

**Оглавление**

[ВВЕДЕНИЕ 3](#_Toc197858654)

[1. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ 6](#_Toc197858655)

[1.1. Выбор и развертывание среды разработки для языка программирования Python. 6](#_Toc197858656)

[1.2. Анализ теоретических аспектов создания нейросетей. 7](#_Toc197858657)

[1.3. Математическая модель нейронной сети. 14](#_Toc197858658)

[1.4. Выводы по разделу. 19](#_Toc197858659)

[2. РАЗРАБОТКА НЕЙРОННОЙ СЕТИ 21](#_Toc197858660)

[2.1. Описание и анализ данных. 21](#_Toc197858661)

[2.2. Подготовка данных. 28](#_Toc197858662)

[2.3. Выбор топологии сети. 33](#_Toc197858663)

[2.4. Создание и обучение моделей. 35](#_Toc197858664)

[2.5. Тестирование моделей. 38](#_Toc197858665)

[2.6. Улучшение моделей. 42](#_Toc197858666)

[2.7. Выводы по разделу. 49](#_Toc197858667)

[ЗАКЛЮЧЕНИЕ 50](#_Toc197858668)

[СПИСОК ЛИТЕРАТУРЫ 51](#_Toc197858669)

# **ВВЕДЕНИЕ**

Эксплуатационная практика проходила в Московском университете им. С.Ю. Витте (далее - Университете) на кафедре информационных систем.

От руководителя практики получено индивидуальное план-задание прохождения практики. В рамках структурного подразделения Университета с помощью руководителя практики и сотрудников кафедры было организовать рабочее место. На персональный компьютер была установлена среда разработки для языка программирования Python. Все задачи практики решались с помощью языка программирования Python.

В первый день практики были изучены правила внутреннего трудового распорядка Университета, в том числе с нормативная документация по мерам противодействия коррупционному поведению. Пройден инструктаж по технике безопасности и правилах поведения при возникновении чрезвычайных и нештатных ситуаций.

Перед выполнением индивидуального задания был изучен материал, необходимый для того, чтобы приступить к его выполнению.

В связи с широким распространением сервисов, связанных с отзывами пользователей на товары и услуги, возникает необходимость сбора обобщенной информации об эмоциональном отклике пользователей, путем их распределения по категориям. При ручной оценке задача усложняется по мере увеличения числа отзывов, и, соответственно, возрастает риск возникновения ошибок, которые, в данном случае, зависят от человеческого фактора.

В данном отчете описан ход разработки системы оценки эмоционального состояния пользователя, в зависимости от содержания комментария.

**Цель практики**

Получение знаний о возможности использования информационных технологий для решения прикладных задач, а также выработка практических навыков по их анализу, выбору и применению информационных технологий в Университете.

Так как в рамках данной работы прикладной задачей является разработка системы оценки эмоционального состояния пользователя, в зависимости от содержания комментария, цель работы - создание работоспособной модели нейронной сети, способной определять эмоциональную направленность текста и относить его к строго определенному диапазону эмоциональных состояний.

**Задачи практики**

* Закрепление приобретенных теоретических знаний;
* Приобретение опыта разработки нейронных сетей на языке программирования Python;
* Использование отечественных и международных стандартов при решении прикладных задач;
* Поиск исходных данных, соответствующие цели данной работы
* Анализ и модификация исходных данных, для дальнейшего их использования в обучении модели нейронной сети
* Разработка математической модели нейронной сети;
* Произвести сбор и нормализацию исходных данных, соответствующие цели данной работы, для последующего обучения нейронной сети;
* Произвести обучение модели нейронной сети, способную определять эмоциональное состояние текста, определить точность обучения;
* Произвести тестирование модели нейронной сети, оценить точность распределения текста по эмоциональным категориям;
* Улучшение модели нейронной сети;
* Подготовка отчета по результатам профессиональной деятельности.

**Источники информации и исходный код**

Источниками информации явились организационно-правовые документы, распорядительные и информационно-справочные документы, регламентирующие деятельность специалиста данной предметной области.

Набор данных, представляющий из себя набор комментариев каждый из которых отнесен к определенной категории эмоций, доступен в формате \*.csv по ссылке: <https://www.kaggle.com/api/v1/datasets/download/abdallahwagih/emotion-dataset?dataset_version_number=1>.

Файл кода в формате .ipynb доступен по ссылке: <https://github.com/soggetto-ventuno/ExPractice.git> (также файлы можно скачать по запасной ссылке: <https://drive.google.com/drive/folders/1BLNbLZEAvKmJqE-bOWSJrPm6PYns0ZQS?usp=sharing>).

В ссылке для скачивания доступны два файла requirements.txt:

* В папке "Requirements" - библиотеки и модули, установленные в процессе разработки;
* В папке "Requirements (все библиотеки среды)" - все библиотеки и модули, установленные в среде разработки.

При установке библиотек из файла requirements.txt, установка этих же библиотек с помощью "!pip install" не требуется.

# **1. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ**

## **1.1. Выбор и развертывание среды разработки для языка программирования Python.**

Для выполнения поставленной задачи, в рамках индивидуального задания, в качестве среды разработки был выбран бесплатный облачный сервис Google Collab.

Данный облачный сервис обладает следующими преимуществами:

1. Интерактивное выполнение кода - код можно писать в отдельных блоках и запускать в любом порядке. Графическое отображение данных также происходит в рамках того блока, в котором прописан вывод графических данных. Это позволяет видеть все стадии выполнения программы, а не только ее итог;
2. Предустановленные библиотеки – избавляет от необходимости устанавливать библиотеки в среду разработки, т.к. большинство библиотек уже установлены;
3. Использование графического процессора для вычислительных задач – позволяет ускорить выполнение кода;
4. Онлайн-сервис – нет необходимости устанавливать среду разработки и дополнительные программы или плагины на рабочее место.

Благодаря вышеуказанным преимуществам Google Collab снискал популярность в области программирования, анализа данных и машинного обучения.

В процессе разработки были установлены следующие библиотеки и модули:

!pip install ipykernel

!pip install kagglehub

!pip install tensorflow

!pip install spacy

!python -m spacy download en\_core\_web\_sm

!pip install wordcloud

!pip install keras\_tuner

!pip install ipywidgets

!pip install pandas

!pip install seaborn

!pip install scikit-learn

Данные библиотеки также можно установить из файла requirements.txt, который можно скачать по ссылке (помимо файла .ipynb для скачивания, доступны два файла requirements.txt, тот что в папке «Requirements» – библиотеки и модули, установленные в процессе разработки, в папке «Requirements (все библиотеки среды)» - библиотеки и модули, установленные в среде разработки), которая указана во введении. Для этого, перед запуском кода, необходимо раскомментировать (удалить #) следующую строку:

#!pip install -r requirements.txt

Файл requirements.txt должен находиться в той же директории, что и файл .ipynb.

## **1.2. Анализ теоретических аспектов создания нейросетей.**

Искусственная нейронная сеть (ИНС) — это математическая модель, которая имитирует работу биологических нейронных сетей, то есть сетей нервных клеток в живых организмах. Идея создания ИНС возникла в результате изучения процессов, происходящих в мозге, и стремления смоделировать их.

Первые попытки создать искусственную нейронную сеть были предприняты У. Маккалоком и У. Питтсом. После разработки алгоритмов обучения полученные модели стали применяться в практических целях, таких как прогнозирование, распознавание образов и управление.

ИНС представляет собой систему взаимосвязанных и взаимодействующих между собой простых процессоров, которые часто называют искусственными нейронами. Эти процессоры обычно довольно просты по сравнению с теми, что используются в персональных компьютерах. Каждый процессор в сети получает и отправляет сигналы, а также обрабатывает информацию, которую получает от других процессоров.

Однако, будучи соединёнными в сеть с управляемым взаимодействием, такие простые процессоры способны выполнять сложные задачи. С точки зрения машинного обучения, ИНС представляет собой частный случай методов распознавания образов (рис. 1).

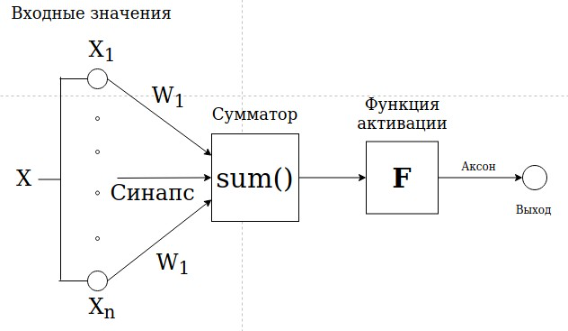


Рисунок 1

Вот некоторые распространённые применения нейронных сетей:  
 Классификация — это процесс распределения данных по категориям на основе определённых параметров. Например, если у вас есть список людей, вы можете использовать нейронную сеть для определения, кому из них следует предоставить кредит, а кому нет. Для этого сеть анализирует такие данные, как возраст, платёжеспособность, кредитная история и т. д.  
 Предсказание — это способность предвидеть будущие события. Например, нейронная сеть может прогнозировать рост или падение цен на акции, основываясь на данных фондового рынка.  
 Распознавание — это наиболее распространённое применение нейронных сетей в настоящее время. Они используются в Google, когда вы ищете фотографии, а также в камерах смартфонов для определения местоположения лица и выделения его на изображении.

Теперь давайте разберёмся, как работают нейронные сети, изучив их компоненты и параметры.

Нейрон — это вычислительный элемент, который получает информацию, выполняет над ней простые операции и передаёт её дальше. Нейроны (рис. 2) делятся на три основных типа: входной (синий), скрытый (красный) и выходной (зелёный).

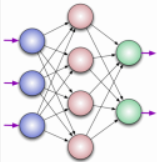


Рисунок 2

Если нейронная сеть состоит из большого количества нейронов, то для их обозначения используется термин «слой». В такой сети есть входной слой, который получает информацию, один или несколько скрытых слоёв (обычно их не больше трёх), которые обрабатывают информацию, и выходной слой, который выдаёт результат.

У каждого нейрона есть два основных параметра: входные данные (input data) и выходные данные (output data). В случае входного нейрона input = output. В остальных нейронах input — это сумма всех входных данных с предыдущего слоя, после чего они нормализуются с помощью функции активации (обозначим её как f(x)) и попадают в поле output.

Важно помнить, что нейроны работают с числами в диапазоне от 0 до 1 или от -1 до 1. Как обрабатывать числа, выходящие за эти пределы? На этом этапе самый простой способ — разделить 1 на это число. Этот процесс называется нормализацией и часто используется в нейронных сетях.

Существует несколько типов нейронов:

Входные нейроны получают данные на вход в виде суммы значений от других нейронов.

Скрытые нейроны обычно выполняют определённые преобразования информации и могут получать данные от нейронов, не являющихся входными.

Выходные нейроны получают значения в виде вероятности того или иного действия.

Синапс — это связь между двумя нейронами. У синапсов есть один параметр — вес. Благодаря ему входная информация изменяется при передаче от одного нейрона к другому (рис. 3).

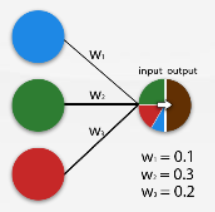


Рисунок 3

Допустим, есть три нейрона, передающих информацию следующему. Тогда у нас есть три веса, соответствующие каждому из этих нейронов. Нейрон с наибольшим весом будет доминировать в следующем нейроне (например, при смешении цветов).

Совокупность весов нейронной сети, или матрица весов, — это своего рода мозг всей системы. Именно благодаря этим весам входная информация обрабатывается и превращается в результат.

Важно помнить, что при инициализации нейронной сети веса устанавливаются случайным образом.

Сумматор — это функция, которая суммирует все значения, полученные от умножения веса на значение нейрона.

Аксон — это выходное значение, которое записывается в выходной нейрон.

Функция активации — это способ нормализации входных данных. То есть если на входе у вас большое число, то после применения функции активации вы получите результат в нужном диапазоне.

Функция активации представляет собой своего рода барьер, который:

— отсеивает несущественные данные, оставляя только значимые в определённом виде;  
— добавляет необходимую гибкость, позволяя сети адаптироваться и выявлять сложные взаимосвязи.

В действительности, можно использовать любую функцию, но некоторые из них работают эффективнее. Главное отличие между ними — это диапазон значений. Среди наиболее популярных можно выделить следующие:

Линейная функция (рис. 4):

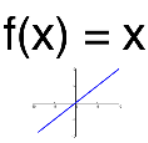


Рисунок 4

Линейная функция активации имеет вид прямой линии и зависит от входа (то есть от взвешенной суммы на этом нейроне).

Сигмоидная функция (рис. 5):

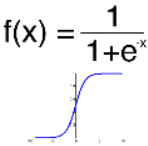


Рисунок 5

Функция сигмоиды отлично подходит для задач, где требуется разделить данные на две категории (да/нет). Она способна обрабатывать любые действительные числа и преобразовывать их в значения в диапазоне от 0 до 1. Именно поэтому она часто применяется в выходных слоях нейронных сетей (подробнее об этом читайте ниже).

Эта функция применяется крайне редко, за исключением ситуаций, когда необходимо проверить работу нейронной сети или передать значение без изменений.

Гиперболический тангенс (рис. 6):

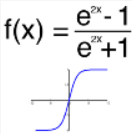


Рисунок 6

Использование гиперболического тангенса целесообразно, если ваши данные могут быть как положительными, так и отрицательными, поскольку диапазон функции составляет [-1,1]. Применение этой функции только с положительными значениями не имеет смысла, так как это может привести к искажению результатов работы нейросети.

ReLU (Rectified Linear Unit) (рис. 7):

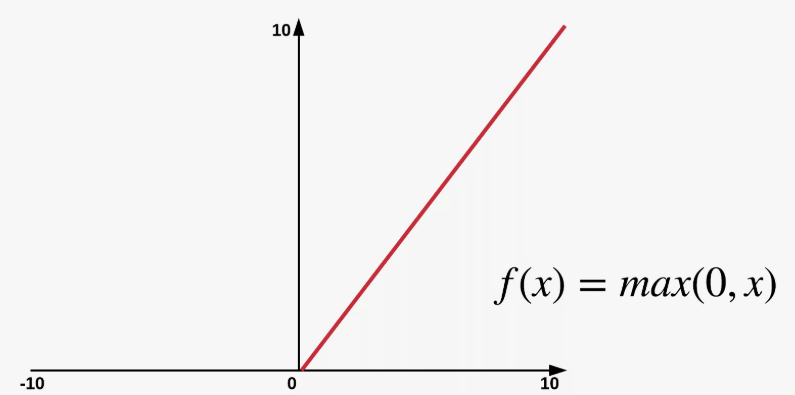


Рисунок 7

ReLU позволяет пропускать положительные значения без изменений, а все отрицательные значения обнулять.

Преимущества ReLU :

— Простота и результативность: ReLU выполняет элементарную математическую операцию — находит максимум из нуля и входного значения. Благодаря этой простоте вычисления происходят быстро;

— Решение проблемы исчезающего градиента: в глубоких нейронных сетях градиенты могут стать очень маленькими, что затрудняет обучение сети. ReLU помогает решить эту проблему, поскольку его производная всегда равна единице.

Softmax:

Функция softmax получает на вход вектор, содержащий K действительных чисел, и преобразует его в распределение вероятностей, состоящее из K вероятностей, которые пропорциональны экспонентам входных чисел.

Вкратце – функция получает на вход набор произвольных чисел и формирует из них список вероятностей, которые соответствуют этим числам.

Предположим, у нас есть список [1, 2, 3]:

softmax([1, 2, 3]) = [0.09003, 0.24473, 0.66524].

Соответственно, вероятность того, что:

входные данные принадлежат к первой группе (соответствующей 1), составляет примерно 0,09003;

входные данные относятся ко второй группе (соответствующей 2), равна примерно 0,24473.

Вероятность того, что входные данные относятся к третьей группе (соответствующей 3), составляет примерно 0,66524.

Таким образом, сумма возвращаемых вероятностей равна 1 (или 100%). Чем больше число во входном списке по сравнению с другими, тем выше вероятность, в которую оно будет преобразовано на выходе.

Важно отметить, что функция принимает вектор (список) входных данных, а не одно число. Это отличие от предыдущих функций, и причина в том, что она работает с несколькими нейронами (обычно со всем слоем). Таким образом, она сворачивает результаты слоя в выходное распределение вероятностей. Поэтому она часто используется в выходных слоях для получения финального результата.

Это делает функцию softmax особенно ценной для задач классификации в нейронных сетях, когда необходимо определить вероятность принадлежности данных к одной из нескольких категорий. Например, при попытке определить, является ли изображение изображением собаки (1), кошки (2) или рыбы (3).

## **1.3. Математическая модель нейронной сети.**

Нейрон (Xi) — это элементарная вычислительная единица, которая является упрощённой моделью естественного нейрона. Он получает значение от предыдущего нейрона, выполняет определённые действия и передаёт информацию дальше.

Функция, описывающая нейрон, выглядит следующим образом:

где:

w0 - смещение

wi−1 - вес от предыдущиx нейронов

Xi - значение текущего нейрона

Xi−1 - значение предыдущего нейрона

Значение нейрона обычно лежит в диапазоне от −∞ до +∞, но в реальности невозможно точно указать его значение, так как оно зависит от функции активации.

Веса искусственной нейронной сети обозначаются как Wi.

Нейронная сеть может быть как однослойной (также ее называют перцептрон), так и со скрытыми слоями с разным количеством нейронов в каждом из них. Ниже представлены несколько возможных вариаций таких сетей, а также их математическое представление.

1. Однослойная нейронная сеть (перцептрон).

Перцептрон — это алгоритм машинного обучения, имитирующий работу нейронов в мозге. Его также называют однослойной нейронной сетью, т.к. результат определяется на основе только одной функции активации, которая представляет собой нейрон (рис. 8).

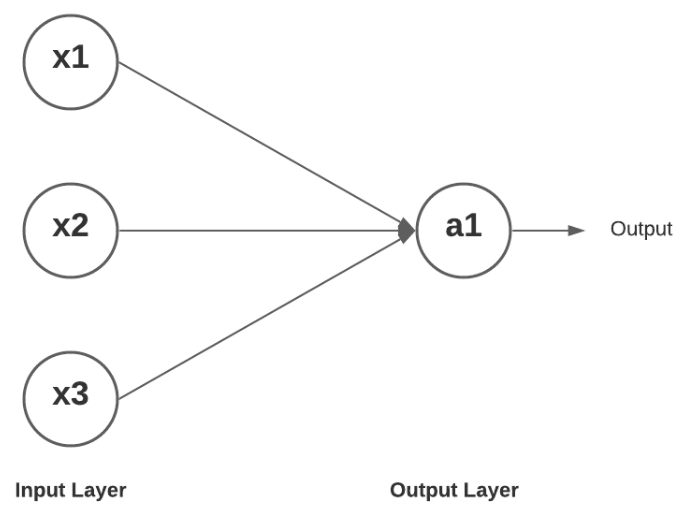


Рисунок 8

Математическое уравнение для получения значения выходного узла a1 в качестве функции входных x1, x2, x3 выглядит следующим образом:

В приведенном выше уравнении верхний индекс веса представляет слой, а нижний индекс весов представляет вес соединения между входным и выходным узлами. Таким образом,  представляет вес первого уровня между узлом 1 в следующем слое и узлом 2 в текущем слое.

1. Нейронная сеть с одним скрытым слоем.

Ниже представлена нейронная сеть с входным слоем, имеющим три нейрона, одним скрытым слоем с тремя входными нейронами и выходным слоем с одним нейроном (рис. 9).

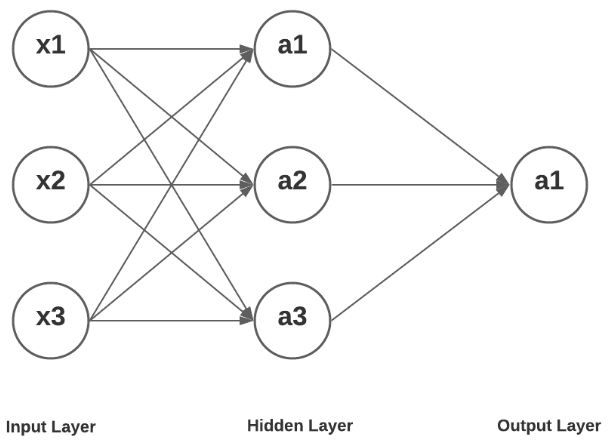


Рисунок 9

Таким образом, значения a1, a2 и a3 в слое 2 представлены в качестве функции входных x1, x2, x3, а значение a1 в слое 3 представлено как функция значений a1, a2 и a3 в слое 2.  
Сначала давайте представим выходные значения, обработанные в трех скрытых нейронах скрытого слоя. Входной слой представлен как слой 1, скрытый слой как слой 2 и выходной слой — это слой 3:

Определим выходное значение узла в выходном слое. Значение представляется как функция от a1, a2 и a3 в предыдущих узлах, которые могут быть представлены как значения x1, x2 и x3 во входном слое:

1. Нейронная сеть с одним скрытым слоем (3 нейрона) и выходным слоем (2 нейрона).

Ниже представлена нейронная сеть с входным слоем, имеющим два нейрона, одним скрытым слоем с тремя входными нейронами и выходным слоем, имеющим 2 нейрона (рис. 10).

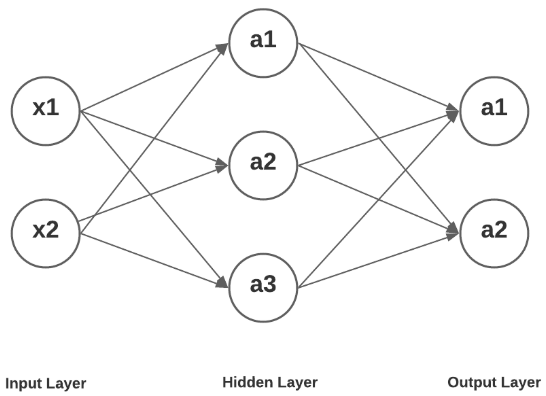


Рисунок 10

Таким образом, значения a1, a2, a3 в слое 2 представлены в качестве функции входных значений x1, x2, а значение a1, a2 в слое 3 представлено как функция значений a1, a2, a3 в слое 2.  
Сначала давайте представим выходные значения, обработанные в трех скрытых нейронах скрытого слоя. Входной слой представлен как слой 1, скрытый слой как слой 2 и выходной слой — это слой 3:

Определим выходное значение узлов в выходном слое. Значение представляется как функция от a1, a2, a3 в предыдущих узлах, которые могут быть представлены как значения x1, x2, x3 во входном слое:

1. Сети глубокого обучения с двумя скрытыми слоями.

Нейронная сеть глубокого обучения, в данном примере, имеет входной слой с двумя нейронами, два скрытых слоя с тремя и двумя входными нейронами соответственно, а также выходным слоем, имеющим один нейрон (рис. 11).

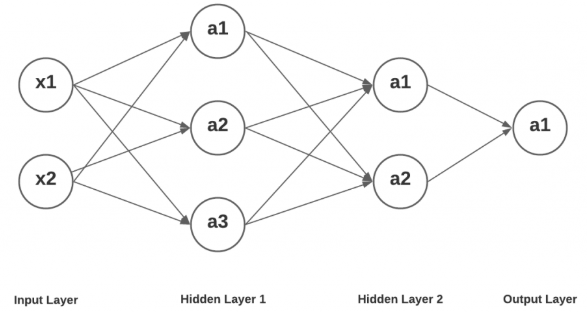


Рисунок 11

Таким образом, значения a1, a2, a3 в слое 2 представлены в качестве функции входных значений x1, x2, а значение a1, a2 в слое 3 представлено как функция значений a1, a2, a3 в слое 2.  
Сначала давайте представим выходные значения, обработанные в трех скрытых нейронах скрытого слоя. Входной слой представлен как слой 1, первый скрытый слой как слой 2 и второй скрытый слой — это слой 3:

Определим значение узлов во втором слое. Значение представляется как функция от a1, a2, a3 в предыдущих узлах, которые могут быть представлены как значения x1, x2, x3 во входном слое:

Таким образом, значение в выходном слое представим в качестве функции значений a1 и a2 в слое 3:

## **1.4. Выводы по разделу.**

Таким образом, было проведено обследование предметной области. В результате проведенного обследования можно сделать следующие выводы (табл.1):

Таблица 1

|  |  |  |
| --- | --- | --- |
| Выводы | Сформированные компетенции | |
| Код компетенции | Содержание компетенции |
| Проведен выбор и развертывание среды разработки для языка программирования Python. | **ПК-1** | Обладать способностью разработки прикладного программного обеспечения, автоматизации работы с базами данных и документами, программирования бизнес-логики приложений, интеграции разнородных данных |
| В соответствие с международными стандартами проектирования и обеспечения качества, проанализированы требования и составлены модели решения задачи. | **ПК-7** | Способность использовать отечественные и международные стандарты при проектировании и обеспечении качества прикладного программного обеспечения. |
| Проведен анализ теоретических аспектов создания нейросетей  Разработана математическая модель нейронной сети | **ПК-8** | Знать методы и инструментальные средства интеллектуального анализа больших данных |

# **2. РАЗРАБОТКА НЕЙРОННОЙ СЕТИ**

## **2.1. Описание и анализ данных.**

Для анализа предметной области был выбран набор данных, содержащий в себе комментарии строго определенного эмоционального диапазона.

Набор данных содержит в себе следующие колонки (табл. 2):

Таблица 2

|  |  |  |
| --- | --- | --- |
| Имя столбца | Тип данных | Описание |
| ID | Integer, Categorical, Nominal | Уникальный идентификатор записи |
| Сomment | String, Categorical, Nominal | Комментарий с определенным эмоциональным содержанием |
| Emotion | String, Categorical, Nominal | Наименование эмоции:  (fear, anger, joy) |

Набор данных был загружен в программу в форме DataFrame:

import kagglehub

# Download latest version

path = kagglehub.dataset\_download("abdallahwagih/emotion-dataset")

Далее, для вывода датасета на экран, обращаемся к загруженному набору данных, предварительно импортировав библиотеки «Pandas» и «os»:

import pandas as pd

import os

data = pd.read\_csv(os.path.join(path, 'Emotion\_classify\_Data.csv'))

data

Таким образом, в выведенном датасете, получено количество строк – 5937 (рис. 12):



Рисунок 12

Далее, была проведена проверка на missing values – наличие пропущенных (пустых) значений в колонках:

data.info()

По результатам проверки пропущенные значения отсутствуют, т.е. в каждой из 5937 строк имеются заполненные данные по всем трем полям (рис. 13):

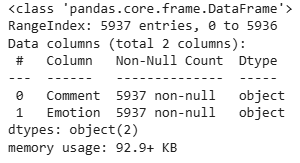


Рисунок 13

Также, была проведена проверка данные на повторяющиеся ряды:

duplicate\_rows = data.duplicated()

print("Number of duplicate rows:", duplicate\_rows.sum())

По результатам проверки дубликатов не найдено (рис. 14):



Рисунок 14

В датасете присутствуют три категории эмоций. Для того, чтобы посчитать количество строк, относящихся к каждой из эмоций, напишем следующее:

data['Emotion'].value\_counts()

Разница между наибольшим и наименьшим классом составляет всего 63 примера (менее 2%), что считается незначительной (рис. 15):

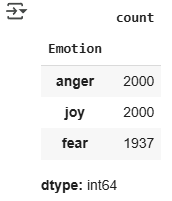


Рисунок 15

Такой баланс позволяет обучать модели машинного обучения без необходимости в дополнительных методах балансировки.

Для визуализации распределения данных в датасете, обратимся к инструментарию библиотеки mathplotlib, предварительно импортировав ее:

import matplotlib.pyplot as plt

Визуализация данных реализована следующим образом:

plt.figure(figsize=(10, 6))

sns.countplot(data=data, x='Emotion', hue ='Emotion', order=data['Emotion'].value\_counts().index,palette='viridis',legend=False)

plt.title('Распределение эмоций в датасете')

plt.xlabel('Эмоция')

plt.ylabel('Количество')

plt.xticks(rotation=45)

plt.tight\_layout()

plt.show()

Таким образом, визуализация наглядно представляет равномерное распределение количества эмоций в данных (рис. 16).

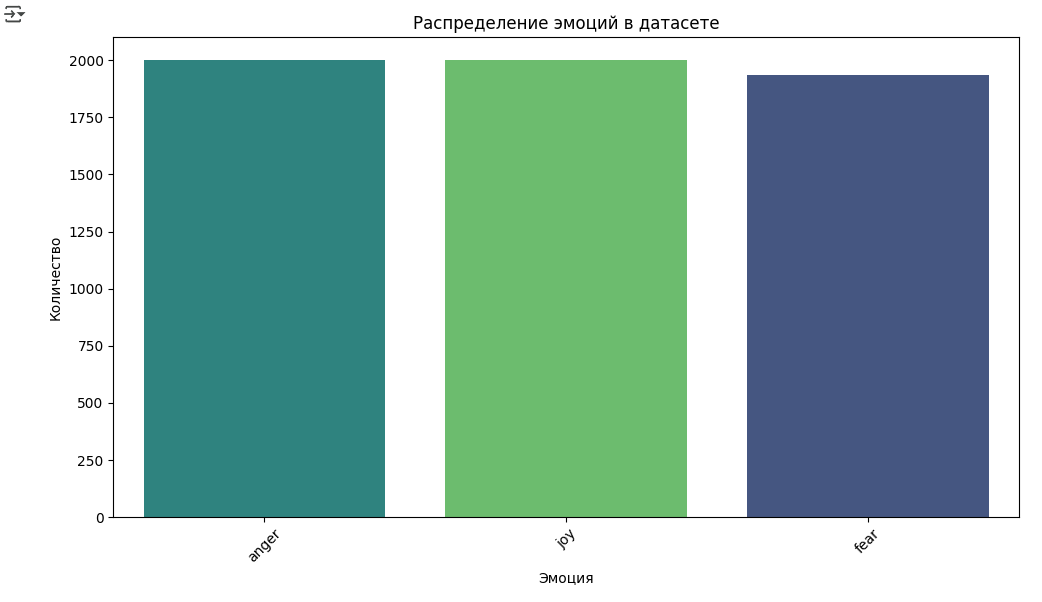


Рисунок 16

Также, для наглядного представления часто встречающихся слов в комментариях с соответствующей эмоцией, были применены методы из библиотеки wordcloud:

from wordcloud import WordCloud

Создание облака слов для эмоции «joy»:

text\_joy = " ".join(data[data['Emotion'] == 'joy']['Comment'])

wordcloud = WordCloud(width=800, height=400, background\_color='white').generate(text\_joy)

plt.figure(figsize=(10, 5))

plt.imshow(wordcloud, interpolation='bilinear')

plt.axis('off')

plt.title('Облако слов для эмоции: joy')

plt.show()

Облако слов для эмоции joy (радость) показывает (рис. 17), что наиболее часто встречающиеся слова связаны с позитивными переживаниями, такими как: "feel", "feeling", "good", "happy", "better", "positive", "friend", "life", "excited", "hope". Часто используются глаголы и выражения желаний: "want", "make", "know", "will", "need". Это говорит о том, что тексты с эмоцией "joy" в основном описывают положительные состояния, желания, уверенность, вдохновение и социальную вовлечённость (например, "friend", "people"):

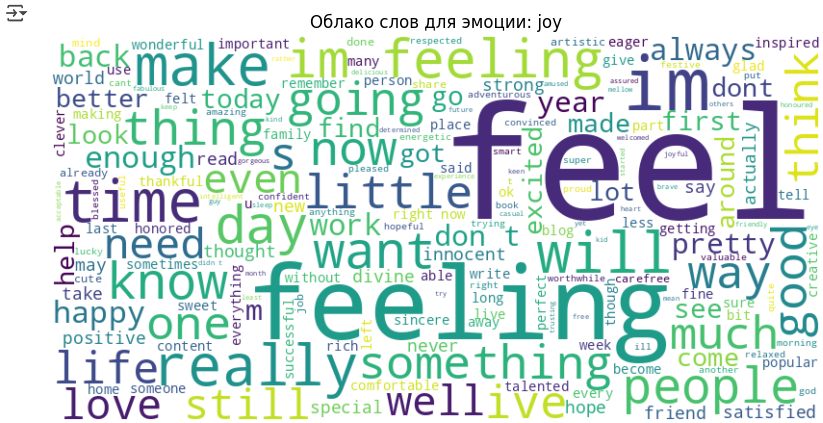


Рисунок 17

Создание облака слов для эмоции «anger»:

text\_joy = " ".join(data[data['Emotion'] == 'anger']['Comment'])

wordcloud = WordCloud(width=800, height=400, background\_color='white').generate(text\_joy)

plt.figure(figsize=(10, 5))

plt.imshow(wordcloud, interpolation='bilinear')

plt.axis('off')

plt.title('Облако слов для эмоции: anger')

plt.show()

Облако слов для эмоции anger подчёркивает слова, связанные с раздражением, агрессией и негативным восприятием (рис. 18). Часто встречаются слова, отражающие эмоциональное раздражением:"stressed", "irritable", "annoyed", "bothered", "mad", "angry", "agitated".Много слов с негативной окраской, указывающих на конфликт или обиду:"selfish", "rude", "cold", "greedy", "wronged", "hated", "frustrated", "offended", "hostile".Встречаются грубые и враждебные выражения, что подчёркивает интенсивность и искренность чувств.

Это говорит о том, что эмоция гнева выражается через слова, связанные с внутренним дискомфортом, обидой и агрессией по отношению к другим или к себе.

Создание облака слов для эмоции «fear»:

text\_joy = " ".join(data[data['Emotion'] == 'fear']['Comment'])

wordcloud = WordCloud(width=800, height=400, background\_color='white').generate(text\_joy)

plt.figure(figsize=(10, 5))

plt.imshow(wordcloud, interpolation='bilinear')

plt.axis('off')

plt.title('Облако слов для эмоции: fear')

plt.show()

Облако слов для эмоции fear подчёркивает слова, связанные со страхом, эмоциональным потрясением и переживанием (рис. 18). Часто встречаются слова, отражающие эмоциональную скованность: "unsure", "nervous", "strange", "uncertain", "confused", "shaky", "anxious".Много слов указывающих на сильное эмоциональное потрясение: "scared", "terrified", "afraid", "distressed", "paranoid", "overwhelmed". Это говорит о том, что эмоция страха проявляется через слова, связанные с эмоционально нестабильным состоянием человека, и, зачастую имеют негативный окрас:

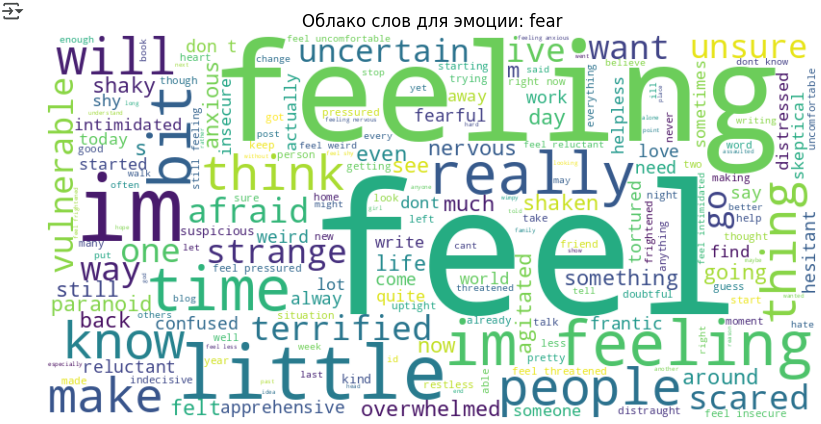


Рисунок 18

Для анализа длины комментариев для каждой из эмоций, был использован инструмент для выявления ключевых характеристик распределения из библиотеки seaborn.

import seaborn as sns

Визуализация длины комментариев в разрезе каждой из эмоций реализована следующим образом:

data['Text\_Length'] = data['Comment'].apply(len)

plt.figure(figsize=(15, 10))

sns.boxplot(data=data, x='Emotion', y='Text\_Length', hue = 'Emotion', order=data['Emotion'].value\_counts().index, palette='viridis')

plt.title('Распределение длины текста по эмоциям')

plt.xlabel('Эмоция')

plt.ylabel('Длина комментария (в символах)')

plt.xticks(rotation=45)

plt.show()

По результатам полученной диаграммы (рис. 19) можно сделать вывод, что комментарии, связанные с эмоцией «anger» имеют наибольший диапазон по длине комментариев, т.к. это связано с интенсивной передачей негативны автора комментария. Комментарии эмоции «fear» - наименьший, что указывает на немногословность автора комментария, испытывающий чувство страха. Соответственно, комментарии эмоции «joy» имеют усредненные показатели длинны текстов.

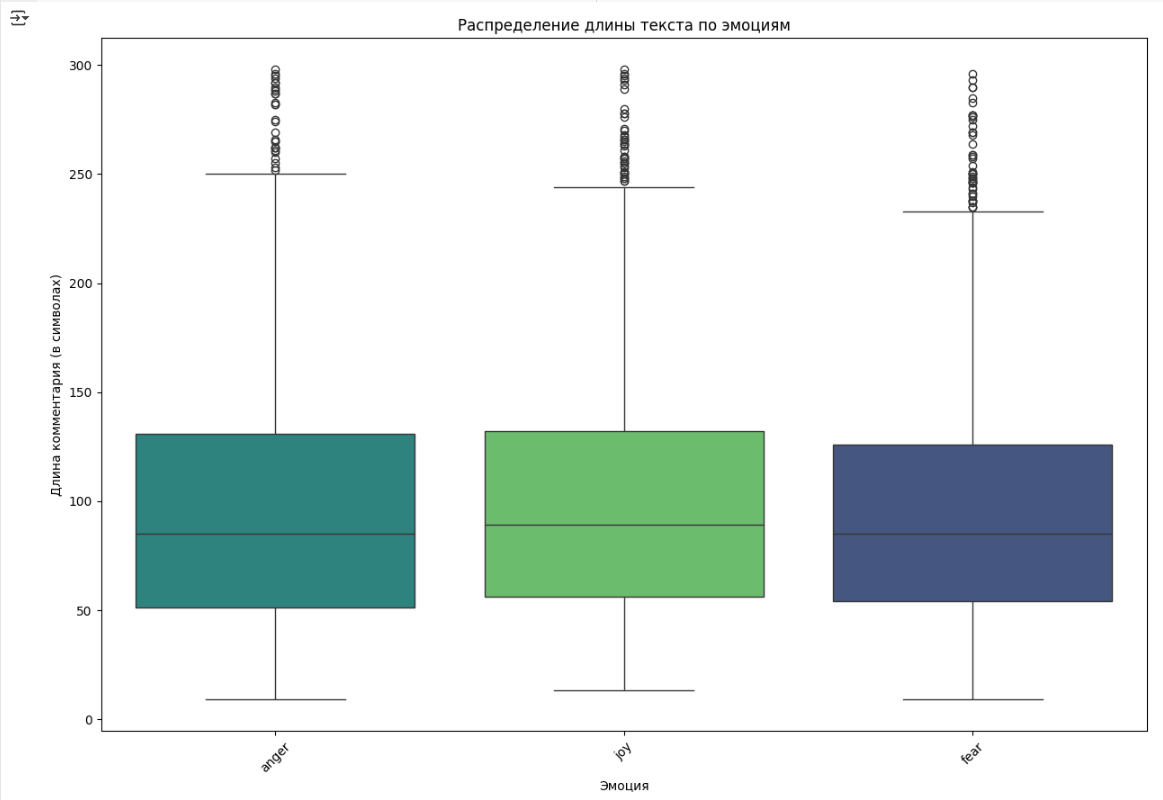


Рисунок 19

Для получения обобщенной информации о состоянии набора данных был проведен первичный анализ:

data['Text\_Length'].describe()

Медианная длина комментариев по всем эмоциям составляет около 86 символов (рис. 20). Распределение длины текста по эмоциям anger, joy и fear примерно одинаковое — все имеют похожие медианы и диапазоны.Большинство комментариев находятся в пределах 54–130 символов (межквартильный размах):

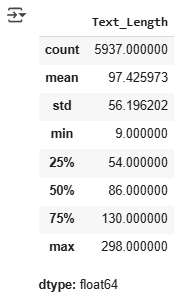


Рисунок 20

Максимальная длина комментария — 298 символов, минимальная — 9, что говорит о наличии коротких и длинных сообщений, а также выбросов (видны на boxplot в виде точек выше усов).

Стандартное отклонение — 56, что свидетельствует о умеренной вариативности длины сообщений в датасете.

В целом, длина текста не сильно зависит от категории эмоции, поэтому эта переменная может быть вспомогательной, но не ключевой для классификации эмоций.

## **2.2. Подготовка данных.**

Далее была произведена подготовка данных для дальнейшего их использования при обучении, тестировании и улучшении модели нейросети.

Для подготовки текстовых данных к классификации с использованием нейросетей была выбрана библиотека spaCy:

import spacy

Она обеспечивает высокую скорость обработки и содержит мощный набор инструментов для лингвистического анализа, что делает её особенно подходящей для задач машинного обучения.

Среди ключевых характеристик spaCy — встроенные предобученные модели, поддержка лемматизации, POS-теггинга (определение частей речи), распознавания именованных сущностей (NER), а также удобная система пайплайнов для последовательной обработки текста. Библиотека оптимизирована для производительности и хорошо интегрируется с современными ML-фреймворками.

SpaCy работает на предобученных языковых моделях. Предобученная языковая модель — это модель обработки естественного языка, заранее обученная на больших корпусах текстов. Она содержит встроенные знания о грамматике, синтаксисе и лексике языка и может быть использована "из коробки" без дополнительного обучения. Такие модели значительно ускоряют процесс анализа текста и обеспечивают высокое качество лингвистической обработки, включая токенизацию, лемматизацию, определение частей речи и распознавание именованных сущностей.

Модель знает грамматику, словоформы, части речи, контекст, и может:

* Определить, где слово заканчивается (токенизация);
* Найти базовую форму слова (лемматизация);
* Понять, к какой части речи оно относится (POS-теггинг);
* Узнать, где в тексте имя, место, организация (NER).

Для английского текста лучше всего подойдёт en\_core\_web\_sm — лёгкая и быстрая модель.

!python -m spacy download en\_core\_web\_sm

# Загрузка модели

nlp = spacy.load("en\_core\_web\_sm")

**1. Лемматизация, удаление стоп-слов и пунктуации**

Каждый комментарий преобразован по следующему принципу:

* Преобразование текст в леммы (базовые формы слов);
* Удаление стоп-слов (например: "the", "is", "and");
* Удаление пунктуации и пробелов;
* Сохранение "очищенного" текста в новом столбце.

Преобразование комментариев реализована следующим образом:

from tqdm import tqdm  # для прогресс-бара, опционально

tqdm.pandas()

def preprocess\_spacy(text):

    doc = nlp(text.lower())  # приведение к нижнему регистру

    tokens = [

        token.lemma\_ for token in doc

        if not token.is\_stop and not token.is\_punct and not token.is\_space

    ]

    return " ".join(tokens)

# Применим ко всем комментариям

data['Cleaned\_Comment'] = data['Comment'].progress\_apply(preprocess\_spacy)

*doc* — это результат обработки текста через nlp(text) в spaCy. Он содержит список token-ов, где каждый токен — это слово, знак препинания или пробел.

*token.lemma* - это лемма токена, то есть базовая форма слова. Например:

* "running" → "run"
* "cars" → "car"
* "better" → "good"

*if not token.is\_stop* - убирает стоп-слова, то есть частые слова, которые не несут смысловой нагрузки: "and", "the", "is", "it", "was", "are" и т.п.

*and not token.is\_punct* -убирает знаки препинания: точки, запятые, восклицательные знаки и т.д.

*and not token.is\_space* - убирает пробелы, табы и прочие "пустые" токены.

*.progress\_apply* – отображение прогресс бара, из библиотеки tgqm.

**2. Токенизация и преобразование текста в числовой формат**

Нейросеть не может "читать" слова — ей нужны числа. Для этого необходимо:

* Преобразовать текст в последовательности токенов (слов);
* Каждому слову присвоить уникальный индекс;
* Преобразовать весь текст в последовательность чисел;
* Привести все тексты к одинаковой длине (padding)

Нейросети, особенно рекуррентные (RNN, LSTM, GRU) и свёрточные (CNN), не умеют работать с "рваной" формой данных. Они ожидают, что все входные данные будут одной и той же формы (одинаковой размерности),padding — "подгонка" под общую длину с помощью нулей. Массивы разной длины невозможно объединить в один батч — модель "сломается" при обучении. Когда все последовательности одной длины, они удобно обрабатываются векторизованно и эффективно подаются в GPU.

Для токенизации и преобразование текста в числовой формат используются Tokenizer и pad\_sequences из библиотеки Keras:

from tensorflow.keras.preprocessing.text import Tokenizer

from tensorflow.keras.preprocessing.sequence import pad\_sequences

Инициализация токенизатора.

Tokenizer() - специальный инструмент из Keras:

* Сканирует весь текст, находит все уникальные слова;
* Присваивает каждому слову уникальный номер (индекс);
* Потом может превращать слова в числа и наоборот.

tokenizer = Tokenizer()

tokenizer.fit\_on\_texts(data['Cleaned\_Comment'])

Преобразование текста в последовательности чисел:

sequences = tokenizer.texts\_to\_sequences(data['Cleaned\_Comment'])

Применение паддинга, для одинаковой длины всех комментариев.

* По умолчанию добавляются нули в начало или конец (в зависимости от параметра padding);
* При обучении модель научится игнорировать нули (если ты используешь masking или слои, которые это учитывают).

padded\_sequences = pad\_sequences(sequences, padding='post')

Таким образом все комментарии преобразованы в числовой формат, и сведены к одинаковой длине (рис. 21):

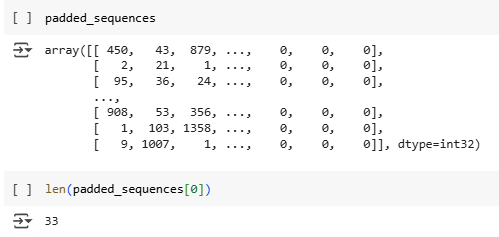


Рисунок 21

**3. Кодирование целевых меток (эмоций)**

Для кодирования целевых меток необходимо:

* Преобразовать категории в числа: "joy" → 0, "anger" → 1, "fear" → 2;
* Преобразовать эти числа в one-hot векторы, будем использовать;
* categorical\_crossentropy как функцию потерь. RAM - Использует больше (больше нулей),one- hot проще анализировать;
* Можно оставить метки как просто числа, тогда функция потерь - sparse\_categorical\_crossentropy.

from sklearn.preprocessing import LabelEncoder

from tensorflow.keras.utils import to\_categorical

Преобразование текстовых меток в числа:

label\_encoder = LabelEncoder()

integer\_labels = label\_encoder.fit\_transform(data['Emotion'])

Преобразование числовых меток в one-hot формат:

one\_hot\_labels = to\_categorical(integer\_labels)

data['Label'] = integer\_labels

Получение имен классов (anger, fear, joy):

class\_names = label\_encoder.classes\_

Преобразование one-hot в DataFrame:

one\_hot\_df = pd.DataFrame(one\_hot\_labels, columns=[f"Emotion\_{cls}" for cls in class\_names])

Объединение с data по индексу:

data = pd.concat([data, one\_hot\_df], axis=1)

В результате кодирования целевых меток, в датасете реализованы новые поля (рис. 22):

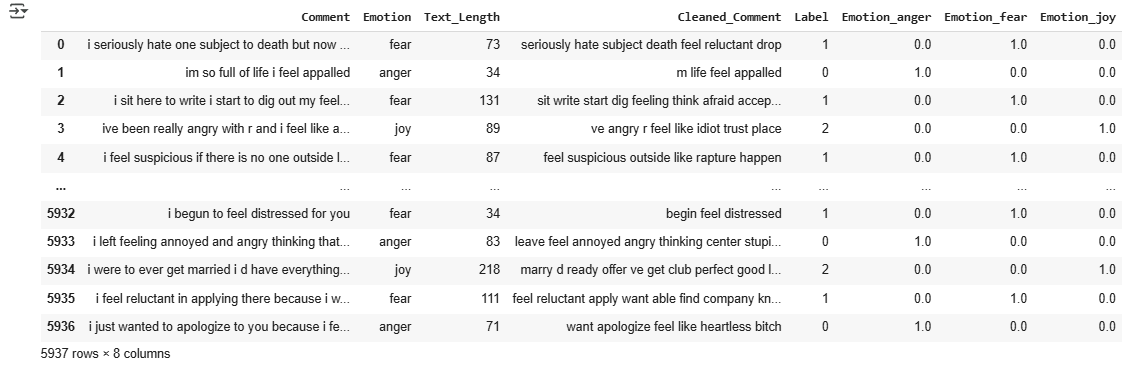


Рисунок 22

Таким образом, данные для обучения модели подготовлены (рис. 23, 24):

X = padded\_sequences

print("X shape:", X.shape)



Рисунок 23

y = one\_hot\_labels

print("y shape:", y.shape)

print("Количество классов:", y.shape[1])



Рисунок 24

## **2.3. Выбор топологии сети.**

Топология модели — это структура нейронной сети, определяющая то, как слои модели организованы и соединены друг с другом.

Топология включает в себя:

* Типы слоёв (например, Embedding, LSTM, Dense, Dropout, Conv2D, Flatten и т.д.);
* Количество слоёв;
* Порядок следования слоёв;
* Число нейронов в слоях;
* Функции активации;
* Параметры каждого слоя (например, return\_sequences=True, units=64 и т.д.);
* Типы связей между слоями (например, последовательные, параллельные, ветвящиеся).

Топология нейронной сети, для определения эмоциональной составляющей комментария, представляет из себя линейную последовательность слоев, реализуемую посредством класса Sequential из библиотеки Keras.

*Sequential модель* это наиболее простой способ построения нейросети в Keras, когда:

* Модель строится последовательно — слой за слоем;
* Каждый слой имеет только один вход и один выход.

Модель включает в себя следующие слои в соответствующем порядке:

* Embedding – входной слой, преобразующий дискретные или категориальные данные в непрерывные векторы.

Имеет следующие параметры:

input\_dim = 6963 (общее количество слов в наборе данных + 1);

output\_dim = 100 (размер векторного пространства);

input\_length = 33 (длина каждой фразы).

* LSTM – слой, определяющий долгосрочные зависимости между временными шагами данных о последовательности.

Имеет следующие параметры:

return\_sequences = False (слой будет выдавать только последний элемент последовательности, а не всю последовательность целиком);

64 – количество нейронов.

* Dropuot – слой, исключающий определенный процент (в данном случае 50%) случайных нейронов во время обучения нейронной сети, с целью предотвращения переобучения модели на тренировочных данных. Соответственно, данная модель включает в себя 32 нейрона.
* Dense – слой, получающий информацию со всех узлов предыдущего слоя.

Имеет следующие параметры:

32 – количество нейронов;

activation = «relu».

* Dense – выходной слой.

Имеет следующие параметры:

3 – количество нейронов (столько же, сколько и классов)

activation = «softmax».

## **2.4. Создание и обучение моделей.**

Перед началом обучения моделей необходимо разделить данные на тренировочные и тестовые, для этого была использована библиотека SkLearn:

from sklearn.model\_selection import train\_test\_split

Можно зафиксировать random seed для моделей с LSTM, чтобы результаты были максимально воспроизводимыми (устанавливает постоянство тренировочных и тестовых данных данных):

import random

import tensorflow as tf

import numpy as np

SEED = 42

os.environ['PYTHONHASHSEED'] = str(SEED)

random.seed(SEED)

np.random.seed(SEED)

tf.random.set\_seed(SEED)

Разделение происходит в пропорции 80 к 20 тренировочных данных к тестовым соответственно:

X\_train, X\_test, y\_train, y\_test = train\_test\_split(

    X,               # входные данные (padded\_sequences)

    y,               # целевые метки (one-hot)

    test\_size=0.2,   # 20% на тест

    random\_state=42, # фиксируем результат для воспроизводимости

    stratify=y       # чтобы классы равномерно распределялись

)

print("X\_train shape:", X\_train.shape)

print("X\_test shape:", X\_test.shape)

print("y\_train shape:", y\_train.shape)

print("y\_test shape:", y\_test.shape)

В результате, данные разделены на тренировочные (X\_train, y\_train) и тестовые (X\_test; y\_test) (рис. 25):

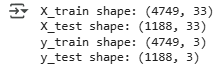


Рисунок 25

В соответствии с описанной в главе 2.2 топологией нейронной сети, реализация модели осуществлена следующим образом:

# Параметры (для моделей)

vocab\_size = len(tokenizer.word\_index) + 1

input\_length = X.shape[1]

embedding\_dim = 100

num\_classes = y.shape[1]

# Модель нейронной сети Sequential (последовательный тип связей между слоями)

model = Sequential()

model.add(Embedding(input\_dim=vocab\_size, output\_dim=embedding\_dim, input\_length=input\_length))

model.add(LSTM(64, return\_sequences=False))

model.add(Dropout(0.5))

model.add(Dense(32, activation='relu'

model.add(Dense(num\_classes, activation='softmax'))

# Укажем конфигурацию обучения (оптимизатор, функция потерь, метрики)

model.compile(loss='categorical\_crossentropy', optimizer='adam', metrics=['accuracy'])

Для конфигурации обучения был выбран стандартный набор классов:

* loss(функция минимизации потерь) = 'categorical\_crossentropy' (вычисляет кросс-энтропийные потери между истинными метками и прогнозируемыми метками);
* optimizer (Оптимизатор) = 'adam' (метод стохастического градиентного спуска, основанный на адаптивной оценке моментов первого и второго порядка);
* metrics = 'accuracy' (вычисляет, как часто предсказания совпадают с метками).

Для обучения модели были выбраны следующие параметры:

* Тренировочные данные: X\_train, y\_train;
* *epochs =10* - количество эпох для обучения модели. Эпоха — это итерация по всем предоставленным данным x и y;
* *batch\_size = 32* (по умолчанию) - количество выборок, которые будут использоваться на каждой итерации;
* *validation\_split=0.1* - 10% от обучающих данных будет использовано как валидация (10 % для тестирования внутри модели).

history = model.fit(

    X\_train, y\_train,

    epochs=10,

    batch\_size=32,

    validation\_split=0.1

)

Для оценки адекватности результатов обучения модели необходимо определить их точность во время обучения, а также построить функцию потерь:

# Точность

plt.figure(figsize=(10, 4))

plt.plot(history.history['accuracy'], label='Train Accuracy')

plt.plot(history.history['val\_accuracy'], label='Validation Accuracy')

plt.title('Accuracy во время обучения')

plt.xlabel('Epoch')

plt.ylabel('Accuracy')

plt.legend()

plt.grid(True)

plt.show()

# Функция потерь

plt.figure(figsize=(10, 4))

plt.plot(history.history['loss'], label='Train Loss')

plt.plot(history.history['val\_loss'], label='Validation Loss')

plt.title('Loss во время обучения')

plt.xlabel('Epoch')

plt.ylabel('Loss')

plt.legend()

plt.grid(True)

plt.show()

В результате получены два графика (рис. 26):

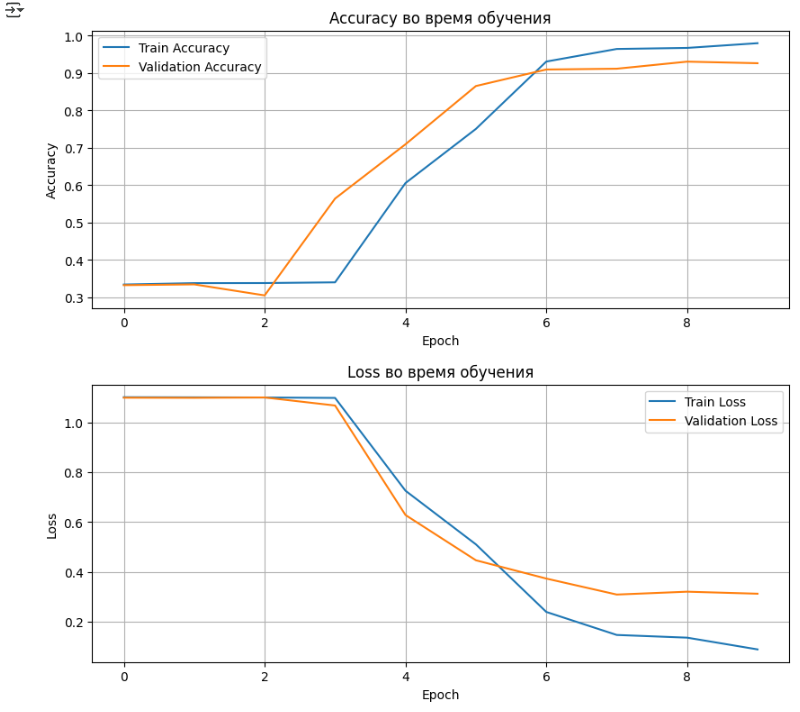


Рисунок 26

На первом графике показана точность на обучающей и валидационной выборках. Видно, что:

* Точность на тренировке быстро достигает почти 100%;
* Валидационная точность начинает стагнировать и колебаться после 4-й эпохи.

На втором графике представлена функция потерь (loss):

* Потери на обучающей выборке стабильно уменьшаются;
* Потери на валидации перестают снижаться и даже немного растут — признак начинающегося переобучения.

## **2.5. Тестирование моделей.**

После создания и обучения модели нейронной сети было проведено тестирование, в соответствии со следующими шагами:

1.Получение вероятностей:

y\_pred\_proba = model.predict(X\_test)

2.Преобразование вероятностей в метки классов:

y\_pred = np.argmax(y\_pred\_proba, axis=1)

3. Истинные метки (тоже в виде индексов):

y\_true = np.argmax(y\_test, axis=1)

axis = 1 – индекс максимальных значений по строке.

Для наглядного представления результатов тестирования, была построена матрица ошибок (confusion matrix) (рис. 27):

from sklearn.metrics import confusion\_matrix, ConfusionMatrixDisplay

cm = confusion\_matrix(y\_true, y\_pred)

disp = ConfusionMatrixDisplay(confusion\_matrix=cm, display\_labels=label\_encoder.classes\_)

plt.figure(figsize=(6, 6))

disp.plot(cmap='Blues', values\_format='d')

plt.title("Матрица ошибок (Confusion Matrix)")

plt.show()

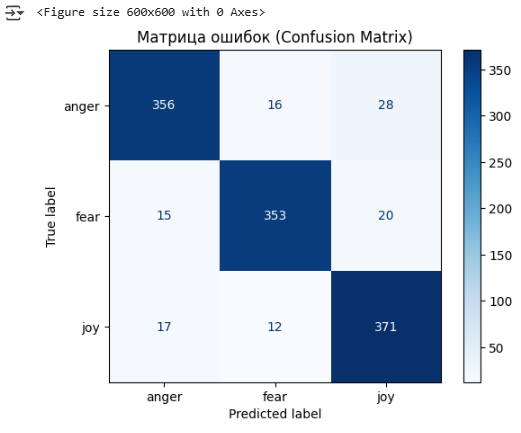


Рисунок 27

Модель обучалась классифицировать тексты на три категории эмоций: anger (злость), fear (страх) и joy (радость).

* Диагональные значения (356, 353, 371) — это количество правильно классифицированных примеров для каждой эмоции;
* Остальные значения — это ошибки классификации, когда модель спутала одну эмоцию с другой:

anger:

* 356 — правильно классифицировано как "anger";
* 15 — ошибочно классифицировано как "fear";
* 17 — ошибочно классифицировано как "joy".

fear:

* 353 — правильно классифицировано как "fear";
* 16 — перепутано с "anger";
* 12 — перепутано с "joy".

joy:

* 371 — правильно распознано;
* 28 — ошибочно определено как "anger";
* 20 — перепутано с "fear".

Также был построен классификационный отчет, позволяющий вывести показатели точности (рис. 28):

from sklearn.metrics import classification\_report

report = classification\_report(y\_true, y\_pred, target\_names=label\_encoder.classes\_)

print("Классификационный отчёт:\n")

print(report)

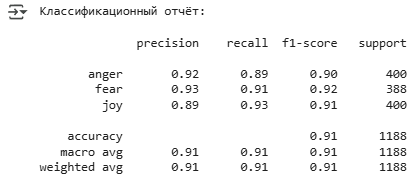


Рисунок 28

*Precision* показывает насколько точны предсказания (из всех "joy", сколько было правильно). Какая доля объектов, предсказанных как интересующий нас класс (например, "joy"), действительно являются таковыми (рис. 29).

*Recall* показывает, какую долю объектов исследуемого класса модель смогла правильно распознать (рис. 29).

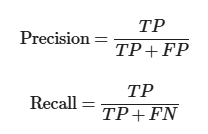


Рисунок 29

*Где:*

*TP (True Positives) —* модель правильно предсказала рассматриваемый класс;

*FP (False Positives) —* модель ошибочно предсказала рассматриваемый класс (но это был другой класс);

*FN (False Negatives) —* модель не смогла распознать рассматриваемый класс (ошибочно предсказала другой класс).

*F1-score —* гармоническое среднее между точностью и полнотой (рис. 30):

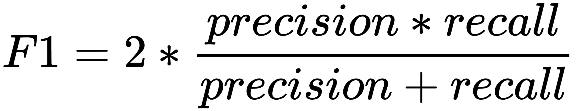


Рисунок 30

Пример на основе класса "joy":

Из матрицы ошибок:

TP = 366 (правильно предсказано как joy);

FP = 20 + 12 = 32 (joy было предсказано для других классов);

FN = 13 + 21 = 34 (настоящее joy предсказано как другие).

Значит:

Precision\_joy = 366 / (366 + 32) = 0,9196

Recall\_joy = 366 / (366 + 34) = 0,915

По результатам тестирования были сделаны следующие выводы:

* В целом модель работает очень хорошо, с точной классификацией в большинстве случаев;
* Наибольшая путаница наблюдается между anger и joy, а также между fear и другими эмоциями, что может быть связано с лингвистической схожестью или пересечением выражений эмоций;
* Это также подтверждает, что оптимизация гиперпараметров может помочь уменьшить количество таких ошибок и улучшить различение схожих по контексту эмоций.

## **2.6. Улучшение моделей.**

Чтобы сделать модель более устойчивой и улучшить её обобщающую способность, мы применим Keras Tuner с Hyperband, чтобы подобрать:

* оптимальное количество нейронов;
* уровень dropout;
* скорость обучения;
* другие важные параметры модели.

Keras Tuner имеет четыре доступных тюнера: RandomSearch, Hyperband, BayesianOptimization и Sklearn.

Hyperband эффективен, поскольку он динамически распределяет ресурсы среди наиболее эффективных моделей на ранних этапах — отлично, когда вам нужны хорошие результаты без безумного количества попыток.

Для начала была определена функция построения модели:

import keras\_tuner as kt

def build\_model(hp):

    model = Sequential()

    model.add(Embedding(

        input\_dim=vocab\_size,

        output\_dim=100,  # фиксированное значение

        input\_length=input\_length

    ))

    model.add(LSTM(

        units=hp.Int('lstm\_units', min\_value=32, max\_value=128, step=32),

        return\_sequences=False

    ))

    model.add(Dropout(

        rate=hp.Float('dropout\_rate', min\_value=0.2, max\_value=0.5, step=0.1)

    ))

    model.add(Dense(

        units=hp.Int('dense\_units', min\_value=16, max\_value=64, step=16),

        activation='relu'

    ))

    model.add(Dense(num\_classes, activation='softmax'))

    model.compile(

        optimizer=Adam(

            learning\_rate=hp.Choice('learning\_rate', values=[1e-2, 1e-3, 1e-4])

        ),

        loss='categorical\_crossentropy',

        metrics=['accuracy']

    )

    return model

Модель hyperband для улучшения гиперпараметров, со следующими параметрами:

* *objective = ‘val\_accuracy’* - параметр, который указывает на точность валидации в качестве цели для алгоритма Hyperband при оптимизации гиперпараметров;
* *max\_epochs = 20* – максимум эпох для одной конфигурации;
* *factor = 3* – определяет, как быстро отбрасываются плохие модели (значение 3 – по-умолчанию);
* *directory = ‘1my\_tuner\_dir’;*
* *project\_name = ‘lstm\_hyperband’.*

tuner = kt.Hyperband(

    build\_model,

    objective='val\_accuracy',

    max\_epochs=20,

    factor=3,

    directory='1my\_tuner\_dir',

    project\_name='lstm\_hyperband'

Далее, было произведено обучение модели с наилучшими гиперпараметрами:

# нилучшие  hyperparameters

best\_hps = tuner\_1.get\_best\_hyperparameters(num\_trials=1) [0]

# тренировка лучшей модели

best\_model\_1 = tuner\_1.hypermodel.build(best\_hps)

history\_1 = best\_model\_1.fit(

    X\_train, y\_train,

    epochs=10,

    validation\_split=0.1,

    batch\_size=32

)

Для оценки адекватности результатов обучения модели необходимо определить их точность во время обучения, а также построить функцию потерь:

# Точность

plt.figure(figsize=(10, 4))

plt.plot(history.history['accuracy'], label='Train Accuracy')

plt.plot(history.history['val\_accuracy'], label='Validation Accuracy')

plt.title('Accuracy во время обучения')

plt.xlabel('Epoch')

plt.ylabel('Accuracy')

plt.legend()

plt.grid(True)

plt.show()

# Функция потерь

plt.figure(figsize=(10, 4))

plt.plot(history.history['loss'], label='Train Loss')

plt.plot(history.history['val\_loss'], label='Validation Loss')

plt.title('Loss во время обучения')

plt.xlabel('Epoch')

plt.ylabel('Loss')

plt.legend()

plt.grid(True)

plt.show()

В результате получены два графика (рис. 31):

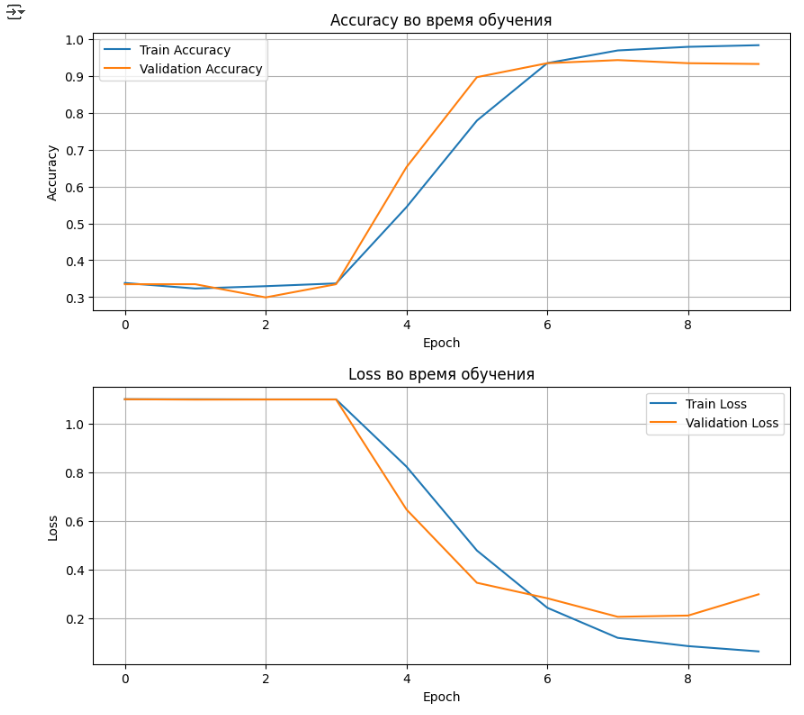


Рисунок 31

На первом графике показана точность на обучающей и валидационной выборках. Видно, что:

* Точность на тренировке быстро достигает почти 100%;
* Валидационная точность начинает стагнировать и колебаться на 6-й эпохе.

На втором графике представлена функция потерь (loss):

* Потери на обучающей выборке стабильно уменьшаются;
* Потери на валидации начинают расти после 8-й эпохи — признак начинающегося переобучения.

Процесс тестирования был проведен, в соответствии с шагами, описанными в главе 2.4 данного отчета, и были получены следующие результаты (рис. 32, 33):

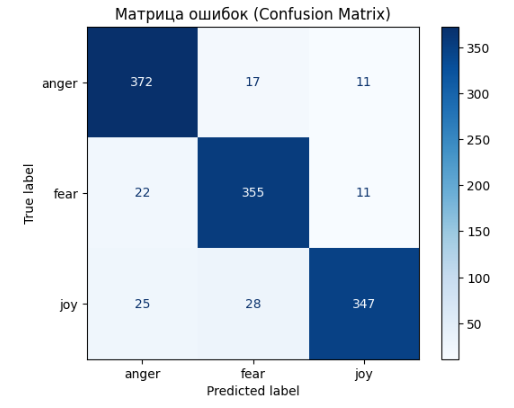


Рисунок 32



Рисунок 33

Соответственно, можно сделать вывод, что точность предсказания по классу «anger» выше, чем у оригинальной модели (без улучшений). Однако значительно упала точность по классам «fear» и «joy», что повлияло на общий показатель *macro avg*, который равен 0,90, против 0,91 у оригинальной модели.

Таким же образом был проведен повторный тюнинг, с целью получения более точных данных. Для этого, при построении модели, были изменены factor (=2, теперь подбор будет более точным, но дольше), project\_name и directory, т.к. Keras Tuner сохраняет результаты тюнинга в папке, которую ты указываешь. Если оставить те же значения, он подгрузит старые результаты и не начнёт тюнинг заново, даже если factor другой:

tuner\_1 = kt.Hyperband(

    build\_model,

    objective='val\_accuracy',

    max\_epochs=20,

    factor=2,

    directory='my\_tuner\_dir',

    project\_name='lstm\_hyperband\_factor2'

)

Графики точности во время обучения, а также функции потерь (рис. 34):

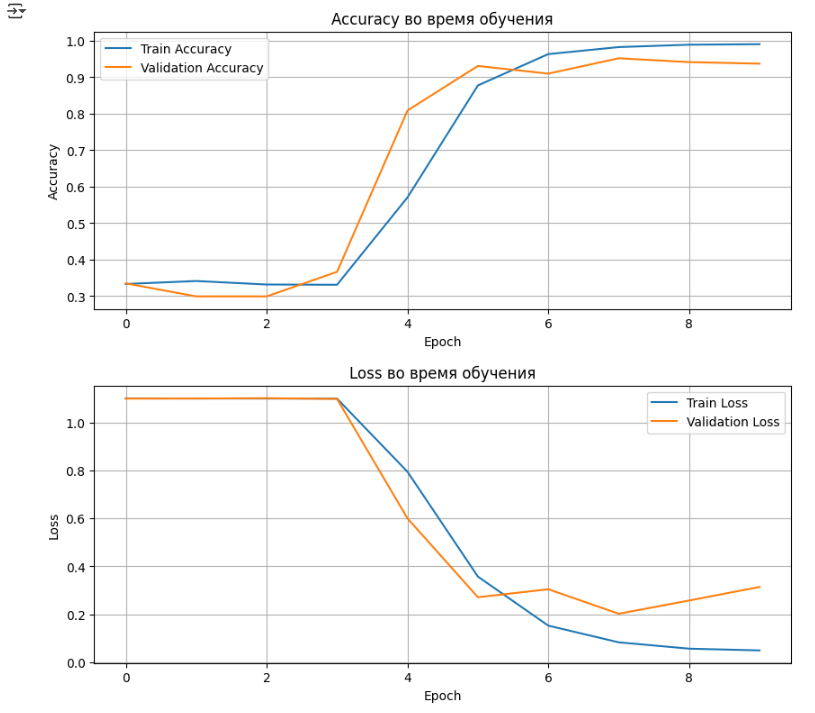


Рисунок 34

На первом графике показана точность на обучающей и валидационной выборках. Видно, что:

* Точность на тренировке быстро достигает почти 100%;
* Валидационная точность начинает стагнировать и колебаться после 4-й эпохи.

На втором графике представлена функция потерь (loss):

* Потери на обучающей выборке стабильно уменьшаются;
* Потери на валидации начинают расти уже после 4-й эпохи, что свидетельствует о переобучении модели.

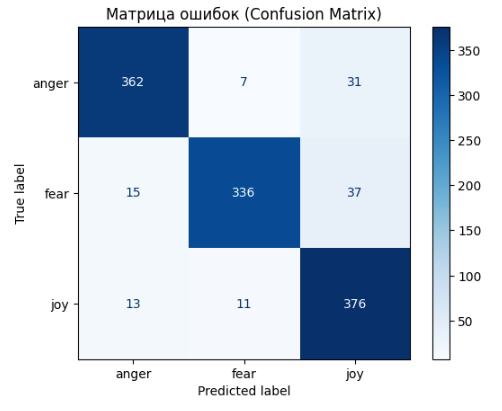


Рисунок 35

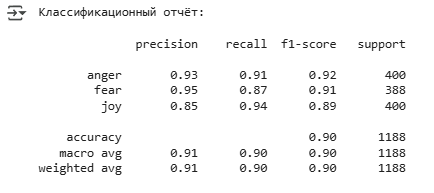


Рисунок 36

Матрица ошибок и классификационный (рис. 35, 36) отчет показывают, что точность предсказания по классу «anger» выше, чем у оригинальной модели (без улучшений). Однако значительно упала точность по классам «fear» и «joy», что повлияло на общий показатель *macro avg*, который равен 0,90, против 0,91 у оригинальной модели.

Ситуация аналогична, что и с первой попыткой улучшения модели, однако судя по данным классификационных отчетов, во втором случае точность предсказания по классам «anger» и «fear» возросла, но компенсировалась потерей точности предсказания для класса «joy».

## **2.7. Выводы по разделу.**

Таким образом, была разработана модель нейронной сети, определяющая эмоциональную направленность комментария. В результате выполненной работы можно сделать следующие выводы (табл.3):

Таблица 3

|  |  |  |
| --- | --- | --- |
| Выводы | Сформированные компетенции | |
| Код компетенции | Содержание компетенции |
| Реализованы алгоритмы сбора информации и создания информационных объектов на Python | **ПК-1** | Обладать способностью разработки прикладного программного обеспечения, автоматизации работы с базами данных и документами, программирования бизнес-логики приложений, интеграции разнородных данных |
| Осуществлено проектирование решения, проведено тестирование и оценено качество работы. | **ПК-7** | Способность использовать отечественные и международные стандарты при проектировании и обеспечении качества прикладного программного обеспечения. |
| Разработаны нейроны и нейронные блоки на языке программирования Python.  Создана нейронная сеть. Экспериментальным способом подобраны характеристики нейронной сети: определено количество слоев, число блоков в скрытых слоях, наличие или отсутствие обходных соединений, передаточные функции нейронов.  Собраны данные для обучения нейронной сети. Параметры обучения подобраны экспериментальным способом.  Произведено обучение нейронной сети на собранных данных. Проверена адекватность результатов обучения сети. | **ПК-8** | Знать методы и инструментальные средства интеллектуального анализа больших данных |

# **ЗАКЛЮЧЕНИЕ**

В результате выполнения поставленных в предметной области задач был проведен анализ и подготовка данных, выбрана топологии нейронной сети, построена модель этой сети, произведено обучение, тренировка и улучшение данной модели посредством оптимизации гиперпараметров.

Несмотря на то, что попытки улучшения модели не дали положительных результатов, в результате чего был сделан вывод, что наилучшей по точности предсказания является именно оригинальная модель, точность ее предсказания, судя по показателю *macro\_avg = 0,91* из классификационного отчета, является довольно высокой. Данный факт позволяет сделать вывод о том, что данная модель нейронной сети способна выполнить поставленную перед ней задачу.

Таким образом, цель практики ‑ закрепление теоретических знаний, полученных в процессе обучения; приобретения практических навыков, компетенций и опыта деятельности по направлению подготовки; ознакомления на практике с вопросами профессиональной деятельности, направленными на формирование знаний, навыков и опыта профессиональной деятельности в полной мере достигнута.

# **СПИСОК ЛИТЕРАТУРЫ**

* 1. Рашитов, Э. Э. Модель математической нейронной сети / Э. Э. Рашитов, К. Л. Стоякова, Р. Р. Ибраев. — Текст : непосредственный // Молодой ученый. — 2017. — № 15 (149). — С. 77-81. — URL: <https://moluch.ru/archive/149/42296/>;
  2. Куликова И.В, Нейросети на Python. Основы ИИ и машинного обучения. Издание: КТК «Галактика» - 2025 г. – стр.86-89, стр. 128-142;
  3. Murphy, K.P. and Massachusetts Institute Of Technology (2022). Machine learning : a probabilistic perspective. Cambridge (Ma): Mit Press. – стр. 546-548;
  4. Raschka, S. and Olson, R.S. (2020). Python machine learning: unlock deeper insights into machine learning with this vital guide to cutting-edge predictive analytics. Birmingham, Uk: Packt Publishing. – стр. 105;
  5. Интернет-источник: <https://habr.com/ru/companies/raft/articles/784964/> (дата обращения к источнику – 12.04.2025);
  6. Интернет-источник: <https://www.geeksforgeeks.org/relu-activation-function-in-deep-learning/> (дата обращения к источнику – 13.04.2025);
  7. Интернет-источник: <https://wiki.merionet.ru/articles/teoreticheskie-osnovy-nejronnyx-setej> (дата обращения к источнику – 13.04.2025);
  8. Интернет-источник: <https://habr.com/ru/companies/skillfactory/articles/525260/> (дата обращения к источнику – 15.04.2025);
  9. Интернет-источник: <https://keras.io/api/losses/probabilistic_losses/#categorical_crossentropy-function> (дата обращения к источнику – 22.04.2025);
  10. Интернет-источник: <https://keras.io/api/optimizers/adam/> (дата обращения к источнику – 25.04.2025);
  11. Интернет-источник: <https://ru-keras.com/home/> (дата обращения к источнику – 27.04.2025).