работы) был проведен методом калькулирования.

Статья 1 «Материалы, покупные изделия и полуфабрикаты + ТЗР (20%) от суммы итога по материалам.

Статья 2 «Специальное оборудование» – чаще всего, затрат нет.

Статья 3 «Основная заработная плата».

Статья 4 «Дополнительная заработная плата» 20-30% от основной заработной платы.

Статья 5 «Страховые отчисления» – 30% от ФОТ.

Статья 6 «Командировочные расходы» – чаще всего, затрат нет.

Статья 7 «Контрагентские услуги» – чаще всего, затрат нет.

Статья 8 «Накладные расходы» – 200% от основной заработной платы.

Статья 9 «Прочие расходы».

В статье «Материалы, покупные изделия и полуфабрикаты» рассчитывается стоимость сырья, полуфабрикатов, покупных изделий, комплектующих товаров и прочих материальных ценностей, необходимых в процессе выполнения научно-исследовательской работы.

На основании спецификации норм материальных затрат, в которых будет указано наименование, цена и количество расходуемых в процессе работы материалах, определяется потребность в физических ресурсах. Транспортно-заготовительные расходы включаются в стоимость материальных затрат. Возьмем их на уровне 20% от общей стоимости затрат по статье. Расходы на оформление комплекта документов будут также включены в статью.

Полуфабрикаты, покупные изделия и материалы описаны в таблице 3.2.

Таблица 3.2 – Материалы, покупные изделия и полуфабрикаты

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| № пп | Наименование  материалов | Единицы измерения | Количество | Цена за единицу (руб) | Стоимость (руб) |
| 1 | Флешка 32 Гб | шт | 1 | 390 | 390 |
| 2 | Бумага А 4 | пачка | 3 | 249 | 747 |
| 3 | Картридж для принтера HP 650 (чёрный) | шт | 2 | 920 | 1840 |
| 4 | Ручка | шт | 2 | 78 | 156 |
| 5 | Карандаш  механический | шт | 1 | 39 | 39 |
| Итого материалов | | | | | 3172 |
| Транспортно-заготовительные расходы | | | | | 634 |
| Итого | | | | | 3806 |

В статью «Специальное оборудование» входят затраты, необходимые на приобретение специального оборудования, информационных стендов, измерительных приборов, выполненных по документациям главного конструктора и/или «исполнителя». Для расчёта итоговой суммы по статье специальное оборудование необходимо учесть расходы на монтаж и доставку в размере 10% от их полной стоимости.

Расходы в статье «специальное оборудование» отсутствуют.

Расчет по статье «Основная заработная плата» приведен в таблице 3.3.

Таблица 3.3 – Расчет основной заработанной платы

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| № пп | Наименование этапа | Исполнитель (должность) | Мес. оклад (руб) | Трудоемкость (чел/дни) | Оплата за день (руб) | Оплата за этап (руб) |
| 1 | ТЗ | Руководитель | 45 000 | 5 | 2045 | 10225 |
| Разработчик | 24 000 | 5 | 1091 | 5455 |
| 2 | ТП | Руководитель | 45 000 | 3 | 2045 | 6135 |
| Консультант | 31 000 | 2 | 1409 | 2818 |
| Разработчик | 24 000 | 3 | 1091 | 3273 |
| 3 | Эскизный проект | Руководитель | 45 000 | 3 | 2045 | 6135 |
| Консультант | 31 000 | 2 | 1409 | 2818 |
| Разработчик | 24 000 | 12 | 1091 | 13092 |
| 4 | Технический проект | Руководитель | 45 000 | 4 | 2045 | 8180 |
| Консультант | 31 000 | 2 | 1409 | 2818 |
| Разработчик | 24 000 | 8 | 1091 | 8728 |
| 5 | Рабочий проект | Руководитель | 45 000 | 1 | 2045 | 2045 |
| Консультант | 31 000 | 2 | 1409 | 2818 |
| Разработчик | 24 000 | 32 | 1091 | 34912 |
| Итого | | | | | | 109452 |

Для статьи «Дополнительная заработная плата» рассчитывают все выплаты, предустановленные законодательством за неотработанные по уважительной причине периоды времени; оплата времени, потраченного в результате исполнения общественных и государственных обязанностей; дополнительного или очередного отпуска; учитываются вознаграждения за стаж службы и т.п. (приближенно она составляет около 20% от статьи основная заработная платы). На стадии определения плана расходов появляется «фонд оплаты труда», представляющий из себя основную и дополнительную заработную плату в сумме. Фонд оплаты труда необходим при расчете суммы, вносимой в социальный фонд. Во всех оставшихся ситуациях (командировки, накладные расходы и др.) расчеты ведутся от итоговой суммы по статье основная заработная плата по формуле:

В которой ДЗП – дополнительная заработная плата научного и производственного состава.

Дополнительная заработная плата научного и производственного штата составляет по проекту 21890 руб.

В статье «Страховые отчисления» на отчисления по социальным нуждам отводится 30% от ФОТ (фонда оплаты труда), который будет состоять из основной и дополнительной заработной платы вместе взятых и вычисляться по формуле (3.2-3.3).

В которой: ФОТ – фонд оплаты труда;

* ОЗП – основная заработная плата;
* ДЗП – дополнительная заработная плата;
* СВ – страховые взносы.

В статье «Командировочные расходы» значение расходов определяется или с помощью прямого расчёта, или их можно установить равными от 8% до 10% от статьи основная заработная плата научного и производственного состава.

Расходы по данному разделу отсутствуют.

К статье «Контрагентские услуги» относится стоимость контрагентских работ, осуществляемых сторонними организациями и предприятиями непосредственно для создания модуля служебного оформления, в частности, стоимость изготовления и испытания макетов и опытных образцов, стоимость других работ и услуг опытного производства, испытательных баз, полигонов и т.п.

В ходе разработки данного проекта услуги сторонних организаций не использовались.

К статье «Накладные расходы» относятся расходы на ремонт и содержание зданий, сооружений, оборудования, инвентаря. Эти затраты, сопутствуют всей работе, но не связанны с ней напрямую, и не входят в стоимость материалов и труда.

Данные расходы рассчитываются в размере 200% от заработной платы сотрудника и вычисляются по формуле:

где НР – накладные расходы,

ОЗП – основная заработная плата.

Расчёт по статье «Прочие расходы».

Рассматривается стоимость электроэнергии. На реализацию проекта необходимо 60 рабочих дней, в каждом по 8 часов. В среднем каждый час расходуется 1 кВт энергии, стоимостью 5 рублей за кВт/час. Из этого следует, что прочие расходы составляют 2400 рублей.

Полная себестоимость проекта приведена в таблице 3.4.

Таблица 3.4 – Полная себестоимость проекта

|  |  |  |
| --- | --- | --- |
| № пп | Номенклатура статей расходов | Затраты (руб) |
| 1 | 2 | 3 |
| 1 | Материалы, покупные изделия и полуфабрикаты (исключая отходы) | 3806 |

Продолжение таблицы 3.4

|  |  |  |
| --- | --- | --- |
| 1 | 2 | 3 |
| 2 | Специальное оборудование для научных (экспериментальных) работ | - |
| 3 | Основная заработная плата научного и производственного персонала | 109 452 |
| 4 | Дополнительная заработная плата научного и производственного персонала | 21 890 |
| 5 | Страховые взносы в социальные фонды | 39 402,6 |
| 6 | Расходы на научные и производственные командировки | - |
| 7 | Оплата работ, выполненных сторонними организациями и предприятиями | - |
| 8 | Накладные расходы | 218904 |
| 9 | Прочие прямые расходы | 2400 |
| Итого | | 395854,6 |

Если предполагается, что «продукт» в дальнейшем будет реализован, то необходимо рассчитать договорную цену:

Цена договорная = себестоимость + НДС+ прибыль.

Размер прибыли составит 30% от стоимости разработки.

Прибыль будет равна:

Разработки, ведущиеся для коммерческой организации, облагаются налогом на добавочную стоимость (НДС) в размере 20%:

Из этого следует, что договорная цена будет составлять:

**ЗАКЛЮЧЕНИЕ**

Выполнена работа по разработке машинной модели для задачи анализа тональности текста новостей. Были рассмотрены разные способы классификации и продемонстрированы их отличия. Также применены разные методы оценки точности классификаторов. Для визуальной оценки были составлены матрицы ошибок. Произведена предобработка данных различными методами и выявление ключевых признаков текста. Получен положительный результат работы реализованных моделей: для байесовского классификатора с униграммами – 91% точности; с биграммами – 93%. Результат более эффективной модели в части производительности TF-IDF с логистической регрессией – 72,4%.

Существует много направлений в обработке естественного языка, в частности анализа тональности текста, которые можно исследовать. Полученные модели можно исследовать на разных выборках, которые не обязательно могут содержать данные с новостными текстами. Модели можно масштабировать и дополнять в зависимости от постановки задач обработки текста. Применение и востребованность подобных технологий крайне высока и постепенно применяется во всех областях коммерческой разработки автоматизированных систем.

**СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ И РЕСУРСОВ**

1. **Федоров, Д. Ю.  Программирование на языке высокого уровня Python**: учебное пособие для прикладного бакалавриата / Д. Ю. Федоров. – 2-е изд., перераб. и доп. – М.: Издательство Юрайт, 2019 – 161с.
2. Kaggle [Электронный ресурс]. – URL: <https://www.kaggle.com>.
3. Нейский И. М. Характеристика технологий и процессов интеллектуального анализа данных//Интеллектуальные технологии и системы. Сб. учебно-методических работ и статей аспирантов и студентов. Вып. 7. М.: Изд-во ООО «Эликс+», 2006 – С. 111-122.
4. Сайт национального корпуса русского языка [Электронный ресурс] – URL: <http://ruscorpora.ru/index.html>.
5. Stackoverflow [Электронный ресурс]. – URL: <https://stackoverflow.com>.
6. Документация Keras [Электронный ресурс] – URL: <https://keras.io/>.
7. Статья «Микро-Макро точность, полнота и F-мера» [Электронный ресурс] – URL: <https://medium.com/@ramit.singh.pahwa/micro-macro-precision-recall-and-f-score-44439de1a044>.
8. Python [Электронный ресурс]. – URL: https://www.python.org.

# **ПРИЛОЖЕНИЕ 1**

**Листинг кода**

Установка и подключение необходимых библиотек.

!git clone https://github.com/Samsung-IT-Academy/stepik-dl-nlp.git && pip install -r stepik-dl-nlp/requirements.txt

import sys; sys.path.append('./stepik-dl-nlp')

!pip install deeppavlov

!pip install -U scikit-learn

from sklearn.feature\_extraction.text import TfidfVectorizer

from nltk.corpus import stopwords

from nltk import word\_tokenize, sent\_tokenize

import pandas as pd

import numpy as np

from sklearn.model\_selection import train\_test\_split

from collections import defaultdict

from sklearn.linear\_model import LogisticRegression

from sklearn.preprocessing import LabelEncoder

from sklearn.metrics import accuracy\_score, classification\_report

import json

from dlnlputils.data import tokenize\_text\_simple\_regex, tokenize\_corpus

from \_\_future\_\_ import division

from collections import Counter

import numba as nb

import re, nltk

import os

nltk.download('punkt')

nltk.download('stopwords')

Загрузка и форматирование данных.

from google.colab import drive

drive.mount('/content/drive')

with open('/content/drive/MyDrive/data/train.json', encoding="utf-8") as f:

raw\_data = json.load(f)

texts, sents = [raw['text'] for raw in raw\_data], [raw['sentiment'] for raw in raw\_data]

raw\_df = pd.DataFrame.from\_dict({'text': texts, 'sentiment': sents})

raw\_df.head

Токенизация.

train\_tokenized = tokenize\_corpus(texts)

import nltk

from nltk import FreqDist # Частотный словарь

nltk.download('punkt') # Модуль с пунктуацией

from nltk.tokenize import word\_tokenize # Пространство токенизатора

tokens\_word = word\_tokenize('\n'.join(list(map(lambda i : ' '.join(i),train\_tokenized))))

tokens\_word[:10]

Выделение униграмм и биграмм

unigram = list(nltk.ngrams(tokens\_word, 1)) # Выделение токенов в качестве кортежей

bigram = list(nltk.ngrams(tokens\_word, 2)) # Выделение биграмм в список кортежей

print(unigram[:5]) # Первые 5 токенов

print(bigram[:5]) # Первые 5 биграмм

unigram\_out = FreqDist(unigram)

bigram\_out = FreqDist(bigram)

print('Популярные униграммы: ', FreqDist(unigram).most\_common(5)) # подсчёт популярности кортежей из токенов

print('Популярные биграммы: ', FreqDist(bigram).most\_common(5))

bigram\_out.plot(30,cumulative=False,title ='Распределение относительных частот слов биграмм')

unigram\_out.plot(30,cumulative=False,title ='Распределение относительных частот слов униграмм')

Получение частотных характеристик

bigram\_out.plot(30,cumulative=False,title ='Распределение относительных частот слов биграмм')

unigram\_out.plot(30,cumulative=False,title ='Распределение относительных частот слов униграмм')

Создаем тренировочную и тестовую выборку из преобразованного файла с данными

def make\_training\_test\_sets(data):

data\_shuffled = data.iloc[np.random.permutation(len(data))]

data\_shuffled = data\_shuffled.reset\_index(drop=True)

data\_shuffled.News = data\_shuffled.News.apply(lambda tweet: "".join(tweet))

positive\_data = data\_shuffled[data\_shuffled.sentiment == 'positive']

negative\_data = data\_shuffled[data\_shuffled.sentiment == 'negative']

positive\_data\_cutoff = int(len(positive\_data) \* (3./4.))

negative\_data\_cutoff = int(len(negative\_data) \* (3./4.))

training\_data = pd.concat([positive\_data[:positive\_data\_cutoff], negative\_data[:negative\_data\_cutoff]])

test\_data = pd.concat([positive\_data[positive\_data\_cutoff:], negative\_data[negative\_data\_cutoff:]])

training\_data = training\_data.iloc[np.random.permutation(len(training\_data))].reset\_index(drop=True)

test\_data = test\_data.iloc[np.random.permutation(len(test\_data))].reset\_index(drop=True)

return training\_data, test\_data

training\_data, test\_data = make\_training\_test\_sets(data)

print ("size of training set: " + str(len(training\_data)))

print ("size of test set: " + str(len(test\_data)))

Создаем Байесовский классификатор с методом оценки F1-мерой.

def classify(training\_data, test\_data, ngram=(1, 1)):

scores = []

k\_fold = KFold(n\_splits=10)

count\_vectorizer = CountVectorizer(ngram\_range=ngram)

training\_data.sentiment = training\_data.sentiment.replace('negative',0)

training\_data.sentiment = training\_data.sentiment.replace('positive',1)

confusion = np.array([[0, 0], [0, 0]])

for training\_indices, validation\_indices in k\_fold.split(training\_data):

training\_features = count\_vectorizer.fit\_transform(training\_data.iloc[training\_indices]['News'])

training\_labels = training\_data.iloc[training\_indices]['sentiment'].values

validation\_features = count\_vectorizer.transform(training\_data.iloc[validation\_indices]['News'].values)

validation\_labels = training\_data.iloc[validation\_indices]['sentiment'].values

classifier = MultinomialNB()

classifier.fit(training\_features, training\_labels)

validation\_predictions = classifier.predict(validation\_features)

confusion += confusion\_matrix(validation\_labels, validation\_predictions)

score = f1\_score(validation\_labels, validation\_predictions)

scores.append(score)

return (sum(scores) / len(scores)), confusion

score, confusion = classify(training\_data, test\_data)

print ('Total classified: ', len(training\_data))

print ( 'Score: ',score)

print ('Confusion matrix:')

print(confusion)

Создаем функцию вывода матрицы ошибок.

%matplotlib inline

from matplotlib import pyplot as plt

labels = ['Positive', 'Negative']

def plot\_confusion\_matrix(cm, labels, title='Confusion matrix', cmap=plt.cm.Blues):

plt.imshow(cm, interpolation='nearest', cmap=cmap)

plt.title(title)

plt.colorbar()

tick\_marks = np.arange(len(labels))

plt.xticks(tick\_marks, labels, rotation=45)

plt.yticks(tick\_marks, labels)

plt.tight\_layout()

plt.ylabel('True label')

plt.xlabel('Predicted label')

print ('Confusion matrix without normalization')

plt.figure()

Применяем Байесовский классификатор совместно с униграммами и выводим его матрицу ошибок.

score, confusion = classify(training\_data, test\_data, (1, 1))

print ('Total classified: ' ,len(training\_data))

print ('Score: ' , score)

print ('Confusion matrix:')

print(confusion)

plot\_confusion\_matrix(confusion, labels)

Байесовский классификатор совместно с биграммами.

score, confusion = classify(training\_data, test\_data, (2, 2)) #последний параметр как раз это и применение биграмм

print ('Total classified: ' ,len(training\_data))

print ('Score: ' , score)

print ('Confusion matrix:')

print(confusion)

plot\_confusion\_matrix(confusion, labels)

Предобработка текста. Для начала, удаляем эмодзи с использованием таблицы юникодов.

# избавляемся от эможди

emoji\_pattern = re.compile("["

u"\U0001F600-\U0001F64F" # emoticons

u"\U0001F300-\U0001F5FF" # symbols & pictographs

u"\U0001F680-\U0001F6FF" # transport & map symbols

u"\U0001F1E0-\U0001F1FF"

u"\U00002702-\U000027B0"

u"\U000024C2-\U0001F251"

u"\U0001f926-\U0001f937"

u'\U00010000-\U0010ffff'

u"\u200d"

u"\u2640-\u2642"

u"\u2600-\u2B55"

u"\u23cf"

u"\u23e9"

u"\u231a"

u"\u3030"

u"\ufe0f"

"]+", flags=re.UNICODE)

def remove\_emoji(string):

return emoji\_pattern.sub(r'', string)

Скачиваем частотный корпус русского языка.

if not os.path.isfile('./corpus/1grams-3.txt'):

if not os.path.exists('./corpus'):

os.makedirs('./corpus')

print("Скачивание архива с файлом, содержащим частоты словоформ.")

import urllib.request

import zipfile

url = 'http://www.ruscorpora.ru/ngrams/1grams-3.zip'

urllib.request.urlretrieve(url, './corpus/1grams-3.zip')

with zipfile.ZipFile('./corpus/1grams-3.zip','r') as zip\_ref:

zip\_ref.extractall('./corpus/')

os.remove('./corpus/1grams-3.zip')

print("Скачивание окончено.")

Функции для разделения слипшихся слов.

def get\_corpus\_dict():

dic = dict()

with open('corpus/1grams-3.txt') as f:

for line in f.readlines():

freq, word = line.strip().split("\t")

dic[word] = int(freq)

return dic

corpus\_dict = get\_corpus\_dict()

print("Количество слов в корпусе:", len(corpus\_dict))

def print\_dict(dic, n=5):

i = 0

print("word -- key")

for k, v in dic.items():

if i == n:

break

print(k, "--", v)

i += 1

Как вы можем видеть сверху, у каждого слова есть значение частоты. Однако, нам нужно привести это в вероятностную величину в рамках [0, 1]. Сумма частот оказалась слишком большой, поэтому будем использовать иной метод нормализации частоты для получения вероятности. Так мы будем нормализовать частоты, деля на максимальное значение в словаре, то есть на значение слова "и" = 6829968.

FREQ\_MAX = max(corpus\_dict.values())

FREQ\_MAX

Функция нормализации.

freq\_normalize = lambda x: x / FREQ\_MAX

Нормализируем частотный корпус русского языка

corpus\_dict = {k: freq\_normalize(v) for k, v in corpus\_dict.items()}

print\_dict(corpus\_dict)

Кеширование данных.

def memo(f):

"Memoize function f, whose args must all be hashable."

cache = {}

def fmemo(\*args):

if args not in cache:

cache[args] = f(\*args)

return cache[args]

fmemo.cache = cache

return fmemo

Функционал для сегментации слов на части на основе частотных вероятностей.

def pdist(word):

"Make a probability distribution, given evidence from a Counter."

if word in corpus\_dict:

return corpus\_dict[word]

else:

return 0

def Pwords(words):

"Probability of words, assuming each word is independent of others."

return np.prod([pdist(w) for w in words])

def splits(text, start=0, L=20):

"Return a list of all (first, rest) pairs; start <= len(first) <= L."

return [(text[:i], text[i:])

for i in range(start, min(len(text), L)+1)]

@memo

def segment(text):

"Return a list of words that is the most probable segmentation of text."

if not text:

return []

else:

candidates = ([first] + segment(rest)

for (first, rest) in splits(text, 1))

return max(candidates, key=Pwords)

Тест сегментации слов.

%time print(segment('приветмир'))

Функция сегментации данных новостей.

def segment\_words(words):

new\_words = []

for word in words:

if word in corpus\_dict:

new\_words.append(word)

else:

new\_words.extend(segment(word))

new\_words = list(filter(lambda w: len(w)>2, new\_words))

return new\_words

data\_df['text'] = data\_df['text'].apply(segment\_words)

data\_df.head

Лемматизация и стемминг. Для данной цели, будем использовать токенайзер с лемматизацией русского языка от DeepPavlov.

from deeppavlov.models.tokenizers.ru\_tokenizer import RussianTokenizer

rus\_tokenizer = RussianTokenizer(lemmas=True, stopwords=stopwords.words('russian'))

def final\_tokenizer(string):

words = segment\_words(ru\_token(string))

return rus\_tokenizer([" ".join(words)])[0]

TF-IDF. Будем использовать векторизированные данные для обучения моделей. Также удаляем стоп-слова.

params = {}

params['tokenizer'] = final\_tokenizer

params['stop\_words'] = stopwords.words('russian')

params['ngram\_range'] = (1, 3)

params['min\_df'] = 3

tfidf = TfidfVectorizer(\*\*params)

tfidf.fit([i for i in raw\_df['text']])

Стандартизируем выборки по признакам для TF-IDF.

train = {}

test = {}

tmp = defaultdict(list)

for \_, row in raw\_df.iterrows():

tmp[row['sentiment']].append(row['text'])

for l in tmp:

train[l], test[l] = train\_test\_split(tmp[l], test\_size=0.2, random\_state=2019)

def upsampling\_align(some\_dict, random\_state=2018):

rand = np.random.RandomState(random\_state)

upper = max([len(some\_dict[l]) for l in some\_dict])

print('upper bound: {}'.format(upper))

tmp = {}

for l in some\_dict:

if len(some\_dict[l]) < upper:

repeat\_time = int(upper/len(some\_dict[l]))

remainder = upper % len(some\_dict[l])

\_tmp = some\_dict[l].copy()

rand.shuffle(\_tmp)

tmp[l] = some\_dict[l] \* repeat\_time + \_tmp[:remainder]

rand.shuffle(tmp[l])

else:

tmp[l] = some\_dict[l]

return tmp

train\_data = upsampling\_align(train)

Обучение Logistic Regression классификатора с Softmax функцией для мультиклассификации.

m\_params = {}

m\_params['solver'] = 'lbfgs'

m\_params['multi\_class'] = 'multinomial'

softmax = LogisticRegression(\*\*m\_params)

x\_train = [j for i in sorted(train\_data.keys()) for j in train\_data[i]]

y\_train = [i for i in sorted(train\_data.keys()) for j in train\_data[i]]

x\_test = [j for i in sorted(test.keys()) for j in test[i]]

y\_test = [i for i in sorted(test.keys()) for j in test[i]]

print(x\_train[:1])

Пропускаем данные через TF-IDF векторизатор.

x\_train\_tr = tfidf.transform(x\_train)

x\_test\_tr = tfidf.transform(x\_test)

softmax.fit(x\_train\_tr, y\_train)

Оценка модели с применением TF-IDF и логистической регрессии.

pred = softmax.predict(x\_test\_tr)

accuracy\_score(y\_test, pred)

lab = LabelEncoder()

c\_test = lab.fit\_transform(y\_test)

c\_pred = lab.transform(pred)

print(classification\_report(c\_test, c\_pred, target\_names=lab.classes\_, digits=5))