Санкт–Петербургский государственный университет

Кафедра технологии программирования

Цимбалов Андрей Евгеньевич

Выпускная квалификационная работа бакалавра

Применение методов машинного обучения в задаче агрегации новостных статей

Направление 01.03.02

Прикладная математика, фундаментальная информатика и программирование

Научный руководитель:

ст. преподаватель, Стученков Александр Борисович,

Научный руководитель:

канд. техн. наук, доцент, Блеканов Иван Станиславович

Санкт-Петербург

2021 г.

Содержание

Введение

В настоящее время существует множество новостных сайтов, генерирующих содержимое разностороннего характера. Для объединения всей информации в одном удобном пользователю месте, были созданы новостные агрегаторы. Однако для разделения новостей по темам они либо используют ручной подход, либо ориентируются на то, к какой теме принадлежит новость в оригинальном источнике. В первом случае разметка будет весьма субъективна, к тому же могут допускаться ошибки. Также наличие большого объема источников пропорционально увеличивает необходимый штат сотрудников. Во втором случае необходимо настраивать точную сеть тематического соответствия между новостным агрегатором и каждым сайтом в отдельности. А также исключается возможность использования ресурсов, на которых отсутствует тематическая разметка.

Актуальность работы заключается в исследовании методов автоматического разделения коллекции новостей на заранее заданные тематики. Это поможет автоматизировать новостные агрегаторы и позволит им пользоваться новостными ресурсами без предварительной разметки.

Объект исследования – применение методов классификации для предоставления пользователю средств навигации по коллекции документов.

Предмет исследования – разбиение новостных документов на темы при помощи классификации и векторных моделей.

Цель работы – сравнение методов машинного обучения в задаче классификации и векторизации новостных статей.

Постановка задачи

Целью данной работы является построение автоматических классификаторов и векторизаторов новостных документов и сравнение методов их работы на заданном корпусе.

Поставленную задачу можно разбить на следующие подзадачи:

1. сбор базы данных новостных статей;
2. предварительная обработка текстов коллекции;
3. выбор, разработка и применение методов векторизации документов;
4. описание классов и разбиение данных при помощи тематической разметки;
5. выбор, разработка и применение методов классификации данных;
6. оценка качества классификации;
7. проведение сравнительного анализа полученных результатов.

Обзор литературы

Книга [1] предназначена для ознакомления с классическими приемами обработки естественного языка. В ней не только описаны теоретические аспекты предметной области и основная терминология, но и приведены примеры практически решаемых задач на языке Python с использованием библиотеки `NLTK`. В данной книге рассмотрены такие задачи как: предобработка текста, написание структурированных программ, извлечение информации из текста, классификация документов, управление лингвистическими данными, смысловой анализ предложений.

В статье [2] охватывает комплекс проблем задачи классификации документов. В ней рассматривается ряд методов предварительной обработки текста: токенизация, удаление стоп-слов, приведение слов к нижнему регистру, обработка сленга и аббревиатур, исправление орфографии, стемминг и лемматизация. Проводится сравнительный анализ различных классических методов классификации и векторизации текста. Расписаны достоинства и недостатки каждой метода. Приведены методики оценки: точность, полнота, F-мера, а также микро и макро усреднение.

В работе [6] изучается влияние методов векторизации документов (учитывающих и не учитывающих семантику слов в тексте) на точность классификации. Авторы статьи рассматривают векторизаторы: Term Frequency, TF-IDF, Word2Vec (CBOW и Skip-Gram архитектуры), Doc2Vec (DBOW и DM архитектуры) и Sparse Composite Document Vectors - и применяют их к различным классификаторам: Logistic Regression, Naive Bayes, Random Forests, SVM. Точность работы алгоритмов оценивается F-мерой. Для каждой пары векторизатор-классификатор вычисляются и сравниваются показатели работы моделей. Для Logistic Regression, Naive Bayes наибольшая точность была достигнута при использовании Term Frequency векторизатора. Методы Doc2Vec показали низкие результаты с Random Forests. Для классификатора SVM не

было выявлено существенной разницы в выборе векторизатора. Авторы делают вывод, что Term Frequency почти всегда было достаточно, так как он стабильно работал с простыми и сложными методами классификации.

Статья [13] написана создателями нейросетевой модели Bert, основанной на архитектуре transformers. В ней описан принцип работы модели, а также приведены различия двух её разновидностей: Bert_base и Bert_large. В статье [3] показаны способы тонкой настройки Bert для задачи классификации и проведена оценка точности работы на восьми различных датасетах, рассмотрены способы решения основных проблем модели. К ним относится предобработка длинных документов с целью применения их в Bert, так как размер подаваемого вектора ограничен 512 токенами. Также показано влияние выбора слоя для классификации (различные слои нейронной сети могут захватывать разные уровни синтаксической и семантической информации). Приведена проблема возможного переобучения модели.

1 Анализ предметной области

Классификация документов - одна из задач информационного поиска, заключающаяся в отнесении документа к одной из нескольких категорий на основании содержания документа. Информационный поиск до некоторого времени оставался скромной научной и прикладной областью, в которой работало относительно небольшое количество ученых. Однако в последние годы эта область получила сильное развитие. Современный информационный поиск позволяет эффективно работать с большими объемами данных, которые генерируются ежедневно [4].

В машинном обучении задача классификации относится к разделу обучения с учителем. Для эффективной работы алгоритма классификации необходима обучающая выборка - некоторое количество хороших образцов документов из каждого класса. Однако в обучении с учителем сохраняется необходимость предварительной подготовки (ручной разметки) данных.

1.1 Анализ существующих решений

В настоящее время, для объединения новостных статей с разных сайтов в одном месте, существуют новостные агрегаторы. Для конкретики, агрегатор СМИ - это компьютерная программа или информационный интернет-ресурс, отображающий структурированную новостную ленту, формируемую им подборку из новостей различных СМИ.

Примерами новостных агрегаторов является Google News, Яндекс.Новости. Оба сайта имеют большую аудиторию, но при этом они совершенно разные по принципу работы. (см. рисунки 1.1 и 1.2).

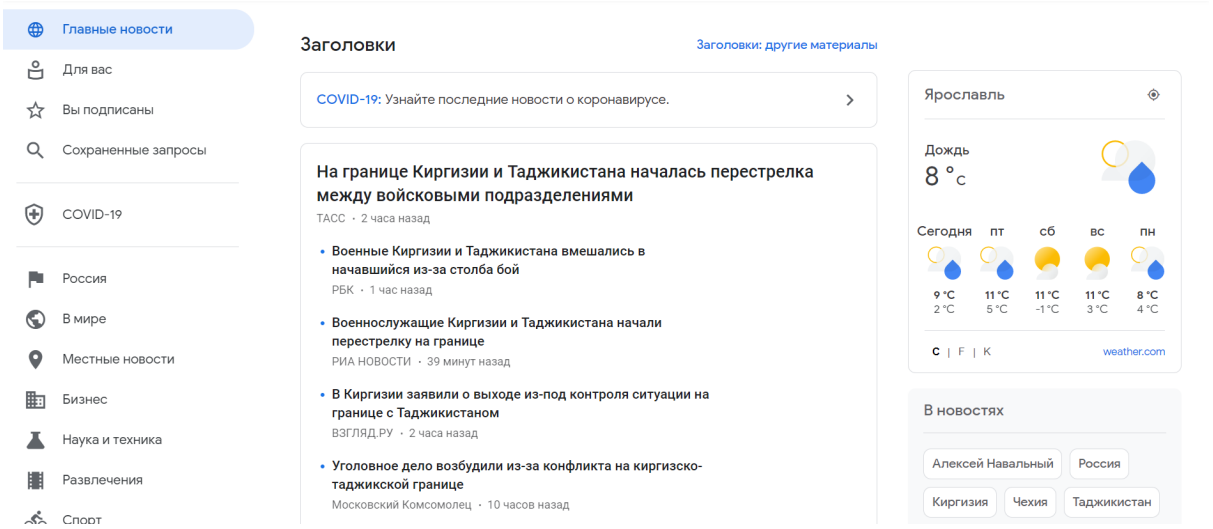

Рисунок 1.1 - Интерфейс Google News.

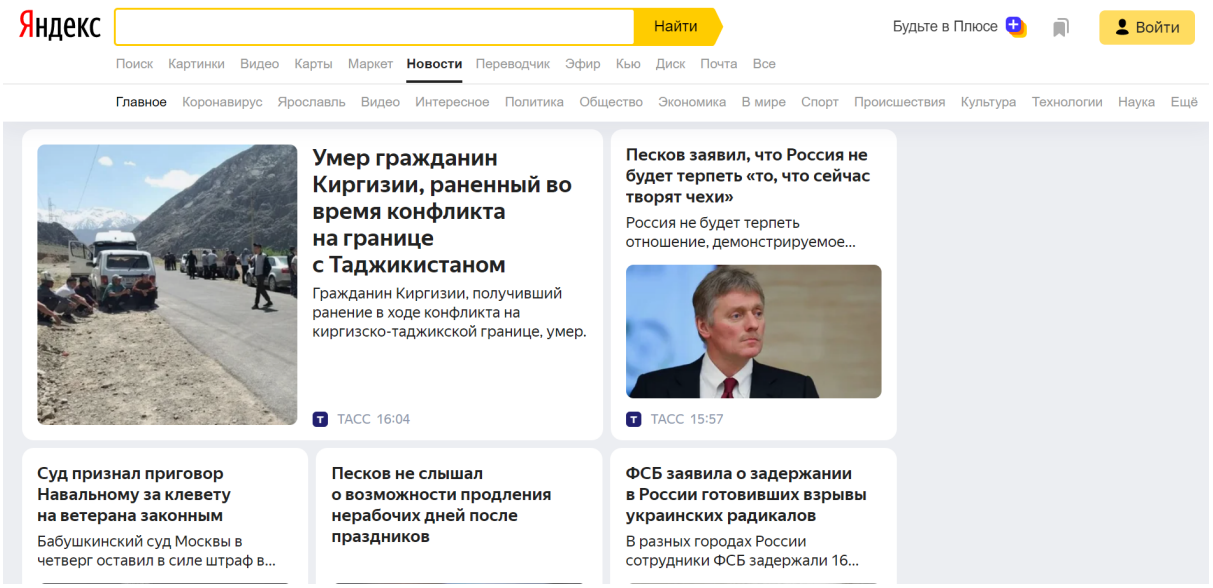

Рисунок 1.2 - Интерфейс Яндекс.Новости.

Google News разделяет новости на категории, а затем генерирует ленту для каждой из них. Новости со схожей тематикой и близкие по времени написания объединяются в блоки. (см. рисунок 1.3). Также сайт имеет персонализированную ленту новостей, которая строится, опираясь на более ранние поисковые запросы пользователя в Google.

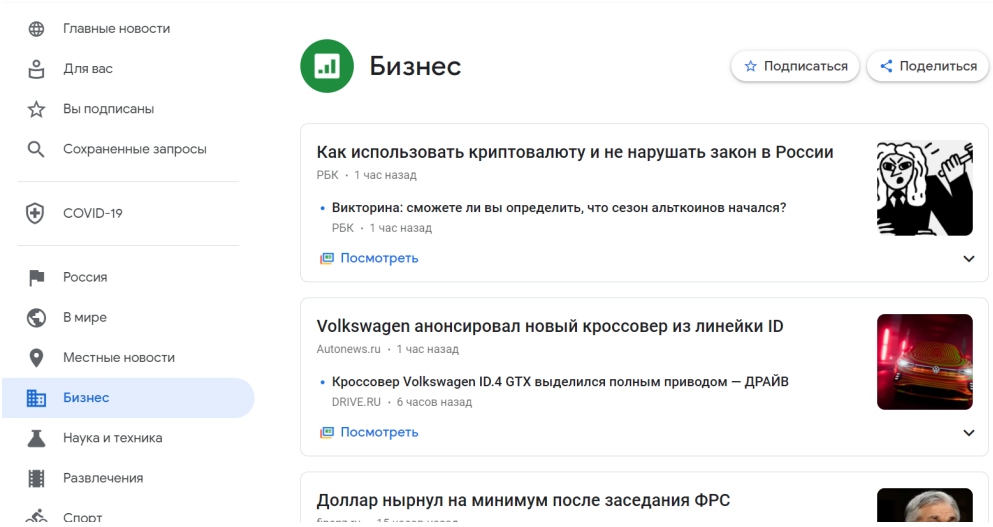

Рисунок 1.3 - Категория Бизнес на Google News.

Яндекс.Новости - самый популярный новостной агрегатор в РФ на данный момент. Сайт имеет иерархическую категоризацию статей. Также он определяет новости со схожей тематикой и близкие по времени написания, затем выводит пользователю одну, опираясь на внутренний рейтинг новостного ресурса. (см. рисунок 1.4)

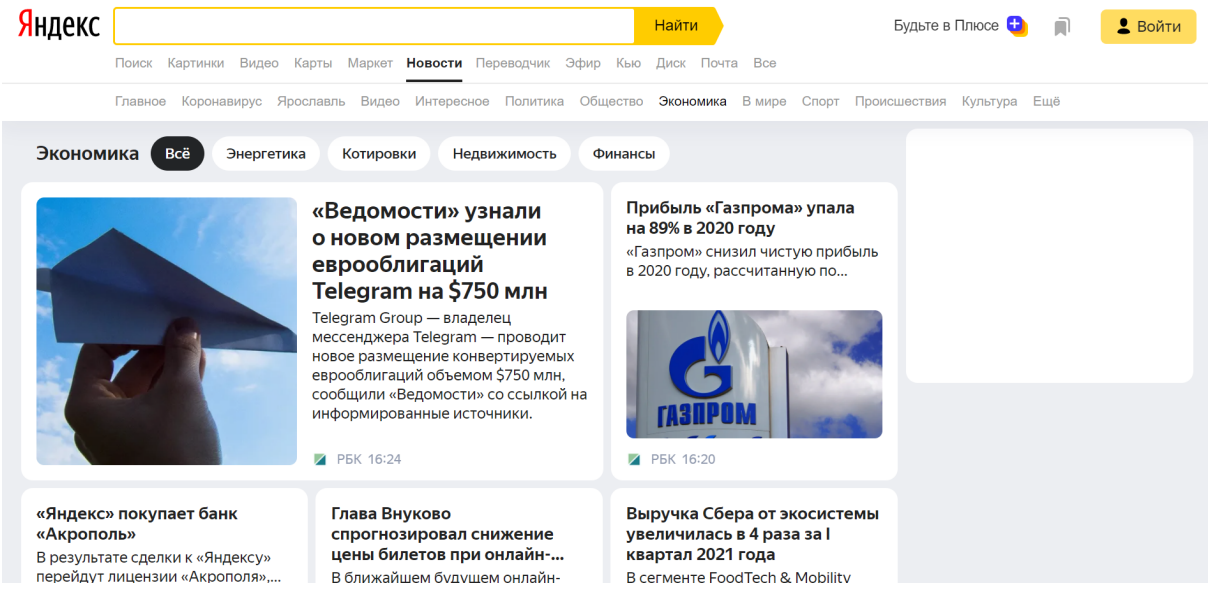

Рисунок 1.4 - Категория Экономика на Яндекс.Новости.

Оба из представленных агрегаторов являются большими и плотно интегрированными в свои инфраструктуры сайтами. Также они имеют строго ограниченный и заранее заданный набор категорий. Эти факторы лишают данные агрегаторы возможности их сторонней интеграции. Другим интернет-порталам приходится писать собственные новостные классификаторы. Зачастую они не имеют возможности проводить исследования по поиску оптимального решения этой проблемы.

В данной работе мы займемся рассмотрением популярных и классических методик классификации документов и применим их на корпусе новостных статей.

2 Классификация текста

Классификация текста - задача компьютерной лингвистики, заключающаяся в отнесении документа к одной из нескольких категорий на основании содержания документа [1]. Классификация текстов как правило применяется для разделения сайтов по тематическим каталогам, борьбы со спамом, распознавания эмоциональной окраски текстов, персонификации рекламы и т. д. [11] В данной работе рассматривается именно тематическая каталогизация.

Формализовано задачу классификации можно поставить следующим образом:

X - множество описаний объектов,

Y - конечное множество номеров (имён, меток) классов,

$X^m = \{(x_1, y_1), \ldots, (x_m, y_m)\}$ - конечная обучающая выборка,

y^* - целевая зависимость, отображение значения которой известны только на объектах обучающей выборки.

Требуется построить алгоритм $a: X \to Y$, способный классифицировать произвольный объект $x \in X$ [5].

В задаче агрегации новостных статей необходимо провести классификацию документов для определения тематики содержания. В общей сложности было выявлено и размечено 7 классов.

- общество,
- экономика,
- спорт,
- наука,
- технологии,
- новости культуры,

- традиции.

Для классификации текстов методами машинного обучения зачастую необходимо произвести предобработку текста, а затем извлечь признаковые составляющие текста или выполнить векторизацию [2].

2.1 Предобработка и векторизация текста

2.1.1 Общий подход к предобработке текста

Классификаторы, как правило, работают с объектами числового типа, точнее - с векторами. *Векторизация* - преобразование текста в числовой формат. Алгоритму векторизации подаётся на вход корпус текстов, а на выходе получаем векторы каждого документа [8]. *Документ* - это совокупность токенов, которые принадлежат одной смысловой единице. *Корпус* - это генеральная совокупность всех документов. При сравнении векторов можно делать выводы о тематической близости текстов [10].

Для очистки документа от возможных шумов и аномалий, которые могут помешать построению достоверного вектора, производится предобработка текста. Предобработка включает в себя: приведение слов к нижнему регистру, токенизацию, удаление стоп-слов, стемминг и лемматизацию [9].

Приведение слов к нижнему регистру необходимо для устранения помех, связанных с расстановкой одинаковых слов в начале и середине предложения, что меняет их регистр написания.

Токенизация - процесс разбиения текста на текстовые единицы, например, слова или предложения.

Удаление стоп-слов - извлечение слов с большой частотой встречаемости, которые не несут большой смысловой нагрузки.

Стемминг - нахождения основы слова.

Лемматизация - процесс приведения слова к нормальной (словарной) форме.

После предобработки документов необходимо выбрать алгоритм векторизации текста. В данной работе будут рассматриваться несколько векторизаторов, а именно: TF-IDF, Word2Vec, Doc2Vec, FastText, GloVe, Universal-Sentence-Encoder и Bert. Будут рассмотрены принципы их работы и произведено сравнение их влияния на точность классификаторов. Суть векторизации документов заключается в построении векторов для каждого текста. Близкие по смыслу с точки зрения модели документы (похожие по смыслу слова используются в сходных контекстах) будут близки по косинусной мере.

Косинусное сходство - это мера сходства между двумя векторами, которая используется для измерения косинуса угла между ними [8].

$$similarity(A, B) = cos(\theta) = \frac{A\,B}{||A||\,||B||} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}},$$

где A и B - векторы признаков,

$cos(\theta)$- косинусное сходство.

2.1.2 TF-IDF

TF-IDF - статистическая мера, используемая для оценки важности слова в контексте документа. Вес некоторого слова пропорционален частоте употребления этого слова в документе и обратно пропорционален частоте употребления слова во всех документах коллекции. Размер вектора в данной модели растет пропорционально размеру словаря, однако его можно ограничить, используя только самые частотные слова.

TF-IDF состоит из двух компонентов: Term Frequency (частотность слова в документе) и Inverse Document Frequency (инверсия частоты документа).

$$TF_{token_i} = \frac{n_i}{N_i} \qquad IDF_{token} = log\frac{p}{P}$$

где n_i - сколько раз встречается токен в i-ом документе,

N_i - общее количество токенов в i-ом документе,

p - количество документов, в которых встречается токен,

P - общее количеств документов.

В конечном счете, TF-IDF – это произведение TF на IDF:

$$TF\,IDF = TF \times IDF$$

2.1.3 Word2Vec

Word2Vec - общее название для совокупности моделей на основе искусственных нейронных сетей, предназначенных для получения векторных представлений слов на естественном языке. Архитектура Word2vec подразделяется на два вида – Skip-gram и Continuous Bag of Words (CBOW). Оба эти метода изучают веса, которые действуют как представления слов в векторе. В корпусе модель CBOW предсказывает текущее слово из окна окружающих контекстных слов, в то время как модель Skip-gram предсказывает окружающие контекстные слова по текущему слову.

Принцип работы Word2Vec заключается в нахождение связей между контекстами слов согласно предположению, что слова, находящиеся в схожих контекстах, имеют тенденцию значить похожие вещи, т.е. быть семантически близкими. Более формально задача стоит так: максимизация косинусной близости между векторами слов (скалярное произведение векторов), которые появляются рядом друг с другом, и минимизация косинусной близости между векторами слов, которые не появляются друг рядом с другом. Рядом друг с другом в данном случае значит в близких контекстах.

Word2vec строит вектор не для документа, а для отдельного слова, опираясь на контекст. С целью нахождения вектора цельного текста был выбран

подход суммирования векторов слов, с учетом весов их частоты встречаемости. Частоту встречаемости слов можно взять из TF-IDF модели.

$$DV = \frac{\sum_{i=1}^{n} w2v_i \times tfidf_i}{n}$$

где DV - итоговый вектор документа,

$w2v_i$ - векторное представление i-го слова в документе,

$tfidf_i$ - частота встречаемости i-го слова во всем корпусе,

n - количество слов в документе.

2.1.4 Doc2vec

Doc2vec - инструмент для представления документов в виде вектора и является обобщением Word2Vec метод. Doc2vec использует ту же логику, что и Word2Vec, но применяет ее к уровню документа. Каждый абзац и каждое слово отображается в уникальные векторы. Вектор абзаца и векторы слова усредняются или объединяются, чтобы предсказать следующее слово в контексте. Маркер абзаца можно представить как другое слово. Он действует как память, которая запоминает то, чего не хватает в текущем контексте или в теме абзаца.

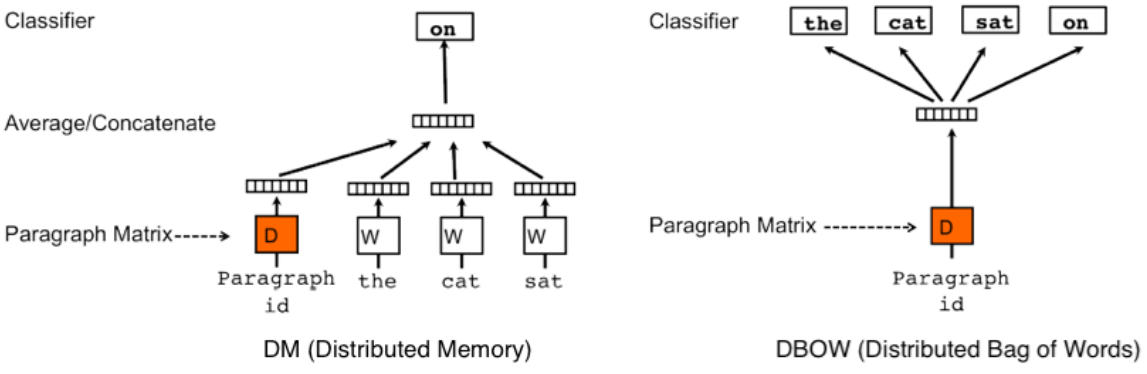

Рисунок 2.1 - Архитектуры DM и DBOW

На рисунке 2.1 изображены основные архитектуры Doc2Vec: DM и DBOW. DM - модель Doc2Vec, аналогичная модели CBOW в Word2Vec. Векторы абзаца получаются путем обучения нейронной сети задаче определения центрального слова на основе контекстных слов и контекстного абзаца. DBOW - модель Doc2Vec, аналогичная модели Skip-gram в Word2Vec. Векторы абзаца получаются путем обучения нейронной сети задаче прогнозирования распределения вероятности слов в абзаце по случайно выбранному слову из абзаца.

2.1.5 FastText

FastText - инструмент для представления как слов, так и документов в виде вектора. Для получения векторного представления слов применяются модели Skip-gram и CBOW. Также к основной модели была добавлена subword-модель. Subword-модель - это представление слова через цепочки символов (n-граммы) с n символов от начала до конца слова плюс само слово целиком. Например слово “замок” с n = 3 будет представлено n-граммами <за, зам, амо, мок, ок> и последовательностью <замок>. Таким образом, для модели есть разница между последовательностью <зам> в слове “зам” - и n-граммой <зам> из слова “замок”. Такой подход позволяет работать с теми словами, которые модель ранее не встречала. FastText одинково эффективно использует N-граммы символов и N-граммы слов. Признаки, полученные при помощи разбиения на n-граммы, имеют огромную размерность. Это может замедлить работу обучаемой на этих признаках модели. Для фиксирования размерности признаков применяется хэширование признаков. Признаки получают хэш-индексы, что помогает считывать их быстрее.

2.1.6 GloVe

GloVe - алгоритм машинного обучения без учителя для векторизации слов. Они работают с матрицей совместного вхождения - для всего корпуса и

подсчитывают, как часто одно слово появляется в контексте другого. GloVe минимизирует разницу между произведением векторов слов и логарифмом вероятности их совместного появления с помощью стохастического градиентного спуска.

GloVe строит вектор не для документа, а для отдельного слова. С целью нахождения вектора цельного текста был выбран подход суммирования векторов слов, с учетом весов их частоты встречаемости. Частоту встречаемости слов можно взять из TF-IDF модели.

$$DV = \frac{\sum_{i=1}^{n} glove_i \times tfidf_i}{n}$$

где DV - итоговый вектор документа,

$glove_i$ - векторное представление i-го слова в документе,

$tfidf_i$ - частота встречаемости i-го слова во всем корпусе,

n - количество слов в документе.

2.1.7 Universal-Sentence-Encoder

Universal-Sentence-Encoder (USE) кодирует текст в многомерные векторы, которые можно использовать для классификации текста, поиска семантического сходства, кластеризации и других задач обработки естественного языка.

Модель обучена и оптимизирована для документов длиной больше слова, таких как предложения, фразы или короткие абзацы. Она обучена работе с множеством источников данных с целью динамического приспособления к большому количеству задач понимания естественного языка. Входными данными является текст переменной длины, а на выходе - 512-мерный вектор.

USE обучается с помощью кодировщика Transformer. Сложность модели составляет $O(n^2)$ для документов длины n. Она строит вложения предложений с использованием подграфа кодирования архитектуры Transformer. Этот подграф

использует механизм Attention для вычисления контекстно-зависимых представлений слов в предложении, которые учитывают как порядок, так и идентичность всех других слов. Представления слов с учетом контекста преобразуются в вектор кодирования предложений фиксированной длины путем вычисления поэлементной суммы представлений в каждой позиции слова [12].

2.1.8 BERT

Bidirectional Encoder Representations from Transformers (BERT) - это основанный на Transformer нейронная сеть для предварительного обучения и обработки естественного языка (NLP), разработанный Google. Особенности BERT заключаются в том, что технология обучается на основе всего набора слов в документе. Она предназначена для предварительного обучения глубоких двунаправленных представлений из немаркированного текста путем совместной обработки левого и правого контекста на всех уровнях [3]. Ранее нейросети обучались на упорядоченной последовательности слов (слева направо или слева направо и справа налево).

BERT обеспечивает плотные векторные представления для естественного языка. Он кодирует информацию о взаимных употреблениях слов, словосочетаний и других языковых единиц по отношению друг к другу, то есть учитывает контекст. Если поменять порядок слов в предложении, то вектора на выходе будут другими [13].

2.2 Разметка и балансировка данных для обучения

Классификации является задачей машинного обучения с учителем. Для обучения модели требуется тренировочная выборка - размеченный и сбалансированный набор данных, содержащий экземпляры каждого класса. В данной работе используется датасет, собранный с трех новостных сайтов

(lenta.ru, РБК, РИА Новости) за 2020 год. Особенности датасета и детали его получения описаны ниже в разделе 3.1.

Разметка данных необходима для работы алгоритмов машинного обучения с учителем. Каждому тексту в тренировочной выборке ставится метка, которая присваивает его к одному из перечисленных классов. В моём датасете каждой новостной статье соответствует набор “тегов”, которые были проставлены вручную на самом сайте. На основании “тегов” велась разметка данных для обучения. Из 151841 новостной статьи были размечены 69955 (рисунок 2.2).

```
'Экономика': "economy",

'Шоу-бизнес': "entertainment",
'Новости культуры': "entertainment",
'Культура': "entertainment",

'Наука':"science",
```

Рисунок 2.2 - Разметка новостный статей по “тегам”

Также была произведена балансировка классов. Самый малочисленный класс насчитывает 4845 статей, а самый обширный - 20141. Балансировка производилась вниз, т.е. из каждого класса удалялись экземпляры до тех пор, пока их количество не останется равным 4845.

2.3 Методы классификации

В данной работе будут рассматриваться несколько методов классификации текста, а именно: LogisticRegression, SVM, Single-layer perceptron (SLP), Bert и Gpt-2. Первые три модели будут получать предобработанные и векторизированные документы, на них будет оцениваться влияние различных векторизаторов на точность классификации. Для Bert и

Gpt-2 будут использоваться готовые предобученные на собственных токенизаторах модели. Для них предобработка производиться не будет.

2.3.1 Логистическая регрессия

Логистическая регрессия (LogisticRegression) - метод построения линейного классификатора, позволяющий оценивать апостериорные вероятности принадлежности объектов классам. Логистическая регрессия применяется для прогнозирования вероятности возникновения некоторого события по значениям множества признаков. Основная идея логистической регрессии заключается в том, что пространство исходных значений может быть разделено линейной границей на две соответствующих классам области. Эта граница задается в зависимости от имеющихся исходных данных и обучающего алгоритма (рисунок 2.3).

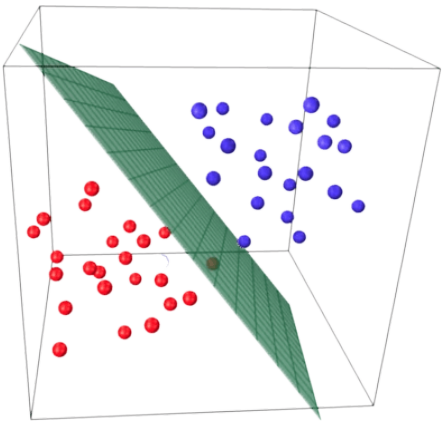

Рисунок 2.3 - Основная идея логистической регрессии

2.3.2 Метод опорных векторов

Метод опорных векторов (SVM) - один из наиболее популярных методов машинного обучения, который применяется для решения задач классификации и регрессии. Основная идея метода заключается в построении гиперплоскости, разделяющей объекты выборки оптимальным способом. Алгоритм работает в предположении, что чем больше расстояние между разделяющей

гиперплоскостью и объектами разделяемых классов, тем меньше будет средняя ошибка классификатора.

2.3.3 Однослойный перцептрон

Однослойный перцептрон (Single-layer perceptron) - нейронная сеть, в которой сигналы от входного слоя сразу подаются на выходной слой, который преобразует сигнал и сразу же выдает ответ (рисунок 2.4). Классический метод обучения перцептрона — это метод коррекции ошибки. Он представляет собой такой вид обучения с учителем, при котором вес связи не изменяется до тех пор, пока текущая реакция перцептрона остается правильной. При появлении неправильной реакции вес изменяется на единицу, а знак (+/-) определяется противоположным от знака ошибки. Входной слой перцептрона равен размерности подаваемого вектора, а выходной - числу классов.

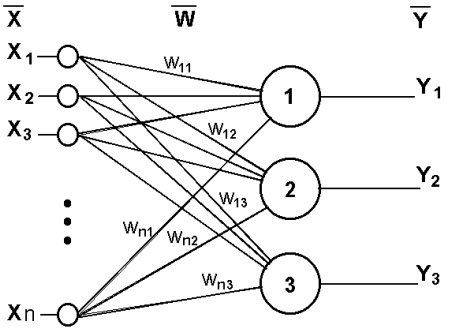

Рисунок 2.4 - Архитектура однослойного перцептрона

2.3.4 BERT

BERT предназначена для предварительного обучения глубоких двунаправленных представлений из немаркированного текста. Она позволяет языковой модели изучать контекст слова на основе всех окружающих его слов. В результате предварительно обученная модель BERT может быть настроена всего с одним дополнительным выходным слоем для создания современных моделей для широкого круга задач, таких как ответы на вопросы и логический

вывод, без существенных модификаций архитектуры для конкретных задач [7]. Для решения задачи классификации используется модифицированная модель BERT с одним дополнительным слоем классификатора.

2.3.5 GPT-2

Generative Pre-trained Transformer 2 (GPT-2) - это искусственный интеллект с открытым исходным кодом, созданный OpenAI в феврале 2019 года. GPT-2 применяется в переводите текста, отвечаетах на вопросы и генерции текста. Особенность GPT-2 заключаются в том, что он предсказывает следующее слово в текста, опираясь на предыдущий контекст. Данная модель не была специально обучена для выполнения какой-либо конкретной задачи, и ее способность выполнять их является расширением ее общей способности точно синтезировать следующий элемент в произвольной последовательности. Архитектура GPT-2 реализует глубокую нейронную сеть, которая использует механизм внимания (attention) вместо предыдущих архитектур, основанных на регрессии и свертке. Attention позволяют модели выборочно фокусироваться на сегментах входного текста, которые, по ее прогнозам, наиболее актуальны.

В GPT-2 последний токен входной последовательности используется для прогнозирования следующего токена, который должен следовать за входом. Это означает, что последний маркер входной последовательности содержит всю информацию, необходимую для прогнозирования. Имея это в виду, мы можем использовать эту информацию для прогнозирования в задаче классификации вместо задачи генерации.

3 Процесс разработки

В данной главе описываются все этапы практического выполнения работы: сбор новостных статей, предобработка, векторизация и классификация.

Для разработки был выбран язык программирования Python, так как на нем реализовано множество алгоритмов предобработки текста, машинного обучения и нейросетевые архитектуры. Разработка велась в Google Colab - это бесплатный облачный сервис на основе Jupyter Notebook. Google Colab предоставляет всё необходимое для машинного обучения прямо в браузере, а также дает доступ к быстрой GPU.

3.1 Создание датасета

Поиск и скачивание новостных статей велось с помощью краулинга трёх новостных сайтом: Lenta.ru, РБК, РИА Новости. Данные ресурсы были выбраны из следующих соображений: они предоставляют открытый доступ к текстовым ресурсам на некоммерческой основе, имеют большую и регулярно обновляемую базу новостных статей, ведут ручное тематическое и “теговое” индексирование новостей.

Краулинг – это процесс автоматического скачивания данных с веб-страницы, извлечение гиперссылок, которые на ней есть, и переход по ним. Данные хранятся в базе данных, что позволяет легко запустить поиск по ним.

Для скачивания текстовой информации с веб-страницы использовалась библиотека `requests`, а для быстрой навигации и поиску необходимой информации применялась `BeautifulSoup`. Для каждой новостной статьи извлекались заголовок, основной текст, дата и время публикации, а также размеченные “теги”. Также для каждой новости ставилась метка, с какого новостного сайта она была собрана: lenta, rbc, ria.

Для дальнейшего обучения классификаторов производилась разметка данных. Были проанализированы “теги” с каждого сайта и выстроена сеть соответствия статьи к одному из классов, или её несоответствие ни одному из них. Например, тег "Интернет и СМИ" соответствует классу “технологии”, а “Шоу-бизнес” - "новости культуры".

Все полученные статьи и метки сохранялись в базу данных. Для этого применялась библиотека pandas. В общей сложности было собрано 151841 новостная статья, из которых 69955 - размечены для дальнейшего обучения классификаторов.

База данных содержит столбцы: id (индекс), tags (теги), dt (дата/время), main_text (основной текст), wedsite (метка веб-сайта), label (метка класса) и main_tag (расшифровка класса) (рисунок 3.1).

	tags	dt	main_text	website	label	main_tag
0	[Экономика]	2020-01-01 00:07:00	В России повысили зарплаты. В России повышен м...	lenta	0	economy
1	[Бывший СССР]	2020-01-01 00:27:00	Партия Порошенко потребовала запретить поставк...	lenta	-1	
2	[Новый год—2020, Парковка, Москва, Общество]	2020-01-01 00:33:00	Парковка в Москве в новогодние праздники будет...	ria	-1	
3	[Экономика]	2020-01-01 00:35:00	В России подорожали алкогольные напитки и сига...	lenta	0	economy
4	[Новый год—2020, Шоу-бизнес, Анастасия Заворот...	2020-01-01 00:38:00	Анастасия Заворотнюк поблагодарила россиян за ...	ria	1	entertainment

Рисунок 3.1 - База данных новостных статей

3.2 Предобработка текста

Над текстами из графы main_text производилась предобработка. Все слова в документе были переведены в нижний регистр. Для программной работы с естественными языками на Python была написана библиотер `nltk`. С её помощью производились токенизация текста на предложения, а затем - на отдельные токены, также удаление стоп-слов и пунктуации. Для лемматизации текста использовалась библиотека `pymorphy2`, которая поддерживает русский язык. Она каждое слова приводит к его нормальной словарной форме. Хоть

лемматизация в среднем работает в три раза дольше стемминга, она позволяет достичь лучшие результаты для русских текстов.

В итоге выполнения программы текст был подготовлен к дальнейшей векторизации (рисунок 3.2).

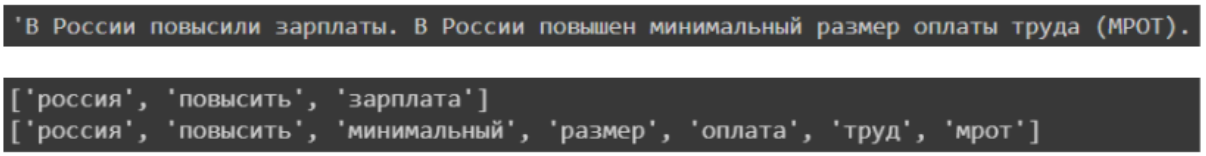

Рисунок 3.2 - Пример предобработки текста

3.3 Векторизация текста

Векторизация документов производилась семью методами: TF-IDF, Word2Vec, Doc2Vec, FastText, GloVe, USE и Bert. - для каждого из которых необходим свой подход к представлению векторизируемого текста.

3.3.1 Алгоритм TF-IDF векторизации был реализован в библиотеке `Scikit-learn (sklearn)`. `sklearn` - один из наиболее широко используемых пакетов Python для Data Science и Machine Learning. Для TF-IDF применялся TfidfVectorizer, с подобранными параметрами: ngram_range=(1,2), min_df = 5, max_features = 500000 - были использованы n-граммы длины 1 и 2; чтобы токен было рассмотрен в модели, необходимо, чтобы он встречался хотя бы 5 раз во всём корпусе; в конечном итоге оставалось только 500000 самый частотных токенов, что также является итоговым размером вектора. Вектор каждого документа представляет из себя разреженную матрицу, т.е. практически весь заполнен нулями, и только на местах, соответствующих определенному токену, имеет ненулевые положительные нормализованные значения. В Python есть отдельный тип данных sparse_matrix, который позволяет экономить память компьютера на хранение векторов большой размерности.

3.3.2 `Gensim` - это библиотека Python для моделирования тем, индексации документов и поиска сходства с большими корпусами. Данный пакет реализует Word2Vec векторизатор отдельных токенов. В данной работе была построена модель с параметрами: длина вектора равна 300; обучение проходит в 8 потоков и 20 эпох; чтобы токен было рассмотрен в модели, необходимо, чтобы он встречался хотя бы 5 раз во всём корпусе; размер окна - 10; использовалась архитектура Skip-Gram с уровнем негативного сэмплирования 0.0001 (рисунок 3.3).

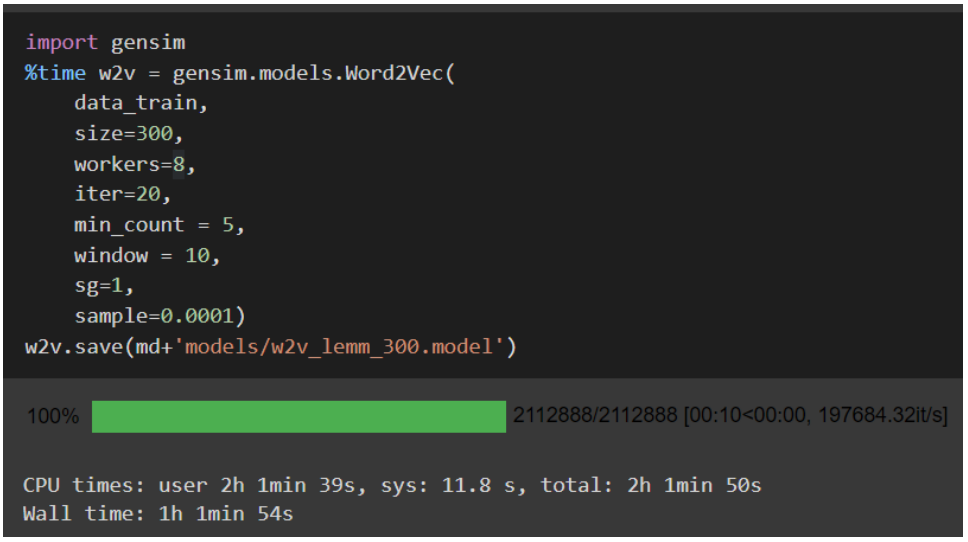

Рисунок 3.3 - Обучение Word2Vec

Так как Word2Vec способен преобразовать лишь отдельные токены, был использован подход суммирования векторов слов с учетом их частотности, ранее подсчитанной в TF-IDF. Необходимо, чтобы слово из документа содержалось в словарях как Word2Vec, так и TF-IDF моделей, иначе оно является низкочастотным и, скорее всего, аномальным. Векторы слов умножаются на коэффициент частоты, затем суммируются и делятся на общую длину документа (с учетом только найденных слов).

3.3.3 В `gensim` также реализован Doc2Vec векторизатор. Для обучения модели необходимо каждому документу в корпусе указать уникальный

идентификатор, а затем преобразовать полученную пару в TaggedDocument тип, специально созданный в `gensim` для Doc2Vec. В данной работе была построена модель с параметрами: длина вектора равна 300; обучение проходит в 8 потоков; чтобы токен было рассмотрен в модели, необходимо, чтобы он встречался хотя бы 5 раз во всём корпусе; использовалась архитектура DBOW с уровнем негативного сэмплирования 0.0001.

3.3.4 Для FastText модели существует отдельная библиотека на Python, однако в `gensim` она также реализована, поэтому будет использоваться она. FastText зачастую применяется и для классификации, но здесь он будет рассмотрен только как векторизатор. В данной работе была построена модель с параметрами: длина вектора равна 300; обучение проходит в 8 потоков и 20 эпох; размер окна - 10; использовалась архитектура Skip-Gram с уровнем негативного сэмплирования 0.0001 (рисунок 3.4).

```
import gensim
%time ft = gensim.models.FastText(
    sentences=data_train,
    size = 300,
    sg = 1,
    sample = 0.0001,
    workers=8,
    iter = 20,
    window = 10)
ft.save(md+'models/ft_lemm_300.model')
```

```
CPU times: user 5h 30min 3s, sys: 24.2 s, total: 5h 30min 28s
Wall time: 2h 47min 38s
```

Рисунок 3.3 - Обучение FastText

3.3.5 Библиотека `Navec` предоставляет предобученную модель GloVe для русского языка. Модель генерирует вектора размерности 300, размер словаря составляет 500000 слов, а обучена она была на 12 миллиардах токенов.

Так как GloVe способен преобразовать лишь отдельные токены, был использован подход суммирования векторов слов с учетом их частотности,

ранее подсчитанной в TF-IDF. Необходимо, чтобы слово из документа содержалось в словарях как GloVe, так и TF-IDF моделей, иначе оно является низкочастотным и, скорее всего, аномальным. Векторы слов умножаются на коэффициент частоты, затем суммируются и делятся на общую длину документа (с учетом только найденных слов).

3.3.6 Для кодирования документов с помощью Universal Sentence Encoder использовалась предобученная модель universal-sentence-encoder-multilingual-large. Она поддерживает обработку 16 языков, в том числе и русский. Из-за квадратичной сложности, данная модель склонна к избыточной загрузке памяти устройства во время кодировки текстов большой длины. Данные документы принудительно сокращались. На вход модели подавался текст, на выходе получались вектора с размерностью 512.

```
import tensorflow_hub as hub

embed = hub.load(
    "https://tfhub.dev/google/universal-sentence-encoder-multilingual-large/3" )

embedding = embed(
    ["съешь ещё этих мягких французских булок, да выпей чаю"])
```

Рисунок 3.4 - Пример работы USE

3.3.7 Для кодирования документов с помощью Bert использовалась предобученная модель bert_multi_cased_L-12_H-768_A-12. Она использует 12 скрытых слоев (блоков Transformer), а размером выходного вектора равен 768.

Данная модель ожидает словарь с тремя int32 тензорами в качестве входных данных: input_word_ids, input_mask, и input_type_ids. Отдельный препроцессор bert_multi_cased_preprocess преобразует ввод простого текста в необходимый формат. В этой модели используется словарь для нескольких языков (в том числе и русский), извлеченных из Multilingual Wikipedia (такой же, как в моделях авторов BERT). Ввод текста был нормализован «по регистру», что означает, что сохранено различие между нижним и верхним регистром, а

также маркеры акцента. Эта модель не имеет обучаемых параметров используется вне цикла обучения.

3.4 Классификация текста

Перед тем, как начать работать с алгоритмами классификации были произведены разбиение размеченных данных на обучающую и тестовую выборки и балансировка классов.

69955 размеченных документов были разделены по категориям, случайным образом перемешаны и разбиты на экземпляры тренировочного и тестового множеств в соотношении 0.8 к 0.2. Затем производилась балансировка классов вниз с коэффициентом 1, т.е. находилась категория, в котором было меньше всего элементов, а во всех остальных удалялись элементы до тех пор, пока размеры классов не будут совпадать. Таким образом все множества являются полностью сбалансированы. В тренировочной наборе минимальный размер класса равняется 3876 элементам, в тестовой - 969. Итоговый размер выборок составляет 27132 и 6783 экземпляра соответственно. После склейки классов в единое множество они опять же были перемешаны.

Перед тем, как начать обучение классификаторов, необходимо произвести очистку элементов выборок. Каждый элемент представлял из себя кортеж, содержащий теги, дату/время, основной текст, метку веб-сайта, метку класса и расшифровку класса. Для обучения потребуются только основной текст и метка класса. Вся выборка разделяется на два массива X_text и labels соответственно. Первый - на чём обучается модель, второй - какой результат она должна получить.

Классификация документов производилась пятью методами: LogisticRegression, SVM, Single-layer perceptron (SLP), Bert и Gpt-2.

3.4.1 LogisticRegression и SVM реализованы в библиотеке `Scikit-learn (sklearn)`. Обе модели поддерживают мультиклассовую классификацию. В данной работе использовалась реализация метода классификации опорных векторов для случая линейного ядра LinearSVC с коэффициентом регуляризации C=0.6. Обе модели работают с числовыми параметрами, поэтому необходимо произвести векторизацию документов. Все экземпляры тренировочной и тестовой выборок проходили этапы предобработки и векторизации текста одним из ранее описанных методом. LogisticRegression и LinearSVC способны обрабатывать как числовые массивы, так и разреженные матрицы, поэтому дополнительная корректировка полученных векторов не требуется.

3.4.2 Single-layer perceptron реализован с помощью `keras`, высокоуровневого API для TensorFlow. Keras - это API, которое предлагает последовательные и простые API-интерфейсы, сводит к минимуму количество действий пользователя, необходимых для распространенных случаев использования, и предоставляет четкие и действенные сообщения об ошибках. TensorFlow - это комплексная платформа с открытым исходным кодом для машинного обучения. Он имеет всеобъемлющую гибкую экосистему инструментов, библиотек и ресурсов сообщества.

Для однослойного перцептрона также производилась предобработка и векторизация текста. Single-layer perceptron является последовательной Sequential моделью с двумя слоями - входным и выходным (рисунок 3.4). Величина входного слоя равна размерности подаваемого вектора (для большинства моделей - 300, для TF-IDF - 100000), а размерность выходного - числу классов (т.е. 7). Однако из-за относительно большого размера подаваемых данных необходимо регулировать затраты памяти на их обработку. Числа во входном векторе являются переменными с плавающей точкой, которые были

ограничены 16 битами (float16), что незначительно уменьшает точность работы классификатора. Обучение модели проводилось в 15 эпох.

```
from keras.models import Sequential
from keras import layers

model = Sequential()
model.add(layers.Dense(32, input_dim=input_dim, activation='relu'))
model.add(layers.Dropout(0.2))
model.add(layers.Dense(len(main_tags_list), activation='softmax'))

model.compile(loss='categorical_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])

model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 32)                9632
_________________________________________________________________
dropout_1 (Dropout)          (None, 32)                0
_________________________________________________________________
dense_2 (Dense)              (None, 7)                 231
=================================================================
Total params: 9,863
Trainable params: 9,863
Non-trainable params: 0
_________________________________________________________________
```

Рисунок 3.5 - Архитектура SLP на Python

3.4.3 В библиотеке `transformers` есть TFBertForSequenceClassification - архитектурная надстройка над Bert с возможностью мультиклассовой классификации. С huggingface.co была загружена bert-base-multilingual-uncased модель, предобученная на 102 языках, с 12-слоями и 110M параметрами. Так как она была обучена на необработанных данных, на вход классификатору также будут подаваться неочищенные документы. У Bert есть свой токенизатор текста, который необходимо применять в данной модели. Он разбивает предложения и слова на заранее сгенерированные токены, и заменяет их на порядковые номера в словаре. Также в начале ставится число “101”, что соответствует ключевому слову “[CLS]” - данный документ будет участвовать в классификации. В конце же предложения ставится число “102” - “[SEP]”, конец предложения. Также у Bert есть

ограничение на величину входного слоя, он не должен превышать 512 элементов. В связи с этим был подобран подходящий по затратам памяти размер батча, равный 8. Обучение модели проводилось в 5 эпох.

3.4.4 GPT2ForSequenceClassification - архитектурная надстройка над Gpt-2 с возможностью мультиклассовой классификации от `transformers`. С huggingface.co была загружена rugpt3medium_based_on_gpt2 модель с 12-слоями и 355M параметрами, предобученная на русских текстах. Так как она была обучена на необработанных данных, на вход классификатору также будут подаваться неочищенные документы. У Gpt-2 есть свой токенизатор текста, который необходимо применять в данной модели. Он разбивает предложения и слова на заранее сгенерированные токены, и заменяет их на порядковые номера в словаре. Для данной модели размер батча был взят равный 8, обучалась она в 4 эпохи.

Полный код приложения доступен по следующей ссылке:
https://github.com/andreytsimbalov/News_Classification_and_Vectorization

4 Результаты исследования

После построения всех классификаторов и векторизаторов проводились обучение моделей, оценка точности, времени работы, а также сравнительный анализ полученных результатов. Кроме того была выведена корреляция между точностью классификации и выбором векторизатора.

Время обучения каждого классификатора с различными векторными моделями отличалось. Для оценки скорости работы оценивались пары векторизатор-классификатор, а также среднее время, которое затрачивает классификатор на обучение [6].

Для оценки точности была использована F-мера, представляющая собой гармоническое среднее между точностью (*precision*) и полнотой (*recall*).

Точность системы в пределах класса – это доля документов действительно принадлежащих данному классу относительно всех документов которые система отнесла к этому классу. Полнота системы – это доля найденных классификатором документов принадлежащих классу относительно всех документов этого класса в тестовой выборке.

$$F = 2\,\frac{Precision \times Recall}{Precision + Recall}$$

$$Precision = \frac{TP}{TP + FP}$$

$$Recall = \frac{TP}{TP + FN}$$

где TP — истинно-положительное решение;

TN — истинно-отрицательное решение;

FP — ложно-положительное решение;

FN — ложно-отрицательное решение.

В ходе исследования проведено 23 различных эксперимента. В таблице 4.1 представлены результаты: точность классификации и время работы в секундах.

vectorizer	classifier	f-score	time_work
w2v+tfidf	LinearSVC	0,885776	51,5
	LogisticRegression	0,881756	8,4
	SLP	0,890656	31,0
d2v	LinearSVC	0,342139	156,0
	LogisticRegression	0,330108	8,5
	SLP	0,315527	30,7
fasttext	LinearSVC	0,876809	6,2
	LogisticRegression	0,874824	8,7
	SLP	0,878888	30,7
tfidf	LinearSVC	0,922397	2,5
	LogisticRegression	0,912016	65,0
	SLP	0,923895	67,0
glove+tfidf	LinearSVC	0,839742	64,0
	LogisticRegression	0,836995	8,1
	SLP	0,853434	31,0
use	LinearSVC	0,874475	4,9
	LogisticRegression	0,866728	13,9
	SLP	0,876990	31,4
bert_multi_cased	LinearSVC	0,833720	165,0
	LogisticRegression	0,778309	17,5
	SLP	0,789664	34,8
	Bert_base	0,926333	18355,0
	Gpt-2	0,923658	23983,0

Таблица 4.1 Результаты классификации

Лучшие показатели точности были достигнуты на Bert_base модели - 92.6333 %. Самое лучшее соотношение скорости обучения к точности были получены на паре TF-IDF + LinearSVC.

Также была выведены среднее значение точности для каждого векторизатора (Таблица 4.2). Данное значение бралось как среднее арифметическое всех показателей. Наилучшие результаты классификации были получены у методов, в которых применялся TF-IDF векторизатор.

Word2Vec + TF-IDF	0.886063
Doc2Vec	0.329258
FastText	0.876840
TF-IDF	0.919436
GloVe + TF-IDF	0.843390
USE	0.872731
Bert	0.800564

Таблица 4.2 Влияние векторизаторов на точность классификации

Выводы

В данном исследовании были решены следующие задачи:

1. собран датасет новостных статей;
2. произведена предобработка текстов коллекции;
3. выбраны методы векторизации и классификации, применяемые в исследовании; выбранные методы реализованы средствами языка Python;
4. проведена векторизация документов;
5. описаны классы; при помощи тематической разметки выполнено разбиение данных по выбранным классам;
6. произведена классификация данных методами обучения с учителем;
7. качество классификации было оценено при помощи F-меры;
8. выполнен сравнительный анализ полученных результатов.

Заключение

В данной работе были исследованы и применены на практике ряд методов по предобработке, векторизации и классифкации текстовых документов. В ходе сравнения алгоритмов векторизации был выявлен один - TF-IDF, при котором достигались высокие результаты при любом из представленных методов классификации. Был найден самый оптимальный классификатор для задачи агрегации новостных статей - Bert_base. Однако классические методы машинного обучения также показали высокие результаты.

Данное исследование имеет перспективы в дальнейшей работе. Существует множество методов машинного обучения, нейросетевых архитектур и векторизаторов, которые можно рассмотреть в данной задаче, а также каждый год появляются новые, более инновационные и прорывные решения.

Список литературы

[1] Bird, Steven, Ewan Klein, Edward Loper. Natural Language Processing with Python - O'Reilly Media Inc, 2009. - 502c.

[2] Kamran Kowsari, Kiana Jafari Meimandi, Mojtaba Heidarysafa, Sanjana Mendu, Laura Barnes, Donald Brown. Text Classification Algorithms: A Survey - 2019

[3] Chi Sun, Xipeng Qiu, Yige Xu, Xuanjing Huang. How to Fine-Tune BERT for Text Classification? - 2020

[4] Г Кристофер Д. Маннинг, Прабхакар Рагхаван, Хайнрих Шютце. Введение в информационный поиск. : Пер. с англ. – М. : ООО “И.Д. Вильямс”, 2014 – 528 с

[5] Айвазян С. А., Бухштабер В. М., Енюков И. С., Мешалкин Л. Д. Прикладная статистика: классификация и снижение размерности. — М.: Финансы и статистика, 1989.

[6] Sousuke Amasaki, Pattara Leelaprute. The Effects of Vectorization Methods on Non-Functional Requirements Classification - 2018

[7] Margarita Bugueno, Marcelo Mendoza. Learning to combine classifiers outputs with the transformer for text classification - 2020

[8] Jurafsky, Daniel; H. James, Martin. Vector Semantics and Embeddings - 2000

[9] V. Srividhya, R. Anitha. Evaluating Preprocessing Techniques in Text Categorization - 2010

[10] С.В. Корелов, А.М. Петров, Л.Ю. Ротков, А.А. Горбунов. Предобработка текстов электронных писем в задаче обнаружения спама - 2020

[11] Малахов Д. П. Методы автоматической рубрикации текстовых документов предметной области // Научный семинар Института системного программирования РАН, 2015

[12] Daniel Cera, Yinfei Yanga, Sheng-yi Konga, Nan Huaa, Nicole Limtiacob, Rhomni St. Johna, Noah Constanta, Mario Guajardo-Cespedes, Steve Yuanc, Chris Tara, Yun-Hsuan Sunga, Brian Stropea, Ray Kurzweila. Universal Sentence Encoder - 2018

[13] Jacob Devlin, Ming-Wei Chang, Kenton Lee, Kristina Toutanova. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding - 2019