МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное бюджетное образовательное учреждение
высшего образования
«Тольяттинский государственный университет»

Институт математики, физики и информационных технологий
(наименование института полностью)

Кафедра «Прикладная математика и информатика»
(наименование кафедры)

02.03.03 Математическое обеспечение и администрирование информационных систем
(код и наименование направления подготовки, специальности)

Технология программирования
(направленность (профиль)/специализация)

БАКАЛАВРСКАЯ РАБОТА

на тему «Применение технологии Text Mining для решения задачи классификации текста»

Студент	М.Р. Мирзоева	
	(И.О. Фамилия)	(личная подпись)
Руководитель	Э.В. Егорова	
	(И.О. Фамилия)	(личная подпись)
Консультанты	К.А. Селиверстова	
	(И.О. Фамилия)	(личная подпись)

Допустить к защите

Заведующий кафедрой к.т.н., доцент, А.В. Очеповский _____
(ученая степень, звание, И.О. Фамилия) (личная подпись)

«_____» _____ 2019 г.

Тольятти 2019

АННОТАЦИЯ

Тема: «Применение технологии Text Mining для решения задачи классификации текста».

В данной работе рассмотрены основные этапы реализации текстового классификатора, решающего конкретную задачу классификации текста. Объектом исследования являются алгоритмы автоматической классификации текста, предметом – эффективность работы алгоритмов в зависимости от используемых параметров. Цель работы – реализация эффективного текстового классификатора.

Под условия решаемой задачи классификации текста отобраны два алгоритма, подходящие по определенным в работе параметрам. Для каждого алгоритма описаны несколько параметров режимов запуска, на основе которых и было проведено тестирование на эффективность. Оценка эффективности классификации проводится с помощью метода кросс-валидации по классическим для данной научной области метрикам – точность (precision) и полнота (recall).

По результатам оценки эффективности выбранных алгоритмов классификации в условиях решения поставленной задачи, для дальнейшего использования выбран метод опорных векторов как наиболее оптимальный. Реализованный на основе данного алгоритма текстовый классификатор предназначен для встраивания в виде программного модуля в целостную информационную систему.

Полученную оценку точности классификации, равную 79%, можно считать достаточной для подобного рода решений. Однако существует возможность ее повышения путем переобучения классификатора на выборке данных, которая может быть собрана за время его эксплуатации.

Данная работа состоит из пояснительной записки общим объемом 45 страниц, включая 6 рисунков, 7 таблиц, список литературы из 30 источников и 1 приложение.

ABSTRACT

The title of the bachelor's thesis is «Application of Text Mining technology for solving text classification problem».

This bachelor's thesis is devoted to the main stages of text classifier implementation that solves specific practical text classification problem. The aim of the work is to implement effective text classifier. The object of the bachelor's thesis is text classification algorithms. The subject of the bachelor's thesis is algorithms' efficiency depending on the parameters used.

We start with the statement of the problem and then follow through with its possible solutions, which includes choice of classification algorithms that can be applied in accordance to problem's condition. Support vector machine and Random forest were chosen as applicable algorithms for a solvable problem. Then we describe several operation modes for two algorithms. For each described operation mode we conduct an experimental launch. Evaluation of obtained results is carried out by cross-validation algorithm with classical for this scientific field metrics (precision, recall, f-measure).

The results show clearly that support vector machine is more applicable for solvable text classification problem. We then present the program implementation of text classifier that using support vector machine as a basis text classification algorithm. Implemented text classifier works in operation mode that was determined as most effective according to experiment results. Developed software will be embedded as a program module to a complete information system.

Obtained f-measure of 79% can be considered as acceptable taking into consideration the fact that text classification problem is quite complicated. However, text classifier efficiency can be increased after retraining on new training set that can be collected during use in practice.

The bachelor's thesis consists of an explanatory note on 45 pages, including 6 figures, 7 tables, the list of 30 references including 25 foreign sources and 1 appendix.

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

В настоящее время применение интеллектуальных алгоритмов, основанных на машинном обучении, широко распространено во многих областях. Такие алгоритмы позволяют решать широкий спектр практических задач, возникающих при работе с данными. К таким задачам относят кластеризацию, прогнозирование, классификацию и прочие. В основе алгоритмов машинного обучения лежит принцип построения специальной модели, которая обучается под конкретную решаемую задачу в процессе решения множества схожих задач. Итоговая модель способна достаточно эффективно решать поставленную перед ней задачу, что обуславливает популярность данного вида алгоритмов.

Другим преимуществом использования алгоритмов машинного обучения является повышение уровня автоматизации задач, в которые не применимы другие широко используемые алгоритмы. Существует определенный круг задач, которые традиционно выполняются с помощью умственного труда специалистов, как правило специально нанимаемых в организации для этого. При этом сами задачи не требуют глубоких знаний от специалиста, то есть данный вид работ способен выполнять практически любой человек после непродолжительного обучения. Проблемой при разработке программных решений, обеспечивающих автоматизацию подобных задач, является слабый уровень формализации самой задачи, что серьезно ограничивает круг доступных алгоритмических решений. Алгоритмы машинного обучения эффективно справляются с такими видами задач, поскольку для эффективного обучения и дальнейшей работы им достаточно предоставить примеры «правильной» работы, без строгой формализации и ручного выявление закономерностей. К множеству таких плохо формализуемых задач относится классификация текста.

Классификация текста (документов) – распространенная задача, связанная с областями информационного поиска и машинного обучения. Суть процесса классификации заключается в категоризации новых элементов, то

есть в присвоении новому элементу какого-либо класса из множества заранее предопределенных классов. В случае задачи классификации текста, классифицироваться может любая информация, выраженная в текстовом виде и имеющая определенный смысл. Использование автоматической классификации текстовых документов позволяет [25] ускорить процесс категоризации документов, при этом избежать потерь точности и «чувствительности», присущих оценкам эксперта-человека.

Помимо этого, за счет отсутствия необходимости оплаты работы эксперта, достигается оптимизация денежных расходов компании, в которой применяется технология автоматической классификации. Таким образом, автоматическая классификация текстов, основанная на применении определенных интеллектуальных алгоритмов, имеет ряд значительных преимуществ по сравнению с процессом классификации экспертом-человеком.

Объектом исследования являются алгоритмы автоматической классификации текста, **предметом исследования** – тестовые классификаторы и методы их реализации в зависимости от используемых параметров.

Целью выпускной квалификационной работы является реализация текстового классификатора, способного классифицировать поступающие в него текстовые данные в режиме реального времени с высоким уровнем эффективности классификации.

Для достижения поставленной цели требуется решение следующих **задач**:

- описать решаемую задачу классификации текста;
- рассмотреть существующие алгоритмы классификации текста и выбрать из них подходящих для использования в имеющихся условиях;
- провести анализ научных источников в целях выявления особенностей, имеющихся при разработке подобных решений;
- реализовать текстовые классификаторы, работающие на основе выбранных ранее алгоритмов классификации;

- провести тестирование реализованных текстовых классификаторов в различных режимах с целью определения условий достижения наилучшей эффективности;
- проанализировать полученные в процессе тестирования результаты, выбрать оптимальный вариант для использования;
- оформить выводы по проделанной работе и дальнейшим перспективам применения реализованного классификатора.

Результаты работы могут быть полезны при реализации схожих программных решений, так как содержат подробное описание всего процесса создания текстового классификатора, а также для общего ознакомления с алгоритмами автоматической классификации текста, поскольку в работе содержится описание наиболее часто используемых алгоритмов с выделением их преимуществ и недостатков.

В первой главе дается описание решаемой задачи классификации текста и исходных данных, приводится анализ алгоритмов классификации текста с выбором используемых в дальнейшей работе, проводится анализ научных публикаций по тематике практического применения алгоритмов классификации текста, а также ставится задача на реализацию текстовых классификаторов с определением используемых при реализации программных средств.

Во второй главе описывается тестирование реализованных текстовых классификаторов на эффективность: обосновывается его необходимость, формируются условия проведения экспериментальных запусков, приводятся метрики оценки результатов тестирования, а также используемые в процессе тестирования программные инструменты. Проводится анализ результатов тестирования, на основании которых определяется оптимальный набор используемых параметров текстового классификатора.

Данная работа состоит из пояснительной записки общим объемом 45 страниц, включая 6 рисунков, 7 таблиц, список литературы из 30 источников и 1 приложение. В процессе выполнения выпускной квалификационной работы

по ее материалам были опубликованы две научные статьи в сборнике трудов научно-практических конференций.

Глава 1 ЗАДАЧА КЛАССИФИКАЦИИ ТЕКСТА И ПОДХОДЫ К ЕЕ РЕШЕНИЮ

1.1 Алгоритмы автоматической классификации текста

Алгоритмы, решающие задачу классификации текста, принято объединять названием Text mining. Text mining (интеллектуальный анализ текстов) является частым случаем data mining (интеллектуальный анализ данных). Это одно из направлений развития искусственного интеллекта, целью которого является получение информации из коллекции текстовых документов путем применения эффективных с практической точки зрения алгоритмов.

Алгоритмы автоматической классификации текста можно разделить на две группы:

1. Алгоритмы, в основе которых лежит построение определенных правил, по совокупности которых впоследствии происходит процесс классификации [28]. Сами правила строятся вручную по принципу «ЕСЛИ-ТО» (англ. “IF-THEN”). Такие алгоритмы принято называть rule-based, то есть основанные на правилах. Основное преимущество rule-based алгоритмов – они могут дать лучшую точность классификации, однако для этого требуется привлечь к составлению правил эксперта в конкретной предметной области, а также периодически обновлять набор правил, если это потребуется.

2. Алгоритмы, основанные на машинном обучении. Перед тем, как приступать к классификации, такие алгоритмы обучаются на специальной выборке элементов, называемой «обучающая выборка». В результате обучения формируются определенные критерии (правила), по которым и происходит дальнейшая классификация. Особенность данных алгоритмов в том, что для их обучения обязательно требуется обучающая выборка, причем ее размер может быть достаточно большим. Данные внутри такой выборки должны быть размечены вручную, как правило с привлечением экспертов.

Очевидно, что применение rule-based алгоритмов требует глубокого знания предметной области для ручного создания набора правил, либо привлечение эксперта. Поскольку привлечение эксперта невозможно, а для

использования алгоритмов машинного обучения имеется подходящая обучающая выборка, то их использование является целесообразным в рамках решаемой задачи.

1.2 Описание решаемой задачи классификации текста

1.2.1 Постановка задачи и исходные данные

Текстовый классификатор можно получить, решив задачу обучения по прецедентам (1):

$$X, Y, y^*, X^M \tag{1}$$

где X – множество классифицируемых документов;

Y – множество меток классов;

$y^*: X \to Y$ – целевая зависимость, значения которой известны только на элементах обучающей выборки $X^M = (x_i, y_i)_{i=1}^M$, в которых x_i – текстовые данные, а $y_i = y^*(x_i)$ – метка класса.

Алгоритм $\alpha: X \to Y$, аппроксимирующий целевую зависимость на всем пространстве X, будет являться решением данной задачи [1].

Требуется реализовать в виде программного обеспечения эффективный текстовый классификатор, способный классифицировать поступающие в него текстовые данные в режиме реального времени. Классификация должна производиться по 8 определенным заранее классам документов. Для предварительного обучения классификатора предоставляется обучающая выборка, состоящая из 11757 элементов. Распределение элементов между классами неравномерное (рисунок 1).

Текстовые данные элементов выборки представлены на естественном языке (русском) без какой-либо предварительной обработки и представляют собой архив обращений граждан города в адрес городской администрации по различным вопросам, распределенные по своей тематике в соответствующие классы документов.

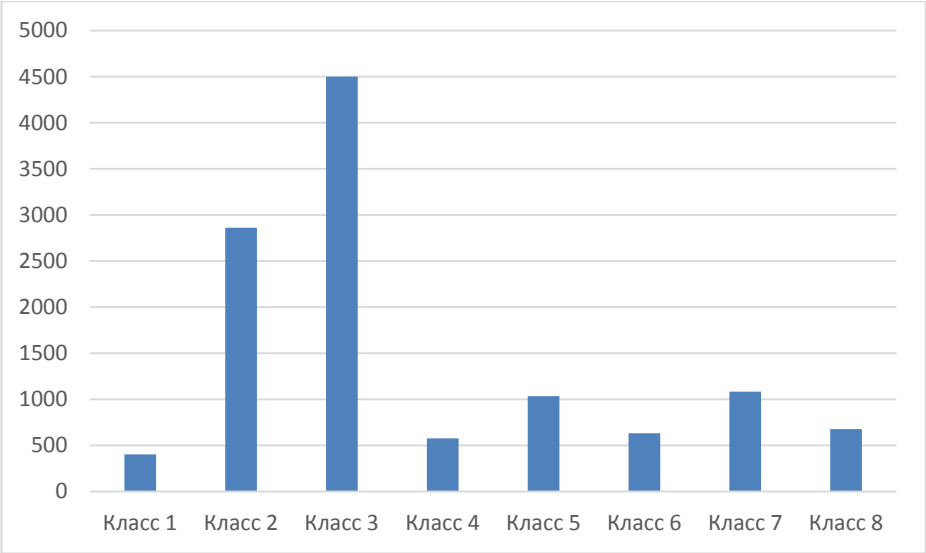

Рисунок 1 – Распределение элементов обучающей выборки по классам

С учетом применения алгоритмов машинного обучения в качестве алгоритмов классификации, общую схему работы текстового классификатора можно разбить на два основных процесса:

1. Обучение алгоритма классификации.

2. Классификация новых текстовых документов.

Процесс обучения текстового классификатора можно представить в следующем виде (рисунок 2):

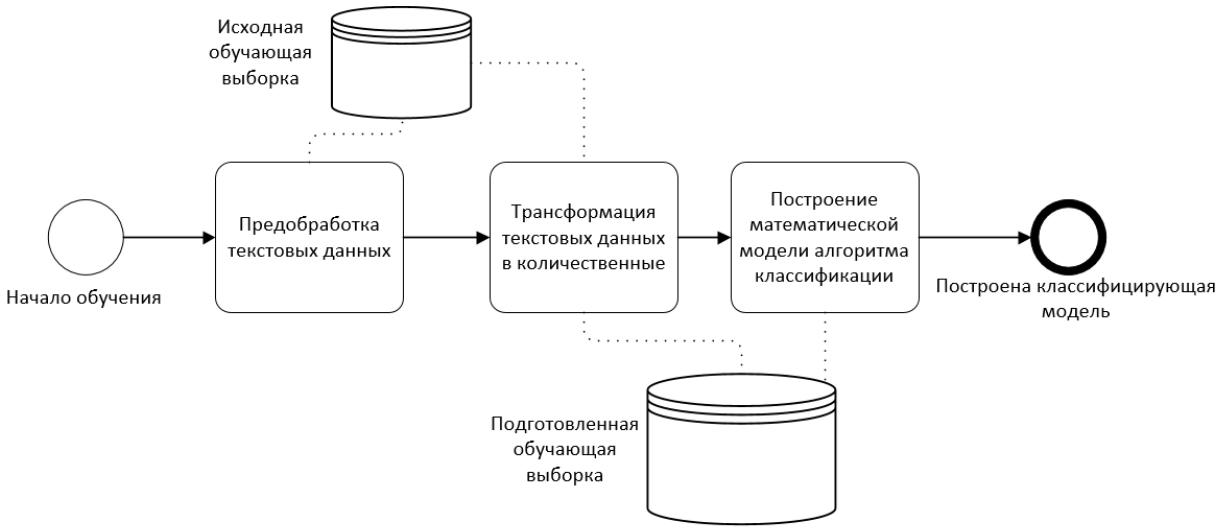

Рисунок 2 – Схема процесса обучения текстового классификатора

После выполнения обучения, текстовый классификатор может выполнять классификацию новых документов (рисунок 3).

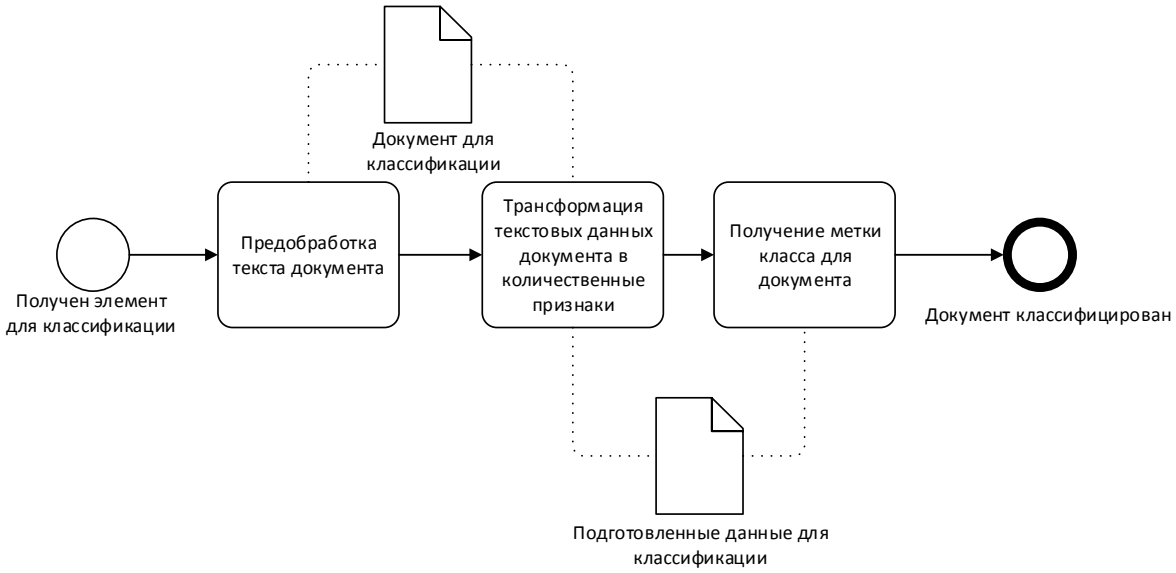

Рисунок 3 – Схема процесса классификации нового документа

Важно отметить, что предобработка текста и его трансформация в количественные признаки должна происходить по одинаковым правилам как в процессе обучения, так и в процессе классификации.

Рассмотрим подробнее этапы предобработки текста и его трансформации в количественные признаки для выявления имеющихся особенностей.

1.2.2 Предобработка текстовых данных

Несмотря на то, что выбор алгоритма может значительно повлиять на реализуемую технологию классификации текста, сравнительные исследования алгоритмов говорят о том, что в различных ситуациях алгоритмы показывают различные результаты [19] [21]. Таким образом, нельзя сказать, что какой-либо алгоритм классификации работает определенно лучше других. Все они имеют определенные особенности работы, которые влияют на оценку их эффективности в зависимости от предметной области, которой принадлежит задача классификации. Гораздо большее влияние на точность работы алгоритма оказывает выборка данных, используемая в процессе обучения классификатора. Поскольку местом для поиска обучающей выборки зачастую служат открытые источники данных, обучающая выборка, собранная, например, из комментариев пользователей социальной сети, либо из другого любого открытого источника,

будет достаточно низкого качества. Это связано с тем, что люди используют в повседневном общении неформальный стиль речи, допускают ошибки при письме, могут некорректно выражать свои мысли и так далее.

С учетом вышесказанного, обучающая выборка, собранная на основе определенного массива данных (источника) без проведения предобработки будет содержать в себе большое количество «шумов» - то есть текстовой информации, не несущей смысловой нагрузки для алгоритма классификации. Такие «шумы» могут значительно снизить точность работы классификатора, так как извлечение полезной информации из зашумленной выборки является более сложной задачей [22]. Именно в связи с этим при реализации технологии классификации текста большее внимание уделяется обучающей выборке и ее предобработке, чем выбору алгоритма и его настройке.

Предобработку текста можно разбить на отдельные операции, при этом действия, выполняемые в процессе каждой операции, обрабатывают текст различными способами. Среди основных методов предобработки текстовых данных следующие [29]:

1. Перевод всех букв в нижний регистр.

2. Удаление знаков пунктуации и чисел.

3. Удаление стоп-слов (часто встречающиеся слова, не несущие значимого смысла при решении задачи классификации текста, например, предлоги, союзы и т.п.).

4. Удаление слов, которые встречаются крайне редко в тексте, либо наоборот чаще остальных.

5. Исправление слов, написанных с ошибками.

При выборе методов предобработки, помимо указанных выше, также используют либо лемматизацию, либо стемминг. Цель обоих процессов – приведение слов к общей базовой форме (основе слова), при этом основа слова может не совпадать с корнем слова. Несмотря на одинаковую цель, стемминг и лемматизация имеют коренные отличия в процессе работы. Алгоритм стемминга как правило подразумевает достаточно грубое отсечение конечных

символов у слов в надежде на получение в результате основы слова, то есть является эвристическим методом. Лемматизация же основана на более формальном подходе, использует морфологический анализ и словарь.

Стоит отметить, что по результатам исследований выбор именно оптимального набора методов предобработки повышает качество классификации текста, в отличии от использования всех доступных способов предобработки или их неиспользования вообще [23]. Выбрать же оптимальный набор используемых методов предобработки возможно только при проведении экспериментальных практических исследований эффективности работы классификатора с различными наборами методов предобработки обучающей выборки.

1.2.3 Трансформация текстовых данных в количественные признаки

Важным шагом при построении текстового классификатора является трансформирование обучающей выборки из текстового представления в числовой формат, поскольку все используемые алгоритмы классификации работают с числовыми значениями. Данный этап выполняется строго после предобработки текстов обучающей выборки, чтобы уменьшить число различных вариантов лексем (слов, встречающиеся в текстах выборки).

Результатом такой трансформации может являться матрица «документ-лексема», которая представляет собой таблицу, в строках которой указываются номера документов (элементов обучающей выборки), в столбцах – лексемы, а на пересечении столбца и строки указывается число повторений лексемы в конкретном документе. Пример такой матрицы представлен в таблице 1.

Таблица 1 – Пример матрицы «документ- лексема»

	Лексема 1	Лексема 2	…	Лексема M
Документ 1	0	1	…	2
Документ 2	0	10	…	5
…	…	…	…	0
Документ N	23	0	6	9

Однако при работе с текстовыми данными различные лексемы могут иметь различный вес в рамках всей выборки. Для нормализации построенной матрицы «документ-лексема» применятся статистическая мера TF-IDF. При ее использовании вес конкретной **лексемы** (lex) пропорционален частоте употребления этой лексемы в документе (d) и обратно пропорционален частоте употребления слова во всех документах выборки (D):

$$TFIDF\ \ lex, d, D\ \ = TF\ \ lex, d\ \ * IDF(lex, D) \tag{2}$$

$$TF\ \ lex, d\ \ = \frac{\text{число вхождений лексемы } lex \text{ в документе } d}{\text{число слов в документе } d} \tag{3}$$

$$IDF\ \ lex, D\ \ = \ ln \frac{\text{число документов в коллекции } D}{\text{число документов,в которых встречается } lex} \tag{4}$$

Частота лексемы (TF) оценивает важность лексемы в пределах одного документа, а обратная частота документа (IDF) уменьшает веса часто встречающихся слов. За счет этого мера TF-IDF позволяет нормировать полученную матрицу.

При реализации текстового классификатора допустимо использовать обычную матрицу «документ-лексема» для представления текстовых данных как количественных признаков, однако в некоторых случаях применение статистической меры TF-IDF для нормализации полученной матрицы может привести к повышению эффективности классификации. Действительную оценку результата возможно получить, проведя экспериментальные сравнения работы текстового классификатора.

1.3 Выбор используемых алгоритмов машинного обучения для решаемой задачи классификации текста

К классическим алгоритмам машинного обучения, способным решить задачу классификации (в том числе текстовых документов) относят [5]:

1. Деревья решений.
2. Метод опорных векторов (Support Vector Machine).
3. Алгоритм K ближайших соседей (KNN).
4. Нейронные сети.

Рассмотрим данные алгоритмы для определения возможности их применения в рамках решаемой задачи. Данные для составления сравнительных характеристик взяты из источников [24]. Характеристика составлена по следующим критериям, важным в условиях решаемой задачи:

1. Требовательность к качеству обучающей выборки – наличие у алгоритма значимого снижения эффективности работы классифицирующей модели под воздействием факта наличия шумов в обучающей выборке. Оценки по данному критерию: 1 балл – алгоритм требователен к качеству обучающей выборки; 0 баллов – алгоритм не требователен к качеству обучающей выборки.

2. «Ленивый» алгоритм – особенность работы алгоритма, заключающаяся в необходимости сохранять в памяти всю обучающую выборку после завершения обучения алгоритма. Оценки по данному критерию: 1 балл – «ленивый» алгоритм; 0 баллов – алгоритм не является «ленивым».

3. Вычислительная сложность обучения – характеризует требовательность к аппаратному обеспечению в процессе обучения алгоритма. Оценки по данному критерию: 3 балла – высокая вычислительная сложность; 2 балла – умеренная вычислительная сложность; 1 балл – низкая вычислительная сложность.

4. Вычислительная сложность классификации – характеризует требовательность к аппаратному обеспечению в процессе классификации алгоритмом новых документов. Оценки по данному критерию: 3 балла – высокая вычислительная сложность; 2 балла – умеренная вычислительная сложность; 1 балл – низкая вычислительная сложность.

5. Сложность адаптации алгоритма для его применения в конкретной задаче – необходимость ручной (либо долгой) настройки довольно многих параметров алгоритма перед его использованием. Оценки по данному критерию: 3 балла – высокая сложность адаптации; 2 балла – умеренная сложность адаптации; 1 балл – низкая сложность адаптации.

Характеристики алгоритмов классификации приведены в таблице 2.

Таблица 2 – Сравнительные характеристики алгоритмов классификации текста

Алгоритм / **Критерий**	**Деревья решений**	**Метод опорных векторов**	**KNN**	**Нейронные сети**
Требовательность к качеству обучающей выборки	1	1	1	1
«Ленивый» алгоритм	0	0	1	0
Вычислительная сложность обучения	2	1	3	2
Вычислительная сложность классификации	2	1	3	2
Сложность адаптации алгоритма для его применения в конкретной задаче	1	2	1	3
ИТОГО баллов (меньше – лучше)	**6**	**5**	**9**	**8**

Анализ характеристик алгоритмов для применения в решаемой задаче классификации текста показывает:

1. Алгоритмы построения деревьев решений обладают умеренными требованиями к аппаратному обеспечению, а также не требуют долгой настройки для использования, поскольку используемые в них методы не привязаны к условиям конкретной задачи, и являются универсальными.

2. Метод опорных векторов является наиболее быстрым методом нахождения решающей функции, а дальнейшая классификация является достаточно точной из-за особенностей математического аппарата [3]. Недостатком является довольно тонкая настройка алгоритма, сложность которой можно считать умеренной. Несмотря на это, алгоритм имеет смысл применить в решаемой задаче, поскольку скорость обучения, скорость классификации и точность классификации алгоритма являются его значимыми преимуществами.

3. Алгоритм K ближайших соседей можно считать практически бесполезным с точки зрения его применимости в рассматриваемой задаче,

поскольку его использование потребует высоких затрат аппаратных ресурсов, что связано с особенностями его работы.

4. Нейронные сети обладают преимуществами, которые присущи методу опорных векторов и алгоритмам построения деревьев решений. Однако практическое применение нейронных сетей для решения задачи классификации текста имеет значительный недостаток – сложность создания архитектуры нейронной сети, требуемой для классификационной модели, может оказаться достаточно высокой. Данная особенность критична при построении текстового классификатора, так как построение модели нейронной сети в данном случае является неоправданно сложной задачей.

Таким образом, в имеющихся условиях оправдано применение алгоритмов построения деревьев решений и метода опорных векторов.

Существует множество алгоритмов построения деревьев решений, однако так как задача классификации текста отличается повышенной сложностью, разумно использовать более эффективные алгоритмы, тем не менее основанные на деревьях решений – алгоритмы построения ансамблей деревьев решений. В отличии от классических алгоритмов создания деревьев решений, таких как CART и C4.5, алгоритмы построения ансамблей деревьев решений в процессе создания классифицирующей модели создают множество небольших деревьев решений, каждое из которых впоследствии используется при классификации новых документов. Такой подход обеспечивает повышенную эффективность при решении задачи классификации текста [27].

Главным параметром метода опорных векторов является выбор используемого алгоритмом ядра. По вопросу сравнения эффективности ядер метода опорных векторов проведено достаточное количество научных экспериментов. В данной работе будет использоваться линейное ядро, поскольку оно обеспечивает наилучшие показатели по скорости обучения, точности и скорости классификации [26].

1.4 Постановка задачи на реализацию текстовых классификаторов

Рассмотрев описание решаемой задачи и используемые средства ее решения, ставится задача на программную реализацию текстовых классификаторов, работающих в различных режимах. Разрабатываемые классификаторы должны соответствовать следующим требованиям:

1. Использовать имеющуюся обучающую выборку для обучения алгоритма классификации.

2. Иметь возможность классификации новых текстовых документов, поступающих в программу в режиме реального времени.

3. В процессе предобработки текстов обучающей выборки должны использоваться различные сочетания описанных ранее методов предобработки.

4. Для представления текстовых данных в виде количественных признаков должна использоваться матрица «документ-лексема» с опциональной возможностью применения меры TF-IDF.

5. В качестве алгоритма классификации использовать либо алгоритм random forest, либо метод опорных векторов

В качестве языка программирования для реализации текстовых классификаторов выбран язык Python, поскольку он широко применяется экспертами в области машинного обучения при решении конкретных задач, обладает большим набором доступных библиотек машинного обучения, и при этом поддерживает объектно-ориентированное программирование, что позволяет применить при разработке ПО стандартные методы объектно-ориентированного проектирования.

Для соответствия поставленным требованиям к текстовым классификаторам, будут применяться программные реализации алгоритмов классификации и предобработки, содержащиеся в библиотеках языка Python:

- метод опорных векторов – класс LinearSVC библиотеки scikit-learn;
- алгоритм Random forest – класс RandomForestClassifier библиотеки scikit-learn.

Используемые технологии предобработки:

- список стоп-слов русского языка берется из модуля nltk.corpus;
- функционал для построения матрицы «документ-лексема» – класс CountVectorizer библиотеки scikit-learn;
- нормализация матрицы «документ-лексема» по TF-IDF – класс TfidfTransformer библиотеки scikit-learn;
- реализация алгоритма стемминга – класс Snowball stemmer модуля nltk;
- реализация алгоритма лемматизации – ПО MyStem от компании «Яндекс».

Реализовав текстовые классификаторы, способные работать в различных режимах, будет проведено их тестирование на эффективность, которое сможет выявить набор оптимальных алгоритмов и параметров, обеспечивающих наилучшую эффективность работы текстового классификатора для решаемой задачи. После чего, полученный набор параметров будет использован в финальном программном продукте – текстовом классификаторе, решающем поставленную задачу классификации, который будет использован как модуль цельной информационной системы.

Основное требование к реализации данного текстового классификатора – возможность его встраивания в виде модуля информационной системы, который будет классифицировать поступающие в него текстовые документы. Так как реализация и текстового классификатора, и информационной системы выполняется на языке Python, для выполнения данного требования достаточно соответствия сигнатуры метода классификации следующим требованиям:

1. Обязательный параметр метода – текст для классификации.
2. Возвращаемое значение – метка класса для входного текста.

С учетом этого, диаграмма классов текстового классификатора представлена на рисунке 4.

Диаграмма последовательности, показывающая работу текстового классификатора как интегрируемого модуля цельной информационной системы, показана на рисунке 5.

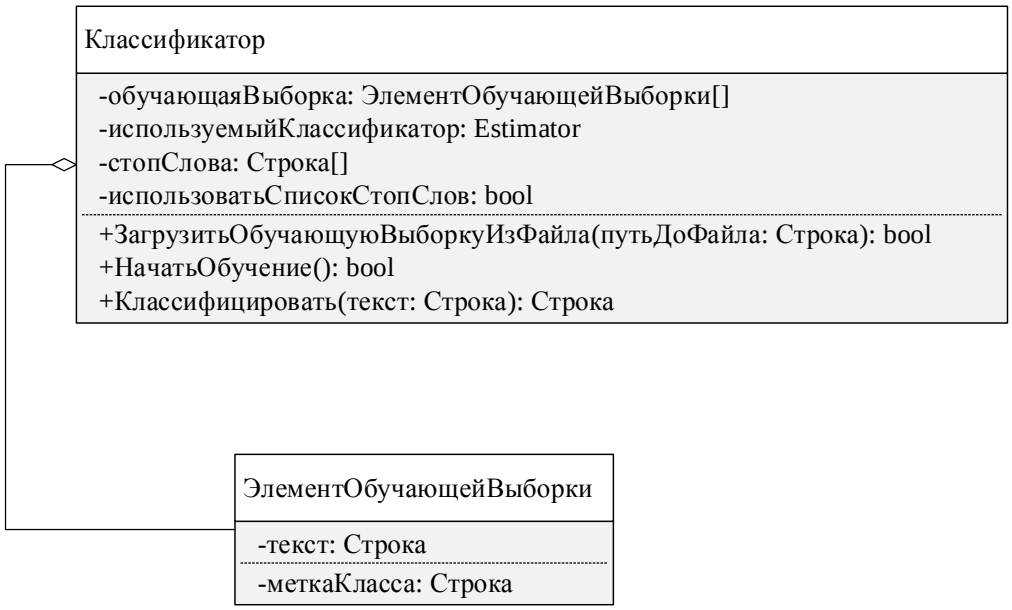

Рисунок 4 – Диаграмма классов текстового классификатора

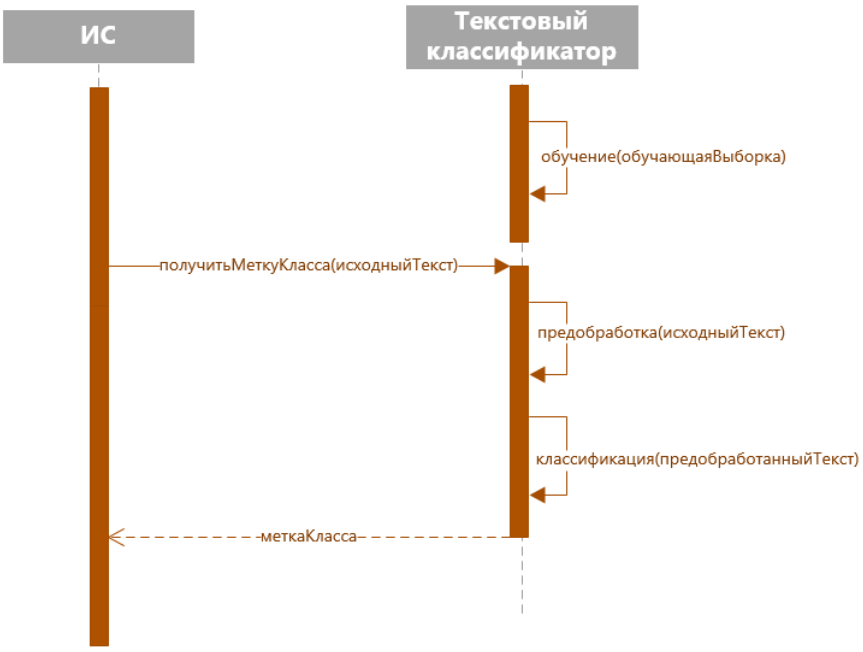

Рисунок 5 – Диаграмма последовательности для процесса взаимодействия классификатора при его встраивании в виде модуля системы

Сам текстовый классификатор представляет собой класс с тремя методами:

- метод для загрузки элементов обучающей выборки из файла в оперативную память;

- метод для обучения классификатора на загруженных элементах;
- метод для классификации новых текстовых документов.

Поля класса имеют следующее назначение:

- обучающаяВыборка: массив для хранения элементов обучающей выборки;
- используемыйКлассификатор: объект, реализующий интерфейс Estimator библиотеки scikit-learn;
- стопСлова: список стоп-слов;
- использоватьСписокСтопСлов: переменная булева типа, определяющая использование списка стоп слов при обучении и классификации.

После встраивания и настройки каналов связи между текстовым классификатором и целостной информационной системой, необходимо провести обучение текстового классификатора, после чего он готов к работе. В процессе работы текстовый классификатор принимает текст, подлежащий классификации, после чего проводит его предобработку (в соответствии с примененными методами предобработки обучающей выборки – они должны совпадать для корректной работы) и непосредственно классификацию. Сформированная метка класса возвращается в место, откуда был получен запрос на классификацию.

1.5 Выводы по главе 1

В главе была описана решаемая задача классификации текста, выбраны используемые средства ее решения и имеющиеся исходные данные. Также была поставлена задача на программную реализацию текстовых классификаторов, способных работать в различных режимах на основе выбранных алгоритмов классификации текста, методов предобработки обучающей выборки и трансформации текстовых данных в количественные признаки. Реализовав данные текстовые классификаторы, с их помощью будет экспериментально определен оптимальный набор параметров алгоритма и методов предобработки в условиях решаемой задачи. Полученный

оптимальный набор параметров будет использоваться в финальной версии разрабатываемого текстового классификатора.

Глава 2 ТЕСТИРОВАНИЕ ЭФФЕКТИВНОСТИ ТЕКСТОВЫХ КЛАССИФИКАТОРОВ

2.1 Обоснование необходимости проведения эксперимента

Чтобы обеспечить наибольшую эффективность финальной версии текстового классификатора, необходимо провести эксперименты, тестирующие его работу в разных условиях.

Целью тестирования является повышение качества конечного программного продукта. Объекты, подлежащие тестированию – алгоритмы классификации текстов, работающие при определяемых правилами эксперимента условиях.

По материалам ранее рассмотренных источников, на эффективность работы текстового классификатора значительное влияние оказывает предобработка обучающей выборки. Но при рассмотрении конкретной практической задачи, правила и закономерности, полученные по результатам других исследований, могут потерять свою силу, из-за особенностей условий решаемой задачи. В данном случае самой значительной особенностью может стать обучающая выборка по причинам, рассмотренным в первой главе. Поэтому в процессе тестирования текстового классификатора необходимо рассмотреть вопрос эффективности его работы при различных наборах используемых методов предобработки обучающей выборки, и выделить наиболее оптимальные комбинации. Однако не исключено, что и используемый алгоритм классификации также окажет значительное влияние на эффективность классификации текстов, что также требуется проверить экспериментально.

Таким образом, для обеспечения максимального качества работы текстового классификатора, необходимо провести серию экспериментов и проанализировать их результаты. На основании полученных данных будет возможно сделать оптимальный выбор используемых алгоритмов и методов предобработки обучающей выборки, которые обеспечат качественную работу финальной версии текстового классификатора.

2.2 Описание условий режимов работы текстовых классификаторов

Тестированию подлежат следующие режимы работы текстовых классификаторов:

1. Используемый алгоритм классификации: метод опорных векторов.

1.1. При построении матрицы «документ-лексема» фильтровать слова, которые встречаются чаще чем (max_df): в 75% документов; в 50% документов; параметр не используется.

1.2. Максимальное количество терминов в матрице «документ- лексема» (max_features): без ограничений; 5000; 10000.

1.3. Минимальная частота повторения слова для его попадания в матрицу «документ-лексема» (min_df): 5; 10.

1.4. Использовать фильтрацию по списку стоп-слов: да; нет.

1.5. Использовать IDF на матрице (use_idf): да; нет.

1.6. Параметр C (определяет соотношение между гладкостью границы гиперплоскости и ошибкой классификации элементов): 0.5; 1; 10.

1.7. Использовать лемматизатор MyStem на обучающей выборке: да; нет.

2. Используемый алгоритм классификации: random forest.

2.1. При построении матрицы «документ-лексема» фильтровать слова, которые встречаются чаще чем (max_df): в 75% документов; в 50% документов; параметр не используется.

2.2. Максимальное количество терминов в матрице «документ-лексема» (max_features): без ограничений; 5000; 10000.

2.3. Минимальная частота повторения слова для его попадания в матрицу «документ-лексема» (min_df): 5; 10.

2.4. Использовать фильтрацию по списку стоп-слов: да; нет.

2.5. Использовать IDF на матрице (use_idf): да; нет.

2.6. Критерий разбиения в узле при построении деревьев решений: Джини; энтропия.

2.7. Количество деревьев решений в ансамбле: 100; 1000; 3000.

2.8. Использовать лемматизатор MyStem на обучающей выборке: да; нет.

При тестировании не рассматривается использование алгоритма стемминга вместо алгоритма лемматизации, поскольку в процессе написания работы было выяснено, что применение алгоритма стемминга на данной обучающей выборке не приводит к значительному улучшению тестируемых показателей, о чем была опубликована статья в сборнике трудов конференции [2].

Для проведения эксперимента используется реализация метода перекрестной проверки (кросс-валидации) GridSearchCV библиотеки scikit-learn. Экспериментальные запуски производятся под ОС Ubuntu 16.04, версия языка Python 3.7.2, версия библиотеки scikit-learn 0.21.0, центральный процесс AMD A10-9620P, распараллеливание вычислений на 4 потока происходит встроенными средствами класса GridSearchCV.

2.3 Метрики оценки эффективности текстовых классификаторов

В качестве метода оценки работы классификатора используется алгоритм кросс-валидации, в процессе которого исходная обучающая выборка разбивается на 5 подмножеств, после чего каждое подмножество выступает как тестовая выборка при условии обучения классификатора элементами всех остальных подмножеств. Таким образом, для получения оценки эффективности текстового классификатора для каждого из наборов параметров запуска требуется провести пять процедур обучения классификатора и проверки тестовой выборки.

В случае мультиклассовой классификации текстов с неравномерным распределением документов по классам, применение очевидной метрики точности, основанной на вычислении отношения количества верно угаданных документов к их общему количеству, не является оптимальным решением. Это связано с тем, что такая метрика считает все имеющиеся текстовые документы равнозначными по весу, хотя размеры классов не равны между собой. Вследствие чего возможна ситуация, при которой текстовый классификатор будет хорошо работать с объектами одного класса, содержащего много

документов, и значение качества классификации будет достаточно высоким, однако на классах меньшего размера точность будет недостаточна, однако используемая метрика не будет учитывать данный факт.

Для решения обозначенной проблемы и правильной оценки эффективности работы текстового классификатора существуют следующие общепринятые метрики [24] [30]:

1. Точность (англ. “precision”) – процент документов, отнесенных классификатором к конкретному классу и действительно являющихся членами данного класса. Precision показывает способность алгоритма отличать конкретный класс документов от других классов.

2. Полнота (англ. “recall”) – количество объектов конкретного класса с верно угаданной меткой класса. Данная метрика показывает, насколько хорошо алгоритм способен обнаруживать данный класс документов в целом.

3. F-мера – среднее гармоническое мер precision и recall. F-мера используется для нахождения оптимального баланса между двумя параметрами, поскольку в практических задачах достичь одновременного максимума обоих метрик практически невозможно. Значение F-меры близко к нулю, если точность или полнота близки к нулю, и вычисляется по формуле (5):

$$F = (\beta^2 + 1)\frac{Precision * Recall}{\beta^2 * Precision * Recall} \quad (5)$$

Изменением параметра β задаются веса параметрам precision и recall. Так, если $0 < \beta < 1$, то приоритет отдается параметру точности, а при $\beta > 1$ больший приоритет имеет полнота. В случае $\beta = 1$, веса для полноты и точности равны, и F-мера в данном случае называется сбалансированной.

Для каждого из семи классов данные метрики рассчитываются отдельно, чтобы обеспечить наглядное представление эффективности условий того или иного запуска текстового классификатора. Итоговое значение по каждой метрике вычисляется как среднее арифметическое значений данной метрики по всем классам.

В экспериментальных запусках используется сбалансированная f-мера, поскольку и полнота, и точность классификации являются одинаково важными параметрами в решаемой задаче классификации.

2.4 Анализ результатов экспериментальных исследований

Результаты экспериментальных запусков для метода опорных векторов, включающие в себя все сочетания указанных выше параметров, представлены в таблице 3 (с предварительным использованием лемматизатора на обучающей выборке) и таблице 4 (без использования лемматизатора на обучающей выборке). В каждой таблице сначала представлены 10 наилучших результатов, затем – 10 наихудших.

Таблица 3 – Результаты экспериментальных запусков для метода опорных векторов (с лемматизатором)

№	Параметры алгоритма и предобработки						Среднее значение f-меры
	C	use_idf	max_df	max_features	min_df	Фильтр стоп-слов	
1	0.5	True	0.5	без огр.	5	Да	0.788968
2	0.5	True	0.75	без огр.	5	Да	0.788968
3	0.5	True	1	без огр.	5	Да	0.788968
4	0.5	True	1	10000	5	Нет	0.787368
5	0.5	True	1	без огр.	5	Нет	0.787368
6	1	True	1	без огр.	5	Нет	0.786863
7	1	True	0.5	без огр.	5	Да	0.786779
8	1	True	0.75	без огр.	5	Да	0.786779
9	1	True	1	без огр.	5	Да	0.786779
10	0.5	True	0.75	без огр.	5	Нет	0.786695
11	10	False	0.75	5000	10	Нет	0.743663
12	10	True	0.5	5000	5	Нет	0.743579
13	10	True	0.5	5000	10	Нет	0.743495
14	10	True	0.75	5000	5	Нет	0.742989
15	10	False	0.75	5000	5	Нет	0.742568
16	10	True	1	5000	5	Нет	0.742232
17	10	False	0.5	5000	5	Нет	0.741558
18	10	False	0.5	5000	10	Нет	0.741474
19	10	True	1	5000	10	Нет	0.740884
20	10	True	0.75	5000	10	Нет	0.740379

Таблица 4 – Результаты экспериментальных запусков для метода опорных векторов (без лемматизатора)

№	Параметры алгоритма и предобработки						Среднее значение f-меры
	C	use_idf	max_df	max_features	min_df	Фильтр стоп-слов	
1	0.5	True	1	без огр.	5	Да	0.774821
2	0.5	True	0.75	без огр.	5	Да	0.774737
3	0.5	True	0.5	без огр.	5	Да	0.774737
4	0.5	True	1	10000	5	Нет	0.773053
5	0.5	True	1	без огр.	5	Нет	0.772968
6	1	True	0.75	без огр.	5	Да	0.772884
7	1	True	1	без огр.	5	Да	0.772884
8	1	True	0.5	без огр.	5	Да	0.772884
9	0.5	True	0.5	10000	5	Нет	0.772800
10	0.5	True	0.75	10000	5	Нет	0.772800
11	10	True	1	5000	5	Нет	0.718232
12	10	True	0.75	5000	5	Нет	0.717726
13	10	True	0.5	5000	5	Нет	0.717474
14	10	True	0.5	5000	10	Нет	0.717474
15	10	True	0.75	5000	10	Нет	0.717389
16	10	True	1	5000	10	Нет	0.716884
17	10	False	1	5000	5	Нет	0.716884
18	10	True	0.5	10000	10	Нет	0.716716
19	10	True	0.75	10000	10	Нет	0.716211
20	10	False	0.5	5000	5	Нет	0.715284

По полученным результатам видно, что наибольшее влияние на работу метода опорных векторов оказывает параметр C, поскольку все наилучшие результаты были получены при C = 0.5, либо C = 1. Также важным параметром можно назвать ограничение по количеству лексем в матрице «документ-лексема»: метод опорных векторов в целом лучше работает, если ограничение по количеству лексем в матрице не существует. Использование лемматизатора дает улучшение f-меры в среднем на 2% по сравнению с запусками без его использования. Наилучшим набором параметров метода опорных векторов и средств предобработки является строка 1 в таблице 3, что обеспечивает значение f-меры, близкое к 79%.

Результаты экспериментальных запусков для алгоритма random forest, представлены в таблице 5 (с предварительным использованием лемматизатора

на обучающей выборке) и таблице 6 (без использования лемматизатора на обучающей выборке). В каждой таблице сначала представлены 10 наилучших результатов, затем – 10 наихудших.

Таблица 5 – Результаты алгоритма random forest (с лемматизатором)

№	Параметры алгоритма и предобработки							Среднее значение f-меры
	Критерий разбиения	**Кол-во деревьев**	**use idf**	**max df**	**max features**	**min_df**	**Фильтр стоп-слов**	
1	Джини	150	Да	0.5	10000	10	Да	0.753095
2	Джини	100	Да	1	10000	5	Да	0.752512
3	Джини	150	Да	0.75	10000	5	Да	0.752253
4	Джини	150	Да	0.5	5000	5	Да	0.752168
5	Джини	150	Нет	1	5000	10	Да	0.752168
6	Джини	150	Да	1	5000	5	Да	0.752084
7	Джини	150	Да	1	5000	10	Да	0.751916
8	Джини	150	Нет	0.5	без огр.	10	Да	0.751747
9	Джини	100	Нет	0.75	10000	10	Да	0.751663
10	Джини	150	Да	1	10000	10	Да	0.751495
11	Энтропия	50	Нет	1	10000	10	Нет	0.719579
12	Энтропия	50	Нет	1	без огр.	5	Нет	0.719242
13	Энтропия	50	Да	0.75	без огр.	5	Нет	0.718400
14	Энтропия	50	Нет	1	5000	5	Нет	0.717811
15	Энтропия	50	Нет	1	10000	5	Нет	0.717221
16	Энтропия	50	Нет	0.75	без огр.	5	Нет	0.717137
17	Энтропия	50	Да	1	без огр.	5	Нет	0.715874
18	Энтропия	50	Да	0.75	10000	5	Нет	0.715705
19	Энтропия	50	Да	1	10000	5	Нет	0.715284
20	Энтропия	50	Нет	0.75	10000	5	Нет	0.715032

Таблица 6 – Результаты экспериментальных запусков для алгоритма random forest (без лемматизатора)

№	Параметры алгоритма и предобработки							Среднее значение f-меры
	Критерий разбиения	**Кол-во деревьев**	**use idf**	**max df**	**max features**	**min_df**	**Фильтр стоп-слов**	
1	Джини	150	Нет	1	без огр.	5	Да	0.726905
2	Джини	150	Нет	0.75	без огр.	5	Да	0.726568
3	Джини	150	Да	0.5	10000	5	Да	0.725221
4	Джини	150	Да	0.75	10000	5	Да	0.724547

№	Параметры алгоритма и предобработки							Среднее
5	Джини	150	Да	0.75	без огр.	5	Да	0.723789
6	Джини	150	Да	1	без огр.	5	Да	0.723705
7	Джини	150	Нет	1	без огр.	10	Да	0.723453
8	Джини	150	Нет	0.75	10000	10	Да	0.723200
9	Джини	150	Нет	0.75	без огр.	10	Да	0.723032
10	Джини	100	Да	0.75	10000	10	Да	0.722947
11	Энтропия	50	Да	0.75	без огр.	10	Нет	0.681516
12	Энтропия	50	Да	1	без огр.	10	Нет	0.681432
13	Энтропия	50	Нет	1	10000	10	Нет	0.681011
14	Энтропия	50	Нет	0.75	без огр.	10	Нет	0.679663
15	Энтропия	50	Да	1	10000	10	Нет	0.679579
16	Энтропия	50	Да	1	10000	5	Нет	0.679158
17	Энтропия	50	Нет	1	10000	5	Нет	0.679074
18	Энтропия	50	Да	1	без огр.	5	Нет	0.678400
19	Энтропия	50	Нет	1	без огр.	5	Нет	0.678316
20	Энтропия	50	Да	0.75	10000	5	Нет	0.678232

Рассмотрев данные экспериментальных запусков алгоритма random forest, можно сделать вывод, что наилучшим набором параметров является строка 1 таблицы 5, что обеспечивает значение f-меры ~0,75.

По результатам экспериментальных запусков алгоритма random forest также видно, что использование критерия Джини для разбиения узла дерева в процессе его построения является предпочтительным вариантом, поскольку экспериментальные запуски с применением данного критерия показывают наилучшие результаты. Помимо этого, повышение количества деревьев решений в ансамбле (в совокупности с применением правильных наборов остальных параметров) ведет к повышению значения f-меры. Как и в случае с методом опорных векторов, предварительное использование лемматизатора на обучающей выборке дает повышение среднего значения f-меры примерно на 2%.

В отличии от метода опорных векторов, для которого оптимальным оказалось отсутствие ограничения по количеству лексем в матрице «документ-лексема», алгоритм random forest работает эффективнее, если такое ограничение присутствует в процессе построения модели. Сравнивая результаты тестирования обоих рассматриваемых алгоритмов также видно, что

эффективной стратегией в обоих случаях является использование фильтра стоп-слов.

На основании проведенного эксперимента можно сделать следующие общие выводы, связанные с вопросами реализации эффективных текстовых классификаторов:

1. В ходе анализа этапов реализации текстового классификатора, были найдены научные работы, содержащие выводы о том, что наиболее действенным подходом к увеличению качества работы текстового классификатора является выполнение предобработки обучающей выборки, при этом воздействие используемого алгоритма классификации на качество классификации представлялось как не столь существенное. Данный вывод может быть частично опровергнут полученными результатами тестирования классификаторов на эффективность. В действительности, проведение правильной предобработки обучающей выборки повышает итоговую оценку качества классификации, и важным фактором в данной ситуации является выявление наиболее оптимального набора методов предобработки, которые могут быть получены только экспериментальным путем. Однако и выбор используемого алгоритма классификации оказывает влияние на оценку качества классификации, что было доказано в работе: метод опорных векторов работает эффективнее, чем алгоритм random forest.

2. Набор методов предобработки обучающей выборки действительно может оказывать различное влияние на оценку точности классификации, как положительное, так и отрицательное. Этим подтверждается вывод из статьи [23], рассмотренный в первой главе, заключающийся в следующем: при решении конкретной практической задачи классификации текста, правильным подходом является поиск оптимального набора методов предобработки, а не использование какого-либо случайного набора таких методов, либо вовсе полное игнорирование этапа предобработки.

В результате проведенного тестирования эффективности работы текстовых классификаторов, для использования выбран вариант набора

параметров метода опорных векторов (таблица 3, строка 1) как обеспечивающий наилучший показатель целевого значения (f-меры). Данный алгоритм с указанным набором параметров будет использоваться в финальной версии реализуемого текстового классификатора.

2.5 Реализованный текстовый классификатор

Диаграмма классов реализованного текстового классификатора, работающего в соответствии с выбранным оптимальным набором параметров, представлена на рисунке 6. Класс “CmsBot” на данной диаграмме классов является программным продуктом, в котором реализованный текстовый классификатор используется в виде модуля (поле “classifier” класса “CmsBot”). Данный программный продукт не является частью разработок, совершенных в рамках данной работы.

Спецификация методов и полей класса ComplaintClassifier:

1. Метод “__init__” – конструктор класса, вызываемый при создании экземпляра класса; в нем выполняется инициализация полей класса, а также вызов методов загрузки обучающей выборки и обучения.

2. Метод “load_train_score” – при вызове загружает обучающую выборку из файла в формате csv, а также производит ее предобработку в соответствии с определенными параметрами.

3. Метод “train” – запускает обучение классификатора на загруженной обучающей выборке, устанавливает поле “trained” в TRUE после завершения обучения.

4. Метод “predict” – возвращает метку класса для нового документа, текст которого помещается в параметр “text”.

5. Поле “vectorizer” – объект класса TfidfVectorizer библиотеки scikit-learn для построения матрицы «документ-лексема» и ее нормализация по TF-IDF.

6. Булево поле “trained” показывает, обучен ли классификатор.

7. Массив “X” – тексты элементов обучающей выборки.

8. Массив “Y” – метки классов элементов обучающей выборки.

9. Поле “classifier” – объект класса LinearSVC библиотеки scikit-learn (программная реализация метода опорных векторов с линейным ядром).

10. Поле “lemmatizer” – объект, реализующий работу лемматизатора MyStem на языке Python.

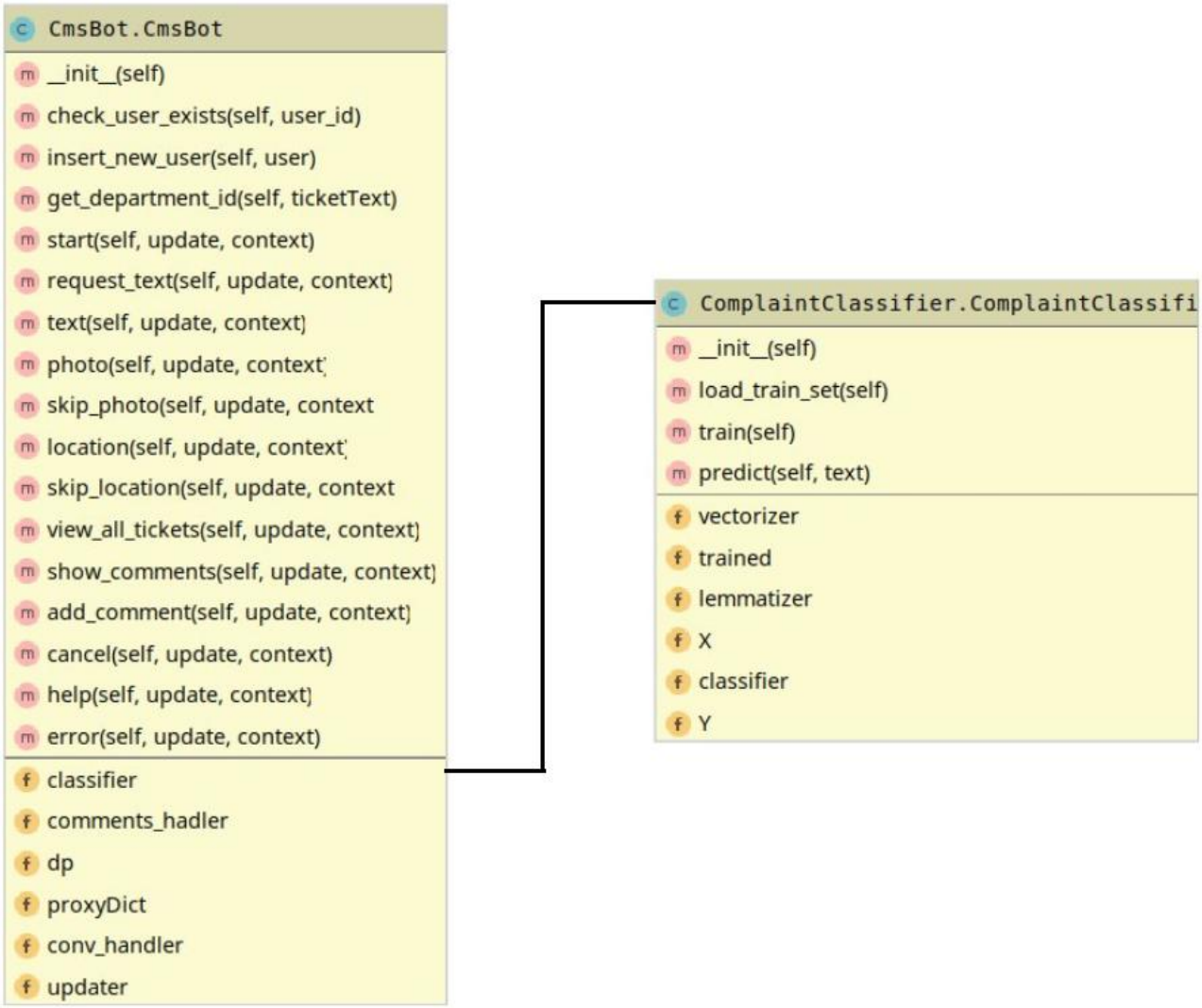

Рисунок 6 – Диаграмма классов реализованного текстового классификатора

При реализации также использована библиотека “joblib” для сохранения и загрузки состояния обученного классификатора, что позволяет не проводить процесс обучения заново при каждом запуске программы. Полный программный код реализованного классификатора приведен в приложении А.

Стоит отметить, что программная реализация классификатора работает в соответствии с приведенными ранее схемами процессов обучения и классификации. Так, прежде всего производится лемматизация текста элемента

обучающей выборки, после чего выполняются другие стадии предобработки и создания матрицы «документ-лексема» (метод “load_train_set”, листинг 1). После получения матрицы «документ-лексема» запускается процесс обучения классификатора (метод “train”, листинг 1).

Листинг 1. **Программный код методов предобработки обучающей выборки и обучения классификатора**

```
def load_train_set(self):
    f = open('./tlt10.csv') # открытие файла с обучающей выборкой
    reader = csv.DictReader(f, delimiter=',') # объект для чтения из файла
    for line in reader:
        #обработка лемматизатором текста элемента выборки:
        lemmatized = "".join(self.lemmatizer.lemmatize(line["текст"]))
        self.Y.append(line["тематика"]) # чтение метки класса
        self.X.append(lemmatized) # добавление обработанного текста
    # создание матрицы «документ-лексема», применение фильтра стоп-слов:
    self.X = self.vectorizer.fit_transform(self.X).toarray()

def train(self):
    self.classifier.fit(self.X, self.Y)
    self.trained = True
```

С точки зрения процесса классификации нового документа (листинг 2), текст также прежде всего обрабатывается лемматизатором, после чего текстовые данные преобразуются в вектор числовых значений по матрице «документ-лексема». Полученный вектор классифицируется методом опорных векторов.

Листинг 2. **Программный код метода классификации нового документа**

```
def predict(self, text):
    # обработка текста лемматизатором:
    text = "".join(self.lemmatizer.lemmatize(text))
    text = self.vectorizer.transform([text]) # перевод текста в вектор по матрице
    result = self.classifier.predict(text)[0] # получение метки класса
    return result
```

Для проверки работоспособности реализованного текстового классификатора перед встраиванием в систему, где он будет использоваться в качестве модуля, а также для демонстрации примеров его работы, были собраны несколько текстов-обращений в адрес городской администрации г.о. Тольятти, которые соответствуют по тематике классам документов из обучающей выборки. Важно отметить, что данные примеры не содержатся в обучающей выборке. Для каждого примера обученным классификатором была проставлена метка класса. Полученные результаты представлены в таблице 7 для удобства. Исходные тексты обращений сохранены без изменений, поскольку именно в необработанном виде с ними должен работать текстовый классификатор.

Таблица 7 – Примеры работы реализованного текстового классификатора

№	Классифицируемый текст (обращение)	Результат работы классификатора (метка класса)
1	Нанесена разметка «Двойная сплошная» по улице Дзержинского (4 квартал). Предусмотрен лишь один въезд в квартал по улице Дзержинского (при направлении от Степана Разина к Юбилейной), в районе бульвара Курчатова. Такой разметки не было никогда. Прошу разобраться в правомерности такого нанесения разметки, и привести разметку в соответствие. Если останется такая разметка,	Департамент дорожного хозяйства и транспорта

№	Классифицируемый текст (обращение)	Результат работы классификатора (метка класса)
	то во время учебного года заехать/выехать будет не просто проблематично, а невозможно.	
2	Маршрутка 124 ходит очень редко, с интервалом 30-40 мин. И более, особенно утром. По Автозаводскому району невозможно уехать с проспекта Степана Разина на Южное шоссе. Другие маршрутки туда не ходят. Ранее был маршрут 132, его отменили. Люди опаздывают на работу, маршрутка набивается ужасно, и этот кошмар постоянно. У маршрута 124 заявлен интервал в 7 минут, он не соблюдается. Ещё проблема с этим маршрутом в выходные вечером. Прошу принять меры к перевозчику.	Департамент дорожного хозяйства и транспорта
3	По адресу ул.Баныкина,66, где находятся организации Росреестр и спортивный клуб Венец территория находится в ужасном состоянии. Зимой дорога вокруг здания не очищалась. После того как снег растаял никто не убирался и не убирается.	Общие вопросы ЖКХ
4	20й квартал. Не стригут траву по всему Рябиновому бульвару.	Общие вопросы ЖКХ
5	С вечера 30.10 по настоящее время в районе ул.Ярославской и Громовой стоит резкий химический запах,что предпринимается властями кроме заседаний по вопросу экологии в городе?	Управление экологического контроля
6	Комзина 2 не горит фонарь уличного освещения	Инженерная инфраструктура
7	на детской площадке дома Мурысева 73 уже два дня лежит поваленное ветром дерево	Общие вопросы ЖКХ
8	на прилегающей территории МБУ школы №1 складированы строительные отходы в виде кирпичного лома и иного мусора, находятся на озеленённой территории, что является недопустимым по СаНиП. Прошу донести до сведения директора школы	Департамент образования

Приведенные примеры показывают эффективность работы реализованного текстового классификатора, его способность верно проставлять

метки классов для необработанных текстов. Следовательно, реализованная версия текстового классификатора может быть использована как практическое решение в виде программного модуля системы.

2.6 Выводы по главе 2

В главе были описаны параметры экспериментальных запусков, включающие в себя различные сочетания средств предобработки и параметров используемых алгоритмов классификации.

Оценивание результатов происходило по стандартным для данной научной области метрикам качества – точность (precision) и полнота (recall) классификации, однако с целью нахождения баланса между двумя метриками, в конечном счете использована f-мера с равными весами для обоих метрик.

Были проведены экспериментальные запуски двух выбранных алгоритмов (метод опорных векторов и random forest) на всех описанных наборах параметров. Полученные результаты тестирования на эффективность показали, что наилучшим с точки зрения максимизации значения сбалансированной f-меры является использование метода опорных векторов с параметром C = 0.5 в совокупности с применением метрики IDF на матрицу «документ-лексема», фильтрацией стоп-слов и применением лемматизатора.

С учетом проведенного тестирования был реализован текстовый классификатор, работающий на основе метода опорных векторов с использованием выявленных оптимальных параметров. Приведена диаграмма классов программного кода классификатора, демонстрирующая имеющиеся методы и поля. Для демонстрации примеров работы классификатора были собраны несколько текстов обращений граждан в органы администрации г.о. Тольятти, которые при этом не содержались в обучающей выборке. Проведена классификация собранных примеров, результаты которой показали высокую эффективность работы реализованного текстового классификатора.

ЗАКЛЮЧЕНИЕ

В ходе работы была описана решаемая задача классификации текста, сформулированы основные этапы реализации текстового классификатора, включающие в себя предобработку обучающей выборки, ее конвертацию в матрицу «документ-лексема» и непосредственно обучение алгоритма классификации на полученной выборке. Для получения наиболее эффективного программного решения – текстового классификатора, были выполнены следующие шаги:

1. Выбраны наиболее подходящие под условия задачи алгоритмы классификации.

2. Реализованы текстовые классификаторы на основе выбранных алгоритмов классификации текста (метод опорных векторов и random forest).

3. Разработан план тестирования реализованных классификаторов на эффективность, включающий в себя запуски классификаторов в различных режимах, в совокупности с различными методами предобработки.

4. Методом перебора всех вариантов, был получен наиболее оптимальный набор параметров, который был использован в финальной версии текстового классификатора.

Можно сделать следующий обобщенный вывод по реализации текстовых классификаторов: при реализации текстового классификатора, практически все имеющиеся параметры модели имеют воздействие на итоговую оценку качества классификации. Единственным способом получить в результате работы максимально эффективный текстовый классификатор является перебор набора доступных параметров классификатора в целях поиска оптимального набора, включающего в себя как используемый алгоритм и его параметры настройки, так и используемые средства предобработки.

Работа выполнена в рамках проектной деятельности по направлению «Умный город» центра «IT-Student» Тольяттинского государственного университета. Результаты, полученные в процессе выполнения работы, были представлены на конференциях «V Международная научно-практическая

конференции (школа-семинар) молодых ученых «Прикладная математика и информатика: современные исследования в области естественных и технических наук» и «Студенческие Дни науки в ТГУ (2019 г.)». Представленные на конференциях материалы были высоко оценены, в результате чего получены грамоты за участие на конференциях и победу в конкурсе докладов («Студенческие Дни науки в ТГУ»), что говорит об успешной апробации работы.

С точки зрения дальнейшего применения реализованного текстового классификатора, работающего на основе набора выявленных оптимальных параметров, полученную оценку точности классификации, равную 79%, можно считать достаточно высокой для подобного рода решений.

Однако эффективность реализованного текстового классификатора может быть дополнительно повышена, но уже после некоторого периода его применения в реальных условиях. В процессе применения на практике, в классификатор будут поступать новые текстовые документы, которым он будет присваивать метки классов в соответствии с имеющейся моделью. Случайным образом можно вручную проверять некоторые оценки, проставляемые классификатором, и в случае ошибки – проставлять верную метку данному документу. Массив полученных документов может быть использован как новая обучающая выборка, на которой можно провести процесс обучения классификатора. Ожидаемая оценка точности на новой выборке должна превышать текущую, поскольку новая выборка будет более ориентирована на конкретные условия применения классификатора. Такой процесс можно проводить периодически, например, раз в год, постепенно все более улучшая эффективность работы текстового классификатора.

СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ

Научная и методическая литература

1. К.В. Воронцов. Лекции по методу опорных векторов // Вычислительный центр им. А.А. Дородницына Российской академии наук Федерального исследовательского центра «Информатика и управление» Российской академии наук, 2007.

2. Меркулов В.Д., Мирзоева М.Р. Влияние алгоритмов стемминга и лемматизации на эффективность работы текстового классификатора // Студенческие дни науки в ТГУ. – Тольятти: Тольяттинский государственный университет, 2019.

Электронные ресурсы

3. Машина опорных векторов [Электронный ресурс] // MachineLearning.ru. – URL: http://machinelearning.ru/wiki/index.php?title=Метод_опорных_векторов (дата обращения: 06.06.2019).

4. Метрики в задачах машинного обучения [Электронный ресурс] // Блог компании Open Data Science. – URL: https://habr.com/ru/company/ods/blog/328372/ (дата обращения: 19.03.2019).

5. CountVectorizer [Электронный ресурс] // scikit-learn 0.21.2 documentation. – URL: https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html (дата обращения: 10.05.2019).

6. GridSearchCV [Электронный ресурс] // scikit-learn 0.21.2 documentation. – URL: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html (дата обращения: 10.05.2019)

7. J. Shaikh. Machine Learning, NLP: Text Classification using scikit-learn, python and NLTK [Электронный ресурс] // Towards Data Science. – URL: https://towardsdatascience.com/machine-learning-nlp-text-classification-using-scikit-learn-python-and-nltk-c52b92a7c73a (дата обращения: 06.06.2019).

8. LinearSVC [Электронный ресурс] // scikit-learn 0.21.2 documentation. – URL: https://scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVC.html (дата обращения: 10.05.2019).

9. MyStem [Электронный ресурс] // Технологии Яндекса. – URL: https://tech.yandex.ru/mystem/ (дата обращения: 10.05.2019).

10. NLTK stop words [Электронный ресурс] // Python tutorials. – URL: https://pythonspot.com/nltk-stop-words/ (дата обращения: 10.05.2019).

11. Random Forest Classifier [Электронный ресурс] // scikit-learn 0.21.2 documentation. – URL: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html (дата обращения: 10.05.2019).

12. Russian stemming algorithm [Электронный ресурс] // Snowball. – URL: http://snowball.tartarus.org/algorithms/russian/stemmer.html (дата обращения: 10.05.2019).

13. Scikit-Learn and GridSearchCV [Электронный ресурс] // Kaggle. – URL: https://www.kaggle.com/cesartrevisan/scikit-learn-and-gridsearchcv (дата обращения: 10.05.2019).

14. SVM Parameter Tuning in Scikit Learn using GridSearchCV [Электронный ресурс] // Medium. – URL: https://medium.com/@aneesha/svm-parameter-tuning-in-scikit-learn-using-gridsearchcv-2413c02125a0 (дата обращения: 06.06.2019).

15. Text Classification: A Comprehensive Guide to Classifying Text with Machine Learning [Электронный ресурс] // MonkeyLearn. – URL: https://monkeylearn.com/text-classification/ (дата обращения: 19.03.2019).

16. TfidfTransformer [Электронный ресурс] // scikit-learn 0.21.2 documentation. – URL: https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfTransformer.html (дата обращения: 10.05.2019).

17. TfifdfVectorizer [Электронный ресурс] // scikit-learn 0.21.2 documentation – URL: https://scikit-

learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html (дата обращения: 10.05.2019).

Литература на иностранном языке

18. A. Dasgupta, P. Drineas, B. Harb, V. Josifovski, M. W. Mahoney. Feature selection methods for text classification. // Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining. – 2007. – pp. 230-239.

19. A. H. Aliwy. Comparative Study of Five Text Classification Algorithms with their Improvements / Ahmed H. Aliwy, Esraa H. Abdul Ameer // International Journal of Applied Engineering. – 2017 – Volume 12, Number 14 (2017) pp. 4309-4319, c. 4314.

20. A. K. Uysal. The impact of preprocessing on text classification / Alper Kursat Uysal, Serkan Gunal // Information Processing & Management. – 2014 – Volume 50, Issue 1, January 2014, Pages 104-112.

21. G. Chandrashekar, F. Sahin. A survey on feature selection methods. // Computers & Electrical Engineering. – 2014. – Volume 40, Issue 1, Pages 16-28.

22. J. Tang. Feature Selection for Classification: A Review. / Jiliang Tang, Salem Alelyani, Huan Liu // Data Classification: Algorithms and Applications. – 2014, c.1.

23. J. Weston, S. Mukherjee, O. Chapelle, M. Pontil, T. Poggio, V. Vapnik. Feature Selection for SVMs // Advances in Neural Information Processing Systems 13, Papers from Neural Information Processing Systems (NIPS). – 2000.

24. O. Ardhapure, G. Patil, D. Udani, K. Jetha. Comparative study of classification algorithm for text based categorization // IJRET: International Journal of Research in Engineering and Technology. - 2016. - Volume: 05 Issue: 02.

25. P. Wang. Automating document classification for the Immune Epitope Database / P.Wang, A. Morgan, Q. Zhang and others // BMC Bioinformatics. – 2007 – 8(1):269.

26. S. Pahwa, D. Sinwar. Comparison Of Various Kernels Of Support Vector Machine // International Journal for Research in Applied Science & Engineering Technology (IJRASET). - 2015. - Volume 3, Issue VII.

27. T. Pranckevicius, V. Marcinkevičius. Comparison of Naive Bayes, Random Forest, Decision Tree, Support Vector Machines, and Logistic Regression Classifiers for Text Reviews Classification // Baltic J. Modern computing. - 2017. - Volume 5, №2.

28. Tung A.K.H. Rule-based Classification // Encyclopedia of Database Systems. Springer, Boston, MA.

29. Ultimate guide to deal with Text Data (using Python) // Analytics Vidhya. – URL: https://www.analyticsvidhya.com/blog/2018/02/the-different-methods-deal-text-data-predictive-python/ (дата обращения: 06.06.2019).

30. V. C.Gandhi. Review on Comparison between Text Classification Algorithms / Vaibhav C.Gandhi, Jignesh A.Prajapati // International Journal of Emerging Trends & Technology in Computer Science (IJETTCS). – 2012 – Volume 1, Issue 3.

ПРИЛОЖЕНИЕ А

Программный код реализованного текстового классификатора

```
# работа с csv-файлом
import csv
# стоп-слова
import nltk
from nltk.corpus import stopwords
nltk.download("stopwords")
# предобработка и создание матрицы
from sklearn.feature_extraction.text import TfidfVectorizer
from pymystem3 import Mystem
# метод опорных векторов
from sklearn.svm import LinearSVC
# сохранение состояния обученного классификатора
from joblib import dump, load
import os.path

class ComplaintClassifier:
    def __init__(self):
        self.trained = False
        self.X = []
        self.Y = []
        self.lemmatizer = Mystem()
        self.vectorizer = TfidfVectorizer(min_df=5, max_df=0.5,
stop_words=stopwords.words('russian'))
        self.load_train_set()
        if os.path.exists('./clf.joblib'):
            self.classifier = load('./clf.joblib')
            self.trained = True
        else:
            self.classifier = LinearSVC(random_state=0)
            self.trained = False
            self.train()

    def load_train_set(self):
        f = open('./tlt10.csv')
        reader = csv.DictReader(f, delimiter=',')
        for line in reader:
            lemmatized = ''.join(self.lemmatizer.lemmatize(line["текст"]))
            self.Y.append(line["тематика"])
            self.X.append(lemmatized)
        self.X = self.vectorizer.fit_transform(self.X).toarray()
```

```
def train(self):
    self.classifier.fit(self.X, self.Y)
    self.trained = True
    dump(self.classifier, './clf.joblib')

def predict(self, text):
    text = ''.join(self.lemmatizer.lemmatize(text))
    text = self.vectorizer.transform([text])
    result = self.classifier.predict(text)[0]
    return result
```