МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего образования
«ТЮМЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

ИНСТИТУТ МАТЕМАТИКИ И КОМПЬЮТЕРНЫХ НАУК
Кафедра программного обеспечения

РЕКОМЕНДОВАНО К ЗАЩИТЕ
В ГЭК И ПРОВЕРЕНО НА ОБЪЕМ
ЗАИМСТВОВАНИЯ
Заведующий кафедрой
к.т.н., доцент
М. С. Воробьева
24.06.2019 г.

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
(магистерская диссертация)

РАЗРАБОТКА ПРИЛОЖЕНИЯ ДЛЯ ВИЗУАЛИЗАЦИИ ЭМОЦИОНАЛЬНОЙ ОКРАСКИ КОММЕНТАРИЕВ ПОД ЗАПИСЬЮ В СОЦИАЛЬНОЙ СЕТИ

02.04.03. Математическое обеспечение и администрирование информационных систем
Магистерская программа «Разработка, администрирование и защита вычислительных систем»

Студент 2 курса
очной формы обучения — Нагибин Денис Сергеевич

Руководитель работы
д.п.н., профессор
кафедры программного обеспечения — Захарова Ирина Гелиевна

Рецензент
к.филол.н., доцент
кафедры информационных систем — Бидуля Юлия Владимировна

г. Тюмень, 2019

Оглавление

Введение 4

Глава 1. Исследование задачи классификации текстов 6

1.1 Обзор существующих исследований классификации текстов .. 6

1.2 Этапы решения задач машинного обучения на примере классификации 7

1.2.1 Задача классификации 7

1.2.2 Модель и обучающая выборка 8

1.2.3 Определение точности решения 9

1.3 Методы решения задач классификации текстов 11

1.3.1 Методы классификации 11

1.3.2 Сравнение методов классификации 12

1.3.3 Наивный байесовский метод 14

1.4 Особенности решения задач классификации текста 17

1.4.1 Признаки объектов при классификации 17

1.4.2 Модель представления текста Bag-of-words 18

1.4.3 Метрика Tf-Idf 19

1.4.4 Уменьшение потери точности 22

1.4.5 Учет ранее отсутствующих слов 22

1.4.6 Предобработка текста 23

Глава 2. Проектирование и разработка приложения для определения эмоциональной окраски постов 27

2.1 Анализ существующих решений 27

2.2 Существующие инструменты обработки данных 31

2.3 Формирование набора данных 32
2.3.1 Поиск обучающей выборки 32
2.3.2 Предобработка обучающей выборки 34
2.4 Обучение и тюнинг модели классификации 36
2.5 Вычислительный эксперимент 39
2.6 Инструменты для Web-разработки 43
2.7 Web-приложение для визуализации эмоциональной окраски комментариев под записями в социальной сети Вконтакте 44
2.8 Результат работы 46
Заключение 50
Список литературы 51
Приложение 1. Описание набора данных 54
Приложение 2. Листинг кода для очистки данных 55
Приложение 3. Листинг кода для обучения модели 58
Приложение 4. Результаты исследований. Таблица зависимости способа очистки данных и размерности обучающей выборки от точности предсказания модели 61
Приложение 5. Листинг кода для работы с VK API 64
Приложение 6. Листинг кода для роутинга в фреймворке Flask 66
Приложение 7. Листинг кода для десериализации модели и предсказания класса эмоциональной окраски 68
Приложение 8. Результирующий рисунок с эмоциональной окраской 69

Введение

В современном мире очень популярны социальные сети. Из-за того, что люди проводят в них очень много времени, в социальных сетях и других медиа-источниках информации появилось большое количество различных тематических сообществ и групп. Но, как и в любой сфере, где проводят время большое количество людей, в социальных сетях есть возможность зарабатывать деньги и продвигать рекламу в сообществах. Одни сообщества развлекательные, и продают рекламные места для бизнеса; другие являются собственностью компании и в основном только рекламируют и оповещают пользователей о новой продукции этой компании. Практически каждая современная компания, которая производит какой-то продукт или предоставляет услугу, желает знать, что понравилось потребителю, а что можно улучшить. С помощью социальных сетей бизнес может проводить различные рекламные кампании и сразу же получать отзывы пользователей о своей продукции или сервисе.

Отзыв пользователя можно отнести к одному из двух классов: класс с позитивными и негативными отзывами. Отнесение отзыва к одному из этих двух классов несут для бизнеса определенную пользу: если пользователь оставил отзыв, относящийся к позитивному классу (позитивный отзыв), значит, ему нравится товар или услуга. Следовательно, компании стоит продолжать выпускать этот товар. И наоборот, если пользователь оставил негативный отзывы, то стоит пересмотреть стратегию выпуска товара или услуги.

Для того чтобы отнести какие-либо объекты (отзывы) к тому или иному классу (в частности, определить позитивную или негативную эмоциональную окраску), необходимо формализовать их представление. Требуется понять, как именно нужно относить объекты к разным классам, и какие для этого использовать признаки. Дальше, необходимо корректно

сравнить объекты по выбранным признакам наиболее подходящим методом. Методы используют специальные метрики для сравнения. От правильности подобранных признаков, метрик и метода зависит эффективность сравнения.

Целью работы является разработка приложения для определения эмоциональной окраски комментариев под записями (постами) в социальной сети Вконтакте.

Для достижения поставленной цели необходимо решить следующие задачи:

- Изучение методов нормализации и анализа текстовых документов.
- Выгрузка и предобработка данных для обучения модели.
- Построение и тестирование модели.
- Изучение API социальной сети Вконтакте.
- Разработка web-приложения для прогнозирования и визуализации предсказания.

Глава 1. Исследование задачи классификации текстов

1.1 Обзор существующих исследований классификации текстов

Задачей классификации текстов занимались многие авторы. Одно из упоминаний в зарубежной литературе датируется еще 1992 годом автором Lewis D. D. в его работе [1]. В работе автор размышляет о базовых вещах классификации, о том, что слова могут быть представлены в виде векторов и о роли классификации в поисковых системах. Но эта тема актуальна и в наши дни: Wang F., Deng X., Hou L. в своей статье 2018 года [2] описали свой опыт в классификации китайских новостных статей с использованием наивного байесовского алгоритма. А. А. Романов в работе 2017 года [3] классифицирует тексты по их тональности с помощью метода опорных векторов.

Также, классификацией текстов занимаются и другие русские авторы. Например, А. Максаков в своей статье [4] описывает две основные проблемы рубрикации текстов: выбор алгоритма классификации и способы предварительной обработки текстов. А. С. Романов в своем исследовании [5] использует классификацию текстов для идентификации автора текста при ограниченном наборе классов. В. В. Осокин, М. В. Шегай в своей работе [6] рассматривают задачу классификации русскоязычного текста на два класса в зависимости от его эмоциональной окраски: положительный и отрицательный. В другой работе [7], автор G. Forman исследует метрики выбора признаков для классификации тестов. В исследовании 2018 года [8] авторы Mohammad A. H., Alwada’n T., Al-Momani O. классифицируют арабские тексты и сравнивают работу трех алгоритмов для классификации: наивный байесовский, метод опорных векторов и многоуровневый персептрон (нейронная сеть).

1.2 Этапы решения задач машинного обучения на примере классификации

1.2.1 Задача классификации

Классификация – это один из разделов машинного обучения, посвященный решению задачи определения классовой принадлежности. Задача классификации представляет собой задачу разбиения пространства признаков на области, по одной для каждого класса [9]. Разбиение это надо производить так, чтобы не было ошибочных решений. Если этого сделать нельзя, то желательно уменьшить вероятность ошибки, или если ошибки имеют различную цену, то сделать минимальной среднюю цену ошибки. При этом задача классификации превращается в задачу статистической теории принятия решений, широко применяемую в различных областях теории распознавания образов.

Существует другая интерпретация задачи классификации. Имеется некоторое множество объектов и задано некоторое конечное число классов. Задача классификации заключается в получении правила, которое относит каждый из объектов к некоторому классу. В машинном обучении задача классификации относится к разделу обучения с учителем, т.е. это правило строится на основе обучающего набора объектов, где каждому из объектов уже поставлен в соответствие какой либо класс. Существует также другой раздел машинного обучения, называемый «обучение с учителем»: в нем нет заранее размеченных данных, на которые можно опираться при построении алгоритма, но нужно найти внутренние зависимости и закономерности, существующие между исследуемыми объектами. К этому разделу относят, например кластеризацию.

Наиболее общей считается вероятностная постановка задачи. Предполагается, что множество пар «объект, класс» является вероятностным

пространством с неизвестной вероятностной мерой P. Имеется конечная обучающая выборка наблюдений, сгенерированная согласно вероятностной мере P. Требуется построить алгоритм a:X→Y, способный классифицировать произвольный объект x из множества X. Результатом работы классификации может являться как точный ответ, к какому именно классу относится объект, так и степень подобия, которая отображает вероятность, с которой объект относится к классу [10]. Тем не менее, это все еще задача классификации. Таким образом, для классификации нужен алгоритм или набор правил, которые смогут определять, к какому классу относится конкретный объект.

1.2.2 Модель и обучающая выборка

Для построения алгоритма классификации нужен набор объектов, которые заранее разделены по классам. Этот набор объектов называют обучающей выборкой. Наблюдения в обучающей выборке содержат опыт, который алгоритм будет использовать для дальнейшей классификации.

Алгоритм, который сможет классифицировать объекты на основе обучающей выборки, называют моделью. Другими словами, модель классификации – это функция, которая на вход получает объекты, которые необходимо отнести к одному из предопределенных классов, а возвращает класс, к которому предположительно относится этот объект.

Нужно тщательно подойти к выбору обучающей выборки – обучать модель следует на том типе данных, которые впоследствии необходимо будет классифицировать. Например, если необходимо классифицировать комментарии или записи в социальной сети, значит необходимо искать или составлять обучающую выборку на основе аналогичных данных из социальных сетей на том же языке. Кроме того, зачастую обучающая выборка содержит множество ненужной информации или шума, который

будет искажать результаты работы классификатора. Для избавления от этого существуют различные способы предобработки обучающей выборки.

1.2.3 Определение точности решения

После обучения алгоритма, необходимо будет проверить, насколько точно он соотносит объекты к классам, т.е. определить точность решения.

Существует несколько подходов для определения точности решения. Одним из них является проверка точности на обучающей выборке. Метод подразумевает под собой оценку точности модели с помощью имеющихся исходных данных (т.е. обучающей выборки), по которым была построена модель. Предположим, имеется обучающая выборка и построенная на основе этой выборки модель. Далее необходимо подать на вход модели те же данные, по которым она обучалась. Следующим шагом необходимо сравнить результат выданный моделью с реальными данными: чем более схожи классы в реальных данных и результат модели, тем точнее модель. Точность определяется соотношением числа правильно соотнесенных объектов к общему числу объектов участвовавших в эксперименте.

Но существует также и другой способ оценки точности модели – перекрестная проверка или cross-validation. Метод подразумевает под собой разбиение обучающей выборки на N равных частей (см. рис. 1). Далее первая часть используется как проверочная часть, а N-1 частей представляются как обучающая выборка и используются для обучения модели. На следующем шаге первая часть возвращается в обучающую выборку, но из обучающей выборки достается вторая часть и снова представляется как проверочная часть. Таким образом, каждая часть должна быть представлена в виде проверочной части по одному разу и N-1 раз как обучающая выборка. При таком подходе результаты получатся более изолированные от конкретных данных.

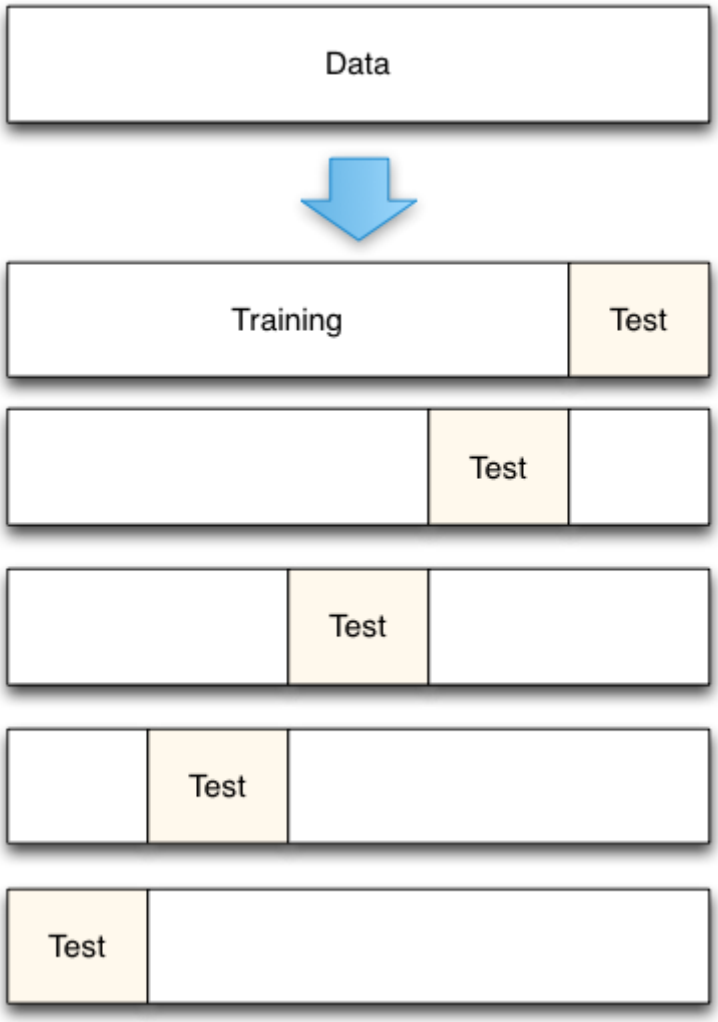

Рисунок 1. Принцип работы кросс-валидации.

Отличие в оценке двух подходов заключается в данных, которые подаются на предсказание. В методе с определением точности на обучающей выборке модель уже «знакома» с этими данными, так как была обучена на них. Следовательно, количество объектов, которое модель предсказала верно, будет выше, чем количество объектов, которое модель предсказала верно, во втором методе. К тому же, модель строится для того, чтобы классифицировать незнакомые ей объекты, следовательно, второй метод покажет более качественную оценку точности. У перекрестной проверки существуют и определенные минусы: при разбиении данных, может получиться так, что тестовые данные будут содержать в себе важную информацию. Так как они не участвовали в обучении, результат работы

модели будет несколько другим, чем, если бы эти данные участвовали в обучении.

1.3 Методы решения задач классификации текстов

1.3.1 Методы классификации

Существует множество методов классификации. У каждого из них существуют свои преимущества и недостатки, а так же своя область применения. Рассмотреть все не представляется возможным, поэтому были рассмотрены только самые основные и наиболее популярные.

Метод логистической регрессии применяется чаще всего для решения задач бинарной классификации, но допускается и многоклассовая классификация (так называемый one-vs-all метод). Достоинством этого алгоритма является то, что на выходе для каждого объекта имеется вероятность принадлежности классу [11]. Практичная реализация должна предусматривать стандартизацию данных, отсев выбросов, регуляризацию (сокращение весов), отбор признаков, и другие эвристики для улучшения сходимости.

Метод K-ближайших соседей (kNN – k-Nearest Neighbors) часто используется как составная часть более сложного метода классификации. Например, его оценку можно использовать как признак для объекта. А иногда, простой kNN на хорошо подобранных признаках дает отличное качество. При грамотной настройке параметров (в основном — метрики) алгоритм дает зачастую хорошее качество в задачах регрессии. К недостаткам относят неэффективный расход памяти, и чрезмерное усложнение решающего правила т.к. необходимо хранить обучающую выборку целиком. Кроме того, поиск ближайшего соседа предполагает

сравнение классифицируемого объекта со всеми объектами выборки, а значит, время на это действие растет линейно [12].

К достоинствам метода опорных векторов причисляют более уверенную классификацию и более существенные методы классификации. К недостаткам – неустойчивость к шуму исходных данных.

Деревья решений (Classification and Regression Trees – CART) часто используются в задачах, в которых объекты имеют категориальные признаки, и используются для задач регрессии и классификации. Также, деревья хорошо подходят для задач многоклассовой классификации. К недостаткам можно отнести нестабильность процесса и сложность контроля размера дерева [13].

Наивный байесовский метод является одним из самых известных и применяемых алгоритмов машинного обучения с учителем, основной задачей которого является восстановление плотностей распределения данных обучающей выборки. Зачастую этот метод дает хорошее качество в задачах именно многоклассовой классификации. Несмотря на его «наивность» и простоту он показывает достаточно точные результаты.

1.3.2 Сравнение методов классификации

Авторы статьи [14] провели ряд сравнений результатов работы разных моделей. Авторы загрузили уже подготовленные модели с сайта Стэнфордского университета [15]. Модели для этого сайта были подготовлены такими лицами как «chief scientist at Salesforce» Richard Socher и профессор стендфордской лаборатории машинного обучения Christopher Manning.

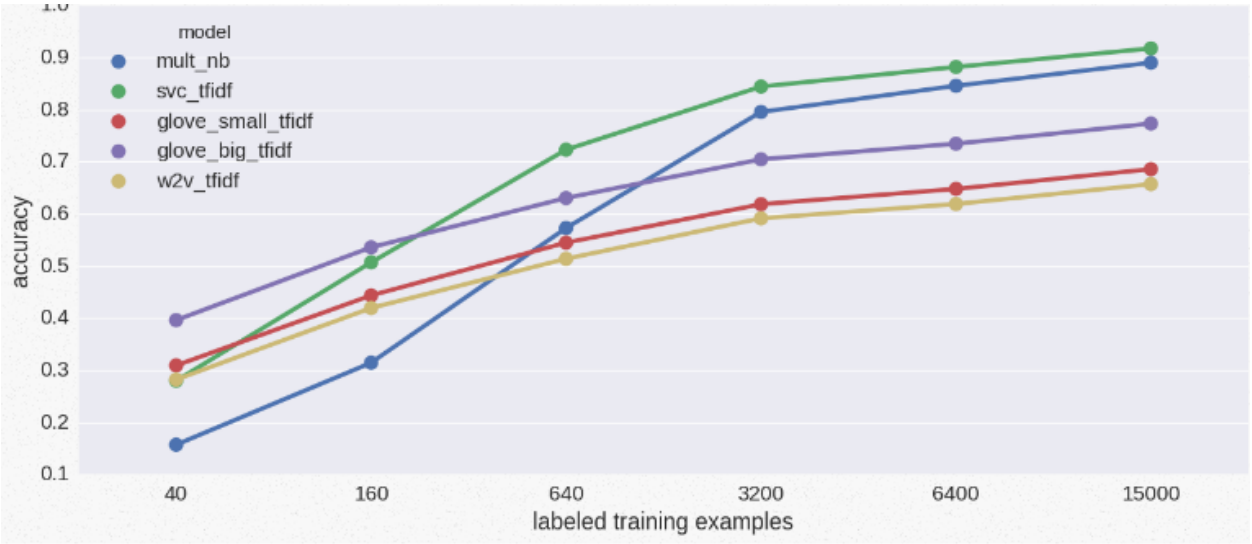

Рисунок 2. Результирующий график точности работы моделей [14].

На шкале X (см. рис. 2) представлена точность модели на обучающей выборке, на Y — количество текстов для обучения. Обозначение mult_nb для мультиномиального байесовского классификатора, svc_tfidf – для метода опорных векторов с Tf-Idf векторизацией. Как видно из графика, наилучший результат получили наивная байесовская модель и модель метода опорных векторов. Причем видно, что в байесовской модели наблюдается достаточно резкий кратковременный рост, по сравнению с методом опорных векторов на 640 текстах. Наилучший же результат, который был получен на 15000 текстах, примерно совпадает по точности у обеих моделей. У обеих моделей рост не прекращается, но можно заметить, что график байесовской модели начинает свой рост со значительным отставанием. Однако, он начинает приближаться к модели метода опорных векторов на дистанции от 3200 до 15000 текстов. Вполне вероятно, что на большем размере обучающей выборки модель наивного байеса вырвется вперед по точности. Именно он и будет в дальнейшем рассматриваться в работе из-за его простоты реализации и хороших результатов на практике.

1.3.3 Наивный байесовский метод

Наивный байесовский алгоритм является одним из самых простых в реализации из всех используемых алгоритмов машинного обучения [16]. Наивным он называется потому, что признаки, используемые для классификации, считают не оказывающими влияния друг на друга. В реальных данных так бывает редко или вовсе не бывает, но, тем не менее, на практике верность данного предположения достаточно высока, даже если предположение о независимости признаков не верно. Обозначения для дальнейшего использования представлены в таблице 1.

Для дальнейшего использования вводятся следующие обозначения (см. табл.1).

Таблица 1. Обозначения для описания наивного байесовского метода.

Переменная	Название
C	Класс текста (положительный или отрицательный).
F_k	В тексте хотя бы один раз встречается слово k.

Классический вариант формулы Байеса имеет вид, представленный в формуле 1:

$$P(A) * P(B|A) = P(B) * P(A|B) \quad (1)$$

После адаптации этой формулы к условиям задачи классификации получится формула 2 (формулы представлены для двух признаков F_1 и F_2).

$$P(\mathrm{C}|F_1, F_2) = \frac{P(C) * P(F_1, F_2|C)}{P(F_1, F_2)} \quad (2)$$

Вероятности в этой формуле описываются в таблице 2.

Таблица 2. Описание вероятностей для формулы 2.

Переменная	Значение
$P(\mathrm{C}\|F_1, F_2)$	Вероятность принадлежности к классу C при известных признаках F. Искомое значение.
$P(C)$	Вероятность класса без каких либо знаний о данных. Иными словами отношение количества примеров какого-то класса ко всем имеющимся примерам обоих классов.
$P(F_1, F_2)$	Вероятность наличия сразу обоих признаков F.
$P(F_1, F_2\|C)$	Вероятность увидеть признаки F если известно, что образец принадлежит классу C.

Рассчитать вероятность увидеть признаки F, при известном классе C достаточно трудоемкая задача и здесь кроется вся суть наивного байесовского метода. Если **наивно** предположить, что все признаки F независимы, т.е. не оказывают друг на друга никакого влияния, тогда вероятность для двух признаков $P(F_1, F_2|C)$ сводится к формуле 3.

$$P(F_1, F_2|C) = P(F_1|C) * P(F_2|C) \quad (3)$$

Подставляя формулу 3 в исходное выражение, получается формула для расчетов 4.

$$P(\mathrm{C}|F_1, F_2) = \frac{P(C) * P(F_1|C) * P(F_2|C)}{P(F_1, F_2)} \quad (4)$$

С теоретической точки зрения нельзя наивно предполагать, что признаки F являются независимыми, но на практике такой подход широко используется и показывает достаточно точные результаты.

Таким образом, для того, чтобы определить к какой из групп по эмоциональной окраске относится тот или иной текст необходимо вычислить две вероятности и сравнить их (см. формулу 5, 6).

$$P(\mathrm{C} = "pos"|F_1, F_2) = \frac{P(C = "pos") * P(F_1|C = "pos") * P(F_2|C = "pos")}{P(F_1, F_2)} \quad (5)$$

$$P(\mathrm{C} = "neg"|F_1, F_2) = \frac{P(C = "neg") * P(F_1|C = "neg") * P(F_2|C = "neg")}{P(F_1, F_2)} \quad (6)$$

Поскольку знаменатели у обеих формул будут одинаковые то можно сократить их. Но в этом случае, в результате, получившемся после вычисления формул, получится уже не действительная вероятность. В действительности, это и не важно, т.к. для того, чтобы определить к какому классу относится текст достаточно сказать какая вероятность больше по отношению к другой, и если сократить обе на одинаковое число, результат сравнения не изменится. Теперь, формулу в общем виде для двух признаков можно привести к общему виду, представленному в формуле 7.

$$\mathrm{C}_{best} = \underset{c \in C}{argmax}\ P(C = c) * P(F_1|C = c) * P(F_2|C = c) \quad (7)$$

1.4 Особенности решения задач классификации текста

1.4.1 Признаки объектов при классификации

Признаками в случае решения задачи классификации текстов называют объекты-маркеры (напр. слова, символы), которые с некоторой вероятностью могут сказать, что текст, в котором находятся эти слова, относится к тому или иному классу. Правильно подобранные признаки залог высокой точности при классификации.

Одним из признаков для данных можно выбрать наиболее часто встречающиеся слова в обучающей выборке. Действительно, если взять обучающую выборку, выделить все слова в массив, предварительно выполнив над ними операции очистки и взять топ-50 самых часто встречающихся слов, то можно сказать, что текст, в котором будут несколько из этих слов с большой вероятностью можно правильно классифицировать. Именно этот подход и использует наивный байесовский классификатор: он берет слова или пары (тройки) слов и определяет количество их повторений в текстах с определенным классом. На основе этих данных и строится модель. В целом для байесовского алгоритма не нужно специально выделять веса для слов и в этом его превосходство: все происходит автоматически. Действительно, если взять из результатов работы алгоритма выборку текстов, которые были отнесены к тому или иному классу с большой долей вероятности, можно предположить, что эти тексты можно использовать к выборке для обучения модели. Таким образом, выборка будет копиться. Но с другой стороны, если текст был классифицирован не правильно, но с большим процентом вероятности, то классификатор все больше и больше будет наращивать неправильную выборку.

1.4.2 Модель представления текста Bag-of-words

Результат работы классификатора напрямую зависит от того, как будет представлен документ для классификатора. Наиболее распространенный способ – в виде набора слов (bag-of-words) или в виде набора N-gramm.

Существует классическая модель для реализации машинного обучения под название «мешок слов» или Bag-of-Words. Формальная постановка задачи для модели:

Пусть $f_1, \dots, f_m$ - множество, состоящее из m признаков (атрибутов), которые могут присутствовать в документе; пусть $n_i\ (d)$ – это количество вхождений признака n_i в документ d. Далее каждый документ d представляется в виде вектора следующим образом: $d = (n_1\ (d), n_2\ (d), \dots, n_m\ (d))$.

Выделяют два основных типа признаков:

- Частотные признаки – каждое значение в векторе d соответствует количеству вхождений атрибутов в документ; тогда $n_i(d) \in (0; +\infty)$
- Бинарные признаки – каждое значение в векторе отражает факт присутствия в документе; тогда $n_i(d) = \{0,1\}$

Далее документы, представленные в виде вектора своих признаков (атрибутов), используются для обучения классификатора. Существует подход с n-граммами – во время обучения модели можно указать, какое количество слов необходимо принимать за единицу измерения: одно – униграмма; одно и два – биграмма; одно, два или три – триграмма (см. рис. 3).

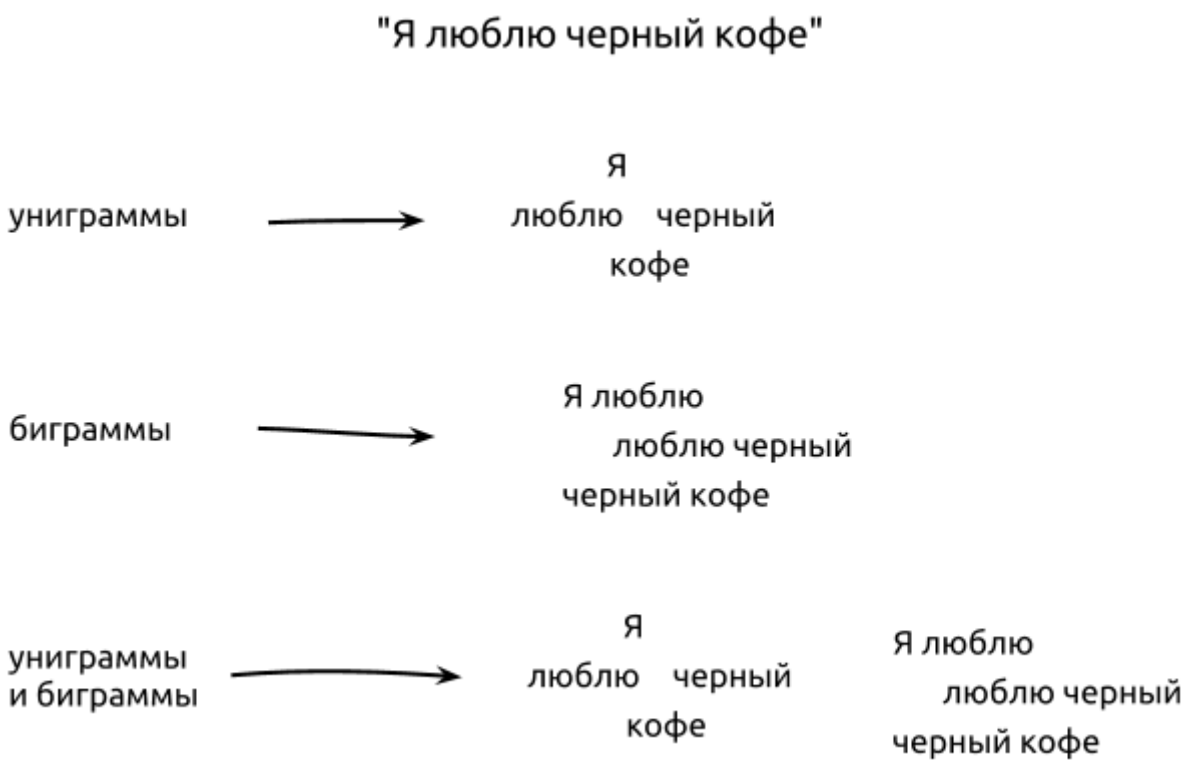

Рисунок 3. Пример разбиение на N-граммы.

Большее количество слов не используется ввиду отсутствия положительного эффекта для точности предсказания модели. Для би- и триграмм классификатор будет считать за повторение только пару слов или тройку. В зависимости от того какой тип грамм будет использоваться, будут розниться результаты. К недостаткам bag-of-words относят отсутствие семантической связи между словами при подсчете [17]. Два текста могут быть похожи только лишь потому, что имеют одинаковые слова в составе.

1.4.3 Метрика Tf-Idf

Как уже было сказано выше, наивный байесовский классификатор не нуждается в специальной маркировке признаков. Общая суть заключается в том, что он автоматически назначает больший вес словам, которые встречаются чаще, чем другие. Но что если предположить что присваивать

большой вес некоторым словам не имеет смысла. Например, некоторые слова могут иметь значение для конкретного текста, но не иметь особого смысла, если рассматривать весь корпус текстов в целом и наоборот. Также можно взять как пример союзы, предлоги (союзы и предлоги взяты как пример, в данной работе все союзы, предлоги и аналогичные стоп-слова будут убраны из выборки) и часто встречающиеся слова: чем чаще они встречаются в разных документах, тем ниже у них должен быть вес.

TF-IDF(TF – term frequency, IDF – inverse document frequency) – статическая мера для оценки важности слова в контексте текста, являющегося частью коллекции текстов или корпуса[11]. Иными словами, это способ оценить важность термина в рамках какого-либо текста относительно всех остальных текстов.

Используется эта метрика для того, чтобы дать больший вес для слов, которые имеют не нейтральную тональность, т.к. именно такие слова определяют тональность всего текста.

TF – это частотность термина в одном тексте. Так как в длинных текстах термин может встречаться большее количество раз, то количество появлений этого слова в тексте делят на общее количество всех слов в конкретном тексте (см. формулу 8). В формуле 8 n_t это число вхождений слова t в документ, а в знаменателе – общее число слов в данном документе.

$$tf(t,d) = \frac{n_t}{\sum_k n_k} \tag{8}$$

IDF – обратная частотность документов. Она обозначает саму важность термина. В TF мы считали общее количество терминов в тексте, и если попадется предлог, встречаться он будет очень часто. Для упразднения этой неточности и считается IDF для конкретного термина – логарифм от общего количества документов деленного на количество документов, в которых

встречается термин (см. формулу 9). В формуле 9 $|D|$ это число документов в коллекции, а $|\{d_i \in D \,|t \in d_i\}|$ это число документов из коллекции D, в которых встречаются t при $n_t \neq 0$.

$$idf(t, D) = \log \frac{|D|}{|\{d_i \in D \,|t \in d_i\}|} \quad (9)$$

В конце, необходимо умножить TF на IDF и получится словарь со значениями, в котором ключом будет выступать сам термин, а значением, на которое указывает ключ – его частота TF-IDF[12]. Больший вес получат слова с высокой частотой в пределах одного конкретного текста, а слова, которые встречаются часто во многих документах, получат вес меньше: например, союзы и предлоги.

Предположим, что необходимо вычислить TF-IDF метрику для следующего набора текстов:

1. «Я люблю пение»;
2. «Я ненавижу пение и танцы»;
3. «Танцы – мое хобби».

В результате вычисления TF-IDF получается следующее представление (см. табл. 3).

Таблица 3. Пример работы TF-IDF векторизации

№	Я	Люблю	Пение	Ненавижу	И	Танцы	Мое	Хобби
1	0,18	0,48	0,18					
2	0,18		0,18	0,48	0,48	0,18		
3						0,18	0,48	0,48

Как видно в таблице 3 больший вес имеют слова, которые встречаются в других текстах реже, чем остальные. С помощью этой метрики можно уйти от bag-of-words и попытаться улучшить модель.

1.4.4 Уменьшение потери точности

При работе с большим количеством исходных данных очень часто бывает, что работа с вероятностями вызывает затруднение, так как точности простых типов данных в языках программирования зачастую не хватает. Для того, чтобы избежать арифметическое переполнение снизу пользуются свойством логарифма произведения.

Так как логарифм функция монотонная, ее применение к обеим частям выражения изменит только его численное значение, но не параметры при которых достигается максимум. При этом логарифм числа близкого к нулю будет числом отрицательным.

Так как вероятность лежит в интервале от 0 до 1, то ее логарифм будет находиться в интервале от –∞ до 0. Чем больше будет число, тем точнее определен класс, только сами числа теперь будут отрицательны [18]. Если же применить это к конечной формуле для N признаков получится формула 10.

$$\mathrm{C}_{best} = \underset{c \in C}{argmax} \left(\log P(C = c) + \sum_{k} \log P(F_k | C = c) \right) \tag{10}$$

1.4.5 Учет ранее отсутствующих слов

В любых исходных данных может случиться так, что какое-то слово там не будет встречаться вообще. И отсутствие этого слова очень сильно скажется на расчетах. В конечном счете, может случиться так, что документ будет иметь нулевые вероятности в каждом классе, если ни одно из его слов

не было встречено в каком-либо классе. Это произойдет потому, алгоритм ни разу не учитывал эти слова и, следовательно, посчитает, что текст относится к каждому классу с вероятностью 0.

Если предположить, что для устранения этой проблемы достаточно лишь увеличивать выборку пока в ней не будут присутствовать все возможные слова, то можно прийти к выводу, что это невозможно. Действительно, нельзя создать выборку, которая бы содержала все возможные слова, включая неологизмы, опечатки, синонимы и т.д.

Типичным решением этой проблемы можно считать сглаживание с прибавлением единицы [19]. Другое название аддитивное сглаживание или сглаживание Лапласа. Заключается оно в том, что к количеству появления какого либо слова всегда добавляется единица. Этот простой прием говорит о том, что даже если какого либо слова нет во всей обучающей выборке, все равно данное слово существует. Таким образом, даже если какое либо слово не встретилось ни разу, т.е. количество его появлений равно 0, то будет добавлена единица и количество появлений этого слова станет минимально возможное значение для любой выборки – единица. Большинство современных классификаторов уже по умолчанию используют сглаживание Лапласа, но имеется возможность указать и другое.

1.4.6 Предобработка текста

Русский язык имеет большое количество различных приставок, окончаний и форм слов. Это позволяет языку быть гибким в построении предложений, но это неудобно для текущей задачи. Перед созданием модели необходимо провести подготовительную работу с текстом, для того, чтобы алгоритмы векторизации соотносили одинаковые слова как один объект. Например, в процессе векторизации слова «красивый» и «красивая» будут рассмотрены как разные слова, потому что они состоят из разных букв. Но

смысл этих слов один и тот же, и рассматриваться для текущей задачи они должны как одно слово. В этом смысле английский язык имеет преимущество. В нем не так много окончаний и слово «nice» имеет множество значений, большая часть, которых относится к одной области применения: красивый, приятный, опрятный.

Существует ряд методик, которые используются для приведения слов к более обобщенному виду [20]. В первую очередь, необходимо провести декапитализацию — преобразование всех символов корпуса к нижнему регистру. Далее, деакциентенизация — прием используемые при обработке текстов, в которых существуют буквы с акцентами. В русском языке такой буквой является «Ё». Также, необходимо удалить часто встречающиеся слова. Иначе их называют стоп-слова. К ним относятся, например союзы, артикли и другие слова, которые слабо влияют на эмоциональную окраску текстов.

Одним из решения проблемы разных окончаний у слов может являться стемминг — процесс выделения первоначальной формы слова или его основы. Метод решения задачи поиска основы слов называется алгоритм стемминга, а конкретная реализация — стеммер. Стемминг(англ. Stemming) – поиск основы слова, учитывающий морфологию исходного слова [21]. Если в тексте встречаются несколько слов с одинаковым корнем, но разными окончаниями, то без стемминга они будут распознаны по-разному. Процесс стемминга должен привести все слова к некоторой основной форме. Конечно, стоит учитывать и особенности языка: например в русском языке очень много различных окончаний, которые изменяются от самых разных условий, и даже простое существительное может иметь множество окончаний; а в английском если у слова другое окончание, значит это уже другая часть речи. Основа слова не обязательно совпадает с морфологическим корнем слова. Пример слов после процесса стемминга представлен в таблице 4.

Таблица 4. Результат стемминга.

№	Оригинальное слово	После стемминга
1	практике	практик
2	возникают	возника
3	используются	использ
4	множественном	множествен

Другим возможным решением выступает лемматизация (англ. lemmatization) — процесс, при помощи которого слово преобразуется в его первоначальную не изменённую форму, иначе говоря, лемму. В русском языке словарной формой считается:

- существительные – именительный падеж единственного числа;
- глаголы – инфинитивная форма;
- прилагательные – единственное число, именительный падеж, мужской род.

Пример обработки слов с помощью лемматизации представлен в таблице 5.

Таблица 5. Результат лематизации.

№	Оригинальное слово	После лемматизации
1	практике	практика
2	возникают	возникать

3	используются	использоваться
4	множественном	множественный

Но лемматизация работает только с одним словом, вырванным из контекста, и не анализирует остальное предложение. Например, слово «вой» можно привести к двум леммам: существительное «вой» и глагол «выть». В связи с этим лемматизация может допускать погрешности.

Глава 2. Проектирование и разработка приложения для определения эмоциональной окраски постов

2.1 Анализ существующих решений

Крупные социальные сети уже имеют некоторый функционал для анализа активности пользователей. Эти инструменты активно используются людьми, которые публикуют контент на этих площадках. Для примера рассмотрим функционал инструмента для сообществ «Статистика» в социальной сети Вконтакте.

С помощью статистики можно просмотреть различную информацию о пользовательской активности в сообществе. Например, можно узнать какое количество уникальных посетителей просмотрело страницу сообщества. Диаграммы покажут информацию о посетителях в разрезе возрастного (см. рис. 4) или географического признака (см. рис. 5).

город	количество
Тюмень	79.20%
Тобольск	5.95%
Ишим	1.90%
Заводоуковск	0.75%
Москва	0.73%
Другие	11.46%

Рисунок 4. Посетители в процентах в разрезе регионов.

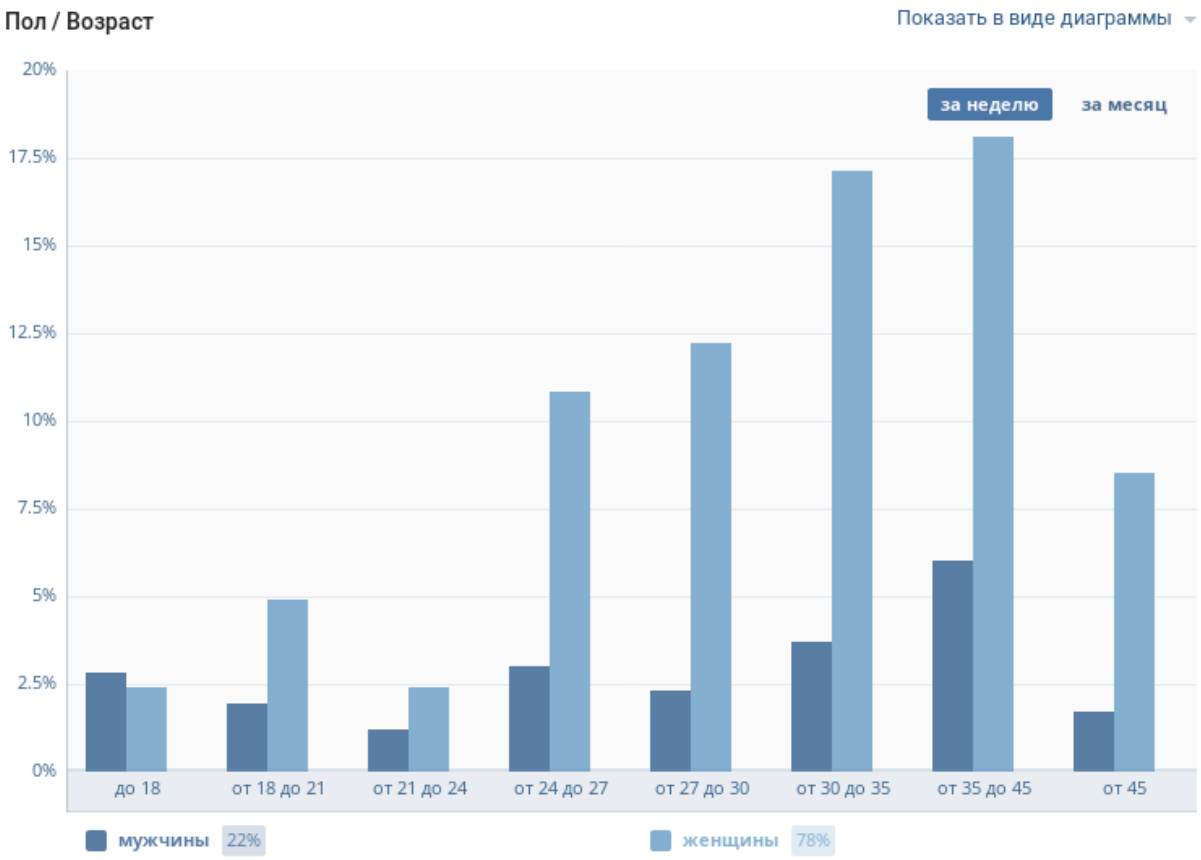

Рисунок 5. Посетители сообщества в разрезе полового и возрастного признака.

Есть возможность посмотреть графики, которые отображают активность пользователей (см. рис. 6). Активность отслеживается по добавлениям комментариев, нажатий кнопки «Мне нравится» и по нажатию кнопки «Рассказать друзьям».

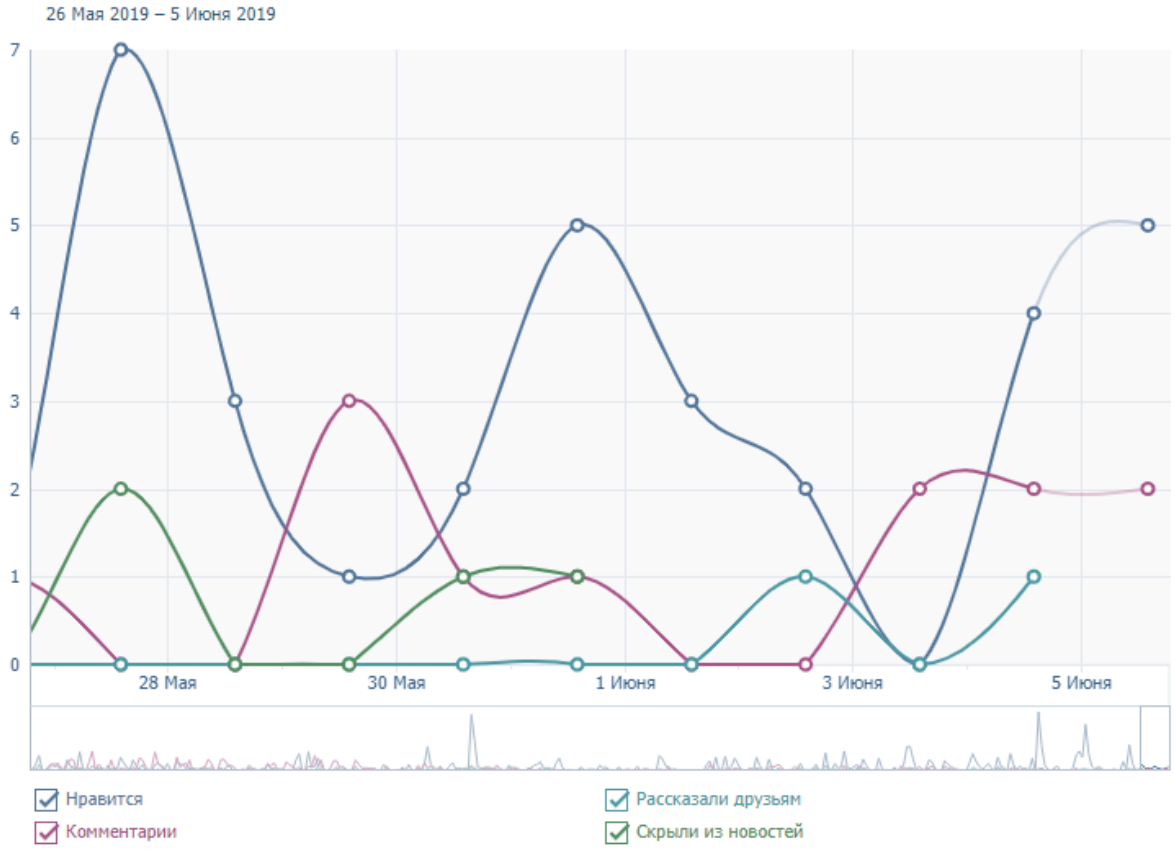

Рисунок 6. График активности пользователей.

Однако, социальная сеть Вконтакте не единственная площадка, на которой есть встроенные инструменты для анализа действий пользователей. Например, Youtube.com — сервис для просмотра и публикации видео — позволяет использовать другой инструмент для аналитики. Этот инструмент больше заточен специально под контент youtube.com: есть различные счетчики количества просмотров видео и активности пользователей. Также, этот инструмент позволяет напрямую отслеживать прибыль от просмотров видео. Ни один из этих инструментов не предоставляет возможность оценить эмоциональную окраску пользовательских комментариев.

Для решения задачи определения эмоциональной окраски можно выгружать данные о комментариях из площадки и анализировать эти данные с помощью уже существующих инструментов анализа данных.

В настоящий момент, большое распространение получили открытые API, которые позволяют пользователям пользоваться определенными инструментами машинного обучения за некоторую плату. Некоторые из них представлены ниже [22]:

- MonkeyLearn
- Google Cloud NLP
- IBM Watson
- Amazon Comprehend
- Lexalytics
- Aylien
- MeaningCloud
- Rosette
- Microsoft Cognitive Services

Преимущество данных решений в том, что они предоставляют большую функциональность в одном сервисе. Т.е. используя Google Cloud NLP возможно не только классифицировать тексты по эмоциональной окраске, но и производить другие действия с данными. Большинство из них представляют API, которое становится доступно после регистрации. Это API можно использовать в других системах. Условно говоря, эти решения предлагают программный каркас, который можно использовать в других системах. В то же время большая функциональность является и недостатком. Прежде чем начать пользователю нужно зарегистрироваться и оплатить доступ или же получить ограниченный функционал бесплатно. Далее нужно прочитать большое количество документации, часть которой представлена только на английском языке; получить персональный токен для работы с API и научиться пользоваться этим API. Данные сервисы достаточно удобны, если пользователь является разработчиком и может разобраться во всем этом. Но если пользователь является менеджером, в обязанности, которого

входит отслеживание комментариев и отзывов пользователей о продукции компании, то все это достаточно нетривиальная работа.

Таким образом, можно прийти к выводу, что излишняя функциональность и чрезмерная сложность, возможно, удобны для интеграции со своей системой, но негативно сказывается на конечном потребителе: люди, не готовые глубоко вникать в теорию программирования или машинного обучения, не смогут быстро начать пользоваться рассмотренными выше сервисами.

2.2 Существующие инструменты обработки данных

В настоящее время применяются самые различные инструменты и библиотеки для обработки данных, а так же языки программирования: R, Java, C#, JavaScript и Swift. У каждого инструмента есть свои плюсы и минусы, но самым перспективным языком в этом направлении получившим общественное признание является язык Python.

На языке Python очень легко писать разного рода программы, а огромное количество уже написанных библиотек на любой случай позволят свести к минимуму написание кода.

Кроме того, существует множество библиотек не только предназначенных специально для работы с данными, но и просто ускоряющих выполнение кода (напр. NumPy):

- Фундаментальные библиотеки, которые применяются не только для Data Science: Numpy, Pandas;
- Библиотеки для машинного обучения: Scikit-learn(sklearn), Scipy, nltk, Shogun, PyBrain;
- Библиотеки для визуализации данных, которые позволяют наглядно представить данные: matplotlib, plotly, seaborn, d3py;

- Предобработка данных: стемминг(выделение основы слов): Stemmer, snowballstemmer, лемматизация(приведение слова к первоначальному виду): pymorphy2.

В работе использовался язык Python и библиотеки pandas, numpy, pymorphy2, nltk, sklearn. Однако не всегда использование библиотек оправдывает себя. Иногда, когда задача весьма специфична, нужно понимать, как работает тот или иной подход на низком уровне, чтобы иметь возможность реализовать его встроенными средствами языка с учетом специфики конкретной задачи. Это позволит добиться более точных результатов. Средой разработки была выбрана IDE от JetBrains под названием PyCharm Professional. Официальная версия IDE была получена после отправки электронного письма с отсканированным студенческим билетом в службу поддержки JetBrains.

2.3 Формирование набора данных

2.3.1 Поиск обучающей выборки

В качестве исходных данных для обучающей выборки можно брать отзывы в различных сервисах. Например, можно использовать отзывы на различные товары в yandex.market или amazon, на сайте магазинов бытовой техники или воспользоваться рецензиями на фильмы с различных сайтов о фильмах. Эти данные находятся в открытом доступе и уже имеют некоторое разграничение на хорошие и плохие отзывы.

За исходные данные можно взять посты из социальных сетей. В социальной сети twitter.com все стены с сообщениями являются открытыми, и любой может зайти и прочитать их. Но здесь уже нельзя точно сказать в каком эмоциональном окрасе находится это сообщение. Можно пойти на хитрость и выбрать стену, на которой ожидается увидеть сообщения строго

негативного или позитивного характера. Например, для негативного окраса это может быть криминальная сводка из новостей или стены различных блогеров. Для позитивного окраса можно рассмотреть страницы молодых мам или политиков.

В данной работе будет использоваться уже размеченный и готовый для разбора набор данных с сайта study.mokoron.com (см. Приложение 1). Этот набор был подготовлен Юлией Рубцовой в рамках диссертационного исследования [23]. На сайте, где данные лежат в свободном доступе, можно скачать 2 набора данных для определения эмоциональной окраски текста: набор, состоящий из позитивно окрашенных текстов из 114 000 записей (25мб) и набор с негативно окрашенными текстами – 111 000 записей (23мб). Все записи взяты из социальной сети twitter.com и содержат следующие значения:

- ID автора сообщения,
- ID сообщения,
- дата публикации,
- само сообщение (не более 140 символов на русском языке),
- количество ретвитов сообщения,
- количество людей добавивших сообщение в избранное,
- и т.д.

При разметке корпуса использовался подход Jonathon Read [24]. Суть подхода заключается в разбиении данных на классы с положительной и отрицательной эмоциональной окраской с использованием признака наличия смайликов в тексте. У этого подхода есть существенный недостаток: тексты с иронией очень сильно искажают реальный результат. Но так, как в работе участвует очень большой корпус данных, это нивелирует некоторые неточности при обучении модели.

2.3.2 Предобработка обучающей выборки

Первоначально необходимо подготовить данные для обучения, чтобы во время обучения классификатор не обращал внимания на зашумления. Под подготовкой данных подразумевается некоторая «очистка» данных, например стемминг или лемматизация.

Первоначально необходимо скачать с сайта study.mokoron.com два файла: positive.csv и negative.csv. В них лежат твиты с позитивной и негативной окраской соответственно. Так как данные в формате csv, то они уже разделены на компоненты с помощью разделителя «;». Как уже было оговорено, в наборах данных представлено множество данных о текстах, но в основном интересен только столбцы содержащие текст сообщения и его окраску. Они будут именоваться label и text (см. Приложение 2). В text будет лежать сам текст твита, в label – его эмоциональный окрас: 1 – положительный, 0 – отрицательный.

Затем с помощью библиотеки Pandas на языке Python необходимо считать эти данные в DataFrame - объект библиотеки Pandas, который позволяет удобно манипулировать данными [25]. Для этого использовалась функция read_csv() (см. приложение 2). Первым аргументов идет путь до файла. Кроме того, можно указать, как назвать столбцы считанных данных, какой использовать разделитель для данных. Также, необходимо изменить label для текстов с негативной окраской с -1 на 0. Это необходимо для дальнейшей корректной работы классификатора, т.к. вероятность находится в диапазоне от ноля до единицы. Далее, необходимо привести текст к нижнему регистру и удалить из текстов все лишние символы включая знаки препинания, оставив только буквы русского алфавита и пробелы. Может случиться так, что один за другим будет идти несколько пробелов. Для этого случая был имплементирован функционал, который оставляет ровно один пробел между словами, даже если пробелов было несколько.

Были проведены эксперименты со стеммингом, лемматизацией и стоп-словами. Для стемминга использовалась сторонняя библиотека для русского языка nltk.stem. На вход функция stem() объекта nltk.stem.SnowballStemmer('russian') получает текст в виде строки. Возвращает функция ту же строку, но слова в ней уже изменены после стемминга.

Стоп-слова были взяты из модуля corpus библиотеки nltk. Простым перебором происходит проверка каждого слова из текста. Если слово принадлежит корпусу стоп-слов, то оно выкидывается из текста.

Для лемматизации использовалась библиотека pymorphy2 и ее класс MorphAnalyzer. На вход в функцию parse подается не все предложение целиком, а лишь только слово. Для разбиения слов использовалась нативная функция split().

В конечном итоге, получались файлы с данными для обучения модели. В файле записывались уже очищенные сообщения и данные о них: индекс сообщения, текст сообщения и его оценка: 1 - для положительно окраски, 0 - для отрицательной. Всего получилось несколько сборок файлов для исследования. Разные размеры данных с разными примененными функциями для их очистки: например использовались функции стемминга и очистки от стоп-слов, но без лемматизации, или только лемматизации, но без приведения к нижнему регистру и лемматизации. Разные вариации очистки данных нужны, чтобы получить данные, которые покажут наилучший результат при обучении модели.

В папке original_data лежат исходные корпусы текстов с текстами в позитивной окраске и негативной. В папке data лежат уже подготовленные данные. Разные названия папок соответствуют разными способам очистки, которые были использованы (см. табл. 6).

Таблица 6. Соответствие названия папки и способу очистки.

Название папки	Способ очистки
lem__stop_stem	Лемматизация, но без стемминга. Стоп-слова оставлены.
lem_stop__stem	Лемматизация, без стемминга. Стоп-слова удалены
stem__stop_lem	Стемминг, без лемматизации. Стоп-слова оставлены.
stem_stop__lem	Стемминг, без лемматизации. Стоп-слова удалены.
lem_stem__stop	Стемминг и лемматизация. Стоп-слова оставлены.

В каждой папке лежат 5 файлов. В файлах, начинающихся с clean_common, лежат очищенные тексты, взятые с начала корпуса. В конце число указывает размер данных, который лежит в папке в тысячах текстов. Т.е. файл clean_common50.csv содержит в себе очищенные данные с начала корпуса исходных данных до 50 тыс. В файлах, название которых начинается с test_common, лежат данные взятые с конца корпуса и они подготовлены для тестирования. Таким образом, данные для обучения и для тестирования не пересекаются.

2.4 Обучение и тюнинг модели классификации

Для обучения модели использовался модуль из библиотеки sklearn. Сначала необходимо считать данные из файла (см. Приложение 3) с помощью все той же библиотеки Pandas и функции read_csv(). Далее в функцию train_model() передается функция для создания классификатора create_ngram_model(), данные для обучения и данные для тестирования.

Внутри функции train_model() все данные перемешиваются внутри себя для детерминированного поведения модели с помощью функции sample() из библиотеки Pandas. Используется реализация векторизатора (векторизатор - объект класса для проведения векторизации) класс TfidfVectorizer из sklearn, который уже реализует в себе Tf-Idf алгоритм. Для классификатора была взята реализация MultinomialNB() из библиотеки sklearn. Далее вызывается функция для создания классификатора: она возвращает объект Pipeline библиотеки sklearn, в котором находится объект векторизатора и классификатора. После чего происходит обучение модели с помощью функции fit(), которая возвращает объект обученной модели. Далее можно предсказать тональность какого либо текста с помощью функции predict_proba() или проверить результаты работы модели на тестовых или обучающих данных с помощью функции score().

Во время реализации модели некоторые параметры менялись, чтобы сравнить полученные результаты и найти наиболее оптимальный вариант работы классификатора для этих данных. Для того чтобы представить данные в допустимом для классификатора формате их нужно перевести в набор векторов. Для этого используются объекты Vectorizer из библиотеки sklearn. Можно использовать обычный векторизатор CountVectorizer, который просто считает количество вхождений элементов в текст и в конце выдает матрицу вхождений. Но существует векторизатор с уже встроенной по умолчанию метрикой TF-IDF – TfidfVectorizer. Он отличается от предыдущего только тем, что не просто считает количество вхождений слова в текст, но и применяет метрику TF-IDF для корректного распределения весов. В таблице 7 представлены параметры конструктора (гиперпараметры), которые менялись при обучении модели.

Таблица 7. Параметры, которые использовались при тюнинге модели.

Наименование параметра	Описание параметра
max_df	Число от 0.0 до 1.0 или больше единицы. При построении словаря игнорируются термины, которую имеют частоту документа строго **выше** заданного порога.
min_df	Число от 0.0 до 1.0 или больше единицы. При построении словаря игнорируются термины, которую имеют частоту документа строго **ниже** заданного порога.
ngram_range	Виды N-грамм.
smooth_idf	Булевское значение. При True добавляет единицу к любому IDF значению, как если бы всегда был один лишний документ, в котором встречается слово. Предотвращает нулевые деления.

Для того чтобы модель показывала наилучший результат, было решено попробовать менять различные параметры векторизатора. Вычислительный эксперимент описан в пункте 2.5. Процесс подбора наиболее оптимальных параметров называется оптимизацией гиперпараметров классификатора или тюнингом модели. Для того чтобы вручную перебрать все допустимые варианты параметров векторизатора и найти наиболее оптимальное решение

потребуется очень много времени. Для того случая был использован класс GridSearchCV из модуля model_selection библиотеки sklearn. Он позволяет передать ему в параметрах Pipeline объект из библиотеки sklearn и параметры для перебора. Дальше внутри этого класса произойдет автоматическое построение модели с каждым параметром и сравнение результатов. Конструктор класса возвращает объект, с помощью которого можно будет представить самые оптимальные параметры для конкретного набора данных.

2.5 Вычислительный эксперимент

Целью вычислительного эксперимента является подбор параметров модели. Для эксперимента использовался компьютер на базе процессора Intel Core i7-6700 с 4 физическими ядрами и 8 потоками. Оперативная память – 2 планки, DDR4, 16 ГБ с частотой 3200 МГц.

Изначально во время тюнинга модели было выставлено слишком много параметров для перебора: алгоритм работал очень долго. В один момент времени выполнения программы на языке python над этой программой может работать только одно ядро (см. рис. 7).

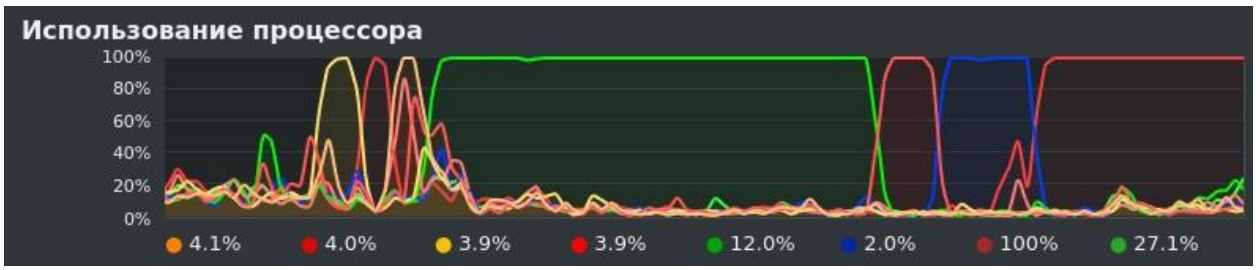

Рисунок 7. Визуализация загруженности процессора при обучении модели.

Видно, что когда работает одно ядро, остальные практически не загружены. При этом видно, что иногда контроль управления передается на другое ядро, и оно продолжает работу над программой. Через несколько часов вылетела ошибка о нехватке оперативной памяти. Тогда были

выставлены флаги на использование максимально доступной оперативной памяти, и на компьютере вся память была занята за 5 часов. Это случилось потому, что был выбран слишком большой корпус данных и слишком много параметров. Было принято решение уменьшить количество параметров (см. рис. 8).

```
parameters = {
    'vect__ngram_range': [(1, 1), (1, 2), (1, 3)],
    'vect__min_df': [1, 5, 10],
    'vect__max_df': [0.1, 0.2],
    'vect__smooth_idf': [False, True]
}
```

Рисунок 8. Параметры, используемые для перебора в вычислительном эксперименте.

Чтобы получить хотя бы какие-то результаты, было решено найти методом перебора оптимальное количество параметром для того, чтобы алгоритм закончил работу. В результате были получены оптимальные параметры для работы векторизатора (см. Приложение 4). Точность работы модели на этих параметрах представлена в таблице 8.

Таблица 8. Точность работы модели при подобранных параметрах классификатора.

Способ очистки / Размер обучающей выборки	10000	50000	80000	105000
Стемминг Стоп-слова	0.6702	0.7167	0.721	0.7235

Лемматизация Стоп-слова	0.6811	0.7205	0.725	0.7263
Стемминг	0.6785	0.7278	0.73365	0.7360
Лемматизация	0.7004	0.7345	0.73941	0.7401
Стемминг Лемматизация	0.687	0.7304	0.7369	0.7374

Были взяты выборки размером 10 тыс., 50 тыс., 80 тыс. и 105 тыс. текстов **каждого вида** (т.е. 10тыс положительной окраски и 10тыс отрицательной). После анализа полученных данных стало понятно, что можно убрать параметры min_df и max_df потому, что эти значения не играют важной роли. Кроме того по оценке видно что очистка без выкидывания стоп-слов дает на 1-1.5% больший показатель. Заметно, что на 50-80 тысячах процент точности уже составляет 72 – 74 процента и не увеличивается с увеличением количества текстов (см. рисунок 8). Все это было учтено в дальнейшем исследовании.

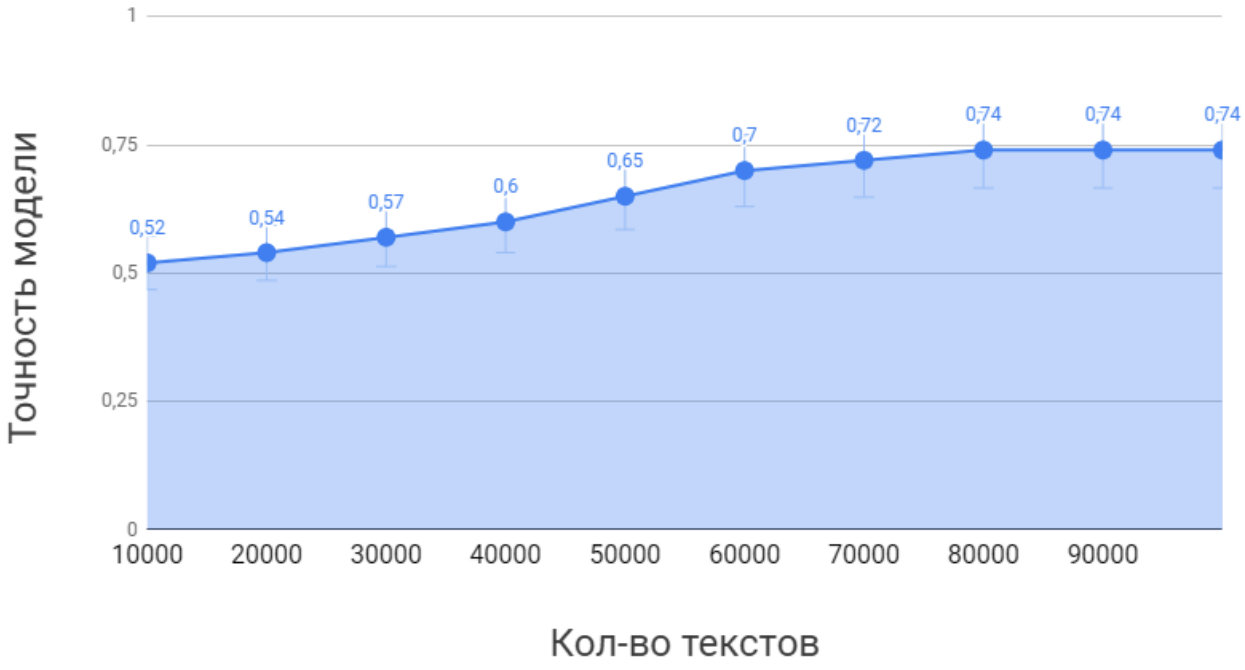

Рисунок 8. Точность модели в зависимости от количества текстов.

Возможно, отбрасывание стоп-слов дает результат хуже потому, что метрика TF-IDF дает им наименьший вес, но они все равно участвуют в классификации. Возможно, какие-то из них чаще встречаются в текстах определенной тональности.

Было решено подготовить выборку без выкидывания стоп-слов, применив к ней лемматизацию и стемминг вместе с лемматизацией. Важно сначала сделать лемматизацию, а затем уже стемминг: иначе при первом стемминге слова обратятся в их морфологическую основу, и лемматизация не сможет распознать никакое слово для преобразования.

Результаты несколько разнятся для 10 тыс. записей – лемматизация выигрывает на 2-4 процента по точности среди всех остальных способов очистки. Для 105 тыс. результаты показали, что очистка текстов с помощью лемматизации имеет преимущество перед стеммингом и стеммингом вместе с лемматизацией, даже несмотря на такую маленькую величину как три десятых процента. Для 50 и 80 тысяч значения смешиваются: где то лидирует стемминг в связке с лемматизацией, где-то лемматизация, а где-то стемминг.

Но в итоге, наилучший процент точности, который удалось получить, составляет 74% при использовании простой лемматизации (См. Приложение 4).

2.6 Инструменты для Web-разработки

Для того, чтобы пользователь мог воспользоваться моделью, необходимо реализовать удобный интерфейс для этого. Для создания веб приложения было произведено сравнение двух наиболее популярных Python фреймворков: Flask и Django. Из-за большого количества фреймворков для Python сравнивать их все не представляется возможным. Были выбраны самые популярные по версии нескольких источников [26] и произведен их краткий анализ.

Django — считается самым популярным фреймворком [26] с большим количеством встроенного функционала. По умолчанию в Django преднастроены ORM, URL-маршрутизация, единая аутентификация, миграция схемы данных и т. д. Кроме того, в Django имеется автогенерируемая админ панель, которая позволяет с легкостью управлять контентом базы данных через web-интерфейс.

Другим популярным фреймворком для Python считается Flask — минималистичный фреймворк, который позволяет быстро реализовать web-приложение. Особенность Flask считается его модульность — по умолчанию в Flask преднастроено минимум компонентов и разработчику предоставляется свобода в выборе необходимых пакетов или модулей для разработки проекта.

В web-приложении данной работы нет нужды в функциональной ORM, миграции базы данных или web-интерфейсе для административной панели, поэтому был выбран Flask за его простоту, как основной фреймворк для разработки.

Для контроля версий использовался такой инструмент как Git. Он позволил вести комфортную разработку на нескольких машинах и дал возможность откатиться к любому моменту разработки, чтобы получить части кода, которые сначала были написаны, а потом удалены.

2.7 Web-приложение для визуализации эмоциональной окраски комментариев под записями в социальной сети Вконтакте

Было реализовано несколько сервисов, каждый из которых отвечал за свой функционал. В файле .gitignore указаны файлы и папки, которые необходимо игнорировать при добавлении в систему контроля версий. Например, папка с конфигурациями Pycharm.

Для выгрузки комментариев из записей необходимо получить доступ к API социальной сети. У социальной сети Вконтакте разносторонне развитое API для сторонних разработчиков. Есть как общее API для простых http/https запросов по сети, так и несколько билиотек-оберток для наиболее популярных языков программирования.

Таблица 9. API функции, которые использовались при взаимодействии с ВК API.

Наименование метода в API	Аргументы	Для чего используется метод
groups.getById	group_id – id группы	Получить информацию о группе (имя, описание, фото). Для разрабатываемого приложения необходимо только имя группы.
wall.get	owner_id – id группы, count – количество	Получение “count” записей со стены сообщества.

	записей	
wall.getComments	owner_id – id группы, count – количество комментариев, post_id – id записи на стене, need_likes – 1 или 0, обозначает нужны ли отметки мне нравится к комментариям	Получение списка из “count” комментариев под записью на стене сообщества. По умолчанию count=10, но может быть увеличен до 100.

При использовании python в любом ответе из библиотеки vk приходит ассоциативный массив. Подробная документация API доступна на официальном сайте Вконтакте в разделе для разработчиков.

Приложение начинает свою работу с файла app.py. В нем импортируется файл src/__init__.py, в котором уже инициализируется flask сервер как переменная. С помощью этой переменной в src/routes.py было настроено API для приложения (см. приложение 6).

Сервис vk_service.py работает с API социальной сети VK и получает информацию о группах, постах в группах и комментариях (см. Приложение 5). Внутри сервиса используется официальная библиотека vk, которая предоставляет API для работы с данными социальной сети. Методы, которые использовались в сервисе, описаны в таблице 9.

Сервис cleaning_data_service.py отвечает за очистку и предобработку входящих данных (см. приложение 2). Чтобы, определение окраски было максимально точным, необходимо обработать данные для определения точно так же, как и данные, которые использовались для обучения модели.

В папке resource лежит сериализованный объект модели, который десериализуется в сервисе predict_service.py (см. приложение 7). Внутри этого сервиса происходит определение эмоциональной окраски тестов с помощью метода predict_proba.

В скрипте charts.py создается график на основе переданных данных. График отображает эмоциональную окраску комментариев под постом. Изначально, в работе использовалась библиотека matplotlib для построения графиков, но она слабо интегрирована с web-интерфейсом. Для отображения графика приходилось строить его, делать снимок этого графика в формате png и отображать на web-интерфейсе как картинку. Когда же понадобилось добавить дополнительную информацию о шкалах графика в подсказку, то было найдено решение – библиотека bokeh. Эта библиотека предоставляет простой API для построения графиков и различных изменений этих графиков: добавить легенду на оси, выбрать тип шкал и цвет, добавить всплывающую подсказку.

2.8 Результат работы

В результате работы были реализованы два проекта-приложения. Первый проект используется для предобработки данных, обучении модели и экспорта объекта этой модели в файл. Второй проект это web-приложение, в которое можно экспортировать объект обученной модели из файла, и через web-интерфейс зайти на сайт. Далее необходимо выбрать интересующую группу в социальной сети Вконтакте и получить информацию об эмоциональной окраске комментариев под записью в виде рисунка. На горизонтальной оси отображается Id записи, чтобы пользователь сайта мог однозначно идентифицировать запись в социальной сети. На вертикальной оси средняя эмоциональная окраска всех комментариев под записью. Если комментарии отсутствуют, на графике будет пустая шкала. При наведении на шкалу высвечивается информация о записи:

Таблица 10. Описание полей подсказки к рисунку 9.

Наименование поля	Описание
Среднее вероятностное значение эмоциональной окраски комментариев	Результат оценки каждого комментария брался как вероятность, с которой этот комментарий относится к классу
Среднее классовое значение эмоциональной окраски комментариев	Результат оценки каждого комментария брался как точное значение: 0 для класса с негативной окраской или 1 для класса с позитивной окраской
Название записи	Наименование записи. Иногда может отсутствовать
Количество комментариев	Количество комментариев под записью, которое было проанализировано
Количество комментариев с позитивной окраской	Количество комментариев с позитивной эмоциональной окраской.

Предположим, необходимо проанализировать записи из группы с татуировками. Группа с татуировками взята потому, что пользователи в комментариях наиболее четко выражают, понравилась им запись с татуировкой или нет. Нужно запросить 4 записи со стены этой группы.

Рисунок 4. Эмоциональная окраска комментариев под 4 записями. По вертикальной оси отображается эмоциональная оценка комментариев. На горизонтальной оси идентификатор записей. В подсказке выводится краткая информация о записи.

На рисунке 9 видно, что большинство записей находятся около отметки в 0.5, т.е. на 50% позитивной эмоциональной окраски (большое разрешение доступно в приложении 8). Это означает, что мнение пользователей разделилось. Не стоит забывать, что сейчас очень распространены боты, которые, преследуя рекламные цели, размещают ссылки в комментариях сообществ на другие сообщества. Но есть и записи, которые выделяются. Например, запись с id 550586 имеет высокий показатель позитивной окраски. Можно предположить, что комментарии будут в большинстве своем положительные. Если посмотреть таблицу с комментариями и их эмоциональной окраской, то можно увидеть, что пользователям действительно понравилась запись (см. рис. 10). Комментарии считаются позитивно окрашенными (относящимися к классу с позитивной эмоциональной окраской), если значение эмоциональной окраски больше 0,5.

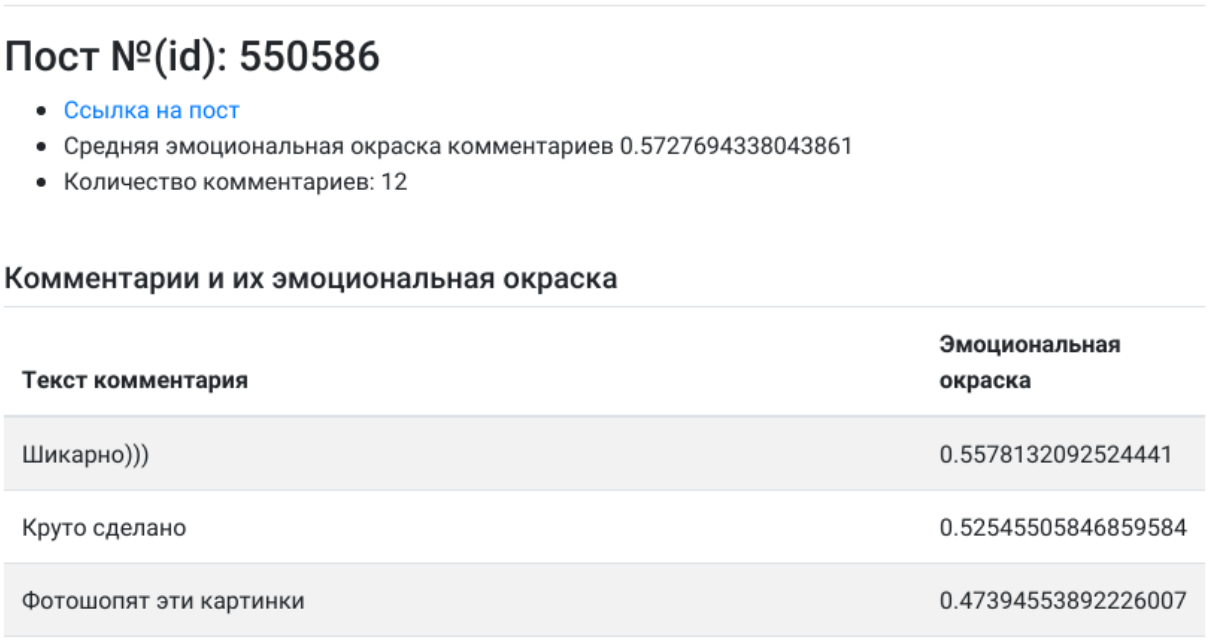

Рисунок 5. Краткая информация о записи.

Таблица с комментариями и их эмоциональной окраской доступна в web-приложении в разделе «Подробности о группе». В этом разделе представлена краткая информация о записях из группы (см. рис. 10): идентификатор записи, ссылка в социальной сети Вконтакте на запись, средняя эмоциональная окраска комментариев и количество комментариев.

Таким образом, администраторы могут оценить какие типы записей нравятся людям, а какие нет. Записи, которые не нравятся пользователям или вызывают у них негативные эмоции лучше исключить из новостной ленты сообщества. Таким образом, можно повысить посещаемость группы и лояльность пользователей.

Заключение

В данной работе были рассмотрены различные подходы и особенности решения задачи классификации текстовых документов.

Обучающая выборка была взята с сайта study.mokoron.com, предоставляющего уже размеченные наборы данных. Данные были очищены и подготовлены для работы с помощью языка программирования Python и его библиотек для работы с данными: Pandas, NumPy и др.

Для построения модели использовался наивный байесовский классификатор и TF-IDF метрика. Модель была реализована с помощью библиотеки sklearn. Объект обученной модели был сериализован и сохранен в файл для дальнейшего использования.

Было изучено API социальной сети Вконтакте. Для получения комментариев из социальной сети использовалась библиотека vk.

Было разработано web-приложения для прогнозирования и визуализации предсказания с использованием фреймворка Flask. Пользователи приложения смогут наглядно увидеть, как участники сообществ Вконтакте реагируют в комментариях на различную информацию, размещаемую в записях сообщества.

В перспективе планируется интегрировать работу сервиса с площадкой youtube.com и анализировать комментарии под видео. В этом будут заинтересованы различные люди, которые ведут свой канал и им необходимо знать общее мнение других людей об их работе.

Список литературы

1. Lewis D. D. Text representation for intelligent text retrieval: A classification-oriented view //Text-based intelligent systems: current research and practice in information extraction and retrieval. – 1992. – С. 179-197.
2. Wang F., Deng X., Hou L. Chinese News Text Multi Classification Based on Naive Bayes Algorithm //Proceedings of the 2nd International Symposium on Computer Science and Intelligent Control. – ACM, 2018. – С. 31.
3. Романов А. А. Классификация тональности текстовых документов с помощью метода опорных векторов. – 2017.
4. Максаков А. Сравнительный анализ алгоритмов классификации и способов представления Web-документов //Труды третьего российского семинара РОМИП. – 2005. – С. 63-73.
5. Романов А. С. Методика идентификации автора текста на основе аппарата опорных векторов //Доклады Томского государственного университета систем управления и радиоэлектроники. – 2009. – №. 1-2 (19).
6. Осокин В. В., Шегай М. В. Анализ тональности русскоязычного текста //Интеллектуальные системы. Теория и приложения. – 2014. – №. 3. – С. 163-172.
7. Forman G. An extensive empirical study of feature selection metrics for text classification //Journal of machine learning research. – 2003. – Т. 3. – №. Mar. – С. 1289-1305.
8. Mohammad A. H., Alwada'n T., Al-Momani O. Arabic text categorization using support vector machine, Naïve Bayes and neural network //GSTF Journal on Computing (JoC). – 2018. – Т. 5. – №. 1.
9. Дуда Р. Распознавание образов и анализ сцен. – Рипол Классик, 1976.
10. Классификация с помощью наивного байесовского подхода [Электронный ресурс]. — Режим доступа: URL:

https://nlp.stanford.edu/IR-book/html/htmledition/naive-bayes-text-classification-1.html (24.02.19)

11. Введение в машинное обучение с помощью Python и Scikit-Learn [Электронный ресурс]. — Режим доступа: URL https://habr.com/ru/company/mlclass/blog/247751/ (25.02.19)
12. Метод k взвешенных ближайших соседей (пример) [Электронный ресурс]. — Режим доступа: URL http://www.machinelearning.ru/wiki/index.php?title=Метод_k_взвешенных_ближайших_соседей_(пример) (2.03.19)
13. Использование деревьев решений в задачах прогнозной аналитики [Электронный ресурс]. — Режим доступа: URL http://www.prognoz.ru/blog/platform/decision-tree-in-predictive-analytics/ (29.03.19)
14. Text Classification With Word2Vec [Электронный ресурс]. — Режим доступа: URL http://nadbordrozd.github.io/blog/2016/05/20/text-classification-with-word2vec/ (21.03.19)
15. Global Vectors for Word Representation [Электронный ресурс]. — Режим доступа: URL https://nlp.stanford.edu/projects/glove/
16. Чусовлянов Д. С., Игнатов Д. И. Машинное обучение для определения тональности и классификации текстов на несколько классов. – 2014. С. 26.
17. Обогащение модели Bag of words семантическими связями для повышения качества классификации текстов предметной области [Электронный ресурс]. — Режим доступа: URL http://www.swsys.ru/index.php?page=article&id=4153 (14.04.19)
18. Бидюк П. И., Терентьев А. Н. Построение и методы обучения байесовских сетей. – 2004.
19. Наивный байесовский классификатор [Электронный ресурс]. — Режим доступа: URL http://bazhenov.me/blog/2012/06/11/naive-bayes.html (10.01.19)

20. Panchenko A. Technology of the automated thesaurus construction for Information Retrieval //Intelligence Systems and Technologies, Bauman Moscow State Technical University, Moscow. – 2009. – Т. 9. – С. 124-140.
21. Шатовская Т., Каменева И. Интегрированный подход текстовой кластеризации для неструктурированных документов. – INTERNET–EDUCATION–SCIENCE: материалы 6-й Международной конференции (Винница, Украина, 7–11 октября, 2008 г.).–Винница, 2008.
22. Sentiment AnalysisSentiment Analysis [Электронный ресурс]. — Режим доступа: URL https://monkeylearn.com/sentiment-analysis/ (14.03.19)
23. Рубцова Ю. Автоматическое построение и анализ корпуса коротких текстов (постов микроблогов) для задачи разработки и тренировки тонового классификатора //Инженерия знаний и технологии семантического веба. – 2012. – Т. 1. – С. 109-116.
24. Read J. Using emoticons to reduce dependency in machine learning techniques for sentiment classification //Proceedings of the ACL student research workshop. – Association for Computational Linguistics, 2005. – С. 43-48.
25. Documentation for pandas.DataFrame [Электронный ресурс]. — Режим доступа: URL https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.html (14.02.19)
26. Лучшие фреймворки для разработки на Python [Электронный ресурс]. — Режим доступа: URL https://techrocks.ru/2018/11/20/best-python-frameworks/ (19.03.19)

Приложение 1. Описание набора данных

В качестве источника текстов была выбрана платформа микроблогинга Twitter. Современные поисковые системы и имеющиеся в открытом доступе инструменты по сбору текстовых отзывов не позволяют собирать актуальные отзывы и оперативно работать с данными. В связи с этим на основе программного интерефейса API twitter был разработан программный инструмент для извлечения отзывов об интересующих товарах, услугах, событиях, персонах из микроблоггинг-платформы twitter, который позволяет учитывать время публикации сообщения и авторитетность автора сообщения. Этот инструмент использовался для сбора неразмеченного корпуса. В корпусе содержится более 15 миллионов записей за время с конца ноября 2013 года до конца февраля 2014 года.

Автоматическая классификация отзывов (и разметка корпуса) осуществляется по методу, предложенному Jonathon Read [2005]. Для разметки на два класса (положительные и отрицательные), тестовая выборка была отфильтрована, согласно следующим критериям:

- Удалялись все твиты, содержащие одновременно и положительные и отрицательные эмоции;
- Как выяснилось, API twitter отдает в результатах выдачи копии twitter- постов. В связи с этим необходимо было удалять одинаковые посты из тестовой выборки;
- Удалялись малоинформативные твиты, длина которых составляла менее 40 символов.

Приложение 2. Листинг кода для очистки данных

```
import pandas as pd
import pymorphy2
import nltk.stem
from nltk.corpus import stopwords
import numpy as np
import re


alphabet = 'абвгдеёжзийклмнопрстуфхцчшщъыьэюя'
s = nltk.stem.SnowballStemmer('russian')
morph = pymorphy2.MorphAnalyzer()
prepare_func = []
stop_words = stopwords.words('russian')

def text_stemming(text):
    return s.stem(text)

def text_remove_stop_words(text):
    return ' '.join([word for word in text.split(' ') if word not in
stop_words])

def text_to_lower(text):
    return text.lower()

def text_letter_only(text):
    new_text = ''
    # is prev a space
    prev = False
    for letter in text:
        if letter in alphabet:
            new_text += letter
            prev = False
            continue
        if letter == ' ' and not prev:
            new_text += letter
            prev = True
    return new_text.replace('ё', 'е')


def text_lemmatization(text):
    lem_text_array = []
    for word in text.split(' '):
        lem_text_array.append(morph.parse(word)[0].normal_form)
    return ' '.join(lem_text_array)


def apply_func(text, func_array):
    for func in func_array:
        text = func(text)
    return text


# manage all preparing in one function
def prepare_text(text):
    return apply_func(text, prepare_func)
```

```
def prepare_array(texts):
    prepare_func = [
        text_to_lower,
        text_letter_only,

        # text_remove_stop_words,
        # text_stemming,
        text_lemmatization,
    ]
    new_texts = []
    for text in texts:
        new_texts.append(apply_func(text, prepare_func))
    return new_texts

def read_csv(filename):
    return pd.read_csv(filename,
                       header=None,
                       delimiter=";")


def clear(pos, neg, index_from, index_to, save_file):
    pos_data = pos
    neg_data = neg

    neg_data = neg_data[index_from:index_to][[3, 4]]
    neg_data.columns = ['text', 'label']
    neg_data['label'] = np.zeros(index_to - index_from)

    pos_data = pos_data[index_from:index_to][[3, 4]]
    pos_data.columns = ['text', 'label']

    print(pos_data['text'][:10])
    common_data = pd.concat([pos_data, neg_data])
    common_data['text'] = common_data['text'].apply(prepare_text)
    print(common_data['text'][:10])

    common_data.to_csv(save_file)
    print('DONE')


if __name__ == '__main__':
    nltk.download('stopwords')

    prepare_func = [
        text_to_lower,
        text_letter_only,

        # text_remove_stop_words,
        # text_stemming,
        text_lemmatization,
    ]
    pos_data = read_csv("original_data/positive.csv")
    neg_data = read_csv("original_data/negative.csv")
    dir_name = 'data/lem.1'

    clear(pos_data, neg_data, 0, 10000, dir_name + '/clean_common10.csv')
    clear(pos_data, neg_data, 0, 50000, dir_name + '/clean_common50.csv')
```

```
    clear(pos_data, neg_data, 0, 80000, dir_name + '/clean_common80.csv')
    clear(pos_data, neg_data, 0, 105000, dir_name + '/clean_common105.csv')

    clear(pos_data, neg_data, 0, 10000, dir_name + '/test_commonTeach.csv')
    clear(pos_data, neg_data, 105000, 110000, dir_name +
'/test_commonNew.csv')
    print('FINISH')
```

Приложение 3. Листинг кода для обучения модели

```
import numpy as np
import pandas as pd
from sklearn.naive_bayes import MultinomialNB, BernoulliNB
from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer
from sklearn.pipeline import Pipeline
from sklearn.model_selection import GridSearchCV
from sklearn.externals import joblib

from clear_data import prepare_array
from word2vecModel import TfidfEmbeddingVectorizer


def create_ngram_model():
    tfidf_ngrams = TfidfVectorizer(ngram_range=(1, 3), analyzer="word")
    clf = MultinomialNB()
    return Pipeline([('vect', tfidf_ngrams), ('clf', clf)])


def get_vectorizer(n_gram, count=False):
    if count:
        return CountVectorizer(ngram_range=n_gram, analyzer="word",
decode_error='ignore')
    return TfidfVectorizer(ngram_range=n_gram, analyzer="word")


def get_best_parameters(clf_factory, data, test_data):
    # перемешиваем индексы, чтобы получить детерминированное поведение
    data = data.sample(frac=1)
    test_data = test_data.sample(frac=1)
    test_data = test_data.reset_index(drop=True)

    # в X лежат текста, в Y их оценка - 0 или 1
    X_train, y_train = data['text'], data['label']
    X_test, y_test = test_data['text'], test_data['label']

    # задаем параметры
    parameters = {
        'vect__ngram_range': [(1, 1), (1, 2), (1, 3)],
        # 'vect__min_df': [1, 5, 10],
        # 'vect__max_df': [0.1, 0.2],
        'vect__use_idf': [False, True],
        'vect__smooth_idf': [False, True],
        'vect__sublinear_tf': [False, True]
    }
    fact_pipeline = clf_factory()
    clf = GridSearchCV(fact_pipeline, parameters, verbose=1)
    # обучаем
    clf.fit(X_train, y_train)

    clf_ = clf.best_estimator_

    # результаты на обучающей выборке
    train_score = clf_.score(X_train, y_train)
    # результаты на тестовой выборке
    test_score = clf_.score(X_test, y_test)
```

```
    print('train: ' + str(train_score))
    print('test: ' + str(test_score))
    return clf


def train_model(data, test_data):
    data = data.sample(frac=1)
    test_data = test_data.sample(frac=1)
    test_data = test_data.reset_index(drop=True)

    # в X лежат текста, в Y их оценка - 0 или 1
    X_train, y_train = data['text'], data['label']
    X_test, y_test = test_data['text'], test_data['label']

    tfidf_ngrams = TfidfVectorizer(ngram_range=(1, 3), smooth_idf=True,
sublinear_tf=True, use_idf=True, analyzer="word")
    # tfidf_ngrams = TfidfEmbeddingVectorizer(word2vec=get_vectorizer())
    clf = MultinomialNB()
    clf = Pipeline([('vect', tfidf_ngrams), ('clf', clf)])
    # обучаем
    clf.fit(X_train, y_train)

    # результаты на обучающей выборке
    train_score = clf.score(X_train, y_train)
    # результаты на тестовой выборке
    test_score = clf.score(X_test, y_test)

    print('train: ' + str(train_score))
    print('test: ' + str(test_score))
    return clf


def best_model(dir, file_n):
    dir_name = dir

    file_name = file_n

    data = pd.read_csv(dir_name + '/' + file_name).dropna()

    test_data = pd.read_csv(dir_name + '/test_commonNew.csv',
index_col=False).dropna()
    test_data = pd.concat([test_data['text'], test_data['label']], axis=1)

    clf = get_best_parameters(create_ngram_model, data, test_data)

    f = open('out/' + file_name + '__' + dir.split('/')[1] + '.txt', 'w')
    f.write(str(clf.best_score_))
    f.write('\r\n\r\n')
    f.write(str(clf.best_params_))
    f.write('\r\n\r\n')
    f.write(str(clf.best_estimator_.steps))
    for i in clf.best_estimator_.steps:
        print(i)

if __name__ == '__main__':
    desired_width = 1080
    pd.set_option('display.width', desired_width)
```

```
    np.set_printoptions(linewidth=desired_width)

    # dirs = ['data/stem_lem',]
    # files = ['clean_common110.csv',]
    # for file in files:
    #     for dir_ in dirs:
    #         best_model(dir_, file)

    data = [
        "Летом я был у бабушки и мне очень понравилост в деревне спасибо
всем",
        "я к сожеланию болею ((((",
        "ура я наконец выздоровел все хорошо здорово прекрасно лето люблю
лето",
        "тоска оружие ужас кошмар убийство смерть раздор ссора ненавижу гроза
страх печаль грусть убил задушил гнев"
                            ]

    # clear_data = prepare_array(data)
    #
    # clf = joblib.load('finalized_model_110.sav')
    print("=========")
    # a = clf.predict_proba(clear_data)
    # print(a)
    print("=========")

    dir = "lem.1"
    data = pd.read_csv('data/'+ dir +'/clean_common80.csv').dropna()

    test_data = pd.read_csv('data/lem/test_commonNew.csv',
index_col=False).dropna()
    test_data = pd.concat([test_data['text'], test_data['label']], axis=1)

    clf = train_model(data, test_data)
    # filename = 'finalized_model_80_' + dir + '.sav'
    # joblib.dump(clf, filename)
```

Приложение 4. Результаты исследований. Таблица зависимости способа очистки данных и размерности обучающей выборки от точности предсказания модели

Способ очистки / Размер обучающей выборки	10000	50000	80000	105000
Стемминг Стоп-слова	0.6702686477 max_df:0.1 min_df:1 ngram_range:(1,2) smooth_idf:False sublinear_tf:False use_idf:False	0.716751726208 max_df:0.1 min_df:1 ngram_range:(1,3) smooth_idf:False sublinear_tf:True use_idf:False	0.72137822337 max_df:0.1 min_df:1 ngram_range:(1,3) smooth_idf:False sublinear_tf:True use_idf:False	0.72355968549 ngram_range:(1,3) smooth_idf:False sublinear_tf:False use_idf:False
Лемматизация	0.68117464605	0.720564395077	0.725775078649	0.726342625685

Стоп-слова	max_df:0.1 min_df:1 ngram_range:(1,3) smooth_idf:False sublinear_tf:True use_idf:False	max_df:0.1 min_df:1 ngram_range:(1,3) smooth_idf:False sublinear_tf:False use_idf:False	max_df:0.1 min_df:1 ngram_range:(1,3) smooth_idf:False sublinear_tf:True use_idf:False	ngram_range:(1,3) smooth_idf:False sublinear_tf:True use_idf:False
Стемминг	0.67855 max_df:0.2 min_df:1 ngram_range:(1,2) smooth_idf:False sublinear_tf:True use_idf: False	0.72788 max_df:0.1 min_df:1 ngram_range:(1,3) smooth_idf:True sublinear_tf:False	0.73365625 ngram_range:(1,3) smooth_idf:True sublinear_tf:True use_idf:True	0.736014285714 ngram_range:(1,3) smooth_idf: True sublinear_tf: False use_idf: True

		use_idf:True		
Лемматизация	0.7004 ngram_range:(1,3) smooth_idf:False sublinear_tf:True use_idf:False	0.73455 ngram_range:(1,3) smooth_idf:True sublinear_tf:True use_idf:True	0.73941875 ngram_range:(1,3) smooth_idf:True sublinear_tf:True use_idf:True	0.740138095238 ngram_range:(1,3) smooth_idf:True sublinear_tf:True use_idf:True
Стемминг Лемматизация	0.687 ngram_range:(1,2) smooth_idf:False sublinear_tf:True use_idf:False	0.73041 ngram_range:(1,3) smooth_idf:True sublinear_tf:True use_idf:True	0.73698125 ngram_range:(1,3) smooth_idf:True sublinear_tf:True use_idf:True	0.737480952381 ngram_range:(1,3) smooth_idf:False sublinear_tf:True use_idf:False

Приложение 5. Листинг кода для работы с VK API

```
import re

import numpy as np
import vk
import time

from src.predict_service import get_predict_for_text_list

session = vk.Session(access_token='xxx')
vk_api = vk.API(session, v=5.0)


def get_groupname_by_group_id(group_id):
    """
    Get group name by the group id.
    If group id starts with '-' char, it need to replace it, because some API
calls need '-' char
    :param group_id:
    :return: String of group name
    """
    if group_id[0] == '-':
        group_id = group_id[1:]
    a = vk_api.groups.getById(group_id=group_id, group_ids=[])
    return a[0]['name']


def get_posts_texts(group_id, count=50):
    """
    Get the @count posts from group with @group_id.
    :param group_id:
    :param count:
    :return:
    """
    items = vk_api.wall.get(owner_id=group_id, count=count)['items']
    result = {}
    for post in items:
        date_key = get_start_day_of_date(post['date'])
        if date_key not in result:
            result[post['id']] = [compose_post_structure(group_id, post)]
        else:
            result[post['id']].append(compose_post_structure(group_id, post))
    return result


def compose_post_structure(group_id, vk_post):
    """
    Compose the data structure for present it on UI
    :param group_id:
    :param vk_post:
    :return:
    """
    post_comments_texts = np.array(get_post_comments(group_id,
vk_post['id']))
    predict = get_predict_for_text_list(post_comments_texts)
    neg = [x[0] for x in predict]
```

```
    pos = [x[1] for x in predict]

    avg_pos = np.nan_to_num(np.mean(pos))
    avg_neg = np.nan_to_num(np.mean(neg))
    mean_pos = np.nan_to_num(np.median(pos))
    mean_neg = np.nan_to_num(np.median(neg))

    pos = [1 if (x[1]>0.5) else 0 for x in predict]
    avg_class_pos = np.nan_to_num(np.mean(pos))
    return {
        'post_id': vk_post['id'],
        'text': vk_post['text'][:30],
        'comments': post_comments_texts,
        'predict': (avg_neg, avg_pos),
        'predict_mean': (mean_neg, mean_pos),
        'date': str(time.ctime(vk_post['date'])),
        'predict_class': avg_class_pos
    }


def get_post_comments(group_id, post_id):
    """
    Fetch post comments and get the text of that comments. Text will not
contain first link to another user like
    "Author, den"
    :param group_id:
    :param post_id:
    :return: list of strings of post comments
    """
    comments_of_post = vk_api.wall.getComments(owner_id=group_id,
post_id=post_id, need_likes=1, count=50)
    return np.array([re.sub('^[\[.(a-zA-Z0-9)+\|.(a-zA-Za-яА-Я0-9)+\].\,.]+',
'',
                            comment['text']).strip() for comment in
comments_of_post['items']])


def get_start_day_of_date(unix_date):
    """
    Get start of a day by unix_date
    """
    return time.strftime("%Y-%m-%d", time.localtime(int(unix_date)))
```

Приложение 6. Листинг кода для роутинга в фреймворке Flask

```
import base64
import io

import matplotlib.pyplot as plt

from charts import create_hover_tool, create_bar_chart
from src import app
from flask import render_template, request
import src.vk_service as vk_service
from bokeh.embed import components
import numpy as np


@app.route('/', methods=['GET'])
def index():
    """
    Return start page
    :return:
    """
    user = {'username': 'User'}
    return render_template('index.html', title='Home', user=user)

def build_graph(x_coordinates, y_coordinates):
    """
    Function for create graph picture. Build it with plt and make a snapshot.
    Result in PNG
    :param x_coordinates:
    :param y_coordinates:
    :return:
    """
    img = io.BytesIO()
    plt.plot(x_coordinates, y_coordinates)
    plt.savefig(img, format='png')
    img.seek(0)
    graph_url = base64.b64encode(img.getvalue()).decode()
    plt.close()
    return '{}'.format(graph_url)

@app.route("/", methods=['POST'])
def process():
    # group id must start with '-' for vk api
    group_id_value = '-' + request.form['group_id']

    data = vk_service.get_posts_texts(group_id_value,
request.form['post_count'])
    values = [np.mean([post['predict'][0] for post in x if post['predict']])
for x in data.values()]
    labels = data.keys()
    result = {
        'group_name': vk_service.get_groupname_by_group_id(group_id_value),
        'posts': data,
```

```
            'labels': labels,
            'values': values,
            'graph': build_graph(labels, values)
        }

        data_for_chart = {
            "ids": [],
            "sentiments": [],
            "sentiments_class": [],
            "texts": [],
            "dates": [],
            "mean": []
        }
        posts = sorted(data.keys())
        for i in range(0, int(len(posts))):
            data_for_chart['ids'].append(str(data[posts[i]][0]['post_id']))

data_for_chart['sentiments'].append(round(data[posts[i]][0]['predict'][0],
2))

data_for_chart['sentiments_class'].append(round(data[posts[i]][0]['predict_cl
ass'], 2))
            data_for_chart['texts'].append(data[posts[i]][0]['text'])
            data_for_chart['dates'].append(data[posts[i]][0]['date'])

data_for_chart['mean'].append(round(data[posts[i]][0]['predict_mean'][0], 2))

        hover = create_hover_tool()
        plot = create_bar_chart(data_for_chart, "1", "2",
                                "sentiments", "sentiments_class", hover)
        script, div = components(plot)

        return render_template('index.html',
                               result=result,
bars_count=request.form['post_count'],
                               the_div=div, the_script=script)
```

Приложение 7. Листинг кода для десериализации модели и предсказания класса эмоциональной окраски

```
from sklearn.externals import joblib

from src.cleaning_data_service import prepare_array

clf = joblib.load('./resource/finalized_model_110_lem.1.sav')

def get_predict_for_text_list(text_list):
    """
    Calculate predict for text from args
    :param text:
    :return: tuple with 2 value: negative and positive predict
    """
    if text_list.size == 0:
        return []
    # clear text before predict
    prepared_text = prepare_array(text_list)
    if len(prepared_text)==0:
        return []
    predict = clf.predict_proba(prepared_text)
    return predict
```

Приложение 8. Результирующий рисунок с эмоциональной окраской