Московский государственный университет имени М. В. Ломоносова

Факультет вычислительной математики и кибернетики

Кафедра алгоритмических языков

Протасов Александр Васильевич

Классификация веб-страниц на основе текстового содержимого с помощью языковой модели BERT

Выпускная квалификационная работа

Научный руководитель: доцент, к.ф.-м.н.

Волкова Ирина Анатольевна

Москва 2023

Аннотация

В работе рассматривается задача классификации веб-страниц на основе текстового содержимого. Для ее решения предлагается использовать языковую модель BERT. В работе показано, как увеличить несбалансированный набор данных с помощью аугментации текстов. В вычислительных экспериментах приводятся результаты работы предложенного классификатора и алгоритмов аугментации на коллекциях от проекта ‘FOSS News’ и веб-страниц lenta.ru. Реализовано приложение с графическим интерфейсом для использования полученных результатов.

Оглавление

1 Введение **4**

2 Постановка задачи **6**

3 Обзор существующих подходов к решению поставленной задачи **7**

4 Языковая модель BERT **10**

4.1 Архитектура Transformer 10

4.2 Архитектура BERT 13

6 Данные и классификатор **16**

7 Методы аугментации текстовых данных **21**

7.1 Обратный перевод (Back Translation) 21

7.2 Использование контекстных представлений 22

7.3 Простая аугментация данных (Easy Data Augmentation) 23

8 Реализованные методы аугментации текстов **24**

9 Описание программной реализации **26**

10 Эксперименты **28**

11 Анализ результатов и выводы **33**

12 Графический интерфейс системы классификации веб-страниц **34**

13 Заключение **35**

Список литературы **36**

1 Введение

На текущий день содержится значительное количество сайтов и страниц с различным содержанием в сети Интернет, из-за чего затруднительно их пытаться вручную индексировать и классифицировать. Очевидно, существует потребность в задаче автоматической классификации веб-страниц.

Проблему классификации веб-страниц можно определить через одну из задач автоматизированной обработки текстов — классификации документов по заранее заданным наборам категорий на основе их текстового содержания. Часто такими документами являются новостные статьи, исследовательские работы. И веб-страница, по сути, является текстовым документом и часто может рассматриваться как относящаяся к определенной категории или теме. Поэтому представляется вероятным, что проверенные алгоритмы для классификации текста можно использовать для присвоения категорий веб-страницам.

Итак, задача классификации веб-страниц – автоматизировать процесс определения категории сайта по его текстовому содержанию.

Данная задача обеспечивает большое удобство пользователям, использующим поисковые системы: быстрее и проще достигать желаемого. Это важно для представления веб-страниц, относящихся к категории, которую ищут пользователи. Кроме того, определение категории веб-страницы также значительную роль играет при выборе рекламы для размещения на веб-сайтах. Размещение рекламы в соответствии с категорией веб-сайта и ограничение места показов рекламы рядом с чувствительным контентом больше повлияет на пользователей и увеличит доходы от рекламы. Кроме того, существуют сайты, содержащие запрещенную распространению информацию, которые нужно блокировать, и учитывая количество пользователей, которые могут создавать нежелательный контент, можно ипользовать классификацию веб-страниц. Задача также важна для того, чтобы дети могли пользоваться Интернетом с большей пользой. Таким образом, можно предотвратить сайты, которые могут оказать негативное влияние на детей, и представить контент, который может быть для них более полезным.

Существует множество подходов в классификации веб-страниц. Основным в использовании и эффективности является анализ текстового содержимого сайта. Но

также, например, можно анализировать URL-адрес страниц (больше в исследовательских целях) или разметку HTML-тегов: проверять наличие или отсутствие определенных тегов и считать их статистику. Другая идея – это задействовать содержимое соседних веб-страниц. Отсюда можно разделить подходы на две категории: подходы, основанные на контенте, и способы, основанные на URL-адресах. В первом случае подходы имеют лучшие характеристики классификации, потому что основаны на содержании и имеют больше информации, чем подходы, использующие только информацию URL.

В XXI веке для решения задачи классификации веб-страниц используются как подходы на основе методов машинного обучения, так и основанные на статистических методах. При этом задача все еще остается открытой.

2 Постановка задачи

Пусть существует конечное множество веб-страниц $\mathfrak{W} = \{w_1, , w_{|\mathfrak{W}|}\}$, фиксированное множество категорий $\mathfrak{C} = \{c_1, ..., c_{|\mathfrak{C}|}\}$, и существует неизвестная целевая функция $f' : \mathfrak{W} \times \mathfrak{C} \to \{0, 1\}$, значения которой известны для каждой пары $(w_i, c_j) \in \mathfrak{Q} \times \mathfrak{C}$ начальной коллекции веб-страниц $\mathfrak{Q} = \{w_1, , w_{|\mathfrak{Q}|}\} \subseteq \mathfrak{W}$.

Необходимо найти максимальное приближение f (классификатор) функции f' с помощью обучающей выборки $W_{tr} = \{w1, ..., w_{|W_{tr}|}\}$.

Качество классификатора оценивается на тестовой выборке $W_{te} = \{w_{|W_{tr}|+1}, ..., w_{|\mathfrak{Q}|}\}$ – результаты, полученные классификатором на тестовой выборке, сравниваются с известными для них значениями функции f' с помощью выбранной метрики.

Для решения поставленной задачи необходимо:

- Изучить существующие алгоритмы решения задачи классификации веб-страниц, выделить их преимущества и недостатки;
- Исследовать особенности аугментации при классификации текстов. Разработать методы решения задачи, основанные на результатах исследований;
- Реализовать систему классификации веб-страниц;
- Сравнить результаты работы алгоритма с результатами существующих систем классификации веб-страниц.

3 Обзор существующих подходов к решению поставленной задачи

Достаточно много публикаций на тему автоматической классификации веб-страниц, документов. Работы посвящены описанию новых методов, сравнению классических или усовершенствованию уже существующих методов.

В работе [1] рассказывается про один из статистических методов классификации – PrTFIDF – вероятностный алгоритм, который является улучшением классификатора TFIDF. Его сравнивают с классическим вероятностным методом Байеса и TFIDF. И несмотря на то, что с TFIDF получились допустимые результаты, первые два метода показали снижение частоты ошибок на 40% в большинстве примеров. Тестирование проводилось на больших коллекциях текстов, приводятся хорошие результаты. Автор отмечает, что с теоретической точки зрения вероятностные методы предпочтительнее.

В работе [2] описан классификатор сайтов по ключевым словам, учитывающий зашумленность и политематичность данных веб-сайтов. Целью авторов является классификация веб-сайтов, а не отдельных веб-страниц. Для такой задачи характерны следующие особенности: в обучающей выборке помимо самого веб-сайта находится репрезентативная выборка его веб-страниц, а также веб-сайт может содержать веб-страницы с тематикой, которая отличается от основной темы сайта. Авторы приводят результаты тестирования для случайно выбранных тем и анализ ошибок классификатора.

В работе [3] рассматривается несколько подходов к решению классификации веб-страниц: обучение классификаторов на основе текстовых данных с веб-страниц, на основе тегов или URL. Также авторы используют разные методы векторизации: Hashing Vectorizer, Tfidf, Word2Vec. В ходе экспериментов показано, что добавление адреса или заголовка в виде атрибутов не позволило увеличить точность модели и использование Word2Vec не внесло значительных изменений в результаты, с соответствующими обоснованиями. Также в работе показано, что добавление к характеристикам информации о страницах, удаленных на 1 или 2 перехода, слабо влияет на точность классификации (рис. 1). В результате SVM с использованием ключевых слов и извлечением данных с веб-страниц показал наибольшую точность решения данной задачи.

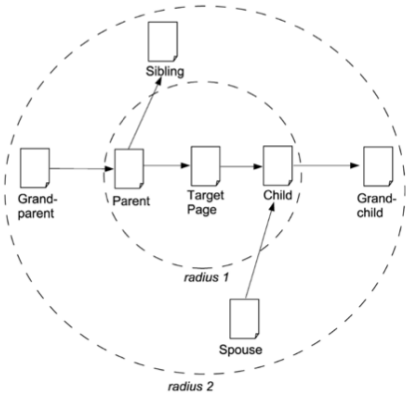

Рис. 1. Схема отношений между веб-страницами со ссылками из работы [3]

В работе [4] построен метод, работающий со статистикой HTML-тегов и решающий проблему путаницы некоторых категорий сайтов при классификации (например, chat – forum – blog). Авторы используют представление html-документа в виде дерева (ключ – тег, значение – все его представители, количество которых подсчитывалось) и классификатор строят на основе Naive Bayes и Decision Tree. Также была проанализирована комбинированная схема классификации, использующая текстовые и структурные признаки сайта одновременно, которая помогла увеличить качество.

В работах [5] и [6] авторы рассматривают только текстовое содержимое веб-страниц. В каждой работе берется множество html-тегов, несколько классических методов машинного обучения, на которых проводят эксперименты. Авторы первой работы показывают результаты подбора гиперпараметров, оценку качества классификаторов по нескольким метрикам и выводят ключевые слова для лучшей модели – Linear SVM. Результаты достаточно хорошие в связи со сбалансированным набором данных (8 категорий, на каждую из которых по 150 текстов). Во второй работе наиболее хорошо показал себя также SVM (8 категорий), но качество сравнимо хуже по сравнению с работой [5] из-за несбалансированных данных. Также авторы приходят к выводу, что части веб-страниц, которые хорошо себя показывают в классификации, – это html-теги <p>.

В качестве сравнения существующих подходов к решению задачи классификации веб-страниц рассматриваются работы [7-10]. Авторы данных работ используют одинаковый датасет – DMOZ, для английского языка он считается более распространенным. Датасет состоит из 13 несбалансированных категорий. В общей сложности около трех миллионов записей. В таблице 1 представлена точность и метод, который задействовал каждый автор для этого набора данных:

Работа	Метод	DMOZ
[7]	CNN	79%
[8]	Logistic Regression	53%
[9]	Бернулли Наивный Байес	58%
[10]	SVM	66%

Таблица 1. Точность классификации методов из работ [7-10] на наборе данных DMOZ

По таблице 1 видно, что лучший результат достигается в работе [7]. Метод в этой работе основан на нейронных сетях (4 слоя, используется 2 эпохи для обучения классификатора). Автор учитывает только заголовки статей (около 5-15 слов), с которыми дополнительной работы не производится. В работе [8] вначале тщательно предобрабатываются заголовки веб-страниц, например, все сокращения заменяются их полными фразами, стоп-слова, частотные слова и специальные символы удаляются и все слова приводятся к нормальной форме. Но результат в итоге не особо значимый по сравнению с другими работами.

Для русского языка сравнительный анализ разных методов провести затруднительно, так как нет общепринятого набора данных для тестирования решений.

В данной главе рассмотрены различные алгоритмы классификации веб-страниц. Подходы, основанные на классических методах машинного обучения не подходят для поставленной задачи в силу невозможности достижения необходимого качества работы алгоритма. Поэтому в рамках ВКР будет использоваться классификатор, основанный на нейронных сетях.

4 Языковая модель BERT

BERT или Bidirectional Encoder Representations from Transformers представляет собой предварительно обученную языковую модель, которая стала активно применяться в области обработки естественного языка (NLP) с момента ее появления в 2018 году [11].

Принцип работы модели BERT основан на применении кодировщиков (encoders) из Transformer. Далее будет рассмотрен подробнее принцип работы кодировщиков, а затем и сам BERT.

4.1 Архитектура Transformer

Архитектура Transformer [12] – архитектура нейронной сети, основанная на механизме внимания и представленная в 2017 году компанией Google. Это первая успешная модель для задачи машинного перевода, не использующая рекуррентные нейросети.

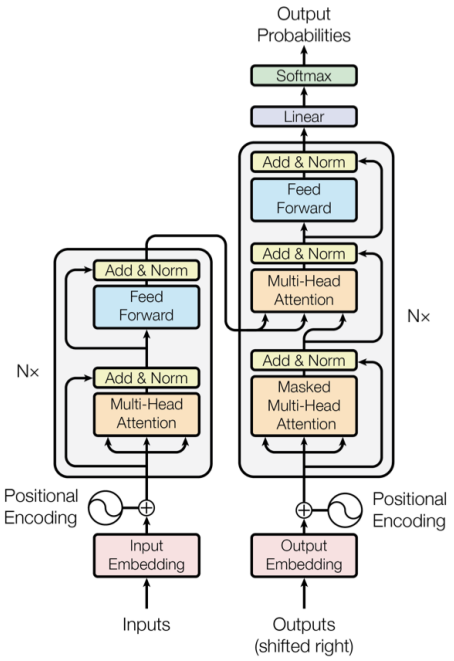

Рис. 2. Архитектура Transformer [12]

На рисунке 2 показана схема трансформера: левый блок представляет собой один слой кодировщика (encoder), а правый — один слой декодировщика (decoder). Далее будет рассматриваться только энкодер.

Механизм внимания (Attention). Данный механизм формулируется с помощью наборов векторов-ключей и соответствующих им векторов-значений (K, V), а также векторов-запросов Q. Требуется сравнить каждый вектор из Q со всеми векторами из K и вернуть наиболее похожие на запросы векторы-ключи и соответствующие им векторы-значения из V.

Более точное рассмотрение алгоритма: на вход блоку подается информация с выхода предыдущего блока, в случае первого блока, на вход подаются неконтекстуализированные эмбеддинги слов. Эмбеддинги слов попадают в слой внутреннего внимания. Векторные представления слов подаются в виде матрицы X размером $L \times N$, где L – длина векторного представления, N – количество слов в предложении.

Первый этап – создание трех матриц:
Q – матрица векторов запросов, q_i является одним вектором-запроса, связанным с одним входным словом.
V – матрица векторов значений, v_i является одним вектором-значения, связанным с одним входным словом.
K – матрица векторов ключей, k_i является одним вектором-ключа, связанным с одним входным словом.

Данные матрицы являются результатом матричного произведения между входными эмбеддингами слов (X) и тремя матрицами обученных весов: W^Q, W^K, W^V.

Следующим шагом необходимо получить для каждого слова в предложении коэффициент схожести (1) со всеми словами в этом же предложении.

Также дополнительно полученные коэффициенты нормируются на некоторую константу β (на практике $\beta = \sqrt{L}$), для стабильности вычислений. Более близкие векторы запросов и ключей будут иметь более высокие скалярные произведения. В свою очередь для их нормирования (от 0 до 1) применяется *softmax* (2).

$$S = QK^T \quad (1), \qquad M = softmax(\frac{S}{\sqrt{\beta}}) \quad (2)$$

Далее полученные веса M умножаются на матрицу значений V и складываются. Конечная формула:

$$Attention(Q, K, V) = softmax\left(\frac{QK^T}{\beta}\right)V$$

В трансформере используется модификация данного механизма: многоголовое внимание (Multi-Head Attention). Основным отличием «многоголового» внимания является наличие нескольких параллельно работающих головок Attention.

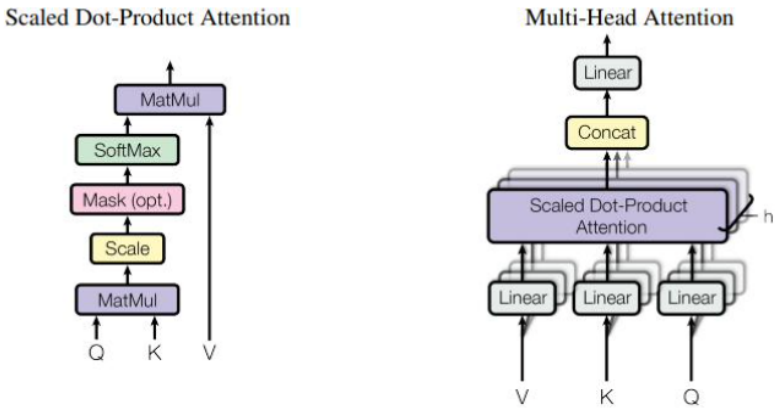

Рис. 3. Схема механизма внимания (слева) и многоголового внимания (справа) [8]

Для начала входные матрицы Q, K, и V разбиваются линейным образом на несколько подпространств, каждое из которых отвечает за изучение отдельного представления (отсюда и разные результаты). После того как каждое из этих подпространств пройдет через механизм внимания (все они обрабатываются одинаково, только размерность матриц меньше), полученные результаты конкатенируются и умножаются еще на одну матрицу весов W^O. Все перечисленные этапы можно описать следующим образом:

$$MultiHead(Q, K, V) = Concat(head_1, \ldots, head_h)W^O$$

где $head_i = Attention(QW_i^Q, KW_i^K, VW_i^V)$.

Стоит отметить, что трансформеры не имеют встроенного механизма для обработки порядка слов. Это важно, потому что расстояние между словами обеспечивает важную контекстную информацию. В трансформере решением данной проблемы является позиционное кодирование. Оно не является частью архитектуры модели. Скорее, это относится к предварительной обработке. Векторы позиционного кодирования генерируется такого же размера, как входящие эмбеддинги, и добавляются к ним.

Encoder. Кодировщик состоит из нескольких идентичных слоев. Каждый из них имеет архитектуру, изображенную слева на рисунке 2, а также два подслоя. Первый –

рассмотренный Multi-Head Attention, а второй – нейронная сеть прямого распределения (feed-forward neural network) или иначе – полносвязная нейронная сеть, применяемая к каждому слову в отдельности. Выход каждого подслоя складывается со своим входом, а к полученной сумме применяется нормализация по слою (Layer Normalization). Данный механизм (который называется skip-connections) нужен для решения проблемы затухающих градиентов.

4.2 Архитектура BERT

Модель BERT состоит из 12 кодировщиков. Ее структура представлена на рисунке 4.

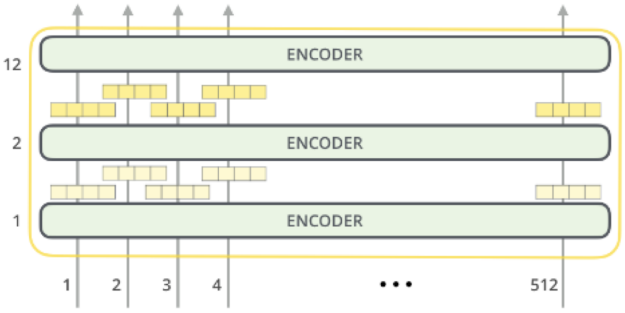

Рис. 4. Структура модели BERT [13]

В процессе работы модель BERT принимает на вход по аналогии с трансформером набор слов, к которому добавляется два специальных токена: токен *[CLS]* – в начало последовательности, конечное скрытое состояние которого используется в качестве представления совокупной последовательности для задач классификации, и токен *[SEP]* – в конец каждого предложения для разграничения предложений. После чего преобразованнная последовательность проходит через слои, каждый слой выполняет над эмбеддингами некоторое преобразование. Выход представляет собой векторы предопределенного размера, соответствующие токенам во входной последовательности. Выходы моделей трансформера и берта отличаются: в первом случае выдается готовое переведенное предложение, в то же время выходные векторы BERT можно использовать в качестве входных данных для различных задач NLP.

Как показывают многочисленные эксперименты, модель, предварительно обученная на некоторые задачи на большом неразмеченном наборе текстов, а потом дообученная на конкретную (downstream) задачу, показывает гораздо лучшие результаты, чем

модель, решающая одну определенную задачу без предобучения. BERT использует первый вариант и соответственно его обучение состоит из двух этапов:

Предобучение (pre-train). BERT предобучается на большом корпусе неразмеченных данных с помощью решения двух задач:

Предсказание замаскированных слов (Masked Language Modelling, MLM). Классическая проблема в однонаправленных языковых моделях (к примеру, в рекуррентных нейросетях): модель видит контекст слова только с одной стороны. Если же использовать двунаправленный подход, то возникает проблема переобучения, поскольку информация о предсказываемом слове содержится в одном из контекстов (левом или правом). Идея авторов: случайным образом замаскировать некоторую долю слов во входных данных, заменив их токеном *[MASK]*, и прогнозировать только замаскированные слова, основываясь на контексте, предоставляемом другими немаскированными словами в последовательности. Однако существует проблема – модель пытается предсказать только тогда, когда токен *[MASK]* присутствует во входных данных, когда нужно, чтобы модель пыталась предсказать правильные токены независимо от того, какой токен присутствует во входных данных. Чтобы решить эту проблему, из доли токенов, выбранных для маскировки, часть фактически заменяется токеном *[MASK]*, некоторые токены заменяются случайным токеном, и оставшаяся часть токенов остается неизменной.

Классификация: является ли второе предложение продолжением первого (Next Sentence Prediction, NSP).

Обученный для решения этих задач BERT оказывается достаточно интеллектуальной системой: задача восстановления замаскированных слов позволяет ей «понимать» связи между словами в предложении, а с помощью задачи NSP она может устанавливать связи между целыми предложениями. Важно отметить, что данные, на которых выполняется pre-train BERT не требуют разметки, что позволяет обучать его на огромном количестве данных.

Дообучение (fine-tuning). После того как модель предобучена на большом корпусе текстов, архитектура BERT незначительно изменяется для решения необходимых задач. В частности, для классификации текстов достаточно добавить полносвязный линейный слой и слой softmax.

Выходные данные 12 слоев BERT являются эмбеддингами в 768-мерное пространство для каждого токена. Поскольку цель данной работы состоит в том, чтобы выполнить классификацию, используется только выходной эмбеддинг для маркера *[CLS]*. Затем он передается через линейный слой (классификатор), выходные данные которого используются для определения прогнозируемого класса. На рисунке 5 показана архитектура модели BERT для классификации текстов.

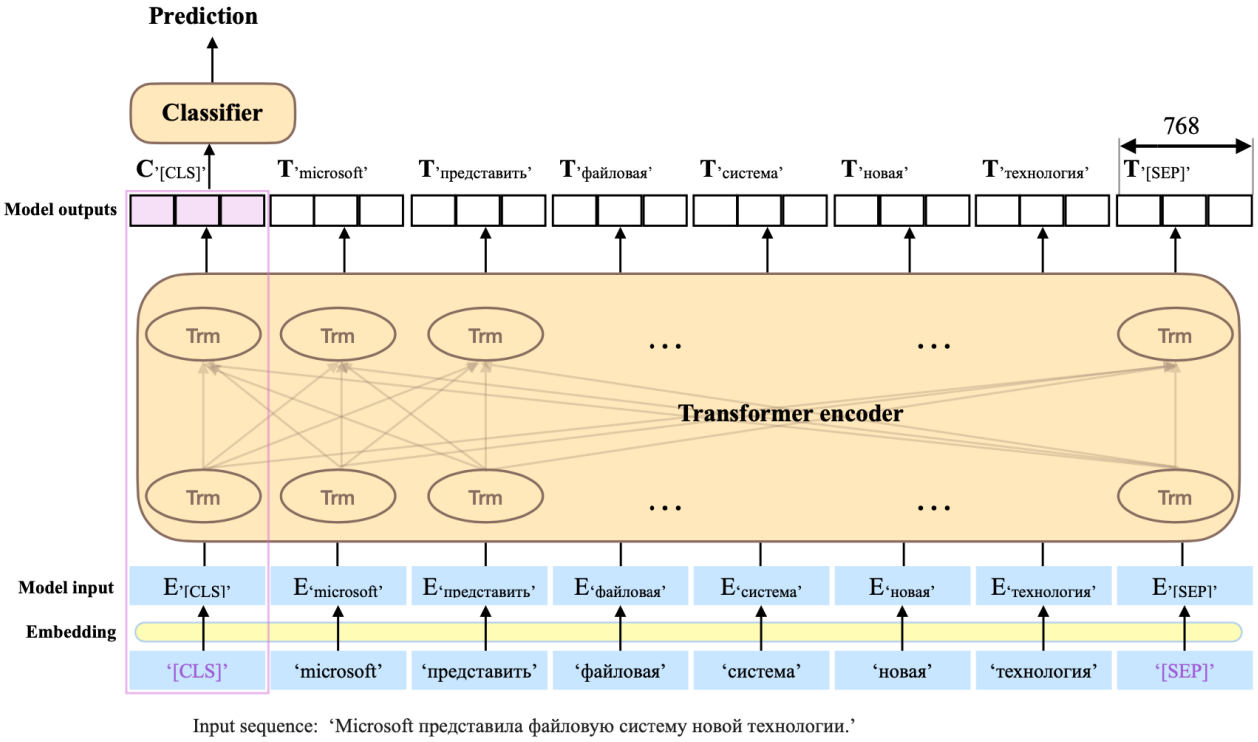

Рис. 5. Пример обученного BERT для классификации текстов

6 Данные и классификатор

Данные. При решении любой задачи с использованием машинного обучения необходим набор размеченных данных, в текущей задаче – русскоязычных веб-страниц. Выбор данных небольшой. Для обучения и проведения различных гипотез и экспериментов используется датасет от проекта ‘FOSS News’. В нем содержится 5007 и 3632 записей на русском и на английском языках соответственно, которые поделены на 25 категорий (Рис.6).

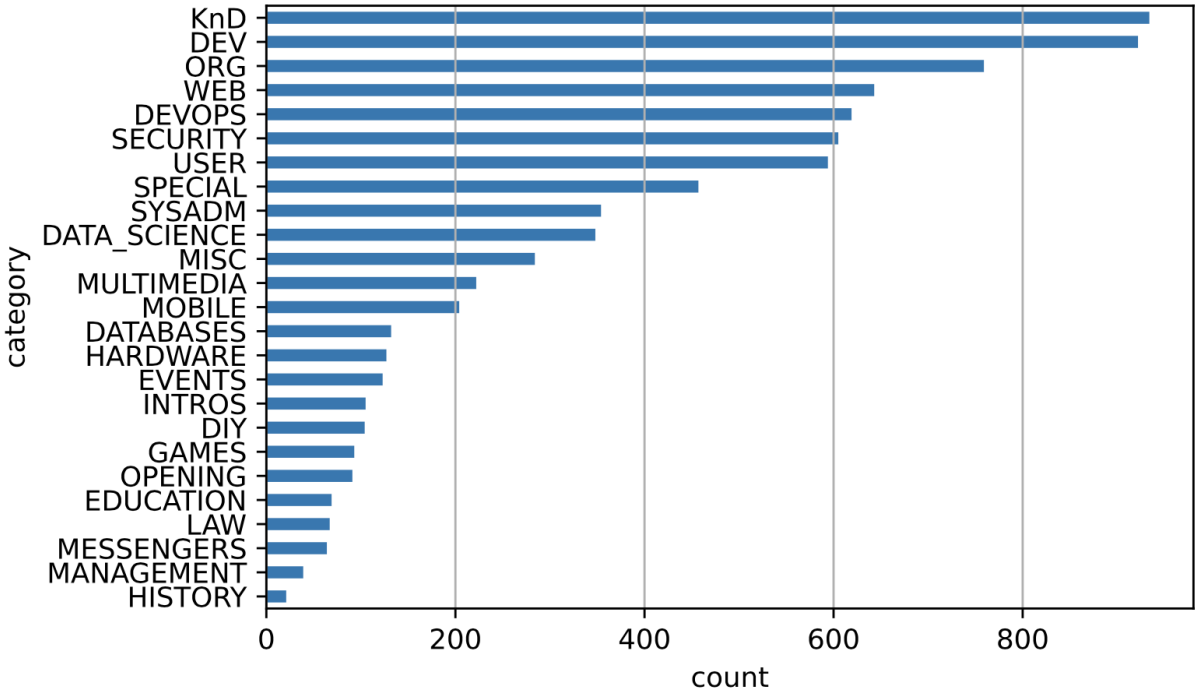

Рис. 6. Частота категорий

По рисунку 6 видно, что есть категории, которые отличаются количеством веб-страниц между собой в 10 и более раз, говоря другими словами, присутствуют миноритарные и мажоритарные классы. Возникает проблема с несбалансированными данными. Это может плохо сказаться на результатах: некоторые классы могут не иметь достаточного веса для воздействия на классификатор. Исходя из этого в работе рассматриваются два подхода к решению этой проблемы:

- субдискретизация (under-sampling) – выбор случайного подмножества мажоритарного класса для создания более сбалансированного набора; [14]

- аугментация – увеличение выборки данных для обучения через модификацию этих данных и создание на этой основе дополнительных данных. Более подробно данный метод будет рассмотрен в следующей главе.

Также был найден и изменен под решаемую задачу датасет с данными с сайта *lenta.ru*. Он содержит 19120 веб-страниц восемнадцати категорий. Количество веб-страниц для большинства авторов одинаковое, но есть и категории, имеющие значительно меньше веб-страниц. Датасет был добавлен в работу с целью верификации классификатора.

Классификация веб-страниц в поставленной задаче будет решаться с помощью их текстового содержимого. Любая web-страница может быть представлена в виде некоторого дерева, узлами которого являются html-тэги. Каждый тег является объектом, текст – тоже объект. Такое представление называется DOM tree, оно удобно для обхода HTML-страницы и так же задействовано в работе. На рисунке 7 показан пример иерархии DOM в документе HTML. Преимуществом такого представления является то, что в листьях получившегося дерева находятся объекты с текстом, имеющие основное содержимое страницы. В главе ‘Эксперименты’ будет показано, какие html-теги являются полезными в решаемой задаче.

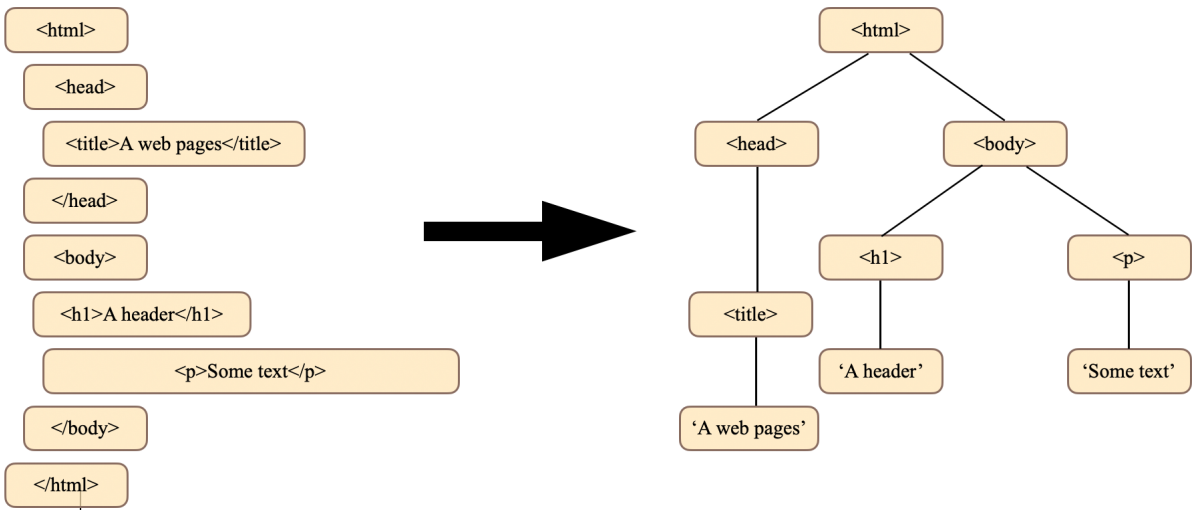

Рис. 7. Представление документа HTML (слева) и его DOM tree (справа)

Классификатор. Для обработки текстов используется модель *DeepPavlov/rubert-base-cased* – модель BERT, обученная специально для русского языка. Она содержит двенадцать слоев, размер эмбеддинга равен 768.

Для подготовленного набора данных нужен токенизатор. BERT использует собственную разметку токенов. Поэтому есть предобученный токенизатор вместе с моделью от *DeepPavlov* – он и будет использоваться в работе. Токенизатор разбивает слова на подслова, благодаря чему обученная модель сможет хорошо понимать слова с опечатками и ошибками, а также сможет обрабатывать возможные новые слова.

Подготовка данных для классификатора. Для проведения экспериментов необходимо подготовить данные для их обработки в модель BERT. Сначала каждый текст проходит этап предобработки (удаление стоп-слов, цифр, пунктуации, лемматизация), далее проводится токенизация (алгоритм wordpiece). Затем выбирается размер входной последовательности, подаваемой нейросети (максимальный допустимый размер для модели – 512). Он должен быть фиксированным, но тексты имеют различную длину. При этом нужно учитывать, хватит ли машине памяти для обработки последовательностей большого размера. В текущей работе оптимальным размером является 500.

Слева на рисунке 8 показано распределение количества токенов в предобработанных текстах. Видно, что почти все тексты имеют длину более 500. Чтобы не обрезать большую часть токенов, которые могут быть полезны при классификации, возникла идея каждый текст разделить на части кратные 500 словам, при этом 50 слов будут перекрываться (Рис. 9). Распределение длины последовательностей поменялось (Рис. 8, справа): теперь максимальная длина не больше 500. В то же время увеличилось и количество последовательностей с 7900 до 13000 текстов (Рис. 10). Выбрав размер, все сэмплы (последовательности) приводятся к единому виду: последовательности малого размера дополнятся паддингами - специальными нулевыми токенами. Потом создается маска внимания для каждого сэмпла. Следующий шаг – разбить данные как на обучающую выборку валидационную - для того, чтобы проверять качество в ходе дообучения модели: нужно разделить как последовательности, так и их маски, так и тестовую. Конечный шаг – загрузка весов предобученной модели и запуск процесса обучения на готовых данных.

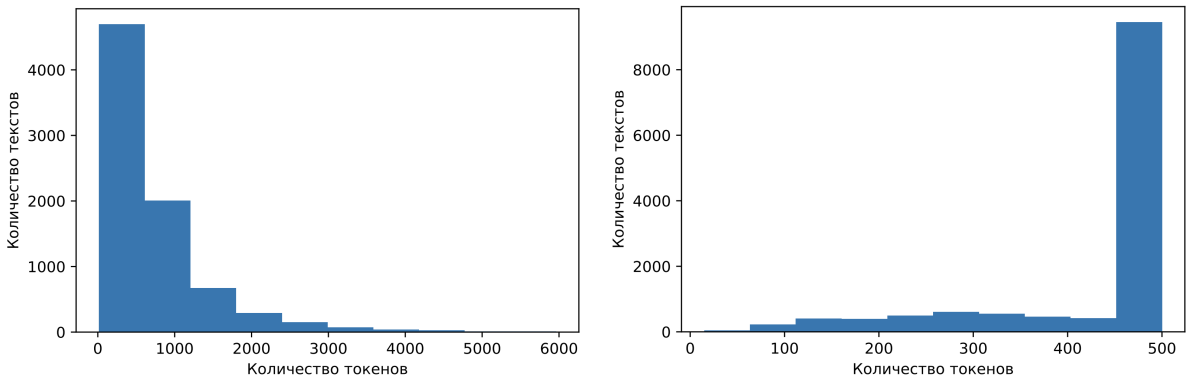

Рис. 8. Распределение длины последовательностей

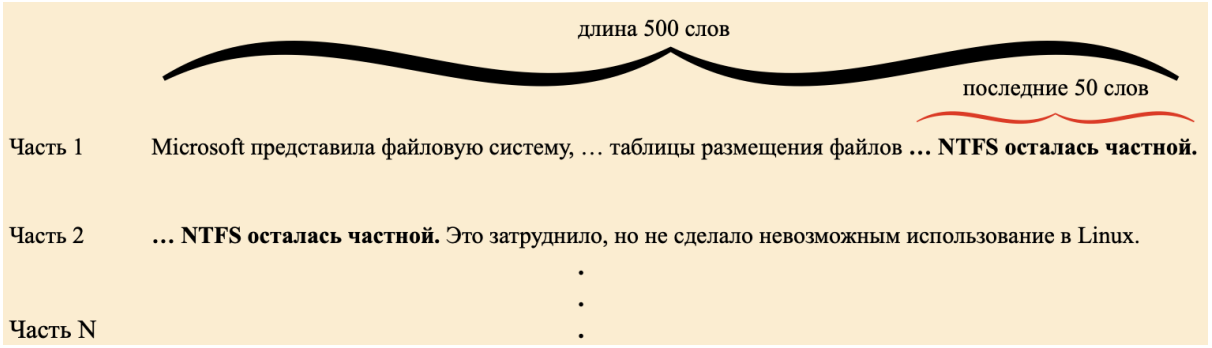

Рис. 9. Разбиение текста на несколько частей

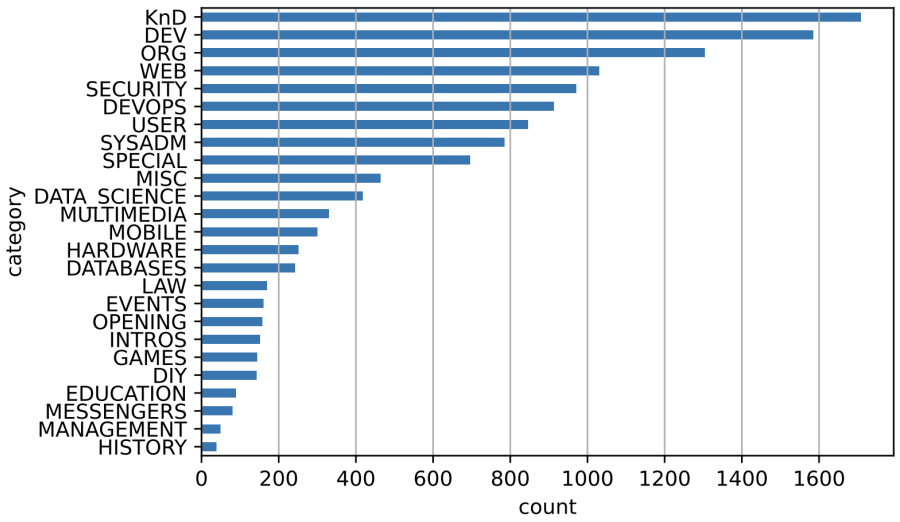

Рис. 10. Частота категорий после разбиения текстов на части

Метрика. В данной задаче критерием качества метрики будет выступать $F1$-мера. Оценка $F1$ принимает значение от 0 до 1, где 0 — наихудшая возможная оценка, а 1 —

наилучшая возможная оценка. Она является средним гармоническим точности (p) и полноты (r):

$$p_i = \frac{TP_i}{TP_i + FP_i}, \quad r_i = \frac{TP_i}{TP_i + FN_i}$$

где TP_i – количество документов, правильно помеченных классом i; FP_i – количество документов, неправильно помеченных классом i;
FN_i – количество документов, которые должны были быть помечены классом i, но не были помечены.

Оценка $F1$ для класса i затем выражается как:

$$F_i = \frac{2p_i r_i}{p_i + r_i}$$

Глобальные значения точности и полноты получаются путем суммирования всех индивидуальных значений:

$$p = \frac{\sum_{i=1}^{M} TP_i}{\sum_{i=1}^{M}(TP_i + FP_i)}, \quad r = \frac{\sum_{i=1}^{M} TP_i}{\sum_{i=1}^{M}(TP_i + FN_i)}$$

где M – количество категорий. В случае же мультиклассовый классификации (данная задача относится к ней) используются [15] термины macro-averaging и micro-averaging:

$$F1(micro) = \frac{2pr}{p + r}, \qquad F1(macro) = \frac{\sum_{i=1}^{M} F_i}{M}$$

Макроусреднение вычисляет метрику независимо для каждого класса, а затем берет среднее значение (отсюда она относится ко всем классам одинаково), в то время как микроусреднение будет агрегировать вклад всех классов для вычисления метрики. В данной задаче будет использоваться мера $F1_{micro}$.

Выбраны данные и классификатор, введена метрика – теперь есть все для решения поставленной задачи и проведения экспериментов.

7 Методы аугментации текстовых данных

В данной работе под аугментацией понимается создание новых данных с помощью модификации уже имеющихся. Использование аугментации может быть особенно полезно для данных с несбалансированными данными или небольшой обучающей выборки и может улучшить обобщающую способность модели, являясь мощным инструментом в борьбе с переобучением.

Исследование методов аугментации данных актуально в настоящее время. Аугментация успешно используется при решении многих задач машинного обучения, связанных с обработкой изображений, звуковых и текстовых данных.

Рассмотрим существующие методы аугментации:

- Обратный перевод (Back Translation);
- Использование контекстных представлений;
- Простая аугментация данных (Easy Data Augmentation).

7.1 Обратный перевод (Back Translation)

Данный метод [16] заключается в переводе текстового содержимого на какой-то язык, а затем обратный перевод на язык оригинала.

Есть несколько приемов, применяемых при обратном переводе, которые увеличивают число возможных аугментаций. Первый – можно осуществлять перевод на разные языки с помощью API переводчиков (Yandex, Google). Второй – можно взять языковую модель, обученную на перевод текстов, и применять ее для аугментации, меняя параметры (например, чтобы генерировать чуть менее вероятные тексты для перефразирования). На рисунке 11 показан пример работы обратного перевода.

Рис. 11. Пример обратного перевода

7.2 Использование контекстных представлений

Преобразуя текст, есть шанс исказить его смысл. С данным ограничением не всегда получается качественно аугментировать текст, но существуют методы, учитывающие контекст слов для замены целевых слов [17]. Этого можно добиться с помощью точной настройки различных предварительно обученных моделей на базе архитектуры трансформера. Например, BERT, GPT, Seq2Seq, BART, T5. Языковые модели получают не конкретное слово, а последовательность слов, поэтому для замены можно использовать любое слово (в частности несколько слов), которому соответствует высокая вероятность.

Ранее была описана задача предсказания замаскированных слов. Как раз названные модели, в частности, можно предобучать на больших данных этой задачи. Поэтому их логично использовать для замены слов: заменить некоторые слова на маски и подать на вход трансформеру, а он уже выдаст текст с вставленными словами на месте масок. Также помимо контекстной замены можно использовать контекстную вставку случайного токена.

Также сюда можно отнести метод **перефразирования**. Перефразировать – значит, не соответствовать источнику дословно, изменить какую-то часть текста своими словами (изменить слова, формулировку), но сохранить и полностью передать первоначальный смысл.

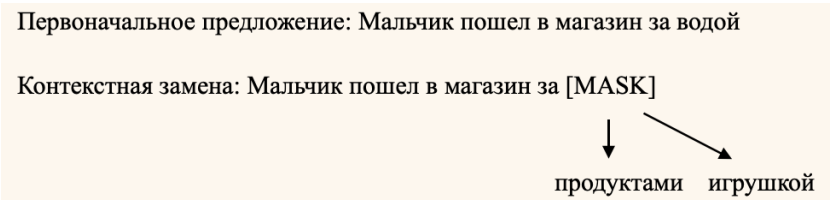

Рис. 12. Пример контекстной замены случайного токена

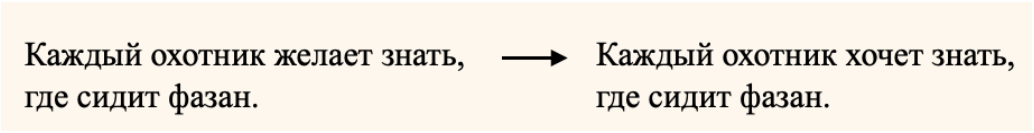

Рис. 13. Пример работы парафразера

7.3 Простая аугментация данных (Easy Data Augmentation)

Авторы статьи [18] утверждают, что использование двунаправленных моделей для аугментации данных вычислительно дорого. Поэтому они предложили свои несколько методов (EDA techniques) для повышения точности работы NLP задач. Четыре простых метода для аугментации текстов:

- **Замена синонимом.** Это самый простой способ перефразирования. Случайным образом выбираются n слов из текста (обычно стоп-слова не входят в данный список) и заменить каждое слово его синонимом. К способам нахождения синонимов можно отнести модели использующие представления слов: нахождение близких слов к заданному (Word2Vec, fasttext, GloVe);
- **Случайная вставка.** Вначале находится случайный синоним случайного слова в тексте (также стоп-слова не учитываются) и потом вставляется в рандомную позицию. Можно повторить n раз;
- **Поменять слова местами.** Случайным образом выбрать два слова и поменять их позиции местами. Можно повторить n раз;
- **Случайное удаление.** Случайно удалить каждое слово с вероятностью p.

Также стоит дополнить данную главу следующими способами аугментации: перестановка предложений, кроссовер, зашумление (ошибки в буквах), генерация (генеративные модели, трансформация синтаксического дерева), замена существенных или несущественных слов.

По итогу, имеется множество методов аугментации текстов. Можно пробовать различные подходы к увеличению данных и проверять, какой из них работает лучше. Также можно сочетать различные способы аугментации.

8 Реализованные методы аугментации текстов

На основе предложенных вариантов аугментации разработано несколько методов, которые в дальнейшем будут сравниваться.

Для каждой несбалансированной категории выбирается определенное количество текстов (*N*), к которым будет применена аугментация. На выходе к имеющимся данным прибавится, как минимум, *N*. На вход любому методу аугментации подается текст без стоп-слов, знаков препинания, цифр.

1. **Перефразирование текстов.** В качестве предобученной модели выбрана модель *cointegrated/rut5-base-paraphraser*, основанная на нейронных сетях и созданная для понимания и генерации текста. Для нее выбраны основные параметры, которые можно задавать перед перефразированием, в том числе для каждого текста выбирается количество новых текстов, которое нужно сгенерировать (по умолчанию = 3). Затем текст векторизуется и подается модели на вход. На выходе три декодированных текста, которые сравниваются с исходным, и если совпадений нет, то к текущей категории добавляется 3 ‘новых’ текста.

2. **Обратный перевод.** В качестве предобученной модели выбрана нейросетевая модель *Helsinki-NLP/opus-mt-{src}-{tgt}* (src = ru/en, tgt = en/ru). На вход модели подается векторизованный русский текст, который генерируется в новую последовательность и декодируется на английский язык. Затем в обратную сторону – модель получает сгенерированный английский текст и переводит на русский. В случае возникновения ошибки нейросетевой модели текст переводится с помощью *Google Translate API*.

3. **Контекстные замена и вставка.** Для контекстного подбора слов выбрана модель *cointegrated/rubert-tiny2* (для задачи *fill-mask*). Основными параметрами являются процент замены (вставки) слов и топ k слов для замены (для случайного выбора). По умолчанию 50% заменяются или вставляется – *‘[MASK]’*. Для замены слова синонимом проверяется: не выдала ли модель то же слово (тогда заменяется следующим словом в топе). На выходе ‘новый’ текст.

4. **EDA.** Для выбора близких слов к заданным реализовано два алгоритма: с помощью *Word2Vec* и API сервиса *RusVectores*

(‘*ruwikiruscorpora_upos_cbow_300_10_2021’*) – вычисляет семантические отношения. В качестве основного параметра является процент замены (вставки, удаления) слов (по умолчанию 50%). Для каждого слова подбирается список самых близких слов (~10), затем случайным образом один из синонимов той же части речи, что и заданное, и, в случае если синоним отличается, заменяет текущее или вставляется на очередную позицию. Также реализованы методы по удалению случайных слов текста и перестановке позиций слов (разница позиций в словах не больше 4).

9 Описание программной реализации

Описанные алгоритмы реализованы на языке Python 3.8. В работе использованы следующие библиотеки:

- **pandas** – библиотека для обработки и анализа данных. В работе используется при сборе и подготовке датасетов и работе с ними;
- **bs4** – библиотека для извлечения данных из файлов HTML и XML. В работе используется для изъятия текстовых данных с веб-страниц;
- **pymorphy3** – морфологический анализатор для русского языка. В программе используется для преобразования слов в нормальную форму;
- **PyQt5** – библиотека для создания приложений с графическим интерфейсом. С помощью нее реализован графический интерфейс задачи классификации веб-страниц;
- **torch** – универсальная библиотека для гибкого использования нейронных сетей. С помощью нее производится работа с тензорами и обучением модели;
- **transformers –** библиотека для предоставления инструментов и интерфейсов для простой загрузки и использования моделей машинного обучения. В программе используется для векторизации BertTokenizer и работы с BERT.

Код был написан в среде разработки Jupyter Notebook при использовании Google Colab, который является бесплатной и мощной платформой для запуска кода. Jupyter Notebok является наиболее удобной платформой для проведения исследований. Также в них можно легко проверить то или иное предположение и работать в любой момент времени с любым фрагментом кода.

По итогу имеется набор файлов **.ipunb**:

- **dataset** – описание данных и изъятие нужных веб-страниц и категорий; работа с веб-страницами: парсинг, получение текстовых данных; перевод английских текстов на русский язык; формирование основного датасета и датасета с веб-страницами с сайта *lenta.ru*;

- **classification_web_pages** – предварительная обработка данных; подготовка данных для классификатора; дообучение языковой модели BERT на задачу классификации; тестирование и проведение экспериментов;

- **augmentation** – реализация всех методов аугментации, описанных ранее: вставка и замена (учитывая контекст, с помощью синонимов), перефразирование, обратный перевод, EDA, а также их практическое применение и верификация;

Приложение с графическим интерфейсом разделено на четыре файла:

- **main.py** – интерфейс, логика приложения и обработка действий пользователя. Содержит описание классов UiMainWindow, в котором описывается расположение элементов интерфейса в окне, Thread и Thread1 – для запуска предобработки веб-страниц, обучения модели и предсказания соответственно – для вычисления сложных и длительных действий в фоновом процессе, чтобы пользователь мог продолжать взаимодействие с приложением;
- **preprocess.py** – подготовка данных для классификатора. разделение текстов на порции данных, изъятие текстового содержимого с веб-страниц и предобработка текстов;
- **berl_model.py** – классификатор текстов. Реализован класс BertClassifier, в основе которого лежит языковая модель BERT с классическими реализованными методами предсказания, обучения, сохранения модели.
- **dataset.py** – реализация класса CustomDataset, который токенизирует текстовые последовательности и затем преобразует целевые данные в тензоры.

.

10 Эксперименты

Первый эксперимент: проверить, как текстовые данные разных html-тегов влияют на качество классификатора. В качестве тегов для проверки использовались:

1. Группа заголовков: теги <title> и <h1>-<h6> – заголовки веб-страниц и заголовки в ходе прочтения веб-страниц;

2. Весь текст веб-страницы (включая комментарии): тег <body> содержит все основные теги: <p>, <h1>-<h6>;

3. Абзацы веб-страниц: тег <p> – определяет параграфы или текстовый абзац.

На рисунке 14 показан результат эксперимента. Видно, что к четвертой эпохе максимальное качество достигается с данными из 2 группы, которых получилось для обучения 17843 текстов. Меньше всего качество получилось на заголовках, потому что они очень короткие и из-за пересекающихся категорий и несбалансированных данных классификатору не хватило знаний. В дальнейших экспериментах будут использоваться текстовые абзацы веб-страниц: качество сравнимо со второй группой и при этом имеет только основной текст веб-страницы, без шума в виде комментариев, которые не всегда несут полезную информацию.

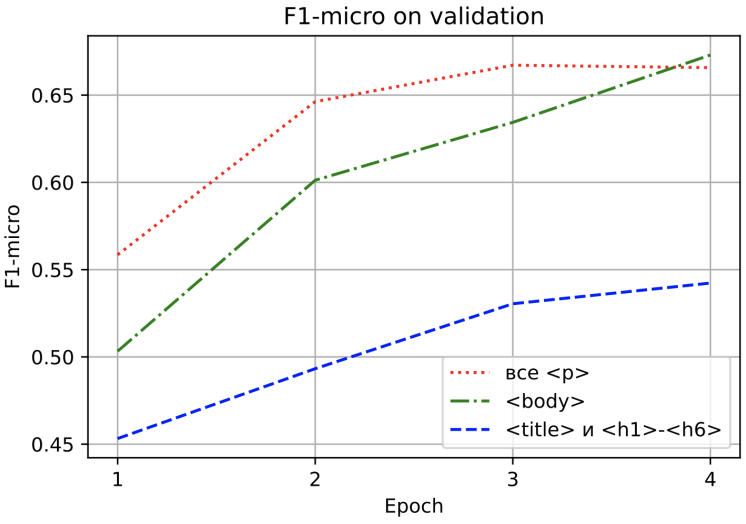

Рис 14. F1-micro на текстовых данных тегов

Ранее было высказано предложение: если текст состоит из более чем 500 токенов, то его стоит разделить на несколько частей. Делается это с предположением о том, что разделение может помочь классификатору собрать дополнительную информацию.

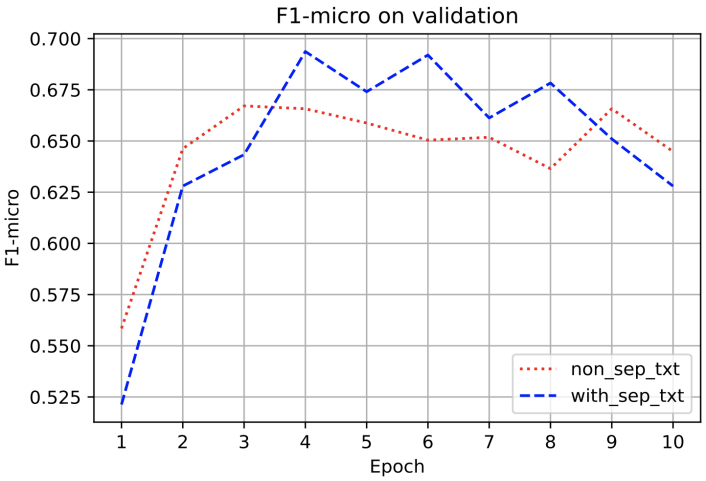

Рис. 15. F1-micro на данных с разделением и без него

Посмотрев на график рисунка 15, можно сделать вывод, что разделенные данные на порции текстов улучшили качество классификатора. Далее работать будет проводиться только с ними.

Следующий базовый эксперимент: посмотреть, как ведет себя метрика F1-micro на обучающей и валидационной выборках. Нужно понять, насколько модель подвержена переобучению.

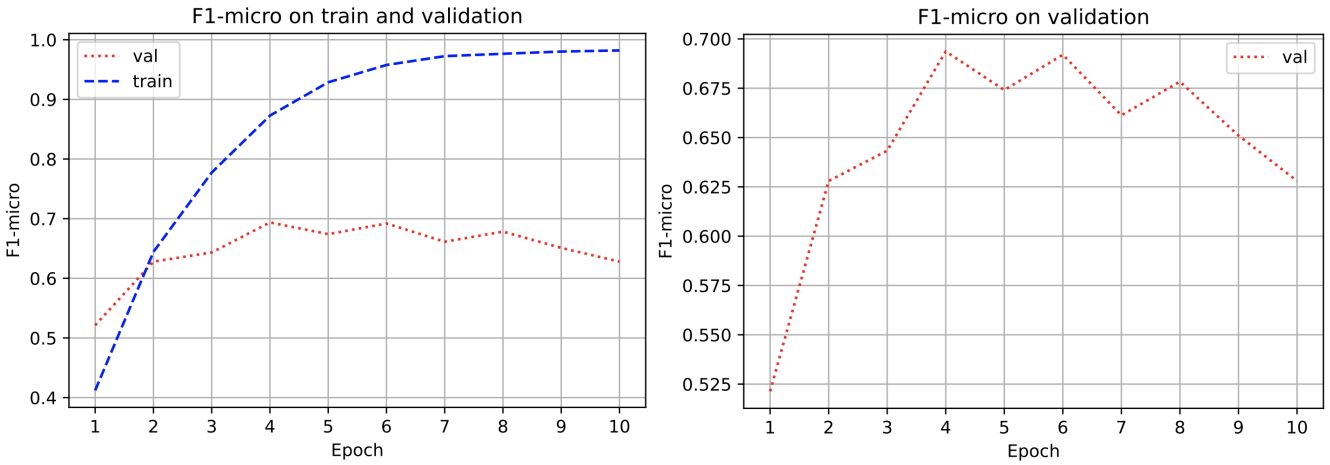

Рис. 16. Качество модели на train и validation

На рисунке 16 видно, что модель очень хорошо подстраивается под обучающую выборку. Но также заметно, что для дообучения достаточно четырех эпох (далее идут лишь флуктуации без улучшений качества), далее идут лишь флуктуации без улучшений качества. Поэтому можно считать, что подстраивание под обучающую выборку не мешает получить хорошую модель.

Далее стоит посмотреть, как гиперпараметры batch и learning rate влияют на качество.

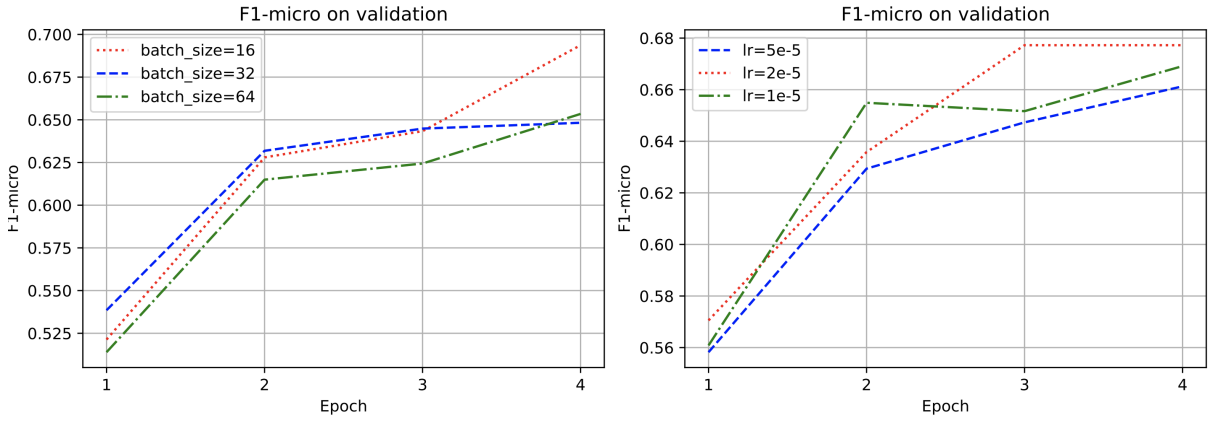

Рис. 17. Сравнение батча (слева) и темпа обучения (справа)

Судя по графикам на рисунке 17, оптимальными параметрами являются: размер батча = 16, шаг обучения = 2e-5 (при большем темпе обучения качество ухудшается).

Предварительно обученные языковые модели, такие как BERT, используют всю информацию в предложении, даже знаки препинания и стоп-слова. Поэтому возникло предположение, что лемматизация слов и удаление стоп-слов может ухудшить результат модели. В таблице 1 показаны результаты тестирования.

Обработка текстовых данных	**F1-micro на тестировании**
Со стоп-словами и лемматизацией	68%
Со стоп-словами и без лемматизации	67.04%
Без стоп-слов и без лемматизации	**68.73%**

Таблица 1. Как влияет предобработка данных на результат модели BERT

Также проведены эксперименты для следующих описанных методов аугментации:

1. Для каждого текста поочередно применяется три метода: контекстная замена случайных слов, контекстная вставка слов и случайное удаление слов;
2. Для каждого текста поочередно применяется три метода: замена случайных слов близкими для них словами (Word2Vec), случайная вставка синонимов и случайное удаление слов;
3. Обратный перевод и затем перефразирование;
4. Замена случайных слов синонимами (Word2Vec) и затем обратный перевод;
5. Замена случайных слов синонимами (Word2Vec), перестановка случайных позиций слов местами и затем обратный перевод.

Результаты тестирования модели BERT показаны в таблицах 2-3 и рисунке 18.

Тестовый датасет (проект FOSS)	
Метод	**F1-micro на тестировании**
1	74,3%
2	77%
3	72.8%
4	76.3%
5	**78%**

Таблица 2. Тестирование модели на данных после аугментации

Веб-страницы сайта lenta.ru	
Метод	**F1-micro на тестировании**
1	76.1%
2	75%
3	79%
4	79.8%
5	**82.3%**

Таблица 3. Тестирование модели на данных с lenta.ru после аугментации

Для верификации алгоритма классификатора был взят датасет с веб-страницами сайта lenta.ru, у которого большинство категорий сбалансировано. По итогу качество модели показало результат в 90% на тесте и 98% на обучении на четвертой эпохе, что говорит о хорошем наборе данных, потому что нет переобучения, и правильной работе алгоритмов классификации. Чтобы удостовериться в правильности реализованных методах аугментации, этот датасет был искусственно уменьшен для несбалансированности данных (на 31%). С таким набором результат стал 74% на тесте. После проведения экспериментов с методами увеличения данных видно (таблица 3), что за счет пятой связки методов аугментации удалось максимально приблизиться к исходному результату.

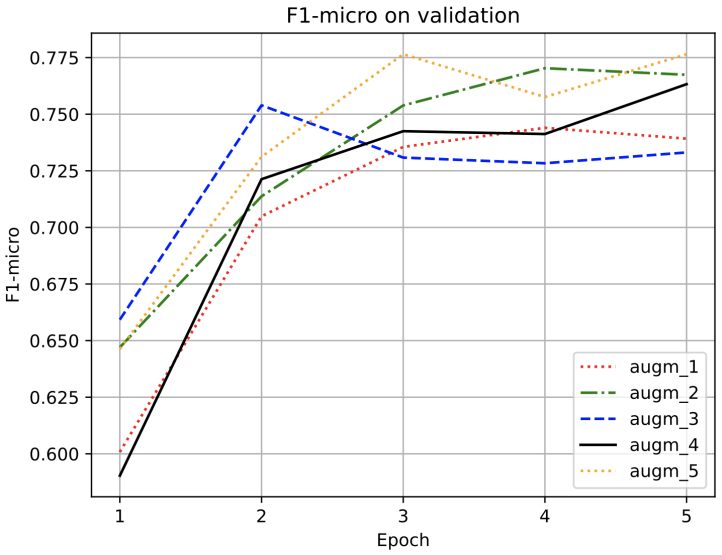

Рис. 18. Качество модели для разных методов аугментации

Сравнение полученных результатов с другими работами

Как было сказано ранее, сравнивать качество работы полученных алгоритмов с методами, представленными в других аналогичных работах, затруднительно в силу того, что для русского языка не существует стандартного датасета для задачи классификации веб-страниц. Вторая проблема заключается в том, что объем данных и количество категорий сильно различаются и где-то в принципе на этом не акцентируют внимание.

Но в одном из соревнований на kaggle.com рассматривается датасет с веб-страницами, который имеет 7 несбалансированных категорий (веб-страницы взяты с российского сайта lenta.ru). В наборе данных помимо веб-страниц есть столбец с их

текстовым содержимым, поэтому, чтобы уменьшить времязатраты, возьмем данный столбец, как после парсинга. Для сравнения возьмем ноутбук участника declot [19] и для аугментации выберем пятый метод, показавший самые высокие результаты среди остальных. Автор работы в качестве классификатора использует SGDClassifier – модель логистической регрессии со стахостическим градиентным спуском, для обработки данных – лемматизацию и TF-IDF. Результат представлен в таблице 4.

	Без аугментации	**С аугментацией (метод 5)**	**News dataset from Lenta.ru**
F1-macro	94%	95.3%	91%

Таблица 4. Результаты сравнения с работой [19]

11 Анализ результатов и выводы

При анализе значимости html-тегов выяснилось, что тег <body> содержит много ненужной информации, которая сказывается плохим образом на времени работы алгоритмов аугментации и классификатора. Из-за этого основным требованием может быть рассмотрение меньшего количества текстовых данных для одной веб-страницы. В данной работе основным тегом для сбора текста выступил <p>, которого может быть много по веб-странице. Но также можно рассматривать, в зависимости от информативности и сложности модели, конкретные абзацы (начало, середину или конец), а не все html-теги <p>.

Лучшим из разработанных методов аугментации оказался метод, основанный на объединении замены случайных слов синонимами (Word2Vec), перестановки случайных позиций слов местами и обратного перевода. Разница в качестве на обучении и валидации достаточно не маленькое, поэтому датасет стоит расширять уже вручную и снова проводить эксперименты. При этом заметно, что из-за несбалансированности качество классификации сильно изменяется в зависимости от категории. Но BERT все равно хорошо справляется с разделением по категориям, учитывая, что большинство может пересекаться друг с другом. В итоге удалось увеличить качество модели на 15% по сравнению с начальными результатами. Также для обученного набора данных, полученного с помощью этого метода, приведены графики с увеличенным объемом данных и матрицей ошибок [19-20].

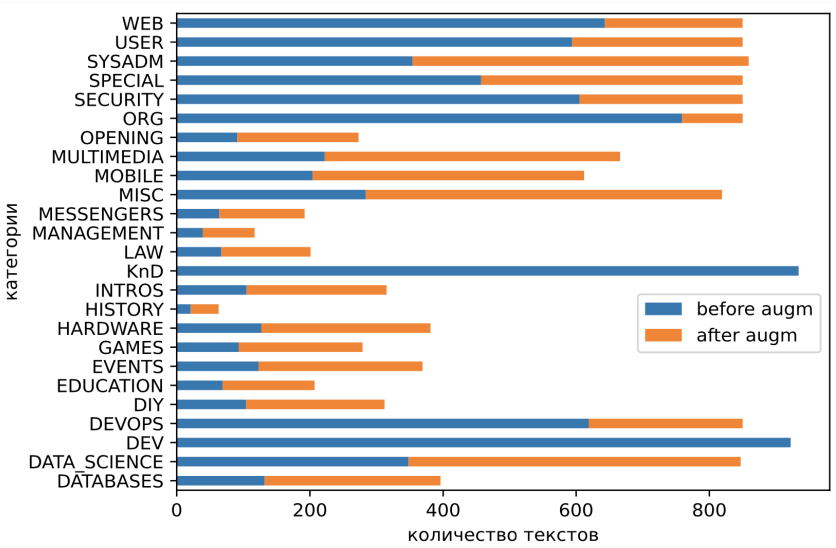

Рис. 19. Объем текстов до и после аугментации

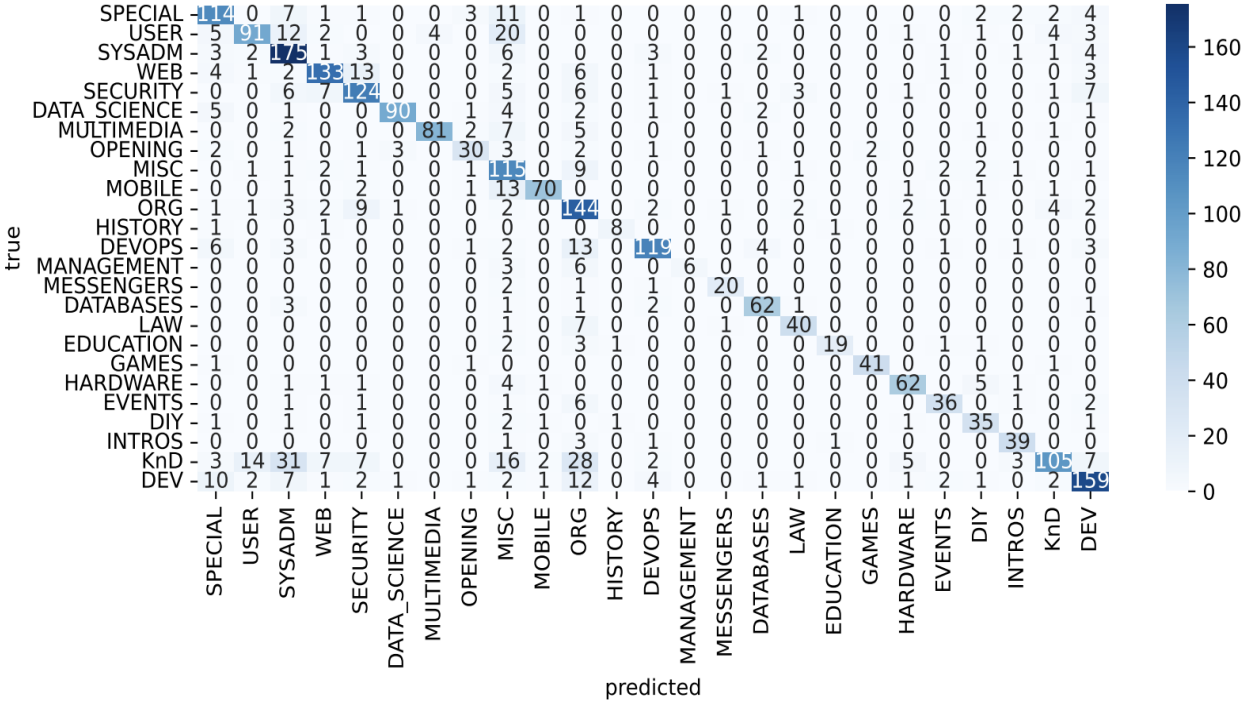

Рис. 20. Матрица ошибок для лучшей связки методов аугментации

По рисунку 20 можно сделать следующие выводы: категории, которые имеют однозначное представление и ни с какими категориями никак не пересекаются, получили почти максимальный результат (‘MULTIMEDIA’, ‘GAMES’, ...); категории ‘USER’, ‘SYSADM’, ‘MISC’, ‘ORG’ получили наиболее низкие результаты, потому что часть текстов была отнесена к категории ‘KnD’. При этом найти какое-то логичное объяснение, связанное с общей тематикой или ошибочно размеченными данными, не удалось.

Также интересно, что, не используя лемматизацию, результат получился лучше. Это можно объяснить тем, что BERT использует архитектуру трансформера, работающую на принципе внимания, и из-за этого он смотрит только на слова, которые влияют на результат, а не на слова, которые распространены во всех данных (можно не удалять стоп-слова). Также BERT использует BPE (Byte-Pair Encoding для сокращения размера словаря), и большинство слов и так в конечном итоге будут декодированы нужным образом. Поэтому лемматизация в данной задаче лишняя и моделе необходима дополнительная информация в качестве словоформ лексем.

Из проведенных исследований можно сделать следующие выводы:

- Для методов аугментации, затрагивающих вероятностной выбор слов, не стоит случайным образом выбирать слова, а, например, выбирать только несущественные слова (слова с маленьким TF-IDF) или наоборот – только значимые (по ключевым словам);

- Объем текстов для языковых моделей играет большую роль при классификации, поэтому разные html-теги могут по-разному влиять на результат и время выполнения нейронных сетей, а также разделять текстовые данные на порции, если их много;

- Парафразер хуже всего справляется с аугментацией текстов в данной задаче. Этот метод может добавить много зашумленности в датасет. Он подходит больше для перефразирования коротких заголовков, предложений;

- Предобученный токенизатор *DeepPavlov/rubert-base-cased* не сильно подходит для решаемой задачи, много специфических слов, которые он не может качественно разделить. Стоит обучить свой токенизатор или провести эксперименты с другими предобученными моделями;

- BERT хорошо справляется с многоклассовой классификацией и даже с несбалансированностью, но, чтобы не было переобучения, нужно достаточно большое количество данных для каждой категории.

12 Графический интерфейс системы классификации веб-страниц

Одной из задач в данной работе является создание пользовательского интерфейса, с помощью которого можно решать задачу классификации веб-страниц.
Графический интерфейс разрабатывался с помощью библиотеки PyQt5. [19]

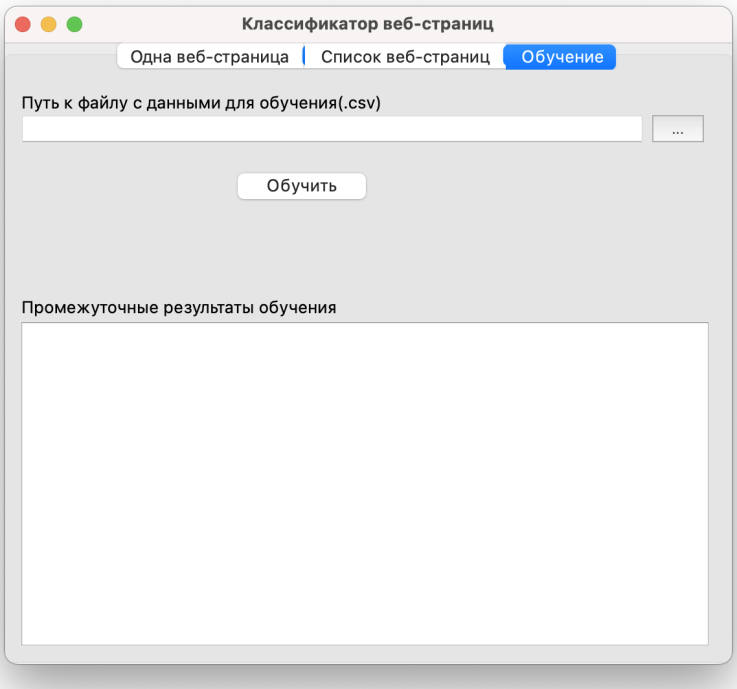

Рис. 21. Графический интерфейс

Приложение (Рис. 21) состоит из одного окна, реализованного на основе класса QMainWindow, внутри которого содержатся три вкладки (QTabWidget).

С помощью первых двух можно классифицировать соответственно одну веб-страницу или несколько. Для этого нужно выбрать обученную модель и веб-страницу или список веб-страниц, категория которых неизвестна. И после будет выдан ответ. С помощью третьей же вкладки происходит обучение. Для этого нужно выбрать путь к файлу .csv с данными (‘url’, ‘category’). Тексты будут предобработаны, векторизованы согласно описанному алгоритму. После построится классификатор на предобученной модели BERT с возможностью его сохранения.

Чтобы программа не зависала, обучение выполняется в отдельном потоке. Это реализовано с помощью класса QThread. За выполнением программы можно следить с помощью индикатора выполнения. Так же, чтобы программа не завершалась аварийно, обработаны основные ошибки (например, ввод неправильной веб-страницы).

13 Заключение

Основные результаты работы:

1. Реализованы четыре метода аугментации текстовых данных;
2. Создана система для определения классификации веб-страниц с графическим интерфейсом на основе языковой модели BERT;
3. Выполнена серия вычислительных экспериментов, результаты которых показали, что при наиболее сбалансированном наборе данных на тестовой выборке качество получилось **78%**.

Тем не менее, методы аугментации все еще требуется улучшать для того, чтобы их было возможно использовать для решения практических задач, в том числе при создании новых датасетов.

Кроме повышения точности классификатора и алгоритмов аугментации можно выделить следующие направления дальнейшей работы:

1. Для более быстрого обучения и работы модели BERT использовать дополнительные методы для нее: knowledge distillation, dynamic quantization;
2. Разработка других методов аугментации: MixUp для текстов, с помощью синтаксического дерева или генеративных моделей;
3. Реализация алгоритма применения методов аугментации текстов с выбором конкретного метода аугментации после каждой эпохи обучения.

Список литературы

1. T. Joachims. A Probabilistic Analysis of the Rocchio Algorithm with TFIDF for Text Categorization // Proceedings of International Conference on Machine Learning (ICML). URL: https://www.cs.cornell.edu/people/tj/publications/joachims_97a.pdf (дата обращения: 02.12.2022)
2. Маслов М., Пяллинг А., Трифонов С. Автоматическая классификация веб-сайтов // Труды РОМИП, 2007. 6 с. URL: http://rcdl.ru/doc/2008/230_235_paper27.pdf(дата обращения: 02.12.2022)
3. Рукавицын А. Н. Разработка модели классификации веб-страниц с использованием методов интеллектуального анализа данных // Материалы 19-й международной конференции по программным вычислениям и измерениям. 2016. URL: https://izv.etu.ru/assets/files/izv-etu-4-2016-12-20.pdf (дата обращения: 02.12.2022)
4. Новожилов Д.А., Чечулин А.А., Котенко И.В. Улучшение категорирования веб-сайтов для блокировки неприемлемого содержимого на основе анализа статистики html-тэгов // Информационно-управляющие системы -2016. - № 6(85). - С. 65-73.
5. Patrick Dave P. Woogue, Gabriel Andrew A. Pineda. Automatic Web Page Categorization Using Machine Learning and Educational-Based Corpus // IJCTE, Vol. 9, No. 6, 2017, URL: http://www.ijcte.org/vol9/1180-IT026.pdf (дата обращения: 20.01.2023)
6. Tobias Eriksson. Automatic web page categorization // Project. – 2013. URL: https://www.diva-portal.org/smash/get/diva2:700316/FULLTEXT01.pdf (дата обращения: 20.01.2023)
7. Web page classification by deep learning [CNN] // Kaggle notebook. URL: https://www.kaggle.com/code/shawon10/web-page-classification-by-deep-learning-cnn/notebook (дата обращения: 20.01.2023)
8. Lisa Hoek. Web classification using DMOZ // Bachelor thesis computing science. 2021. URL: https://www.cs.ru.nl/bachelors-theses/2021/Lisa_Hoek___1009553___Web_classification_using_DMOZ.pdf (дата обращения: 20.01.2023)
9. Building a URL classifier using DMOZ data // Github project. 2017. URL:

https://utatds.github.io/2017-01-18-URL-classification-using-DMOZ-data/ (дата обращения: 20.01.2023)

10. Goran Matosevic. Using Machine Learning for Web Page Classification in Search Engine Optimization // Future Internet. - 2021. URL: https://www.researchgate.net/publication/348201897_Using_Machine_Learning_for_Web_Page_Classification_in_Search_Engine_Optimization (дата обращения: 20.01.2023)
11. Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. 2018.
12. Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. 2017.
13. Yingzhe Dong, Da Yan. BELT: A Pipeline for Stock Price Prediction Using News. 2020.
14. Andreas Mueller. Introduction to Machine Learning with Python: A Guide for Data Scientists. 2017.
15. Sokolova M., Lapalme G. A systematic analysis of performance measures for classification tasks // Information Processing & Management. 2009, с. 427–437.
16. Matthew Ciolin, David Noever, Josh Kalin. Back Translation Survey for Improving Text Augmentation. 2022. URL: https://arxiv.org/pdf/2102.09708.pdf (дата обращения 23.03.2023)
17. Дьяконов А.Г. Аугментация для текстов (Text Augmentation). 2020. URL: https://alexanderdyakonov.wordpress.com/2020/11/09/text-augmentation/ (дата обращения 23.03.2023)
18. Jason Wei, Kai Zou. EDA: Easy Data Augmentation Techniques for Boosting Performance on Text Classification Tasks. 2019. URL: https://arxiv.org/pdf/1901.11196.pdf (дата обращения 23.03.2023)
19. Классификация текстов новостей Lenta.ru // Kaggle notebook ‘Sport news ru’. URL: https://www.kaggle.com/code/declot/sport-news-ru/notebook (дата обращения: 23.03.2023)
20. What is PyQt? // URL: https://riverbankcomputing.com/software/pyqt (дата обращения: 01.02.2023)