

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение

высшего образования

**«КУБАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»**

**(ФГБОУ ВО «КубГУ»)**

**Факультет компьютерных технологий и прикладной математики**

**Кафедра анализа данных и искусственного интеллекта**

**КУРСОВАЯ РАБОТА**

**РЕШЕНИЕ ЗАДАЧИ КЛАССИФИКАЦИИ С ПРИМЕНЕНИЕМ БИБЛИОТЕК TENSORFLOW И KERAS**

Работу выполнила \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Д.В. Кощавцев

(подпись)

Направление подготовки 01.03.02 Прикладная математика и информатика курс 3

Направленность (профиль) Математическое и информационное обеспечение экономической деятельности

Научный руководитель

канд. физ.-мат. наук, доц. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Г.В. Калайдина

(подпись)

Нормоконтролер

канд. физ.-мат. наук, доц. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Г.В. Калайдина

(подпись)

Краснодар

2024

**РЕФЕРАТ**

Курсовая работа 27c., 15 источн., 15 рис.

МЕТОДЫ КЛАССИФИКАЦИИ, МАШИННОЕ ОБУЧЕНИЕ, PYTHON, НЕЙРОННЫЕ СЕТИ, KERAS, ГРАДИЕНТНЫЙ БУСТИНГ, TENSORFLOW, PYCARET, NLP

Объектом исследования данной курсовой работы являются классические модели обучения и нейронные сети.

Целью работы является решение задачи обработки естественного языка и классификации новостных статей по темам, а также сравнение результатов решения задачи классическими алгоритмами и нейронными сетями.

Работа посвящена исследованию основных методов классификации данных, таких как: метод K-ближайших соседей, метод опорных векторов, наивный байесовский метод, деревья принятия решений, градиентный бустинг, нейронные сети на Keras, нахождению оптимальных инструментов для решения задачи классификации, а также решению задачи классификации новостных статей.

**СОДЕРЖАНИЕ**

Введение............................................................................................................. 3

1 Методы классификации................................................................................. 4

* 1. Метод К-ближайших соседей................................................................ 4
  2. Наивный байесовский классификатор.................................................. 4
  3. Деревья принятия решений.................................................................... 6
  4. Метод опорных векторов....................................................................... 7
  5. Градиентный бустинг............................................................................. 8

2 Инструменты для решения задачи классификации.................................... 10

* 1. Предобработка входных данных .......................................................... 10
     1. Стандартизация и токенизация..................................................... 10
     2. Лемматизация и стемминг............................................................. 10
     3. Векторизация.................................................................................. 11
  2. Модель глубокого обучения.................................................................. 12
  3. Классические методы обучения............................................................ 14

1. Решение задачи классификации.................................................................. 15
   1. Нейронная сеть на Keras......................................................................... 15
      1. Предобработка входных данных................................................... 15
      2. Развертывание модели................................................................... 18
      3. Результаты нейронной сети........................................................... 20
   2. Классические методы обучения............................................................ 21
   3. Сравнение результатов........................................................................... 23

Заключение......................................................................................................... 25

Список используемых источников................................................................... 26

**ВВЕДЕНИЕ**

За последние несколько лет тема искусственного интеллекта вызвала сильный резонанс в обществе. Всё больше эта технология внедряется в разные сферы человеческой жизни и помогает оптимизировать производство, обрабатывать большие объёмы данных и автоматизировать рутинные задачи.

С помощью обработки естественного языка компьютеры учатся отвечать на вопросы, вести беседы и понимать человеческую речь. Задача обработки текста долгое время была трудно решаемой, создавались разные методы обработки текста, часть из которых будет освещена в курсовой работе.

В данной курсовой будет продемонстрировано решение задачи классификации текста, а также сравнение результата нейронной сети с классическими методами обучения.

Курсовая работа состоит из трёх разделов, введения, заключения и списка использованных источников. В первом разделе рассмотрены основные методы классификации, второй раздел посвящён выбору инструментов для решения поставленной задачи, в третьем разделе подробно описано программное решение задачи и сравнение результатов.

Целью данной курсовой работы является решение задачи классификации текстовых новостных статей по темам.

Задачами курсовой являются: исследование методов обработки текста,

решение задачи классификации текста, программная реализация решения, сравнение результатов работы нейросети с классическими методами обучения.

**1 Методы классификации**

* 1. **Метод K-ближайших соседей**

Метод K-ближайших соседей представляет собой тип контролируемого алгоритма машинного обучения, используемого для классификации, регрессии, поиска обнаружения выбросов. Он использует для обучения все данные новой точки данных или экземпляра. Этот метод непараметрический алгоритм обучения: он не предполагает ничего о базовых данных.

При добавлении новой точки к набору данных, метод К-ближайших соседей использует только одну часть данных для определения класса добавленной точки. Так как алгоритму не нужно повторно просматривать все точки, это делает его алгоритмом ленивого обучения. Алгоритм также ничего не предполагает о базовых характеристиках данных, он не ожидает, что данные будут соответствовать какому-либо типу распределения, так как это непараметрический алгоритм обучения.

Несмотря на то, что формально фаза обучения отсутствует, алгоритм может легко переобучиться. Из-за этого алгоритму легко подстроиться под конкретные данные.

* 1. **Наивный байесовский классификатор**

Наивный Байес – это самый простой алгоритм, который можно применить к данным. Этот метод – это набор алгоритмов контролируемого обучения, основанный на применении теоремы Байеса с предположением об условной независимости между каждой парой характеристик при заданном значении переменной класса. Теорема Байеса утверждает следующее отношение, учитывая переменную класса и зависимый вектор признаков через :

.

Используя наивное предположение об условной независимости,

,

для всех , это отношение упрощается до:

.

Тогда искомый объект:

Различные наивные байесовские классификаторы различаются в основном предположениями, которые они делают относительно распределения . Несмотря на упрощённые предположения, такие классификаторы хорошо работают во многих реальных ситуациях. Таким классификаторам требуется небольшой объём обучающих данных для оценки параметров.

К плюсам алгоритма можно отнести: быстрое предсказание класса тестового набора; алгоритм справляется с многоклассовым прогнозированием; производительность алгоритма гораздо выше, чем у многих других простых алгоритмов и требует меньшего объёма обучающих данных; метод хорошо работает с категориальными признаками. К минусам относятся: при наличии у переменной категории, которая не наблюдалась в обучающем наборе данных, модель может присвоить ей нулевую вероятность, из-за чего она не сможет сделать предсказание; значения спрогнозированных вероятностей не всегда являются достаточно точными; ограничением данного алгоритма является предположение о независимости признаков.

* 1. **Деревья принятия решений**

Деревья решений – это непараметрический контролируемый метод обучения. Решающее дерево предсказывает значение целевой переменной с помощью применения последовательности простых решающих правил (которые называются предикатами). Обобщающая способность решающих деревьев невысока, их предсказания вычисляются просто, из-за чего решающие деревья часто используют как части моделей, делающих предсказания на основе агрегации предсказаний других моделей.

У дерева решений есть несколько свойств:

– дерево решений не сможет экстраполировать зависимости за границы области значений обучающей выборки;

– выученная функция является кусочно-постоянной, из-за чего производная равна нулю везде, где задана;

– дерево решений способно идеально приблизить обучающую выборку и ничего не выучить.

Можно построить дерево, запоминающее всю выборку, однако на тестовых данных такой алгоритм не покажет высокого качества. Однако, можно поставить другую задачу: построить оптимальное с точки зрения качества на обучающей выборке дерево минимальной глубины, чтобы снизить переобучение. Проблема в том, что поиск такого дерева пока не имеет решения за полиномиальное время. Решить это можно двумя способами:

– Разрешить искать не оптимальное решение, а просто достаточно хорошее. Начать можно с того, чтобы строить дерево с помощью жадного алгоритма, то есть не искать всю структуру сразу, а строить дерево этаж за этажом. Тогда в каждой внутренней вершине дерева будет решаться задача, схожая с задачей построения решающего пня;

– Заняться оптимизацией с точки зрения computer science — наивную версию алгоритма (перебор наборов возможных предикатов и порогов) можно ускорить.

Ставится задача построить решающее дерево, наилучшим образом представляющее целевую зависимость. Однако, оптимизировать структуру дерева с помощью градиентного спуска невозможно, из-за обнуления производных.

Алгоритм построения решающего дерева жадным алгоритмом:

1. Создаём вершину ;
2. Если выполнен критерий остановки, то останавливаемся, объявляем эту вершину листом и ставим ей в соответствие ответ, после чего возвращаем её;
3. Иначе: находим предикат, который определит наилучшее разбиение текущего множества объектов на две подвыборки и , максимизируя критерий ветвления;
4. Для и рекурсивно повторим процедуру.

* 1. **Метод опорных векторов**

Метод опорных векторов (Support Vector Machines — SVM) — это набор контролируемых методов обучения, используемых для классификации, регрессии и обнаружения выбросов. С точки зрения классификации, это дискриминантный классификатор, который используется для поиска оптимальной гиперплоскости с целью классификации данных по различным классам. В двумерном пространстве эту оптимальную гиперплоскость можно представить как линию, разделяющую пространство на две части: где одна часть пространства содержит точки данных, которые принадлежат одному классу, а другая часть пространства содержит точки данных, принадлежащие другому. класс. Концепция линий, действующих как классификатор, верна только в том случае, если точки данных линейно разделимы. SVM также можно использовать для поиска оптимальной кривой, которая может использоваться для классификации точек данных, которые нельзя разделить линейно.

Преимуществами данного метода являются:

– Эффективен в пространствах больших размеров;

– По-прежнему эффективен в случаях, когда количество измерений превышает количество образцов;

– Использует подмножество обучающих точек в функции принятия решений (называемых опорными векторами), поэтому это также эффективно с точки зрения памяти;

– Универсальность: для функции принятия решения могут быть указаны различные функции ядра. Предоставляются общие ядра, но также можно указать собственные ядра.

К недостаткам данного метода можно отнести:

– Если количество функций намного превышает количество выборок, избегайте чрезмерной подгонки при выборе функций ядра, и термин регуляризации имеет решающее значение;

– SVM не предоставляют напрямую оценки вероятностей, они рассчитываются с использованием дорогостоящей пятикратной перекрестной проверки;

**1.5 Градиентный бустинг**

Бустинг, использующий деревья решений в качестве базовых алгоритмов, называется градиентным бустингом над решающими деревьями. Он отлично работает на выборках с «табличными», неоднородными данными. Примером таких данных может служить описание пользователя Яндекса через его возраст, пол, среднее число поисковых запросов в день, число заказов такси и так далее. Такой бустинг способен эффективно находить нелинейные зависимости в данных различной природы. Этим свойством обладают все алгоритмы, использующие деревья решений, однако именно GBDT обычно выигрывает в подавляющем большинстве задач. Хотя деревья решений и являются традиционным выбором для объединения в ансамбли, никто не запрещает использовать и другие алгоритмы, например, линейные модели в качестве базовых. Стоит только понимать, что построенная композиция, по сути, окажется линейной комбинацией линейных моделей.

**2 Инструменты для решения задачи классификации**

Перед началом решения задачи классификации необходимо предобработать входные данные в подходящий для работы модели вид. В машинном обучении в задачах обработки естественного языка есть своя терминология, введём несколько определений.

Стандартизация текста заключается в удалении из текста знаков препинания, приведение его к нижнему регистру и удаление стоп слов. Стоп-слова – это слова или фразы, которые не несут в себе смысловой нагрузки и затрудняют индексирование текста. Документ в nlp является отдельной единицей текста, который часто представлен в виде абзаца, предложения или слова. Токен – это отдельная единица документа, представленная словом, буквой или символом. Токенизация – это процесс разбиения документа на токены.

**2.1 Предобработка входных данных**

**2.1.1 Стандартизация и токенизация**

Для обработки текста есть удобные библиотеки nltk и spacy, которые содержат в себе реализованные функции, такие как токенизатор, векторизатор, стоп-слова, лемматизатор и прочее. Это сильно облегчает решение задачи, так как не придётся реализовывать эти функции вручную.

Для токенизации используется готовый токенизатор языковой модели из библиотеки spacy.

**2.1.2 Лемматизация и стемминг**

При обработке текста использовалась лемматизация. Реализованный лемматизатор взят из библиотеки spacy. Почему же было использована именно лемматизация? Для начала разберёмся в чём различие подходов преобразования слов.

Стемминг – это процесс нахождения основы слова для заданного исходного слова, он осуществляется за счёт удаления окончания и суффиксов слова. В некоторых задачах обработки текста используется именно стемминг, потому что он упрощает слово, что позволяет легче его индексировать, а также этот метод проще остальных в реализации.

Лемматизация основана на словаре, который переводит сложные формы слов в самые простые формы. Её суть схожа со стеммингом, однако лемматизация не просто удаляет суффиксы или окончания, а преобразует слово.

Стемминг чаще всего применяется при расширенном поиске, где степень детализации достаточно грубая. В тоже время лемматизация чаще используется при анализе текста, что больше подходит под поставленную задачу.

**2.1.3 Векторизация**

Векторизация – это подход к преобразованию входных данных из их исходного формата в векторы действительных чисел, которые понятны моделям машинного обучения. Существует много техник векторизации данных, рассмотрим некоторые из них:

а) Мешок слов. Это самая простая техника, которая включает в себя: токенизацию, создание словаря, в который сохраняются только уникальные слова, создание вектора. Из показателей частоты слов получившегося словаря создается разреженная матрица в качестве входных данных. В этой разреженной матрице каждая строка представляет вектор предложения, длина которого равна размеру словаря. Проблема такого метода в том, что модель практически не улавливает взаимосвязи в словах, из-за этого теряется контекст и модель даёт плохие результаты на длинных предложениях.

б) TF-IDF. Это числовой статистический показатель, который отражает важность слова для документа. И хотя эта методика тоже основана на частотности, как и «мешок слов», она использует более сложные расчеты. В чём же различие этих методов? В «мешке слов» векторизация сводилась только к показателям частоты встречаемости слов из словаря по документу. В результате, предлоги или союзы, вклад которых в общий смысл предложения не так важен, получают такой же вес, как и важные для контекста слова. Метод TF-IDF решает эту проблему – слова, которые встречаются слишком часто, не зашумляют менее встречающиеся важные слова.

в) Word2Vec. Этот подход задействует мощь простой нейронной сети для создания эмбеддингов. В «мешке слов» и TF-IDF каждое слово обрабатывалось как отдельная сущность, при этом семантика слова игнорировалась, в то время как word2vec сохраняет контекст и взаимосвязь слов. У этой нейросети есть два способа реализации: skip-gram и CBOW. В методе Skip-Gram, в котором передаётся какое-то слово нашей нейронной сети, и она предпологает возможный контекст, а метод CBOW имеет обратную реализацию: подаётся набор контекстных слов и модель вычисляет текущее слово.

При решении поставленной задачи используется метод Word2Vec, потому что этот метод сохраняет контекст слов в предложении, а это достаточно важный критерий при обработке документов больших размеров. Реализован этот метод с помощью модели Word2Vec из библиотеки gensim, в которой хранятся реализованные модели. Word2Vec обучается на переданном ему датафрейме входных данных.

**2.2 Модель глубокого обучения**

Для решения задачи классификации текста я использовал простую нейронную сеть, реализованную с помощью библиотек Tensorflow и Keras, так как данных недостаточно много для развёртывания сложной сети. TensorFlow – библиотека сквозного машинного обучения Python для выполнения высококачественных численных вычислений. С помощью TensorFlow можно построить глубокие нейронные сети для распознавания образов и рукописного текста и рекуррентные нейронные сети для NLP (обработки естественных языков). Также есть модули для векторизации слов (embedding) и решения дифференциальных уравнений в частных производных (PDE). Этот фреймворк имеет отличную архитектурную поддержку, позволяющую с легкостью производить вычисления на самых разных платформах, в том числе на десктопах, серверах и мобильных устройствах. Основной козырь TensorFlow это абстракции. Они позволяют разработчикам сфокусироваться на общей логике приложения, а не на мелких деталях реализации тех или иных алгоритмов. Keras — одна из основных библиотек Python с открытым исходным кодом, написанная для построения нейронных сетей и проектов машинного обучения. В этой библиотеке реализованы практически все автономные модули нейронной сети, включая оптимизаторы, нейронные слои, функции активации слоев, схемы инициализации, функции затрат и модели регуляризации. Это позволяет строить новые модули нейросети, просто добавляя функции или классы. И поскольку модель уже определена в коде, разработчику не приходится создавать для нее отдельные конфигурационные файлы.

Нейросеть состоит из слоя Dense с 16 нейронами с функцией активации ReLu и выходного слоя Dense с 5 нейронами с функцией активации softmax. В качестве оптимизатора используется Adam, так как он даёт наиболее точные результаты, функция потерь – binary crossentropy, которая чаще всего используется в задачах классификации. Метрики, на которых проверялась модель – precision, F1, accuracy. Размер входных пакетов был равен 128, так как на пакетах больших размеров, качество модели ухудшалось.

**2.3 Классические методы обучения**

Решение задачи классическими методами обучения было реализовано с помощью фреймворка PyCaret, благодаря которому возможно оперативно развернуть множество моделей классического обучения и проверить их на множестве метрик. По сравнению с другими открытыми библиотеками машинного обучения, PyCaret – это low-code альтернатива, которая поможет заменить сотни строк кода. Скорость проведения более эффективных экспериментов возрастет экспоненциально. PyCaret – это, по сути, оболочка Python над несколькими библиотеками машинного обучения, такими как scikit-learn, XGBoost, Microsoft LightGBM, spaCy и многими другими. PyCaret проста и удобна в использовании. Все операции, выполняемые PyCaret, последовательно сохраняются в пайплайне полностью готовом для развертывания.

**3 Решение задачи классификации**

**3.1 Нейронная сеть на Keras**

**3.1.1 Предобработка входных данных**

Приступим к реализации решения задачи. Загружаем данные с помощью метода библиотеки pandas read\_csv(), который позволяет считывать csv файлы, для дальнейшей работы с ними. С помощью метода sample(), перемешиваем весь датафрейм, указав в параметрах frac=1 (Рисунок 3.1).

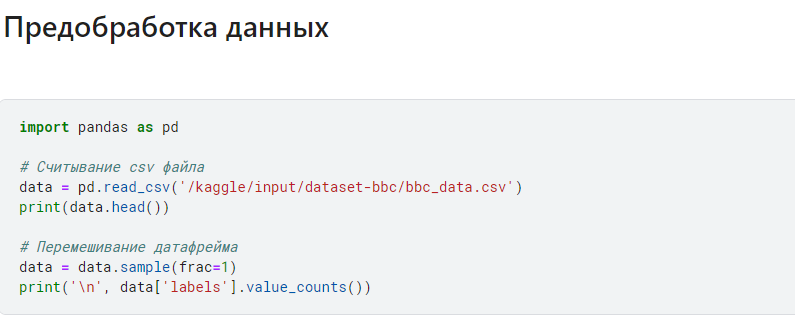


Рисунок 3.1 – Считывание данных

Методом датаффрейма head() отображаем первые несколько элементов. А затем через value\_counts() отображаем, сколько существует экземпляров каждого класса, в которые будут классифицироваться документы (Рисунок 3.2).

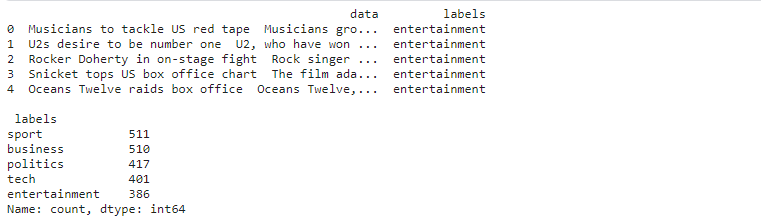


Рисунок 3.2 – Вывод входных данных

Теперь перейдём к обработке входных данных. Первоначально импортируются библиотеки string, содержащая строки и методы работы с ними, spacy, spacy, из которой будет взят лемматизатор, stopwords и word\_tokenize из nltk.

В переменную nlp загружается языковая модель, для работы с английскими словами. В stop\_words загружаем английские стоп-слова из библиотеки nltk.

После подгрузки библиотек и языковой модели реализуется функция обработки preprocess(), в которую в качестве параметра передаётся документ, а именно абзац текста. Функция первоначально приводит весь текст к нижнему регистру, после чего из этого текста исключаются все символы пунктуации, которые достаются из библиотеки string. С помощью языковой модели, которая была сохранена в переменную nlp, текст разбивается на токены. После тонизации, полученный массив обрабатывается с помощью метода lemma\_, который лемматирзирует слова, а также исключаются все слова, которые являются стоп-словами. В конечном итоге, массив разбитый на токены, собирается в строку и возвращается (Рисунок 3.3).

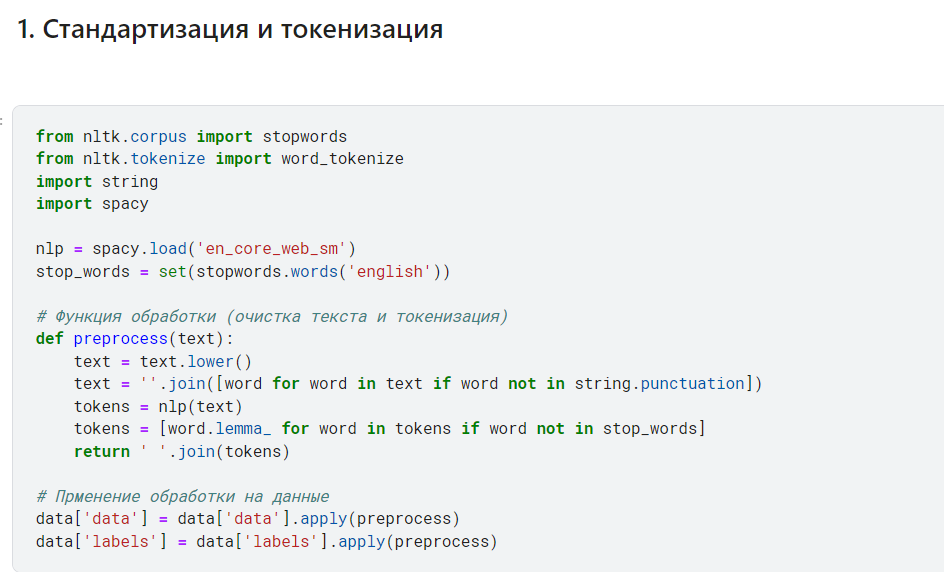


Рисунок 3.3 – Функция обработки документа

С помощью метода apply(), который позволяет применить функцию поэлементно к датафрейму, обрабатываем исходные данные (Рисунок 3.4).

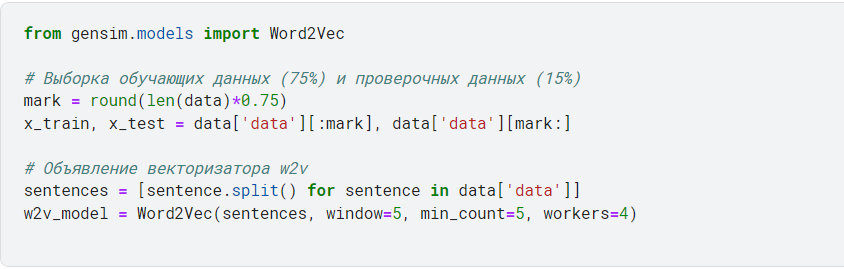


Рисунок 3.4 – Объявление функции векторизации

Теперь перейдём к векторизации текста. Она реализуется через модель Word2Vec, загруженную из библиотеки gensim, и обученную на датафрейме из входных данных. Метки кодируются с помощью OneHotEncoder, загруженного из библиотеки SKLearn (Рисунок 3.5).

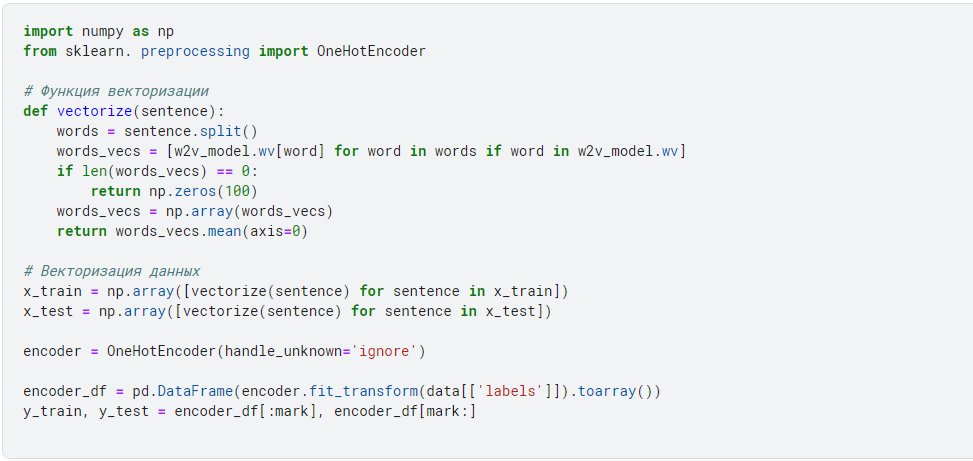


Рисунок 3.5 – Векторизация

**3.1.2 Развёртывание модели**

Узнаем размер обучающей выборки и массива меток (Рисунок 3.6).

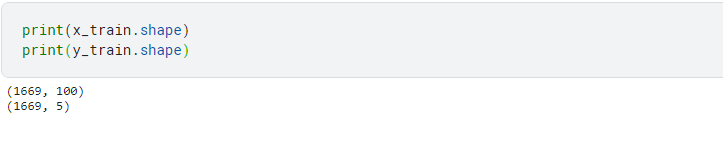
****

Рисунок 3.6 – Размер обучающей выбороки

Нейронная сеть будет реализована с помощью библиотек Tensorflow и Keras. Для начала объявляется последовательность слоёв модели Sequential с 2-мя слоями Dense. После этого модель компилируется методом compile(), в котором в качестве параметров передаются оптимизатор Adam, функция потерь – binary\_crossentropy, метрики – precision, accuracy, F1. С помощью вызова callbacks. ModelCheckpoint() сохраняется лучший результат модели. Метод fit(), запускает обучение модели, где в качестве параметров принимаются: обучающая выборка, набор меток, количество эпох обучения, размеры входных пакетов, данные для проверки модели (Рисунок 3.7).

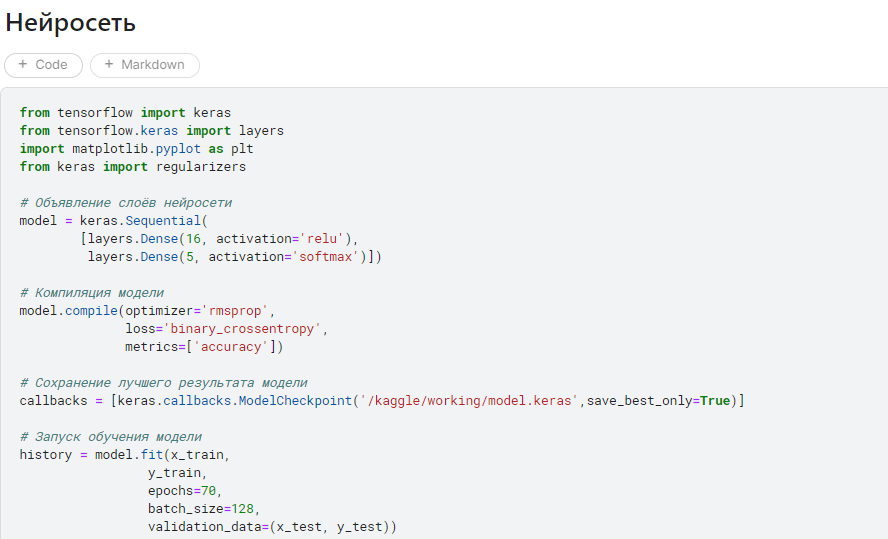


Рисунок 3.7 – Нейросеть

Через метод history получаем значения функции потерь на обучении, потерь на проверке, точность на обучении, точность на проверке (Рисунок 3.8).

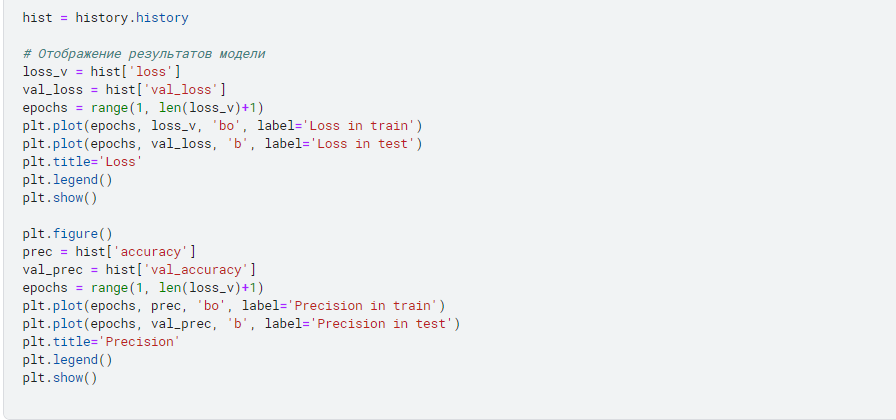


Рисунок 3.8 – Отображение результатов модели

С помощью библиотеки matplotlib.pyplot отображаем графики потерь и точности модели.

**3.1.3 Результаты работы нейросети**

Построим матрицу неточностей на этой метрике и выведем лучший результат модели. Модель достаточно хорошо распределяет классы, но не без ошибок (Рисунок 3.10).

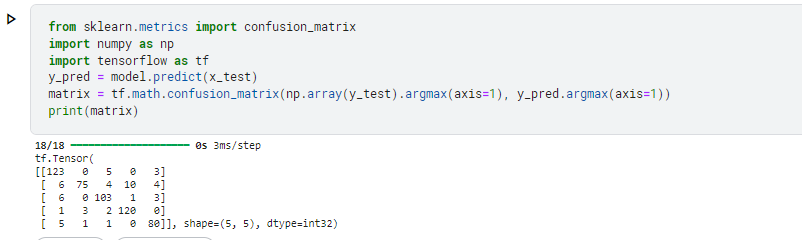


Рисунок 3.10 – Матрица неточностей

На метрике ROC AUC за 50 циклов обучения точность получается в 96%, на precision – 90%, на recall – 88%, потери при обучении – 0.1297 (Рисунок 3.11).

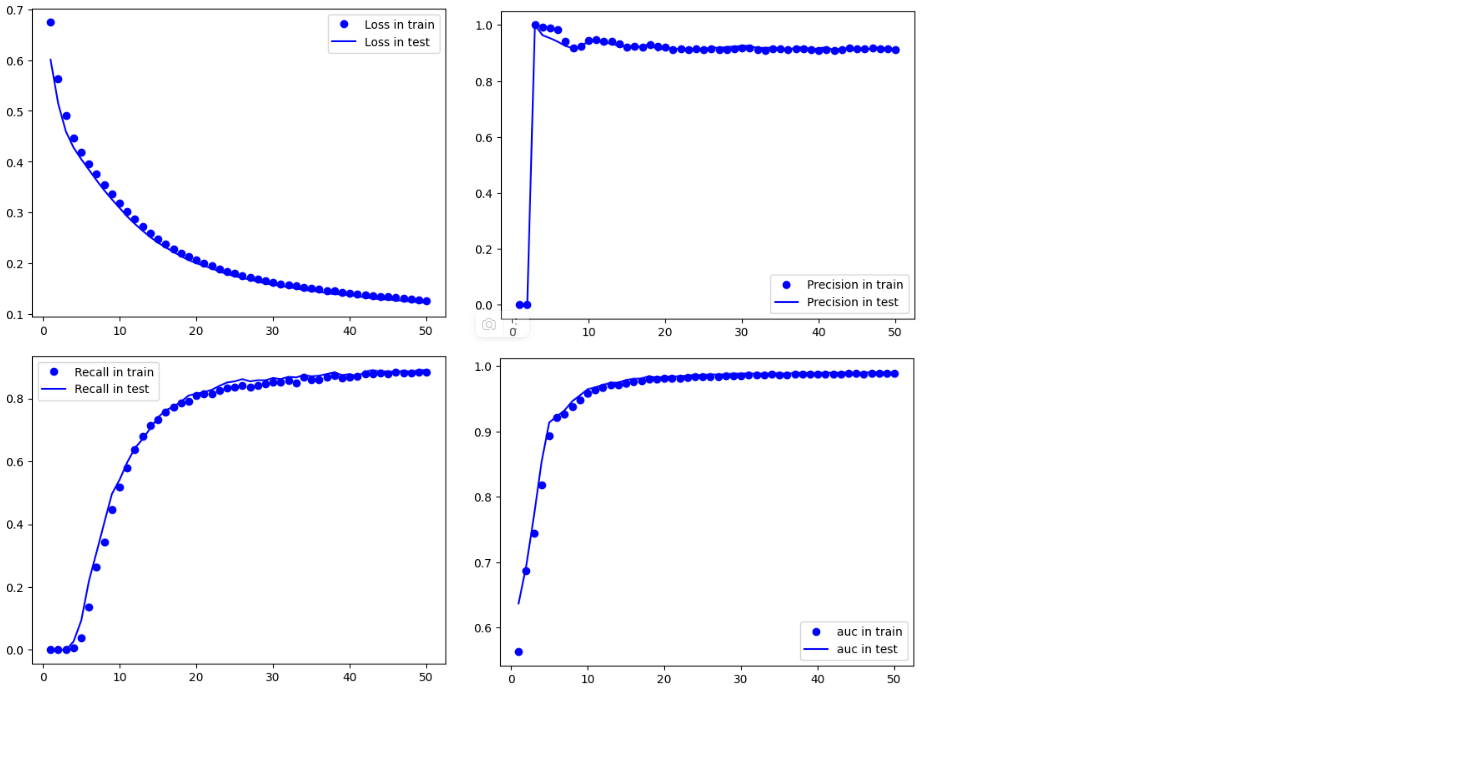
****

Рисунок 3.11 – Графики метрики

**3.2 Классические методы обучения**

Аналогично считываем данные с помощью read\_csv() и перемешиваем датафрейм через метод sample() и обрабатываем данные через токенизатор и лемматизатор (Рисунок 3.12).

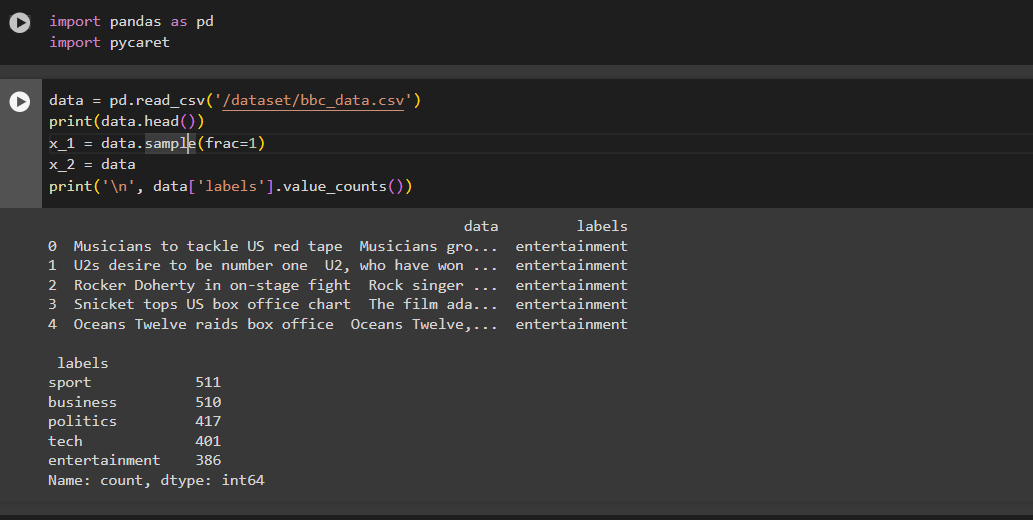
****

Рисунок 3.12 – Считывание данных из файла

После того, как был получен датафрейм данных, импортируется ClassificationExperiment из модуля pycaret.classification, который отвечает за решение задач классификации. Через метод setup(), задаём набор данных, который будет обрабатываться, столбец, отвечающий за метки, номер сессии и размер обучающей выборки (Рисунок 3.13).

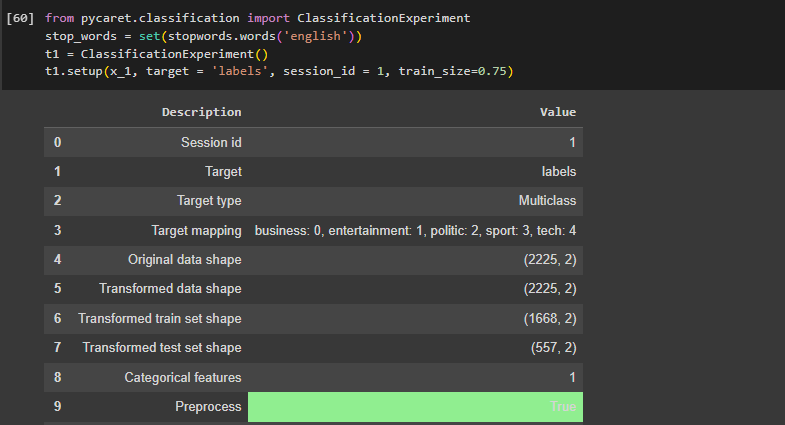


Рисунок 3.13 – Инициализация и настройка среды

После инициализации среды, запускаем обучение моделей, с помощью метода compare\_models(), который автоматически обучает все классические алгоритмы на всех метриках и отображает эти данные в виде таблицы (Рисунок 3.14).



Рисунок 3.14 – Обучение классических моделей

Хорошие результаты показали методы градиентного бустинга на деревьях, метод ближайших соседей и логистическая регрессия.

**3.3 Сравнение результатов**

В конечном итоге нейронная сеть с 2-мя слоями показала результаты:

ROC AUC – 0.9, Precision – 0.9, Recall – 0.88. Классические методы на усреднённых показаниях метрик дали в лучшем случае: ROC AUC – 0.5477, Recall – 0.2452, Precision – 0.734 (на Catboost Classifier), если брать лучшие результаты модели Catboost Classifier, то на precision можно получить 85% (Рисунок 3.15).

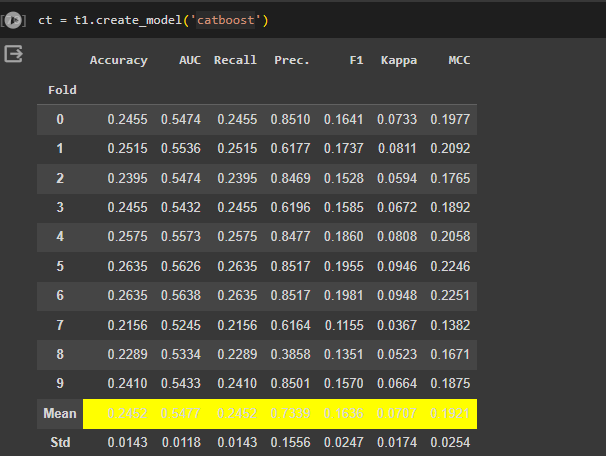
****

Рисунок 3.15 – Работа Catboost Classifier

Таким образом можно заключить, что нейронная сеть лучше классифицировала текст, нежели классические методы, это можно объяснить тем, что нейросети способны находить более сложные зависимости в данных.

**ЗАКЛЮЧЕНИЕ**

Целью данной курсовой работы было решение задачи классификации новостных сводок по заголовкам. Были исследованы классические методы обучения и нейронные сети, составлен алгоритм решения задачи и программная реализация. Нейросеть была реализована с использованием библиотек Keras и Tensorflow и состояла из двух слоёв, где первый содержал 16 нейронов и функцию активации ReLu, а второй был выходным слоём и содержал 5 нейронов, под количество классов, и имел функцию активации Softmax. В качестве оптимизатора модели был выбран Adam, функция потерь – бинарная кроссэнтропия, метрки – ROC AUC, Recall, Precision. Классические методы обучения были рассмотрены с помощью библиотеки Pycaret, которая позволила развернуть и обучить сразу множество алгоритмов.

По итогам сравнения нейросеть за 50 циклов обучения показала результат: ROC AUC – 96%, на precision – 90%, на recall – 88%, потери при обучении – 0.1297. Из классических методов обучения лучшего всего себя показали методы градиентного бустинга над деревьями, из них был выбран CatBoost Classifier, который показал в усреднении метрик: ROC AUC – 55%, Recall – 24%, Precision – 73%, а в самом лучшем результате: ROC AUC – 57%, Recall – 26%, Precision – 85%.

Простая нейронная сеть показала себя лучше, однако не стоит торопиться оставлять классические алгоритмы в прошлом. Есть разные методики обучения, которые могут существенно улучшить результаты классических моделей, к примеру ансамблирование, или использование ансамбля моделей. Нейронные сети также имеют свои недостатки, поэтому инструменты для работы стоит подбирать исходя из задачи и имеющихся входных данных.

**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

1. Берман, К. Основы Python для Data Science : учебное пособие / К. Берман. – СПб. : Питер, 2023. — 272 с. – ISBN 978-5-4461-2251-6
2. Будума, Н. Основы глубокого обучения. Создание алгоритмов для искусственного интеллекта следующего поколения : учебное пособие / Н. Будума, Н. Локашо. – Москва : Манн, Иванов и Фербер, 2020. — 304 с. – ISBN 978-5-00146-472-3
3. Бурков, А. Машинное обучение без лишних слов : учебное пособие / А. Бурков. – СПб. : Питер, 2021. – 192с. ISBN 978-5-4461-1560-0
4. Вейдман, С. Глубокое обучение: легкая разработка проектов на Python : учебное пособие / С. Вейдман. – СПб. : Питер, 2021. — 272 с. – ISBN 978-5-4461-1675-1
5. Вьюгин, В.В. Математические основы машинного обучения ипрогнозирования : учебное пособие / В.В. Вьюгин – Москва : МЦНМО, 2022. – 484 с. – ISBN 978-5-4439-2014-6
6. Мюллер, А.П. Введение в машинное обучение с помощью Python. Руководство для специалистов по работе с данными : учебное пособие / А.П. Мюллер, С. Гвидо. – Москва : Вильямс, 2022. – 480с. – ISBN 978-5-9908910-8-1
7. Орельен, Ж. Прикладное машинное обучение с помощью Scikit-Learn, Keras и TensorFlow. Концепции, инструменты : учебное пособие / Ж. Орельен. – Москва : Диалектика, 2020г. – 1040с. – ISBN 978-5-907203-33-4
8. Постолит, А. В. Основы искусственного интеллекта в примерах на Python : учебное пособие / А.В. Постолит. – СПб. : БХВ-Петербург, 2021. – 448 с. – ISBN 978-5-9775-6765-7
9. Протодьяконов, А. В. Алгоритмы Data Science и их практическая реализация на Python : учебное пособие / А.В. Протодьяконов, П.А. Пылов, В.Е. Садовников – Москва : Инфра-Инженерия, 2022. – 392 с. – ISBN 978-5-9729-1006-9
10. Рашид, Т. Создаём нейронную сеть : учебное пособие / Т. Рашид – СПб. : “Альфа-книга”, 2017. – 272 с. – ISBN 978-5-9909445-7-2
11. Рашка, С. Python и машинное обучение : учебное пособие / С. Рашка, В. Мирджалили. – Москва : ДМК-Пресс, 2020. – 420 с. – ISBN 978-5-97060-409-0
12. Саттон, Р.С. Обучение с подкреплением : учебное пособие / Р.С. Саттон, Э.Г. Барто. – Москва : ДМК-Пресс, 2020. – 552с. – ISBN 978-5-00101-456-0
13. Траск, Э. Грокаем глубокое обучение : учебное пособие / Э. Траск — СПб. : Питер, 2019. — 352 с. – ISBN 978-5-4461-1334-7
14. Флах, П. Машинное обучение. Наука и искусство построения алгоритмов, которые извлекают знания из данных : учебное пособие / П. Флах. – Москва : ДМК-Пресс, 2021. – 402с. – ISBN 978-5-97060-273-7
15. Michie, D. Machine Learning, Neural and Statistical Classification / D. Michie, D. J. Spiegelhalter. – New York : Ellis Horwood, 2020. – 289с. – ISBN 9780131063600

