Федеральное государственное бюджетное образовательное учреждение высшего образования
«Национальный исследовательский университет «МЭИ»

Институт: ИВТИ Кафедра: ВМСС

Направление подготовки/специальность: 09.03.01 Информатика и вычислительная техника

ОТЧЕТ по практике

Наименование практики: Производственная практика: преддипломная практика

СТУДЕНТ

_____ / Челышев Э.А.
(подпись) (*Фамилия и инициалы*)

Группа А-08-17
(номер учебной группы)

ПРОМЕЖУТОЧНАЯ АТТЕСТАЦИЯ ПО ПРАКТИКЕ

_____
(отлично, хорошо, удовлетворительно, неудовлетворительно, зачтено, не зачтено)

_____ / _____ /
(подпись) (*Фамилия и инициалы члена комиссии*)

_____ / _____ /
(подпись) (*Фамилия и инициалы члена комиссии*)

Москва, 2021

ОГЛАВЛЕНИЕ

1. ПОСТАНОВКА ЗАДАЧИ

В последние десятилетия наблюдается быстрый рост объема производимых и накопляемых человечеством данных. Так, например, общемировой объем данных в 2018 году составил 33 зеттабайтов, а прогнозируемый общемировой объем данных к 2025 году составит уже 175 зеттабайтов [1], то есть вырастет более чем в пять раз.

Безусловно, с ростом накопленных данных человеку становится все сложнее ориентироваться в них, все более возрастает потребность в автоматизированной обработке информации. Именно этот факт является одной из ключевых причин роста популярности анализа данных и машинного обучения, позволяющих автоматически обрабатывать большие объемы данных и извлекать из них практическую пользу и коммерческую прибыль. Одной из отраслей машинного обучения является машинная обработка естественного языка.

Машинная обработка естественного языка (англ. Natural Language Processing, сокр. NLP) - это область машинного обучения, позволяющая компьютеру анализировать и потенциально генерировать тексты на естественном языке. Обработка естественного языка используется во многих сферах: машинный перевод, поиск информации, определение настроения и тональности, распознавании речи, прогнозирование и др. [2].

Весьма популярными становятся сейчас новостные агрегаторы, рубрикаторы научных статей и прочие решения по автоматизированной обработке информации, которые в своей работе используют автоматическую рубрикацию текстов на естественном языке. Под **рубрикацией** (рубрицированием) подразумевается распределение некоторой информации по тематическим разделам (рубрикам). С точки зрения машинного обучения

рубрикация является задачей классификации на несколько непересекающихся классов [2, 4]

В данной работе мною с использованием машинного обучения решена задача рубрикации русскоязычных новостных статей на 9 рубрик: Дом, Интернет и СМИ, Культура, Наука и техника, Политика, Путешествия, Силовые структуры, Спорт, Экономика и бизнес. В качестве набора данных для решения задачи был выбран корпус статей новостного издания lenta.ru, который доступен в [5].

По результатам преддипломной практики были разработаны: программный модуль, содержащий функции подготовки текстов на русском языке для подачи их на вход рубрицирующей модели машинного обучения, с использованием данного модуля были подготовлены данные для построения моделей классификации, а также был проведен ряд экспериментов по построения рубрицирующих моделей классификации и подбору значений гиперпараметров, дающих наилучшие показатели точности.

Для реализации данной задачи мною был выбран язык программирования высокого уровня **Python** [6]. В качестве среды разработки был выбран **Jupyter Notebook**, а именно его облачная реализация **Google Colaboratory**.

2. РАЗРАБОТКА ПРОГРАММНЫХ СРЕДСТВ

2.1. РАЗРАБОТКА ПРОГРАММНОГО МОДУЛЯ

Одним из важнейших этапов решения задач машинной обработки естественного языка является подготовка данных. Подготовка данных в рассматриваемой мной задаче включает в себя как общие для всех задач машинного обучения этапы (анализ набора данных, удаление выбросов и пропусков) [7], которые были реализованы мною отдельно, так и специфичные для машинной обработки естественного языка этапы: **удаление нерелевантных символов и приведение к общему регистру, токенизация, удаление стоп-слов, приведение к начальной форме и векторизация** [2, 8], для осуществления которых был разработан программный модуль на языке Python **prepare.py**.

Модуль содержит ряд функций, осуществляющих вышеназванные этапы подготовки, а также функцию **prepare_text**, включающую в себя все этапы подготовки и осуществляющую их путем вызова прочих функций данного модуля. Таблица 1 содержит спецификации всех входящих в данный программный модуль функций.

Таблица 1. Спецификации функций программного модуля prepare.py

Имя функции	**Назначение**	**Входные данные**	**Выходные данные**
load_model	Данная функция загружает предобученную модель векторизации FastText, расположенную по пути **path**	строка пути к модели векторизации: **path**	нет
preprocess_text	Данная функция производит приведение текста к одному регистру и удаляет из него нерелевантные символы	строка, содержащая текст, подлежащий обработке: **text**	строка, содержащая обработанный текст: **preprocessed_text**

Продолжение таблицы 1

preprocess_text	Данная функция производит приведение текста к одному регистру и удаляет из него нерелевантные символы	строка, содержащая текст, подлежащий обработке: **text**	строка, содержащая обработанный текст: **preprocessed_text**
lemmatize	Данная функция осуществляет лемматизацию токенов	список токенов: **text_list**	список лемматизированных токенов: **result**
delete_stop_words	Данная функция удаляет из списка токенов стоп-слова	список токенов: **text_list**	список токенов с удаленными стоп-словами: **result**
build_article_vector	Данная функция формирует вектор по списку токенов	список токенов: **t**	массив NumPy: **res**
prepare_text	Данная функция осуществляет все этапы подготовки данных.	строка текста: **text**	массив NumPy: **vector**

Первый этап подготовки, а именно **удаление нерелевантных символов и приведение символов к общему регистру**, позволяет исключить из текста несущественную или даже вредную для обучения модели машинного обучения информацию. Его производит функция **preprocess_text**. Данная функция осуществляет замену всех символов входной строки на строчные, а также заменяет символ «ё» на «е». Далее с использованием регулярных выражений удаляются присутствующие в тексте URL-ссылки, а также все небуквенные символы, исключая пробелы. Для работы с регулярными выражениями необходимо подключить стандартную библиотеку языка Python **re**.

Токенизация – процесс разбиения текста на токены, то есть текстовые единицы (символ, слово, предложение и т.д.). Как правило, токены отделены друг от друга разделительными символами. В данной работе токенизация осуществляется с использованием встроенного метода **word_tokenize** стандартной библиотеки **NLTK** [9].

Как известно, русский язык обладает богатой морфологической структурой. Слова могу иметь различные падежные и временные формы. Это придает текстам на естественном языке связность, однако, мешает машинной обработке русскоязычных текстов, так как одному и тому же значению соответствуют сразу несколько форм одного слова, которые в дальнейшем при машинной обработке будут восприниматься как различные токены [10].

Этот факт приводит к необходимости **приведения слов к их начальной**, то есть словарной, **форме**. Для решения задачи приведения к начальной форме в данной работе реализована функция **lemmatize**, которая, получая на вход список токенов, приводит их к начальной форме с использованием русскоязычного морфологического анализатора, реализованного в библиотеке **pymorphy2**. Такой подход к решению задачи приведения к начальной форме носит название лемматизация. Подробнее ознакомиться с данной библиотекой и ее возможностями можно в [11] и [12].

Стоп-слова – это часто встречающиеся в текстах слова, которые играют большую роль в обеспечении связности предложения, однако при машинной обработке естественного языка являются шумом и мешают обучению модели, так как не несут практически никакой информации. К ним можно отнести частицы, предлоги, союзы, местоимения и прочие слова-связки. Пакет **NLTK** имеет в своем составе список стоп-слов для русского языка, насчитывающий 151 слово.

Для решения задачи удаления стоп-слов в программном модуле prepare.py реализована функция **delete_stop_words**. Получая на вход список токенов, она возвращает список, очищенный от стоп-слов.

Векторизация – процесс, в результате которого каждому токену ставится в соответствие его векторное представление, то есть вектор v некоторого n-мерного векторного пространства $\mathbb{R}^n$. Для получения векторного представления слов

используются различные методы векторизации, которые могут как сохранять семантическую, то есть смысловую, близость слов, так и игнорировать ее. Модели, сохраняющие семантическую близость, подразделяются на статические модели векторизации и динамические (контекстуализированные) модели векторизации. Последние называются так же языковыми моделями [13]. Модели, сохраняющие семантическую близость, как правило, строятся на основе искусственных нейронных сетей. Они основываются на идее дистрибутивной семантики: слова, встречающиеся в аналогичных контекстах и имеющие близкую частоту употребления, как правило, семантически (т.е. по смыслу) близки. Подробного анализа существующих моделей векторизации в этой работе не привожу, подробнее с ними можно ознакомиться в [13], [14] и [15], остановлюсь только на одной из них – **FastText**.

Модель векторизации **Fasttext**, разработанная компанией **Facebook**, является статической моделью векторизации, сохраняющей семантическую близость. Показывая результаты, схожие с результатами аналогичных моделей векторизации, Fasttext имеет более высокую скорость обучения. Другим достоинством данного метода является то, что он учитывает контекстную близость слов, что значительно увеличивает информативность векторных представлений слов. На рисунке 1 видно, что модель векторизации способна определять семантически (т.е. по смыслу) близкие слова. Именно по этим причинам для выполнения работы мною была выбрана эта модель векторизации.

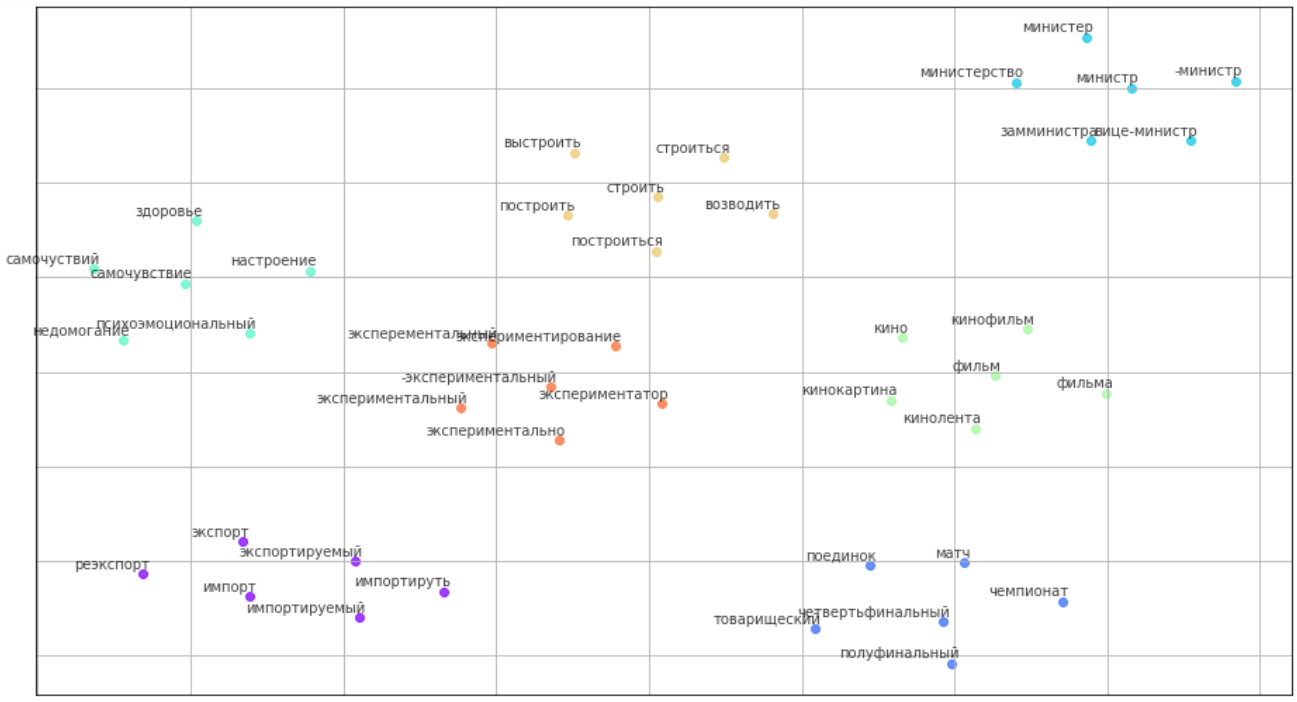

Рис. 1. Пример определения моделью векторизации семантически близких слов

Недостатком метода является то, что для обучения он требует большого объема данных. Кроме того, процесс обучения все же занимает немало времени, несмотря на то, что производится быстрее других статических моделей векторизации. Однако, сейчас нетрудно найти ресурсы, содержащие уже предобученные модели веторизации. Одним из таких ресурсов является [16]. Оттуда была выбрана модель векторизации **geowac_lemmas_none_fasttextskipgram_300_5_2020**. Для работы с данной моделью векторизации необходимо подключить пакет **genism** [17]. Загрузка модели векторизации осуществляется функцией **load_model** модуля prepare.py.

Функция **build_article_vector** данного модуля, получая на вход список токенов, соответствующий некоторому тексту, вычисляет его вектор как среднее арифметическое векторов отдельных токенов, входящих в данный список. Для корректной работы данной функции необходимо подключить стандартную библиотеку **NumPy** [18], позволяющую осуществлять эффективную работу с массивами.

Схема алгоритма функции **prepare_text** представлена на рисунке 2. Данная функция последовательно осуществляет удаление нерелевантных символов, токенизацию, приведение токенов к начальной форме, удаление стоп-слов и векторизацию.

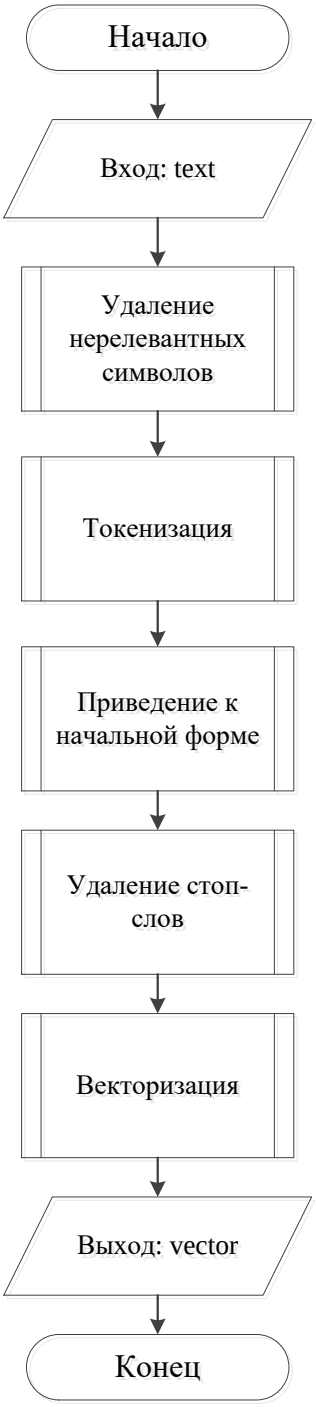

Рис. 2. Схема алгоритма функции prepare_text

2.2. ПОДГОТОВКА ДАННЫХ

Подготовка данных для построения моделей классификации была осуществлена в файле **corpus.py**.

Использованный в данной работе набор данных представлен в виде файла в формате CSV. В дальнейшей работе, в соответствии с принятой в машинной обработке естественного языка терминологией, набор данных будет именоваться **корпусом**.

Для чтения данного корпуса был использован стандартный метод библиотеки **Pandas read_csv**. Подробнее с библиотекой Pandas можно ознакомиться в [19]. Возвращаемым данным методом результатом является объект класса **pandas.DataFrame**, который в среде Jupyter Notebook отображается в виде таблицы, как показано на рисунке 3.

	text	topic	tags
0	Бои у Сопоцкина и Друскеник закончились отступ...	Библиотека	Первая мировая
1	Министерство народного просвещения, в виду про...	Библиотека	Первая мировая
2	Штабс-капитан П. Н. Нестеров на днях, увидев в...	Библиотека	Первая мировая
3	Фотограф-корреспондент Daily Mirror рассказыва...	Библиотека	Первая мировая
4	Лица, приехавшие в Варшаву из Люблина, передаю...	Библиотека	Первая мировая
...	...	...	...
800970	Певец Сергей Шнуров раскритиковал свою коллегу...	NaN	ТВ и радио
800971	Министерство юстиции России предложило изменит...	NaN	Все
800972	Испытание США ранее запрещенной Договором о ли...	NaN	Политика
800973	В ближайшие дни в европейской части России пог...	NaN	Общество
800974	Ведущие футбольные чемпионаты ушли на зимние к...	NaN	Английский футбол

800975 rows × 3 columns

Рис. 3. Пример отображения объекта класса pandas.DataFrame

При анализе корпуса выяснилось, что он включает в себя 23 рубрики. Из их числа были выделены девять рубрик, которые вошли в итоговый корпус. Из корпуса также были удалены пропуски. Названия рубрик и соответствующее каждой из них количество статей в корпусе представлены в таблице 2. Далее словесные названия рубрик были заменены их числовыми эквивалентами, которые также указаны в таблице 2.

Таблица 2. Содержимое корпуса

Рубрика	Числовой эквивалент	Количество статей
Дом	0	21734
Интернет и СМИ	1	44663
Культура	2	53796
Наука и техника	3	53136
Политика	4	40716
Путешествия	5	6408
Силовые структуры	6	19596
Спорт	7	64413
Экономика и бизнес	8	86926

Подготовка корпуса была осуществлена с использованием программного модуля prepare.py.

Корпус после приведения к общему регистру и удаления нерелевантных символов представлен на рис. 4.

	text	topic
0	с января года все телеканалы будут оплачивать ...	8
1	германский автопромышленный концерн volkswagen...	8
2	нераспределенная прибыль оао тюменнефтегаз доч...	8
3	две крупнейших телекоммуникационных компании с...	8
4	оао газ и нижегородский банк сбербанка россии ...	8
...	...	...
372817	законопроект о борьбе с домашним насилием одно...	4
372818	основатель американской частной военной компан...	4
372819	пресс секретарь президента россии дмитрий песк...	4
372820	белый дом ограничил число лиц которые имеют до...	4
372821	испытание сша ранее запрещенной договором о ли...	4

Рис. 4. Изображение очищенного от нерелевантных символов корпуса

Корпус после проведенной токенизации, лемматизации и удаления стоп-слов изображен на рис. 5.

	text	topic
0	[январь, год, всё, телеканал, оплачивать, услу...	8
1	[германский, автопромышленный, концерн, volksw...	8
2	[нераспределённый, прибыль, оао, тюменнефтегаз...	8
3	[крупный, телекоммуникационный, компания, сша,...	8
4	[оао, газ, нижегородский, банк, сбербанк, росс...	8
...	...	...
372817	[законопроект, борьба, домашний, насилие, одно...	4
372818	[основатель, американский, частный, военный, к...	4
372819	[пресс, секретарь, президент, россия, дмитрий,...	4
372820	[белый, дом, ограничить, число, лицо, который,...	4
372821	[испытание, сша, ранее, запретить, договор, ли...	4

Рис. 5. Изображение корпуса после лемматизации

Обработка корпуса в виду его большого размера производилась частями, для чего в библиотеке Pandas есть удобный функционал. Для осуществления над корпусом действий, предусмотренных функциями модуля prepare.py, был использован метод класса Pandas.DataFrame **progress_apply,** для работы которого необходимо подключение библиотеки **tqdm.** Данный метод принимает функцию, которая должна быть применена к записям объекта класса Pandas.DataFrame, в качестве аргумента. Преобразуя корпус в соответствии с этой функцией, метод также в наглядной форме отображает текущий прогресс операции преобразования и выводит оценку оставшегося времени выполнения.

Подготовленный для построения моделей классификации корпус был сохранен в виде файла dataset.csv.

2.3. ПРОВЕДЕНИЕ ЭКСПЕРИМЕНТОВ ПО ПОСТРОЕНИЮ МОДЕЛЕЙ КЛАССИФИКАЦИИ

Для построения моделей классификации и определения гиперпараметров моделей, дающих наивысшие значения точности, была использована стандартная библиотека языка Python **Scikit-learn**. С документацией на нее можно ознакомиться в [20].

Подготовленные ранее данные были разделены на обучающую и тестовую выборки с использованием стандартного метода библиотеки Scikit-learn **train_test_split**. Размер тестовой выборки составил 25% от общего числа статей в корпусе.

Для определения значений гиперпараметров был использован алгоритм решетчатого поиска, заключающийся в последовательном обучении некоторой

модели машинного обучения на одних и тех же данных, но при различных значениях гиперпараметров [21].

Для построения моделей классификации были испробованы три метода классификации: гауссовский наивный байесовский классификатор, логистическая регрессия и случайный лес решающих деревьев.

НАИВНЫЙ БАЙЕСОВСКИЙ КЛАССИФИКАТОР

Наивный байесовский классификатор (НБК) представляет собой простой вероятностный классификатор, использующий теорему Байеса. Он строится на наивном предположении независимости признаков используемого набора данных. Гауссовский НБК предполагает, что входные данные являются непрерывными и распределены в соответствии с нормальным вероятностным распределением. В работе был использован гауссовский НБК, реализованный в библиотеке Scikit-learn.

СЛУЧАЙНЫЙ ЛЕС РЕШАЮЩИХ ДЕРЕВЬЕВ

Решающее дерево представляет собой древовидную структуру условий (тестов), которая разделяет набор данных на отдельные классы. Однако, применять одно решающее дерево может быть нецелесообразным, так как у данного метода есть недостаток, а именно склонность к переобучению [7].

Переобучение – явление, при котором модель машинного обучения слишком точно повторяет зависимости обучающей выборки, что приводит к потере моделью обобщающей способности. Данное явление нежелательно, для борьбы с ним среди прочих используются ансамблевые методы, которые сочетают в себе множество моделей машинного обучения [21, с.100]. Примером ансамблевого метода является **случайный лес решающих деревьев**.

Данный метод заключается в построении множества решающих деревьев, которые в некоторой степени отличаются друг от друга. При построении каждого из деревьев в отдельности модель случайным образом выбирает объекты обучающей выборки, на которых будет обучаться. Для каждого теста модель случайным образом также выбирает значимые признаки. Таким образом, возможное переобучение каждого из деревьев в отдельности на своем наборе данных компенсируется за счет усреднения по множеству различных деревьев.

Оптимизируемыми гиперпараметрами для данной модели машинного обучения в моей работе выступают количество решающих деревьев, и максимальное число признаков, учитываемых моделью на каждом тесте. Схема алгоритма случайного леса решающих деревьев приведена на рис. 6.

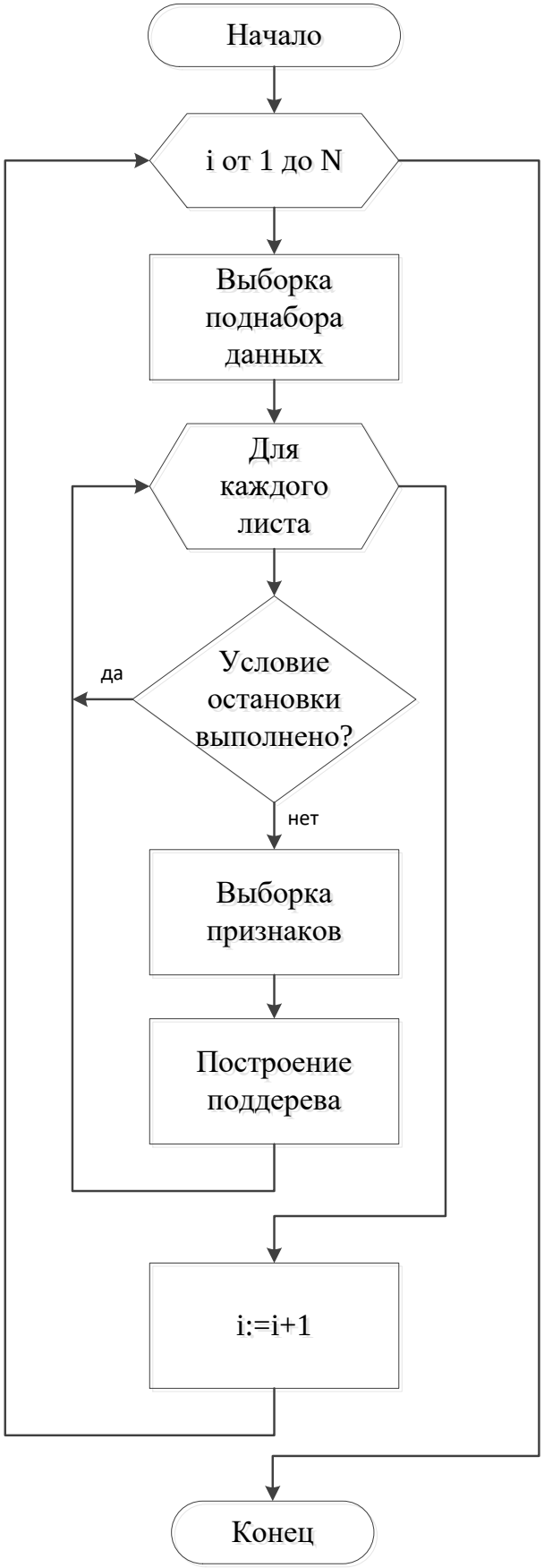

Рис. 6. Схема алгоритма случайного леса решающих деревьев

В работе мною был использован стандартный класс **RandomForestClassifier** библиотеки Scikit-learn. С использованием алгоритма решетчатого поиска были вычислены значения числа деревьев *n_estimators = 150* и максимального числа признаков *max_features = 30.*

ЛОГИСТИЧЕСКАЯ РЕГРЕССИЯ

Несмотря на то, что данный метод имеет в своем названии слово «регрессия», он является методом бинарной классификации. Данный метод вычисляет линейную границу двух классов. В общем случае, это гиперплоскость. Ее уравнение имеет вид (1).

$$f(x_1, x_2, \ldots, x_n) = \beta_0 + \beta_1 x_1 + \beta_0 x_1 + \ldots + \beta_0 x_1 = 0 \quad (1)$$

где β_i, $i = 0 \ldots n$ – вычисляемые коэффициенты, а x_i, $i = 1 \ldots n$ – признаки объектов.

При классификации для объекта вычисляется значение $z = f(x_1, x_2, \ldots, x_n)$, если результат z оказался отрицательным, то объект относится к классу с меткой «0», если положительным, то – к классу с меткой «1». Для нормировки полученному результату применяется логистическая функция (2).

$$P(z) = \frac{1}{1+e^{-z}} \quad (2)$$

Значение логистической функции которой можно интерпретировать как вероятность того, что объект относится к классу с меткой «1». Объекту присваивается метка класса с наибольшей вероятностью принадлежности.

С целью недопущения переобучения в логистической регрессии используется регуляризация, которая «штрафует» модель за переобучение [22]. Параметр регуляризации C является для данной модели гиперпараметром. На рисунке 7 представлен график зависимости точности модели от параметра

регуляризации. График был построен с использованием программных средств языка Python [23]. Хорошо видно, что начиная с некоторого значения *C* точность модели становится константной. В качестве итогового значения параметра регуляризации было выбрано значение *C = 50*.

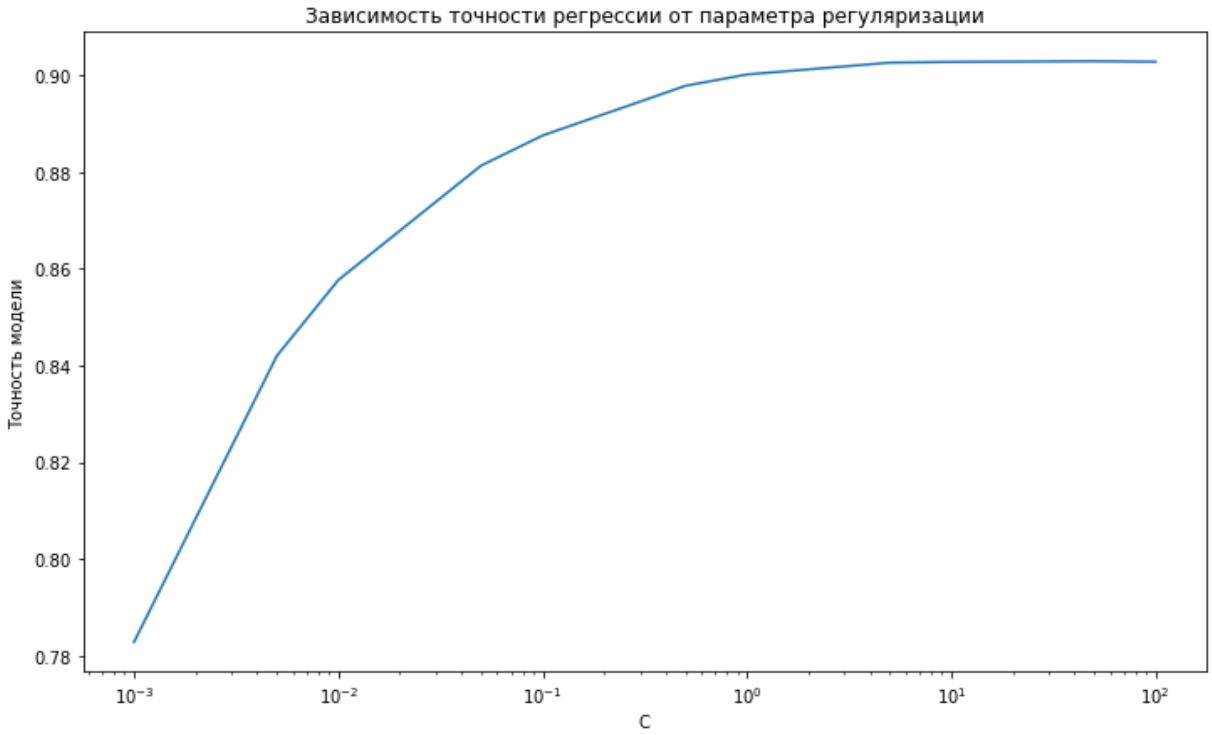

Рис. 7. Зависимость точности модели логистической регрессии от значения параметра регуляризации

3. РЕАЛИЗАЦИЯ И ТЕСТИРОВАНИЕ ПРОГРАММНЫХ СРЕДСТВ

3.1. ОПИСАНИЕ РАЗРАБОТАННЫХ ПРОГРАММНЫХ СРЕДСТВ

В ходе данной работы были разработаны программные средства, позволяющие решать задачу рубрикации русскоязычных текстов с использованием машинного обучения.

Разработанный модуль **prepare.py** позволяет осуществлять обработку русскоязычных текстов перед подачей их на вход модели машинного обучения не только для решения рассматриваемой нами задачи, но и для любого другого случая построения модели машинного обучения для обработки русского языка.

Рассмотренный выше корпус статей новостного издания lenta.ru был обработан в файле **corpus.py** с использованием стандартных библиотек машинного обучения и разработанного в рамках данной работы модуля prepare.py и подготовлен для подачи на вход модели классификации.

В рамках экспериментов по построению моделей классификации в файлк **build_models.py** были построены и обучены при различных значениях гиперпараметров три модели классификации: гауссовский наивный байесовский классификатор, логистическая регрессия и случайный лес решающих деревьев. Модели классификации, дающие наивысшие значения точности, были сохранены в виде отдельных файлов и пригодны для дальнейшего использования.

3.2. ТЕСТИРОВАНИЕ

Тестирование производилось в соответствии с ГОСТ 19.301-79.

Объекты испытания: модели классификации на основе:

- наивного байесовского классификатора;

- логистической регрессии;

- случайного леса решающих деревьев.

Цель испытания: оценка качества каждой модели классификации по ряду метрик классификации.

Средства испытания: испытание проводилось на персональном компьютере с ОС Windows 10 в среде разработки Jupyter Notebook.

Порядок испытаний: последовательная оценка качества каждой модели классификации.

Методы испытания: подача на вход модели классификации подготовленной тестовой выборки, построение матрицы путаницы и вычисление метрик классификации для каждой модели в отдельности.

Матрица путаницы – матрица, у которой на пересечении *i*-ой строки и *j*-ого столбца расположено число, равное числу объектов, которые относятся к *j*-ому классу, но были классифицированы моделью машинного обучения как относящиеся к *i*-му классу [24].

В качестве метрик оценки качества модели классификации были использованы метрики **точности** (англ. precision) и **полноты** (англ. recall), а также **F-мера**, которые рассматриваются для каждого класса в отдельности [21, 25].

Метрики точности $\boldsymbol{precision}$ и полноты $\boldsymbol{recall}$ определяются по формулам (3) и (4) соответственно.

$$precision = \frac{TP}{TP+FP} \tag{3}$$

$$recall = \frac{TP}{TP+FN} \tag{4}$$

TP – количество объектов, которые были правильно классифицированы как относящие к данному классу,

TN – количество объектов, которые были правильно классифицированы как не относящиеся к данному классу,

FP – количество объектов, которые были ошибочно классифицированы как относящиеся к данному классу,

FN – количество элементов, которые были ошибочно классифицированы как не относящиеся к данному классу.

Значения описанных выше метрик может лежать в отрезке от 0 до 1. Чем ближе значения полученных метрик точности и полноты к 1, тем выше точность классификации.

В качестве комбинированной метрики классификации используется F-мера, которая определяется в соответствии с формулой (5)

$$F_\beta = (1+\beta^2)\frac{precision \cdot recall}{(\beta^2 \cdot precision)+recall} \tag{5}$$

где параметр β имеет смысл веса метрики точности.

Частным случаем F-меры является F_1-мера, в которой $\beta=1$.

Для каждой модели классификации представлены матрица путаницы и значения метрик качества.

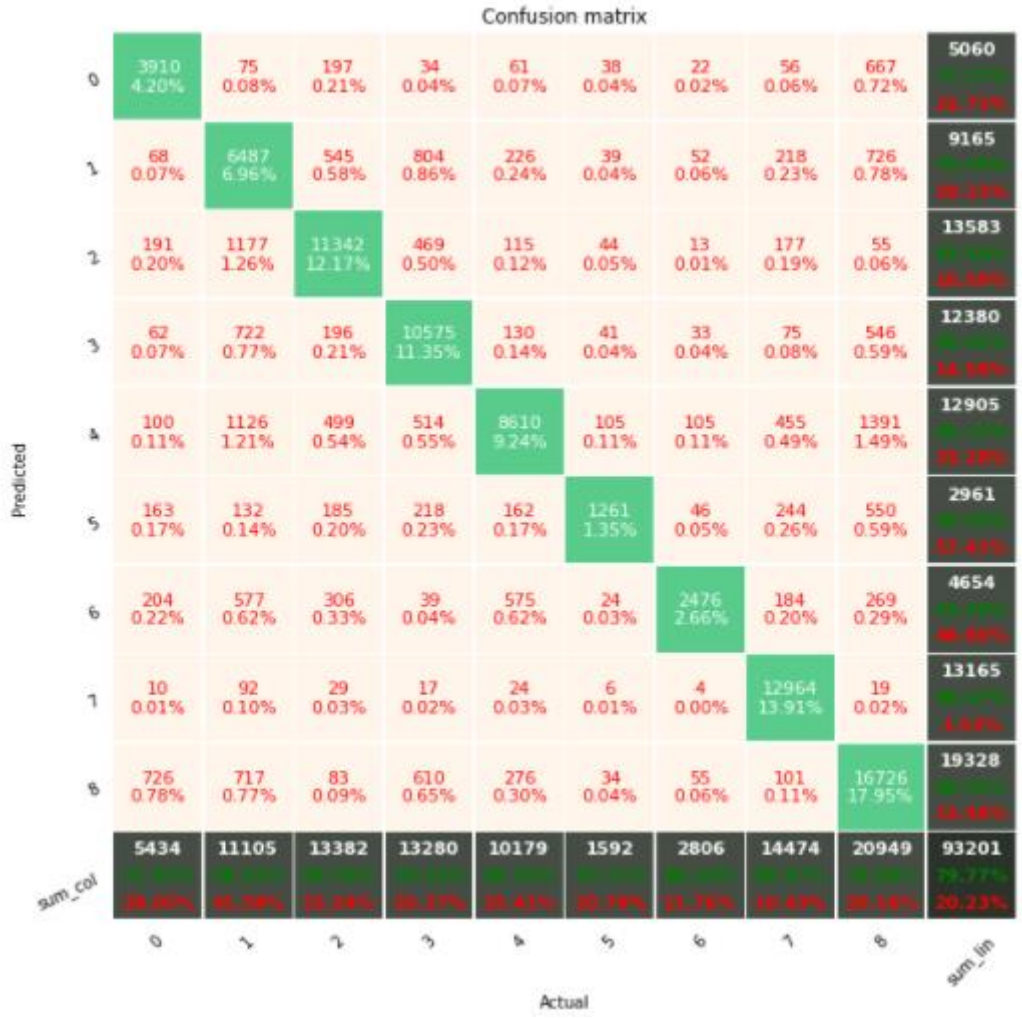

Рис. 8. Матрица путаницы для наивного байесовского классификатора

	precision	recall	f1-score	support
0	0.77273	0.71954	0.74519	5434
1	0.70780	0.58415	0.64006	11105
2	0.83501	0.84756	0.84124	13382
3	0.85420	0.79631	0.82424	13280
4	0.66718	0.84586	0.74597	10179
5	0.42587	0.79209	0.55392	1592
6	0.53202	0.88239	0.66381	2806
7	0.98473	0.89568	0.93809	14474
8	0.86538	0.79842	0.83055	20949
accuracy			0.79775	93201
macro avg	0.73832	0.79578	0.75367	93201
weighted avg	0.81459	0.79775	0.80123	93201

Рис. 9. Отчет о классификации для наивного байесовского классификатора

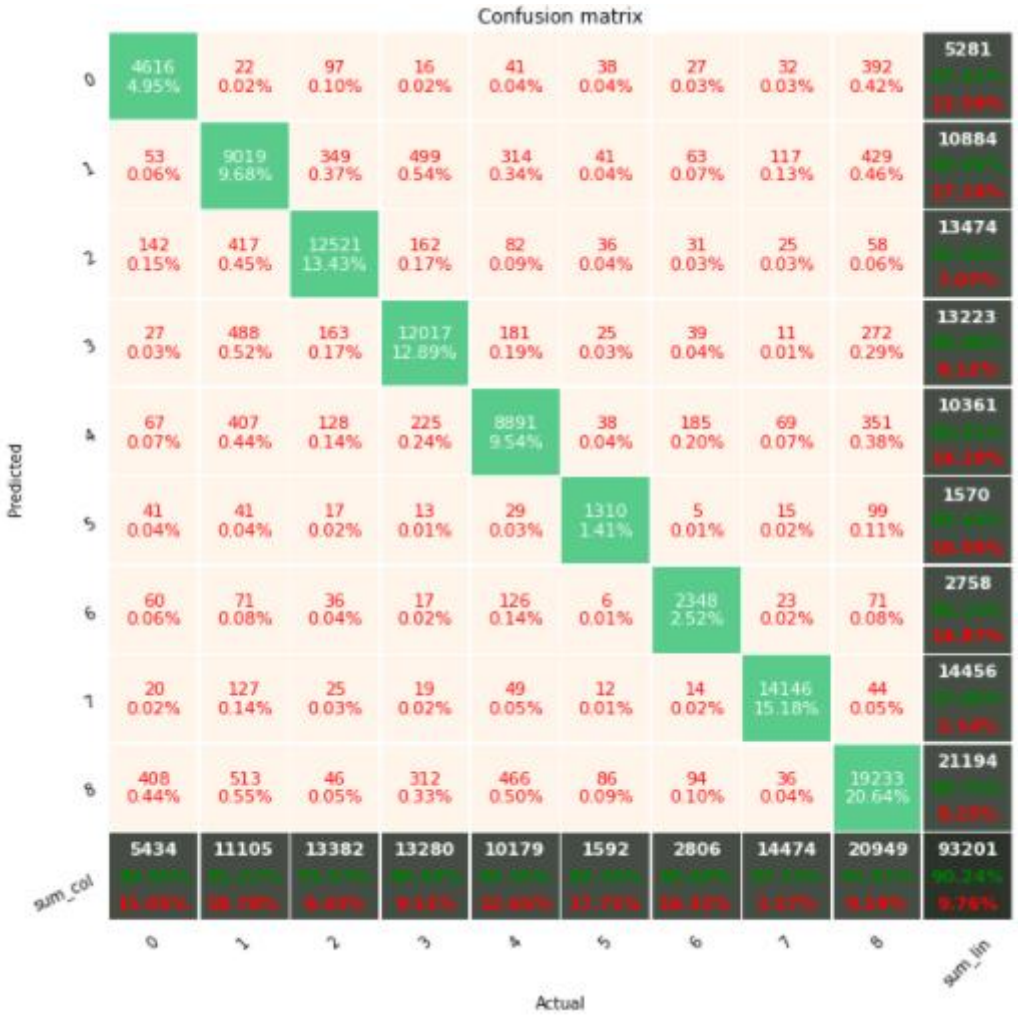

Рис. 10. Матрица путаницы для логистической регрессии

	precision	recall	f1-score	support
0	0.87408	0.84947	0.86160	5434
1	0.82865	0.81216	0.82032	11105
2	0.92927	0.93566	0.93245	13382
3	0.90880	0.90489	0.90684	13280
4	0.85812	0.87346	0.86573	10179
5	0.83439	0.82286	0.82859	1592
6	0.85134	0.83678	0.84400	2806
7	0.97856	0.97734	0.97795	14474
8	0.90747	0.91809	0.91275	20949
accuracy			0.90236	93201
macro avg	0.88563	0.88119	0.88336	93201
weighted avg	0.90216	0.90236	0.90222	93201

Рис. 11. Отчет о классификации для логистической регрессии

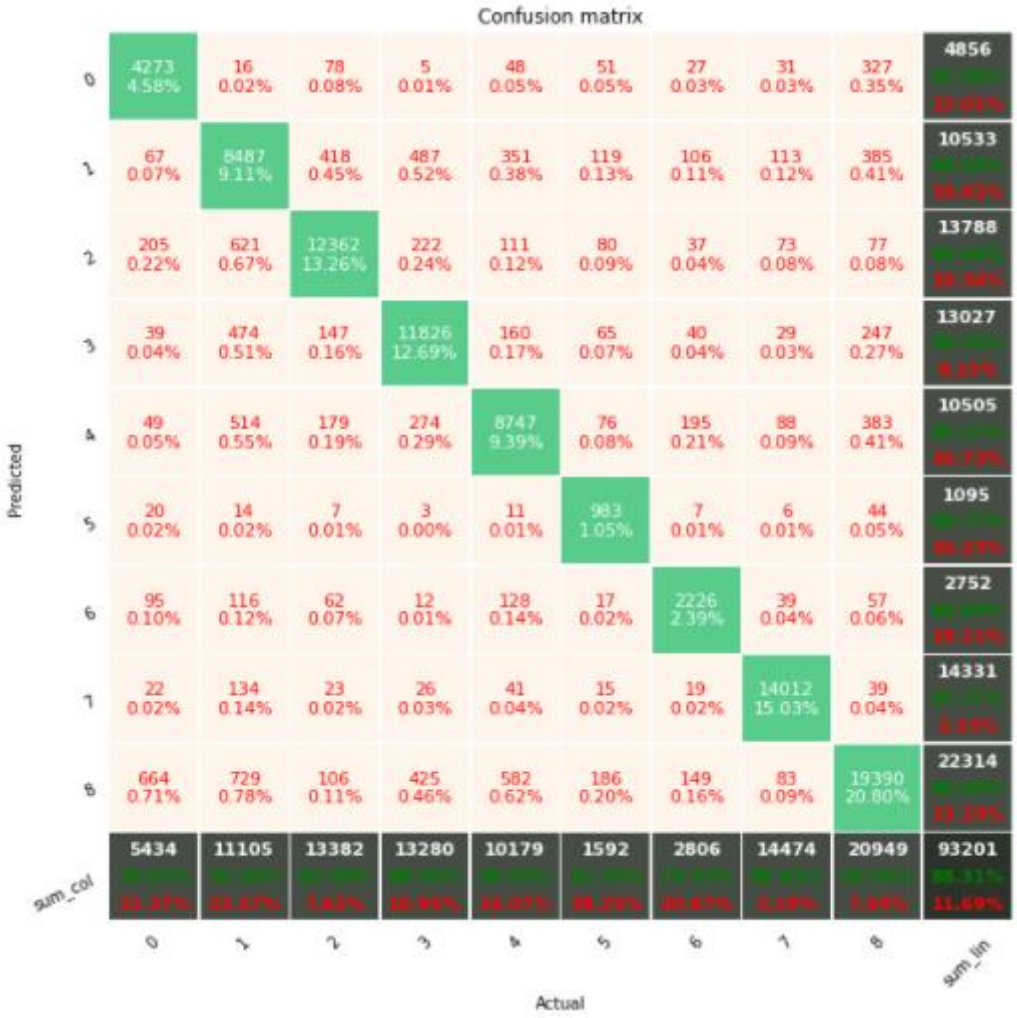

Рис. 12. Матрица путаницы для случайного леса решающих деревьев

	precision	recall	f1-score	support
0	0.87994	0.78635	0.83052	5434
1	0.80575	0.76425	0.78445	11105
2	0.89658	0.92378	0.90997	13382
3	0.90781	0.89051	0.89908	13280
4	0.83265	0.85932	0.84577	10179
5	0.89772	0.61746	0.73167	1592
6	0.80887	0.79330	0.80101	2806
7	0.97774	0.96808	0.97289	14474
8	0.86896	0.92558	0.89638	20949
accuracy			0.88310	93201
macro avg	0.87511	0.83651	0.85242	93201
weighted avg	0.88318	0.88310	0.88221	93201

Рис. 13. Отчет о классификации для случайного леса решающих деревьев

В таблице представлены сводные результаты оценки качества построенных моделей классификации

Таблица 3. Сводная таблица метрик для построенных моделей классификации

Модель классификации	**Среднее взвешенное значение метрики точности**	**Среднее взвешенное значение метрики полноты**	**Среднее взвешенное значение F_1-меры**
Наивный байесовский классификатор	0,81459	0,79775	0,75367
Логистическая регрессия	0,90216	0,90236	0,90222
Случайный лес решающих деревьев	0,88318	0,88310	0,88221

По результатам проведенной оценки качества построенных моделей классификации можно сделать следующие выводы:

- все разработанные в рамках данной работы программные средства протестированы и работают в штатном режиме;

- все построенные модели классификации обладают обобщающей способностью, что означает, что данные модели классификации были успешно обучены;

- наилучшие значения метрик качества имеет модель классификации на основе логистической регрессии;

- случайный лес решающих деревьев показывает значения метрик качества классификации чуть ниже логистической регрессии;

- наивный байесовский классификатор как самая простая из всех построенных моделей показывает наихудшие результаты, хотя даже в этом случае можно признать, что обучение данной модели является успешным;

- для дальнейшего использования наиболее пригодной является модель классификации на основе логистической регрессии.

ЗАКЛЮЧЕНИЕ

Целью преддипломной практики являлось построение системы рубрикации новостных текстов с использованием машинного обучения.

Реализованные в рамках данной работы программные средства полностью решают поставленную задачу. Был разработан программный модуль, позволяющий производить подготовку текстов на естественном языке для подачи их на вход моделей машинного обучения. Удобство данного модуля заключается в возможности его повторного использования. На заранее подготовленных данных были обучены модели классификации, их качество было оценено по ряду метрик. Можно сделать вывод, что качество построенных моделей классификации для задач машинной обработки естественных языков является достаточно высоким.

Результаты, полученные в данной работе, могут быть улучшены при использовании более усовершенствованных подходов к подготовке данных и других методов классификации.

Логичным продолжением данной работы могло быть создание веб-сайта, осуществляющего рубрикацию текстов.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Reinsel, D. The Digitalization of the World / D. Reinsel, J. Gantz, J. Rydning – 2018. – 28 с.: [Электронный ресурс]. – URL: https://www.seagate.com/files/www-content/our-story/trends/files/idc-seagate-dataage-whitepaper.pdf. (Дата обращения: 11.03.2021).

2. Обработка естественного языка (NLP) для машинного обучения: [Электронный ресурс] // Машинное обучение. URL: https://www.machinelearningmastery.ru/natural-language-processing-nlp-for-machine-learning-d44498845d5b/. (Дата обращения: 21.02.2021).

3. Воронцов, К.В. Математические методы машинного обучения по прецедентам (теория обучения машин) / К.В. Воронцов. – 141 с.: [Электронный ресурс]. – URL: http://www.machinelearning.ru/wiki/images/6/6d/Voron-ML-1.pdf. (Дата обращения: 14.01.2021).

4. Шаграев, А.Г. Модификация, разработка и реализация методов классификации новостных текстов: дисс. … канд. технических наук: 05.13.17. – НИУ «МЭИ», Москва, 2014. – 108 с.

5. News dataset from Lenta.ru [Электронный ресурс] // Kaggle: Your Home for Data Science. URL: https://www.kaggle.com/yutkin/corpus-of-russian-news-articles-from-lenta. (Дата обращения 08.02.2021)

6. Лутц, М. Изучаем Python. В 2 тт. Т.1 / М. Лутц; пер. с англ. – 5 изд. – СПб.: Диалектика, 2019. – 832 с. – ISBN 978-5-907144-52-1.

7. Рашка, С. Python и машинное обучение / С. Рашка; пер. с англ. А. В. Логунов; под ред. Д. А. Мовчан. — М.: ДМК Пресс, 2017. — 418 с.: ил. — ISBN: 978-5-97060-409-0.

8. Как решить 90% задач NLP: пошаговое руководство по обработке естественного языка: [Электронный ресурс] // Хабр. URL: https://habr.com/ru/company/oleg-bunin/blog/352614/. (Дата обращения: 12.02.2021).

9. NLTK 3.6.2 documentation: [Электронный ресурс]. URL: https://www.nltk.org/. (Дата обращения 14.04.2021).

10. Предобработка текста в NLP: [Электронный ресурс] // Python School. URL: https://python-school.ru/nlp-text-preprocessing/. (Дата обращения: 02.03.2021).

11. Korobov M.: Morphological Analyzer and Generator for Russian and Ukrainian Languages // Analysis of Images, Social Networks and Texts, pp. 320-332 (2015).

12. Морфологический анализатор pymorphy2: [Электронный ресурс]. URL: https://pymorphy2.readthedocs.io/en/stable/. (Дата обращения: 03.05.2021).

13. Жеребцова Ю.А., Чижик А.В. Сравнение моделей векторного представления текстов в задаче создания чат-бота. // Вестник НГУ. 2020. Т.18. URL: https://cyberleninka.ru/article/n/sravnenie-modeley-vektornogo-predstavleniya-tekstov-v-zadache-sozdaniya-chat-bota/viewer. (Дата обращения: 19.03.2021).

14. Куратов, Ю.М. Специализация языковых моделей для применения к задачам обработки естественного языка: дисс. … канд. физико-математических наук: 05.13.17. – МФТИ, Москва, 2020. – 121 с.

15. Чудесный мир Word Embeddings: какие они бывают и зачем нужны: [Электронный ресурс] // Хабр. URL: https://habr.com/ru/company/ods/blog/329410/. (Дата обращения: 15.04.2021).

16. RusVectores: семантические модели для русского языка: [Электронный ресурс]. URL: https://rusvectores.org/ru/. (Дата обращения: 14.02.2021).

17. Gensim – Руководство для начинающих: [Электронный ресурс] // Еще один блог веб-разработчика. URL: https://webdevblog.ru/gensim-rukovodstvo-dlya-nachinajushhih/. (Дата обращения: 16.03.2021).

18. Varoquaux G. Scipy Lecture Notes / G. Varoquaux, E. Gouillart, O. Vahtras, P. Buyl. и др. – 2020. – 674 с.: [Электронный ресурс]. – URL: http://scipy-lectures.org/. (Дата обращения: 08.10.2021).

19. Pandas – Python Data Analysis Library: [Электронный ресурс]. URL: https://pandas.pydata.org/. (Дата обращения: 05.02.2021).

20. Scikit-learn. Machine Learning in Python: [Электронный ресурс]. URL: https://scikit-learn.org/stable/. (Дата обращения: 12.04.2021).

21. Мюллер, А. Введение в машинное обучение с помощью Python / А. Мюллер, С. Гвидо; пер. с англ. – М.: Альфа-книга, 2018. – 480 с.: ил. – ISBN: 978-1-449-36941-5.

22. Флах, П. Машинное обучение. Наука и искусство построения алгоритмов, которые извлекают знания из данных / П. Флах; пер. с англ. А.А. Слинкина. – М.: ДМК Пресс, 2015. – 400 с.: ил.

23. Абдрахманов М.И. Библиотека matplotlib / М.И. Абдрахманов. – 2019. – 124 с.: [Электронный ресурс]. URL: https://devpractice.ru/files/books/python/Matplotlib.book.pdf. (Дата обращения: 03.05.2021).

24. Что такое матрица путаницы в машинном обучении: [Электронный ресурс] // Машинное обучение. URL: https://www.machinelearningmastery.ru/confusion-matrix-machine-learning/. (Дата обращения: 14.05.2021).

25. Метрики в задачах машинного обучения: [Электронный ресурс] // Хабр. URL: https://habr.com/ru/company/ods/blog/328372/. (Дата обращения: 11.05.2021).

ПРИЛОЖЕНИЕ А. ЛИСТИНГ ПРОГРАММ

Программный модуль prepare.py

```
#!/usr/bin/env python
# coding: utf-8


"""
Программный модуль, содержащий функции для
обработки текстов на естественном языке.

Разработчик: студент группы А-08-17 "НИУ МЭИ"
Челышев Э.А.
e-mail: ChelyshevEA@mpei.ru
Место разработки: каф. ВМСС НИУ "МЭИ"
Дата разработки модуля: 15 мая 2021 г.

Входные и выходные данные описаны для каждой функции в отдельности.
См. поясняющую информацию далее.
Для корректной и полнофункциональной работы необходимо наличие
файла предобученной модели векторизации FastText (файл model.model)
"""

import numpy as np  # импорт пакета для работы с массивами
import re  # импорт пакета для работы с регулярными выражениями
import gensim  # импорт пакета для работы с моделями векторизации
import nltk  # импорт пакета для работы с естественным языком
from nltk import word_tokenize  # импорт функции токенизации
from nltk.corpus import stopwords  # импорт стоп-слов
import pymorphy2  # импорт морфологического анализатора

# создание объекта класса морфологического анализатора
morph = pymorphy2.MorphAnalyzer()
# список стоп-слов русского языка
stop_words = stopwords.words("russian")
# объявление переменной для модели векторизации
w2v_model = None

"""
Данная функция устанавливает путь к внешему файлу модели векторизации
и загружает ее.
На вход функция получает строку - путь к файлу. По умолчанию путь
'model.model'
Каких-либо результатов функция не возвращает.
"""
def load_model(path='model.model'):
    try:
        # загрузка модели векторизации
        global w2v_model
        w2v_model = gensim.models.KeyedVectors.load(path)
```

```
    except:  # если не удалось загрузить модель векторизации
        print('Не удалось загрузить модель векторизации. Проверьте
правильность пути')
    pass

"""
Данная функция осуществляет приведение строки текста к общему регистру
(строчные буквы), а также
производит удаление нерелевантных символов и встречающиеся в тексте URL-
ссылки.
На вход функция получает text - строку текста, подлежащую обработке.
Результатом является обработанная строка
"""
def preprocess_text(text):
    # приведение к общему регистру и удаление буквы "ё"
    text = text.lower().replace("ё", "е")
    # удаление URL-ссылок
    text = re.sub('((www\.[^\s]+)|(https?://[^\s]+))', 'URL',
                  text)
    # удаление небуквенных символов
    text = re.sub('[^а-zА-Za-яА-Я]+', ' ', text)
    # несколько подряд идущих пробелов заменяются на один
    text = re.sub(' +', ' ', text)
    # удаление пробелов с обоих концов строки, если таковые имеются
    preprocessed_text = text.strip()
    return preprocessed_text

"""
Данная функция осуществляет приведение токенов к начальной форме путем их
лемматизации.
На вход функция получает text_list - список строк, каждая из которых
является отдельным токеном, подлежащим лемматизации.
Результатом функции является список токенов, прошедших лемматизацию
"""
def lemmatize(text_list):
    result = list()  # подготовка списка для хранения результата
    for token in text_list:  # цикл по токенам списка
        # получение начальной формы
        modified_token = morph.parse(token)[0].normal_form
        result.append(modified_token)  # добавление результата к списку
    return result

"""
Данная функция осуществляет удаление стоп-слов.
На вход функция получает список строк, являющихся отдельными токенами.
Результатом является список токенов, откуда удалены стоп-слова.
Для корректной работы данной функции токены должны быть приведены
к начальной форме с использованием функции lemmatize()
"""
def delete_stop_words(text_list):
    # подготовка списка для хранения результата
    result = list()
    # цикл по токенам входного списка
    for token in text_list:
        # если токен не является стоп-словом
        if token not in stop_words:
```

```
            # он добавляется к результирующему списку
            result.append(token)
    return result

"""
Данная функция осуществляет построение вектора статьи.
На вход функция получает список токенов.
Результатом выполнения функции является вектор размерности 300,
в формате массива NumPy.
Для корректной работы функции входной список должен быть обработан
с использованием функции prepare_text
"""
def build_article_vector(t):
    # счетчик числа векторизованных токенов
    count = 0
    # подготовка списка для хранения результата
    res = np.zeros(300)
    # цикл по токенам
    for word in t:
        # если токен содержится в модели
        if (word in w2v_model.vocab):
            # прибавляем полученный для токена вектор к результирующему
            res += w2v_model.get_vector(word)
            # увеличиваем счетчик векторизованных токенов
            count += 1
    # определяем среднее по статье
    res /= count
    return res

"""
Данная функция осуществляет полную подготовку текста на естественном языке,
а именно: токенизацию, лемматизацию и удаление стоп-слов.
На вход функция получает строку текста, подлежащую обработке.
Результатом выполнения функции является список токенов, прошедший этапы
подготовки
"""
def prepare_text(text):
    # приведение к одному регистру и удаление нерелевантных символов
    text = preprocess_text(text)
    #  токенизация с использованием пакета nltk
    text_list = word_tokenize(text, language = "russian")
    # лемматизация с использованием функции lemmatize
    text_list = lemmatize(text_list)
    # удаление стоп-слов с использованием функции delete_stop_words
    text_list = delete_stop_words(text_list)
    # векторизация
    vector = build_article_vector(text_list)
    return vector
```

Файл corpus.py

```
#!/usr/bin/env python
# coding: utf-8

"""
Данная программа подготавливает корпус для экспериментов по построению
моделей машинного обучения.

Разработчик: студент группы А-08-17 "НИУ МЭИ"
Челышев Э.А.
e-mail: ChelyshevEA@mpei.ru
Место разработки: каф. ВМСС НИУ "МЭИ"
Дата разработки модуля: 16 мая 2021 г.

Входными данными является корпус статей новостного портала lenta.ru (файл
lentanews.csv).
Результатом выполнения данной программы являются подготовленный для
построения моделей машинного обучения
корпус в виде файла dataset.csv.
Для работы необходимо подключить модуль prepare.py
"""

import pandas as pd  # импорт пакета для работы с датафреймами
# импорт пакета для отслеживания времени выполнения обработки датафрейма
from tqdm import tqdm
import prepare  # импорт модуля с функциями подготовки данных

tqdm.pandas()  # отслеживание времени выполнения для датафреймов pandas

# Считываем данные
data_path = 'C:/Users/Эдуард Челышев/Desktop/мл/Dip/diplom/lentanews.csv'
data = pd.read_csv(data_path,
                   sep=',',
                   error_bad_lines=False,
                   usecols=['text', 'topic', 'tags'])

# подсчет количества статей в рубриках
print(data.groupby(['topic']).count())
print(data.groupby(['tags']).count())

# формирование рубрики Политика
data['topic'] = np.where((data.tags == 'Политика'), 'Политика', data.topic)
# проверим наличие рубрики Политика
print(data.groupby(['topic']).count())

# удаление из корпуса статей, принадлежащих некоторым рубрикам
data = data.drop(
    data[(data['topic'] == '69-я параллель') | (data['topic'] ==
'Библиотека')
         | (data['topic'] == 'Крым') | (data['topic'] == 'МедНовости') |
         (data['topic'] == 'Оружие') | (data['topic'] == 'Сочи') |
         (data['topic'] == 'ЧМ-2014') | (data['topic'] == 'Мир') |
```

```
        (data['topic'] == 'Культпросвет ') | (data['topic'] == 'Легпром')
|
        (data['topic'] == 'Россия') | (data['topic'] == 'Бывший СССР') |
        (data['topic'] == 'Из жизни') | (data['topic'] ==
'Ценности')].index)
# объединение рубрик Бизнес и Экономика
data = data.replace({'topic': {'Бизнес': 'Экономика'}})
# изучение оставшихся рубрик
print(data.groupby(['topic']).count())
# удаление строк, содержащих пропуски
data = data.dropna(axis='index', how='any')

# Построение списка из названий имеющихся рубрик
topics = np.sort(data['topic'].unique())
print(topics)
# построение словаря соответствия названий рубрик и их цифровых
эквивалентов
dct = dict()  # объявление пустого словаря
label = 0  # нумерация рубрик начинается с 0
for item in topics:
    dct[item] = label  # добавляем новую рубрику
    label += 1
# замена названий рубрик на цифровые эквиваленты
data = data.replace({'topic': dct})

# рассмотрим структуру имещихся данных
print('Размерность датафрейма: {}'.format(data.shape))
print('Столбцы датафрейма: {}'.format(data.columns))
print('Информация: {}'.format(data.info()))
print('Статистические характеристики:\n')
data.describe()

# указываем модулю prepare адрес к модели векторизации
model_path = 'model/model.model'
prepare.load_model(model_path)

# произведем подготовку с использованием функции
# prepare_text модуля prepare
# обработка датафрейма осуществляется частями
chunk_list = []
for chunk in data:
    # для каждой части датафрейма осуществляем обработку
    chunk['text'] = chunk['text'].progress_apply(prepare.prepare_text)
    chunk_list.append(chunk)
# объединяем обработанные части датафрейма
data = pd.concat(chunk_list)

# создаем копию датафрейма
vectors = data.copy()
# разбиваем список по колонкам
vectors = pd.DataFrame(vectors['text'].to_list(), columns=range(0, 300))
# объединяем со столбцом рубрик
df = pd.concat([vectors, data['topic']], axis=1)
df = df.dropna(axis='index', how='any')  # удаляем пропуски
df.to_csv('dataset.csv', sep=',', header=True,
          index=False)  # сохрянем итоговый файл
```

Файл build_models.py

```
#!/usr/bin/env python
# coding: utf-8

"""
Данная программа содержит ряд экспериментов по построению моделей машинного обучения.

Разработчик: студент группы А-08-17 "НИУ МЭИ"
Челышев Э.А.
e-mail: ChelyshevEA@mpei.ru
Место разработки: каф. ВМСС НИУ "МЭИ"
Дата разработки модуля: 17 мая 2021 г.

Входными данными является заранее подготовленный набор данных (файл dataset.csv).
Результатом работы являются обученные модели машинного обучения для задачи рубрикации текстов.
"""

# отчет о классификации
from sklearn.metrics import classification_report
# функция разделения на тренировочный и тестовый набор
from sklearn.model_selection import train_test_split
# импорт пакета для работы с датафреймами
import pandas as pd
import numpy as np  # для работы с массивами
import pickle  # для сохранения моделей
# логистическая регрессия
from sklearn.linear_model import LogisticRegression
# решетчатый поиск
from sklearn.model_selection import GridSearchCV
# для отображения матрицы путаницы
import confusion_matrix_pretty_print as cmpp
# гауссовский наивный байесовский классификатор
from sklearn.naive_bayes import GaussianNB
# предобработка данных
from sklearn import preprocessing
# случайный лес решающих деревьев
from sklearn.ensemble import RandomForestClassifier

"""
Данная функция выводит метрики качества модели классификации: отчет о классификации и матрицу путаницы.
На вход функция получает два массива: достоверный массив меток и предсказание модели.
Функция выводит отчет о классификации и отображает матрицу путаницы.
"""
def evaluate_quality(y_test, predicted):
    # отчет о классификации
    print(classification_report(y_test, lr_predicted, digits=5))
    # матрица путаницы
    cmpp.plot_confusion_matrix_from_data(y_test,
```

```
                                        lr_predicted,
                                        columns=range(0, 9),
                                        figsize=[7, 7])
    pass


# чтение данных
path = 'dataset.csv'
df = pd.read_csv(path)
# выделение меток классов в отдельный датафрейм
targets = df.filter(['topic'], axis=1)
df.drop(['topic'], axis='columns', inplace=True)
vectors = df  # содержит только векторы статей
# разделение на тренировочную и тестовую выборки

x_train, x_test, y_train, y_test = train_test_split(vectors,
                                                    targets,
                                                    test_size=0.25,

stratify=targets['topic'],
                                                    random_state=1)
# сжатие массивов меток до одной оси и приведение их к целочисленному типу
y_train = np.ravel(y_train)
y_train = np.array(y_train, dtype='int8')
y_test = np.ravel(y_test)
y_test = np.array(y_test, dtype='int8')

# приведение к типу float32
x_train = x_train.astype('float32')
x_test = x_test.astype('float32')

""" Наивный байесовский классификатор """
# нормализация данных
x_train_norm = preprocessing.normalize(x_train)
x_test_norm = preprocessing.normalize(x_test)

# обучение гауссовского наивного байесовского классификатора
bayes = GaussianNB().fit(x_train_norm, y_train)
bayes_predicted = bayes.predict(x_test_norm)
# оценка качества
evaluate_quality(y_test, bayes_predicted)
# Сохранить модель в файле
pickle.dump(bayes, open('bayes.pkl', 'wb'))

""" Логистическая регрессия """
# создание объекта логистической регрессии
lr = LogisticRegression(random_state=0, max_iter=10000)
# поиск гиперпараметров методом решетчатого поиска
log_params = {'C': [0.001, 0.005, 0.01, 0.05, 0.1, 0.5, 1, 5, 10, 50, 100]}
log_grid = GridSearchCV(lr, log_params, cv=3)
log_grid.fit(x_train, y_train)
# вывод оптимальных гиперпараметров
print(log_grid.best_params_)
print(log_grid.cv_results_)

# обучение регрессии при оптимальных параметрах
```

```
lr_best = LogisticRegression(C=50, max_iter=10000, random_state=0)
lr_best.fit(x_train, y_train)

# оценка качества
lr_predicted = lr_best.predict(x_test)
evaluate_quality(y_test, lr_predicted)
# Сохранить модель в файле
pickle.dump(lr_best, open('log_reg.pkl', 'wb'))

""" Случайный лес решающих деревьев """
# поиск оптимальных параметров
rfc = RandomForestClassifier(random_state=42)
forest_params = {
    'n_estimators': [50, 100, 150, 200],
    'max_features': [5, 10, 15, 20, 25, 30, 35, 40]
}
forest_grid = GridSearchCV(rfc, forest_params, cv=3)
forest_grid.fit(x_train, y_train)
print(forest_grid.best_params_)
print(forest_grid.cv_results_)

# обучение для оптимальных параметров
rfc = RandomForestClassifier(n_estimators=150, max_features=30)
rfc.fit(x_train, y_train)
rfc_predicted = rfc.predict(x_test)

# оценка качества
evaluate_quality(y_test, rfc_predicted)
# Сохранить модель в файле
pickle.dump(rfc, open('forest.pkl', 'wb'))
```