Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение
высшего образования
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт перспективной инженерии
Департамента цифровых, роботехнических систем и электроники

ОТЧЕТ
ПО ЛАБОРАТОРНОЙ РАБОТЕ №4
дисциплины «Основы нейронных сетей»

Выполнил:
Гайчук Дарья Дмитриевна
3 курс, группа ИВТ-б-о-22-1,
09.03.01 «Информатика
и вычислительная техника»,
направленность (профиль)
«Программное обеспечение средств
вычислительной техники и
автоматизированных систем», очная
форма обучения

_____
(подпись)

Руководитель практики:
Воронкин Р.А.-доцент департамента
цифровых, роботехнических систем и
электроники института перспективной
инженерии

_____
(подпись)

Отчет защищен с оценкой _____ Дата защиты_____

Ставрополь, 2025 г.

Тема: «Обработка текстов с помощью нейронных сетей.»

Цель: изучить особенности построения архитектур нейронных сетей для обработки текста и особенности обучения таких НС.

Ссылка на git: https://github.com/Ichizuchi/NN_LR4

Порядок выполнения работы:

1. Базовая часть: Классификация писателей

В первой части была реализована классификация по стилю текста, используя тексты писателей (Булгаков, О. Генри, Стругацкие и др.).

Ключевые шаги:

1) Загрузка и предварительная обработка текстов
2) Построение словаря частотности и токенизация
3) Разбиение текстов на отрезки (скользящее окно)
4) Формирование обучающей и тестовой выборки
5) Обучение моделей на форматах Bag of Words и Embedding

Использованные модели:

1. BoW + Dense: простая архитектура, точность ~76%
2. Embedding(20): слабее BoW, склонна к переобучению (~70%)
3. Embedding(200): немного лучше (~75%)
4. Embedding(200) без Dropout: ухудшение качества, признаки переобучения

```python
# Создание последовательной модели нейросети
model_text_bow_softmax = Sequential()
# Первый полносвязный слой
model_text_bow_softmax.add(Dense(200, input_dim=VOCAB_SIZE, activation="relu"))
# Слой регуляризации Dropout
model_text_bow_softmax.add(Dropout(0.25))
# Слой пакетной нормализации
model_text_bow_softmax.add(BatchNormalization())
# Выходной полносвязный слой
model_text_bow_softmax.add(Dense(CLASS_COUNT, activation='softmax'))

# Входные данные подаются в виде векторов bag of words
compile_train_eval_model(model_text_bow_softmax,
                         x_train_01, y_train,
                         x_test_01, y_test,
                         class_labels=CLASS_LIST,
                         title='BoW')
```

```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/core/dense.py:87: UserWarning: Do not pass an `input_sha
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Model: "sequential"
```

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 200)	4,000,200
dropout (Dropout)	(None, 200)	0
batch_normalization (BatchNormalization)	(None, 200)	800
dense_1 (Dense)	(None, 6)	1,206

```
 Total params: 4,002,206 (15.27 MB)
 Trainable params: 4,001,806 (15.27 MB)
 Non-trainable params: 400 (1.56 KB)
Epoch 1/50
```

Рисунок 1. Структура модели (BoW)

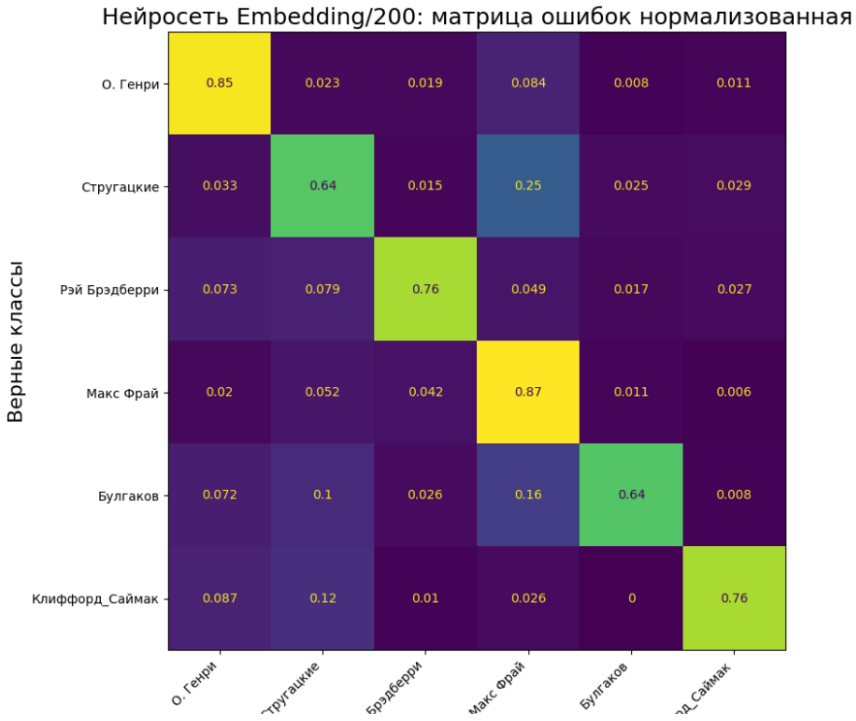


Рисунок 2. График точности во время обучения

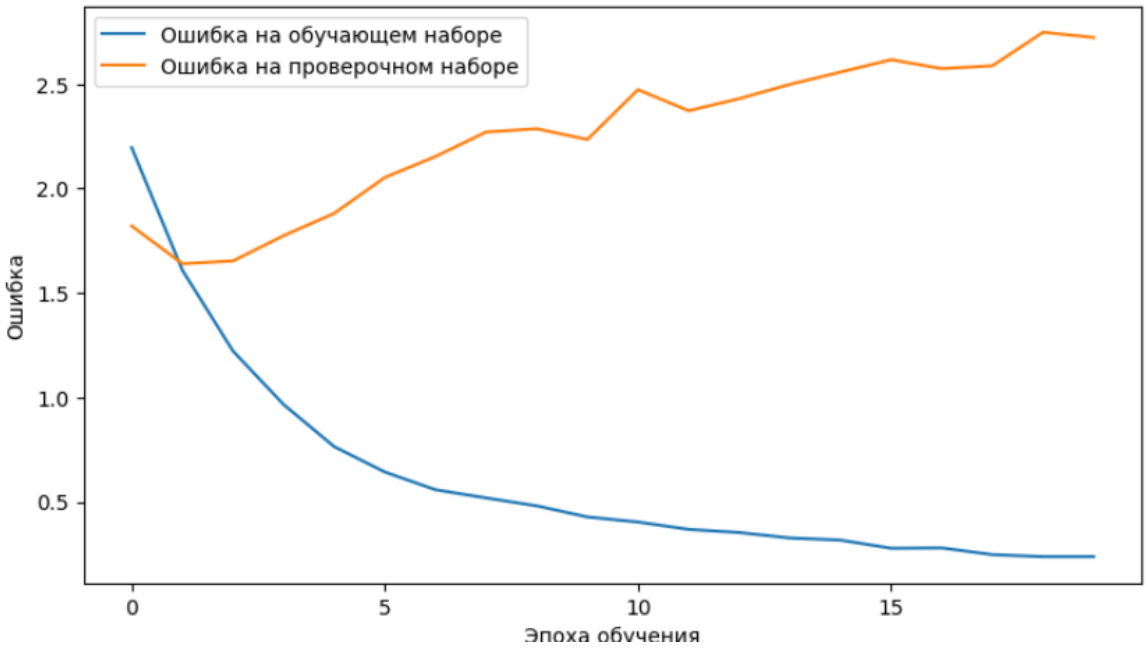


Рисунок 3. График ошибки сети во время обучения

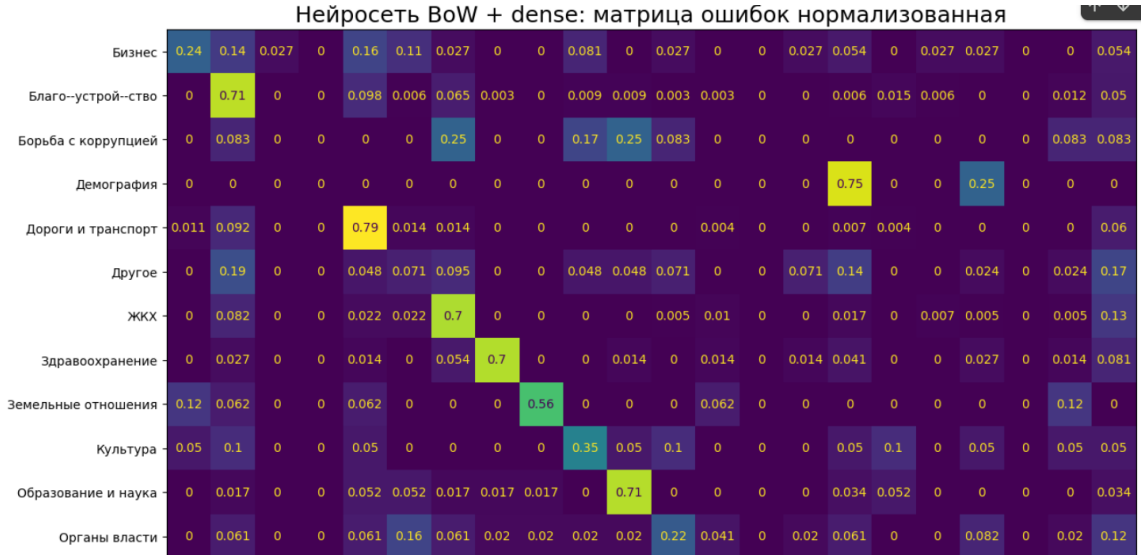


Рисунок 4. Матрица ошибок модели

Выполнение домашних заданий

1. Домашнее задание Lite: Классификация обращений граждан

Задача: Классификация по категориям из обращений граждан (ЖКХ, Дороги, Соцсфера и т.д.)

Этапы:

Описание первого шага: импорт библиотек и загрузка датасета.

```
[1]  # Загрузка датасетов из облака google
     import gdown
```

```
[2]  # Загрузка датасета из облака
     gdown.download('https://storage.yandexcloud.net/aiueducation/Content/base/l7/writers.zip', None, quiet=True)
```

```
'writers.zip'
```

```
# Ваше задание здесь

# Распакуем загруженный архив в папку writers
!unzip -qo writers.zip -d writers/

# Просмотр содержимого папки
!ls writers
```

```
'(Булгаков) Обучающая_5 вместе.txt'
'(Булгаков) Тестовая_2 вместе.txt'
'(Клиффорд_Саймак) Обучающая_5 вместе.txt'
'(Клиффорд_Саймак) Тестовая_2 вместе.txt'
'(Макс Фрай) Обучающая_5 вместе.txt'
'(Макс Фрай) Тестовая_2 вместе.txt'
'(О. Генри) Обучающая_50 вместе.txt'
'(О. Генри) Тестовая_20 вместе.txt'
'(Рэй Брэдберри) Обучающая_22 вместе.txt'
'(Рэй Брэдберри) Тестовая_8 вместе.txt'
'(Стругацкие) Обучающая_5 вместе.txt'
'(Стругацкие) Тестовая_2 вместе.txt'
```

Рисунок 5. Импорт необходимых библиотек и загрузка данных

Следующий этап — очистка данных и замена пропущенных значений.

```
FILE_DIR = "writers"  # Папка с текстовыми файлами
SIG_TRAIN = "обучающая"  # Признак обучающей выборки в имени файла
SIG_TEST = "тестовая"  # Признак тестовой выборки в имени файла

# Загрузка датасета. Добавляются имена классов и соответствующие тексты.
# Все тексты преобразуются в строку и объединяются для каждого класса и выборки
CLASS_LIST = []
text_train = []
text_test = []

for file_name in os.listdir(FILE_DIR):
    # Выделение имени класса и типа выборки из имени файла
    m = re.match(r"\((.+)\) (\S+)_", file_name)
    # Если выделение получилось, то файл обрабатывается
    if m:
        class_name = m[1]
        subset_name = m[2].lower()
        # Проверка типа выборки в имени файла
        is_train = SIG_TRAIN in subset_name
        is_test = SIG_TEST in subset_name

        # Если тип выборки обучающая либо тестовая - файл обрабатывается
        if is_train or is_test:
            # Добавление нового класса, если его еще нет в списке
            if class_name not in CLASS_LIST:
                print(f'Добавление класса "{class_name}"')
                CLASS_LIST.append(class_name)
                # Инициализация соответствующих классу строк текста
                text_train.append("")
                text_test.append("")

            # Поиск индекса класса для добавления содержимого файла в выборку
            cls = CLASS_LIST.index(class_name)
            print(
                f'Добавление файла "{file_name}" в класс "{CLASS_LIST[cls]}", {subset_name} выборка.'
            )
            with open(f"{FILE_DIR}/{file_name}", "r") as f:
                # Загрузка содержимого файла в строку
                text = f.read()
            # Определение выборки, куда будет добавлено содержимое
            subset = text_train if is_train else text_test
            # Добавление текста к соответствующей выборке класса. Концы строк заменяются на пробел
            subset[cls] += " " + text.replace("\n", " ")
```

Рисунок 6. Очистка таблицы и замена NaN

Далее происходит токенизация текста и кодирование классов метками.

```
results[ шаг окна ].extend([win_hope] * 3)

# Используется встроенный в Keras токенизатор для разбиения текста и построения частотного словаря
tokenizer = Tokenizer(
    num_words=vocab_size,
    filters='!"#$%&()*+,-—./…:;<=>?@[\\]^_`{|}~«»\t\n\xa0\ufeff',
    lower=True,
    split=" ",
    oov_token="неизвестное_слово",
    char_level=False,
)
```

Рисунок 7. Токенизация и кодировка меток классов

Тексты преобразуются в формат Bag of Words и формируются выборки:

```
seq_test = tokenizer.texts_to_sequences(text_test)
# Формирование обучающей выборки
x_train, y_train = vectorize_sequence(seq_train, win_size, win_hope)
# Формирование тестовой выборки
x_test, y_test = vectorize_sequence(seq_test, win_size, win_hope)
x_train = tokenizer.sequences_to_matrix(x_train.tolist()).astype("float16")
x_test = tokenizer.sequences_to_matrix(x_test.tolist()).astype("float16")
create_and_train_models(vocab_size, results, x_train, y_train, x_test, y_test)
```

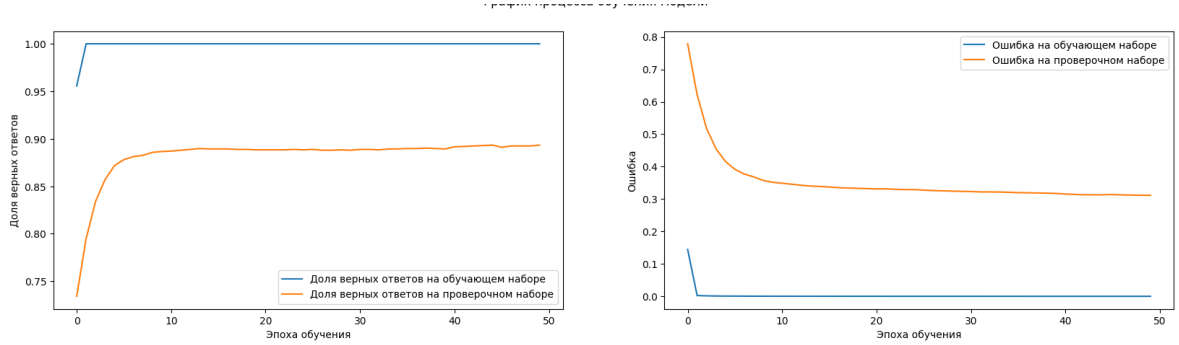
Рисунок 8. Преобразование текстов в BoW и формирование выборок

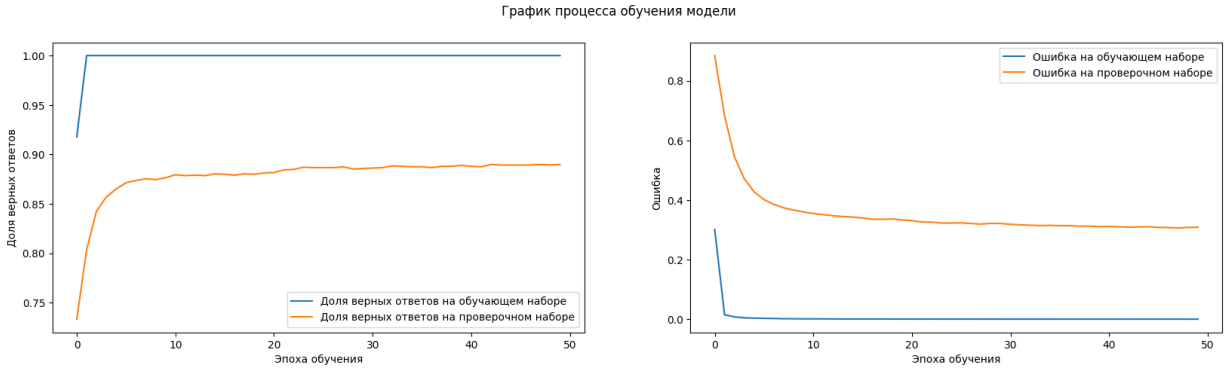
Создаётся модель с полносвязными слоями и dropout:

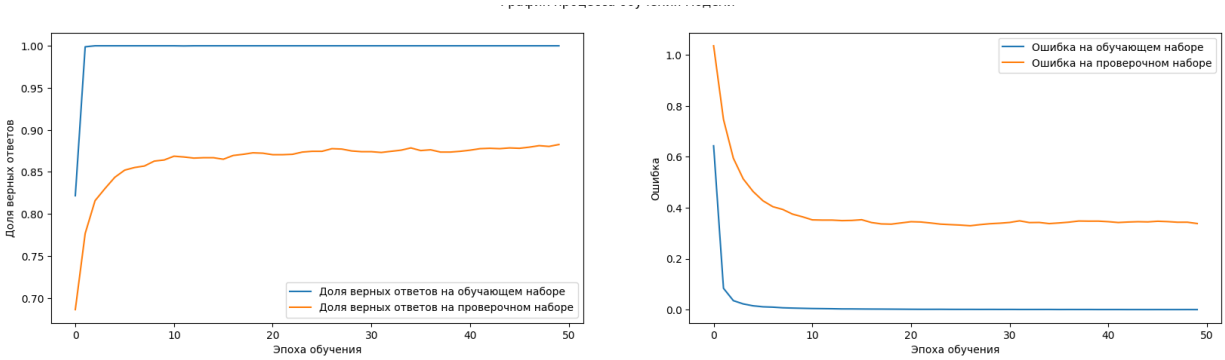
```
# Напишем функцию для создания и обучения 9-ти моделей с разными гиперпараметрами
def create_and_train_models(size, results, x_train, y_train, x_test, y_test):
    model_lite = Sequential(
        [
            Input((size,)),
            Dense(256),
            BatchNormalization(),
            Activation("relu"),
            Dropout(0.25),
            Dense(CLASS_COUNT, activation="softmax"),
        ]
    )

    model_middle = Sequential(
        [
            Input((size,)),
            Dense(256),
            BatchNormalization(),
            Activation("relu"),
            Dropout(0.25),
            Dense(128),
            BatchNormalization(),
            Activation("relu"),
            Dropout(0.25),
            Dense(CLASS_COUNT, activation="softmax"),
        ]
    )

    model_hard = Sequential(
        [
            Input((size,)),
            Dense(256),
            BatchNormalization(),
            Activation("relu"),
            Dropout(0.25),
            Dense(128),
            BatchNormalization(),
            Activation("relu"),
```

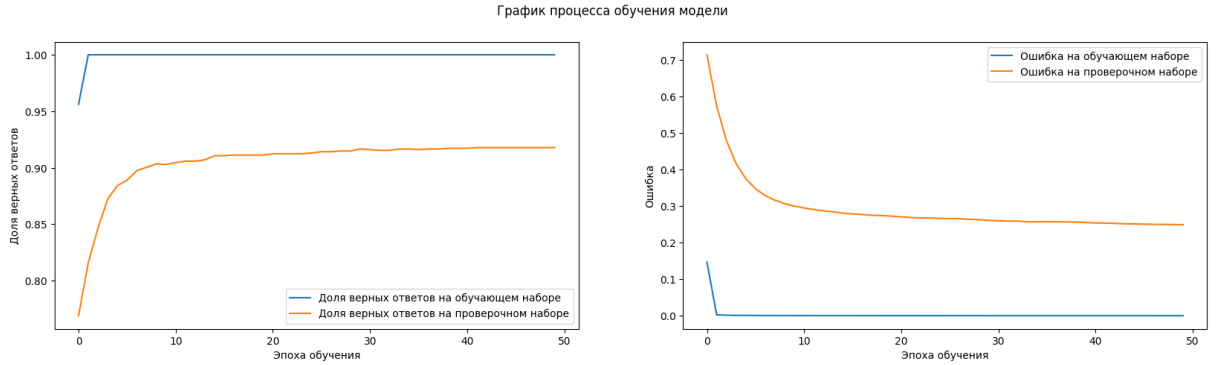
Рисунок 9. Архитектура модели BoW + Dense

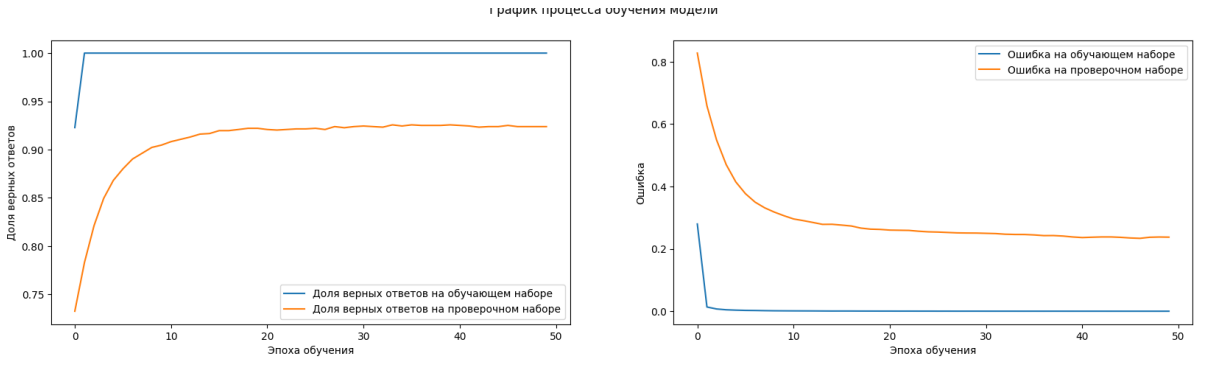
После обучения модели выводятся графики точности и ошибки:











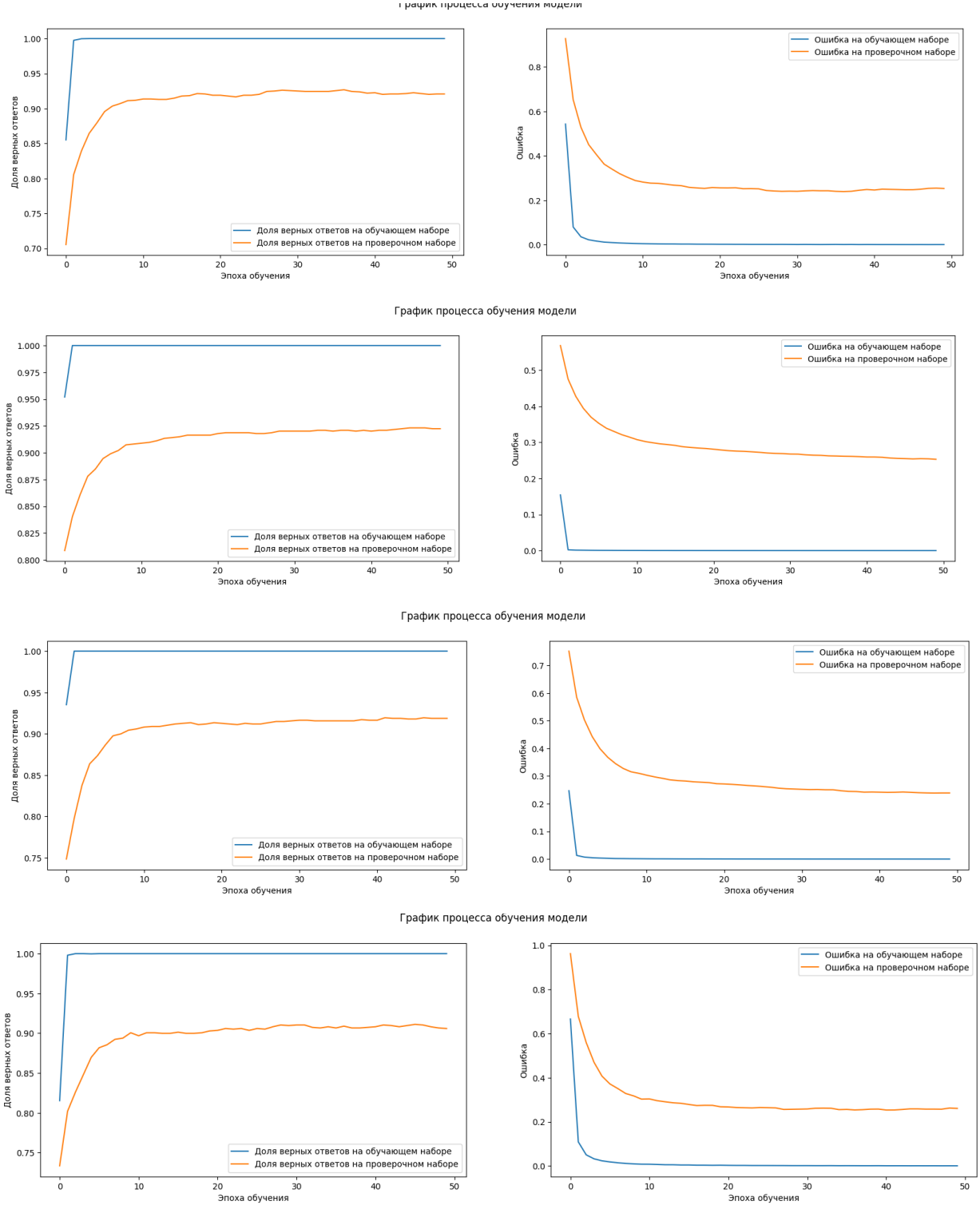


Рисунок 10. Графики точности и ошибки модели

Выполняется балансировка данных по количеству классов:

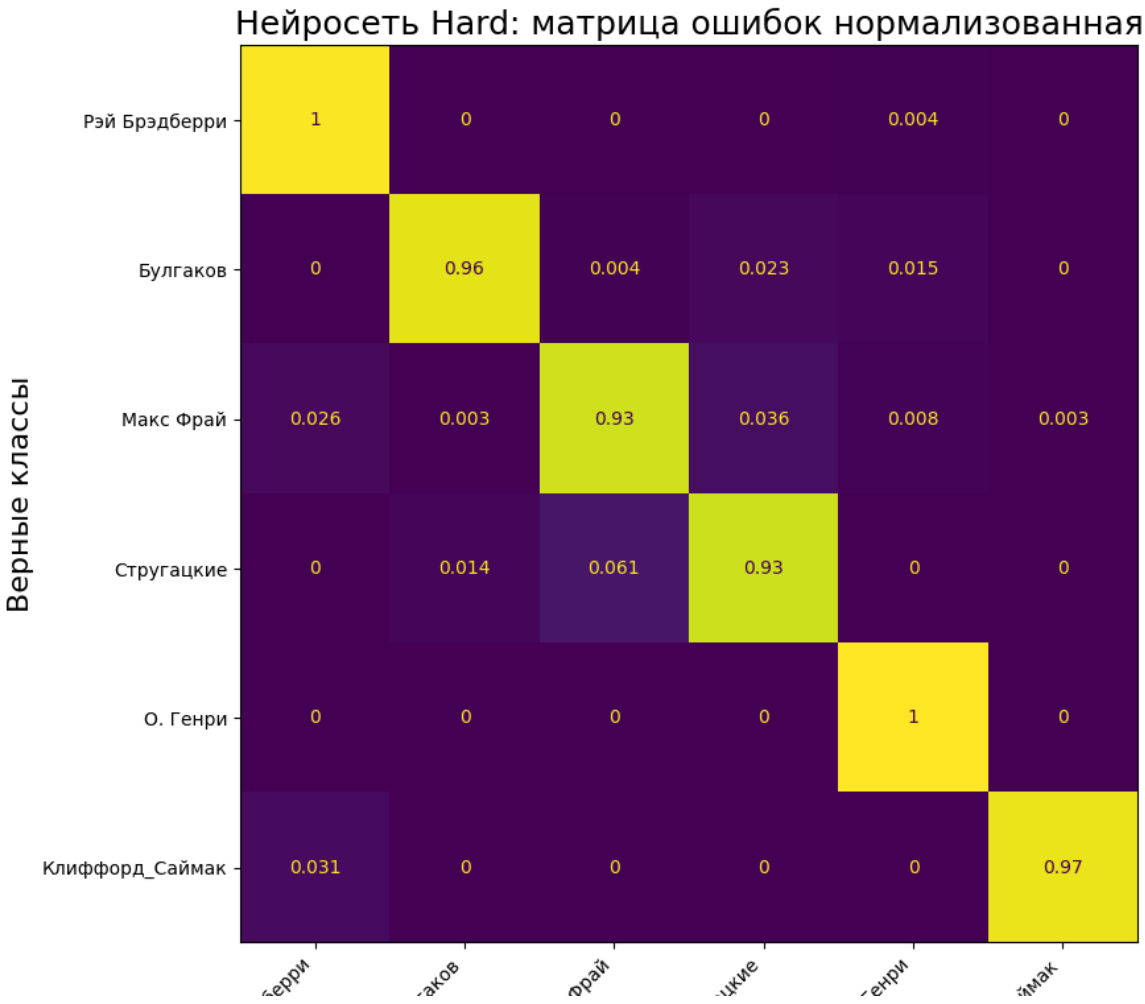


Рисунок 11. Матрица ошибок сети с лучшей точностью

Таким образом можно сделать вывод о том, что увеличение словаря частотности и ширины шага положительно влияют на обучение модели.

3. Домашнее задание Pro: Классификация отзывов Tesla

Задача: Определение тональности (позитив/негатив) отзывов об автомобиле Tesla

Описание подготовки данных — разметка, скользящее окно, токенизация.

```
# Работа с массивами данных
import numpy as np

# Работа с таблицами
import pandas as pd

# Отрисовка графиков
import matplotlib.pyplot as plt

# Функции-утилиты для работы с категориальными данными
from tensorflow.keras import utils

# Класс для конструирования последовательной модели нейронной сети
from tensorflow.keras.models import Sequential

# Основные слои
from tensorflow.keras.layers import Dense, Dropout, SpatialDropout1D, BatchNormalization, Embedding, Flatten, Activation, In

# Токенизатор для преобразование текстов в последовательности
from tensorflow.keras.preprocessing.text import Tokenizer

# Заполнение последовательностей до определенной длины
from tensorflow.keras.preprocessing.sequence import pad_sequences

# Загрузка датасетов из облака google
import gdown

# Для работы с файлами в Colaboratory
import os

# Отрисовка графиков
import matplotlib.pyplot as plt

# Оптимизаторы
from tensorflow.keras.optimizers import Adam

%matplotlib inline
```

```
gdown.download('https://storage.yandexcloud.net/aiueducation/Content/base/l7/tesla.zip', None, quiet=True)
```

Рисунок 12. Подготовка данных и построение словаря

```
def read_text(file_name):

  # Задаем открытие нужного файла в режиме чтения
  read_file = open(file_name, 'r')

  # Читаем текст
  text = read_file.read()

  # Переносы строки переводим в пробелы
  text = text.replace("\n", " ")

  # Возвращаем текст файла
  return text

# Объявляем интересующие нас классы
class_names = ["Негативный отзыв", "Позитивный отзыв"]

# Считаем количество классов
num_classes = len(class_names)
```

Рисунок 13. функция чтения содержимого файла и преобразования текста в одну строку

```
texts_len = [len(text) for text in texts_list]

# Устанавливаем "счётчик" номера текста
t_num = 0

# Выводим на экран  информационное сообщение
print(f'Размеры текстов по порядку (в токенах):')

# Циклом проводим итерацию по списку с объёмами текстов
for text_len in texts_len:

  # Запускаем "счётчик" номера текста
  t_num += 1

  # Выводим на экран сообщение о номере и объёме текста
  print(f'Текст №{t_num}: {text_len}')
```

```
Размеры текстов по порядку (в токенах):
Текст №1: 134535
Текст №2: 213381
```

Рисунок 14. Объемы текстов негативного и позитивного

```
# Создаём список с вложенным циклом по длинам текстов, где i - 100% текста, i/5 - 20% текста
train_len_shares = [(i - round(i/5)) for i in texts_len]

# Устанавливаем "счётчик" номера текста
t_num = 0

# Циклом проводим итерацию по списку с объёмами текстов равными 80% от исходных
for train_len_share in train_len_shares:

  # Запускаем "счётчик" номера текста
  t_num += 1

  # Выводим на экран сообщение о номере и объёме текста в 80% от исходного
  print(f'Доля 80% от текста №{t_num}: {train_len_share} символов')
```

```
Доля 80% от текста №1: 107628 символов
Доля 80% от текста №2: 170705 символов
```

Рисунок 15. Сколько символов составит 80% объёма каждого текста

```
tokenizer = Tokenizer(
    num_words=VOCAB_SIZE,
    filters='!"#$%&()*+,-—./…:;<=>?@[\\]^_`{|}~«»\t\n\xa0\ufeff',
    lower=True,
    split=" ",
    oov_token="неизвестное_слово",
    char_level=False,
)
tokenizer.fit_on_texts(texts_list)
seq_train = tokenizer.texts_to_sequences(text_train)
seq_test = tokenizer.texts_to_sequences(text_validate)
# Формирование обучающей выборки
x_train, y_train = vectorize_sequence(seq_train, WIN_SIZE, WIN_HOPE)
# Формирование тестовой выборки
x_test, y_test = vectorize_sequence(seq_test, WIN_SIZE, WIN_HOPE)
x_train = tokenizer.sequences_to_matrix(x_train.tolist()).astype("float16")
x_test = tokenizer.sequences_to_matrix(x_test.tolist()).astype("float16")
```

Рисунок 16. Словарь частотности на данных выборках

Создание и обучение модели для бинарной классификации:

```
# Создадим модель нейронной сети
model = Sequential(
    [
        Input((VOCAB_SIZE,)),
        Dense(200, activation="relu"),
        Dropout(0.4),
        Dense(100),
        Dropout(0.4),
        Dense(2, activation="softmax"),
    ]
)

model.compile(
    optimizer=Adam(learning_rate=0.001),
    loss="categorical_crossentropy",
    metrics=["accuracy"],
)

model.summary()
```

```
Model: "sequential_2"
```

Layer (type)	Output Shape	Param #
dense_6 (Dense)	(None, 200)	4,000,200
dropout_4 (Dropout)	(None, 200)	0
dense_7 (Dense)	(None, 100)	20,100
dropout_5 (Dropout)	(None, 100)	0
dense_8 (Dense)	(None, 2)	202

```
Total params: 4,020,502 (15.34 MB)
Trainable params: 4,020,502 (15.34 MB)
Non-trainable params: 0 (0.00 B)
```

Рисунок 17. Архитектура модели и параметры обучения

Вывод графиков обучения:

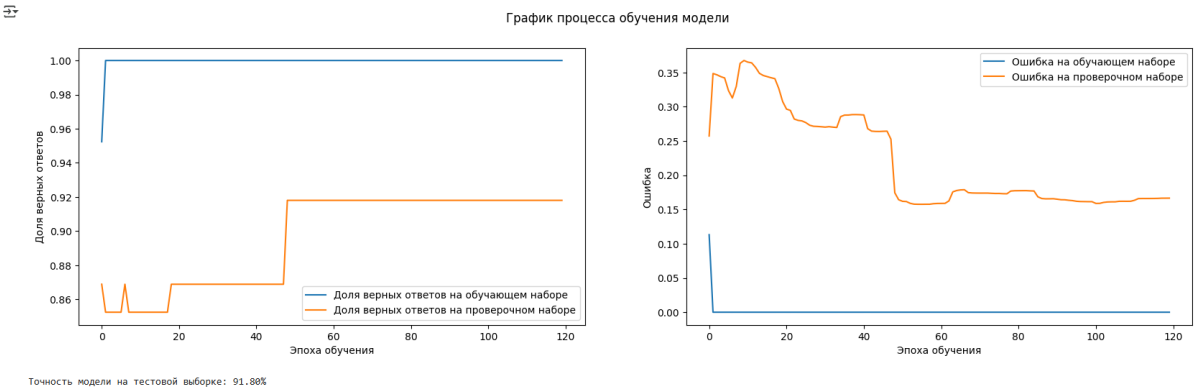


Рисунок 18. Графики точности и ошибки во время обучения

Результат: точность 91.8%, переобучения не наблюдается.

4. Домашнее задание Ultra Pro: Распознавание 20 писателей

Задача: Классификация текста по автору из списка из 20 писателей

Все тексты писателей были объединены в один список. Выборка была разделена на обучающую и тестовую:

```
[4]  # Сохраним путь к папке с выборкой
     path = Path("20writers")

     # Список для текста базы и распознаваемых классов (имен писателей)
     text = []
     class_names = []

     # Объединим тексты писателей в один список
     for f in path.iterdir():
         class_names.append(f.name)
         text.append(f.read_text().replace("\n", " "))
```

```
[5]  # Определим длины текстов 20 писателей
     text_lens = [len(t) for t in text]
     split_indices = [int(lenth * 0.8) for lenth in text_lens]
     print(text_lens)
```

```
[1965119, 2255254, 2952792, 1972541, 6611627, 2579246, 3408339, 2399413, 3089426, 1980206, 1992830, 3104712, 3357061, 2001064, 5178950, 3386268
```

```
[6]  # Разделим выборку на обучающую 80% и тестовую 20%
     train_text = []
     test_text = []
     for i, t in enumerate(text):
         train_text.append(t[:split_indices[i]])
         test_text.append(t[split_indices[i]:])
```

Рисунок 19. Разделение выборки

Проверка сбалансированности выборок по классам, вывод гистограммы:

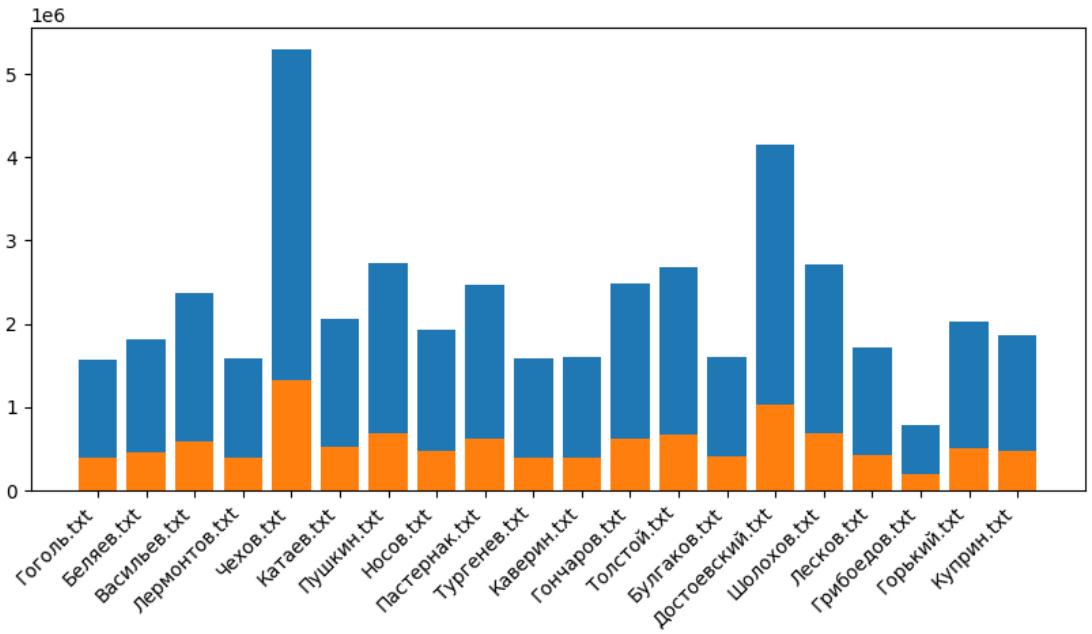
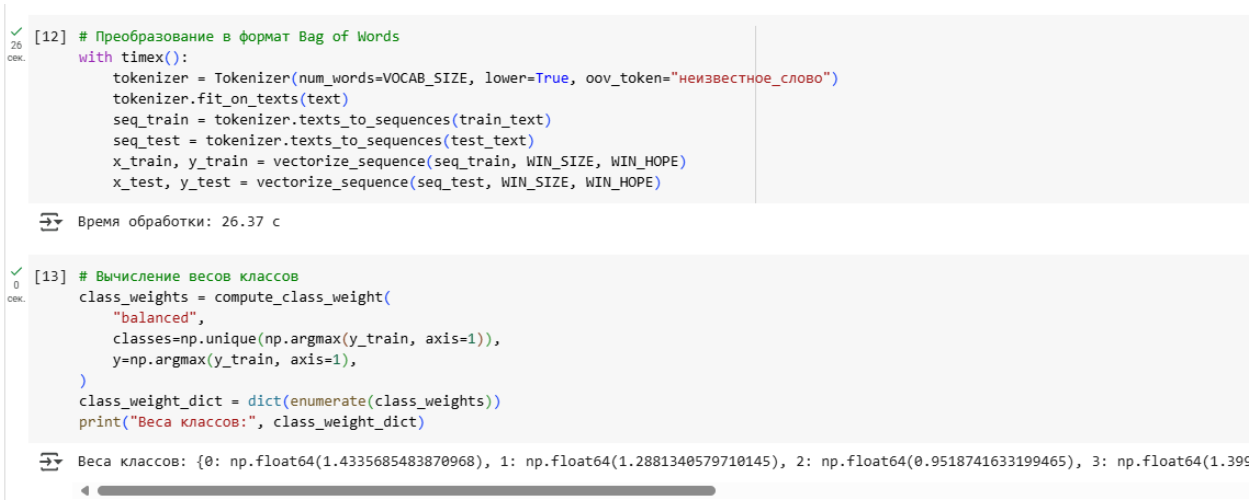


Рисунок 20. Гистограмма объёмов текстов авторов

Создание модели с использованием Bag of Words:



```
[12] # Преобразование в формат Bag of Words
     with timex():
         tokenizer = Tokenizer(num_words=VOCAB_SIZE, lower=True, oov_token="неизвестное_слово")
         tokenizer.fit_on_texts(text)
         seq_train = tokenizer.texts_to_sequences(train_text)
         seq_test = tokenizer.texts_to_sequences(test_text)
         x_train, y_train = vectorize_sequence(seq_train, WIN_SIZE, WIN_HOPE)
         x_test, y_test = vectorize_sequence(seq_test, WIN_SIZE, WIN_HOPE)

     Время обработки: 26.37 с

[13] # Вычисление весов классов
     class_weights = compute_class_weight(
         "balanced",
         classes=np.unique(np.argmax(y_train, axis=1)),
         y=np.argmax(y_train, axis=1),
     )
     class_weight_dict = dict(enumerate(class_weights))
     print("Веса классов:", class_weight_dict)

     Веса классов: {0: np.float64(1.4335685483870968), 1: np.float64(1.2881340579710145), 2: np.float64(0.9518741633199465), 3: np.float64(1.39
```

Рисунок 21. Архитектура модели BoW + Dense

Преобразуем выборку в формат Bag of Words

```
with timex():
    x_train_matrix = tokenizer.sequences_to_matrix(x_train.tolist(), mode="binary").astype("float16")
    x_test_matrix = tokenizer.sequences_to_matrix(x_test.tolist(), mode="binary").astype("float16")

# Вывод формы обучающей выборки
print("Форма обучающей выборки:", x_train_matrix.shape)
print("Форма тестовой выборки:", x_test_matrix.shape)
```

```
Время обработки: 25.17 с
Форма обучающей выборки: (14221, 25000)
Форма тестовой выборки: (3466, 25000)
```

```
# Преобразуем one-hot метки в числовые классы
train_labels = np.argmax(y_train, axis=1)
test_labels = np.argmax(y_test, axis=1)
```

Рисунок 22. Преобразование выборки

Создадим модель сети для формата Bag of Words

```python
model = Sequential(
    [
        Input((VOCAB_SIZE,)),
        Dropout(0.6),
        Dense(128, activation="relu"),
        Dropout(0.6),
        Dense(64, activation="relu"),
        Dropout(0.6),
        Dense(20, activation="softmax"),
    ]
)

model.compile(
    optimizer=Adam(learning_rate=0.0005),
    loss="categorical_crossentropy",
    metrics=["accuracy"],
)

model.summary()
```

```
Model: "sequential"
```

Layer (type)	Output Shape	Param #
dropout (Dropout)	(None, 25000)	0
dense (Dense)	(None, 128)	3,200,128
dropout_1 (Dropout)	(None, 128)	0
dense_1 (Dense)	(None, 64)	8,256
dropout_2 (Dropout)	(None, 64)	0
dense_2 (Dense)	(None, 20)	1,300

```
Total params: 3,209,684 (12.24 MB)
Trainable params: 3,209,684 (12.24 MB)
Non-trainable params: 0 (0.00 B)
```

Рисунок 23. Модель формата «BoF»

```python
fig.suptitle("График процесса обучения модели")
ax1.plot(history.history["accuracy"], label="Доля верных ответов на обучающем наборе")
ax1.plot(
    history.history["val_accuracy"],
    label="Доля верных ответов на проверочном наборе",
)
ax1.xaxis.get_major_locator().set_params(integer=True)
ax1.set_xlabel("Эпоха обучения")
ax1.set_ylabel("Доля верных ответов")
ax1.legend()

ax2.plot(history.history["loss"], label="Ошибка на обучающем наборе")
ax2.plot(history.history["val_loss"], label="Ошибка на проверочном наборе")
ax2.xaxis.get_major_locator().set_params(integer=True)
ax2.set_xlabel("Эпоха обучения")
ax2.set_ylabel("Ошибка")
ax2.legend()
plt.show()
```

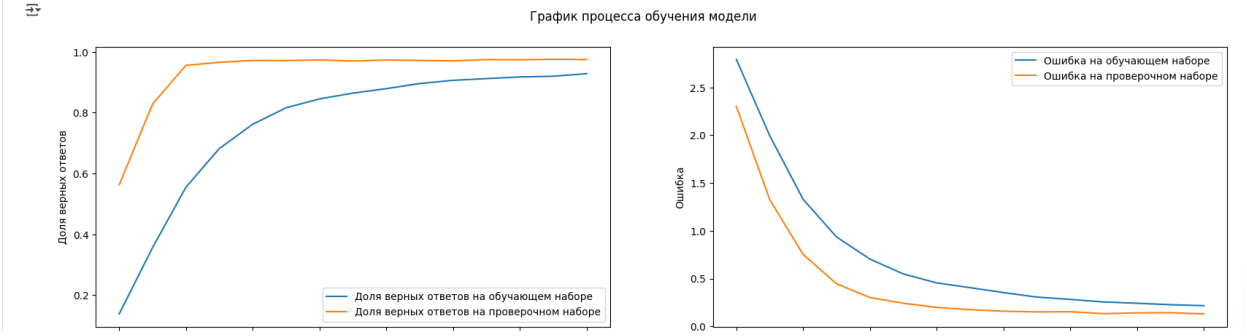


Рисунок 24. Вывод графиков процесса обучения

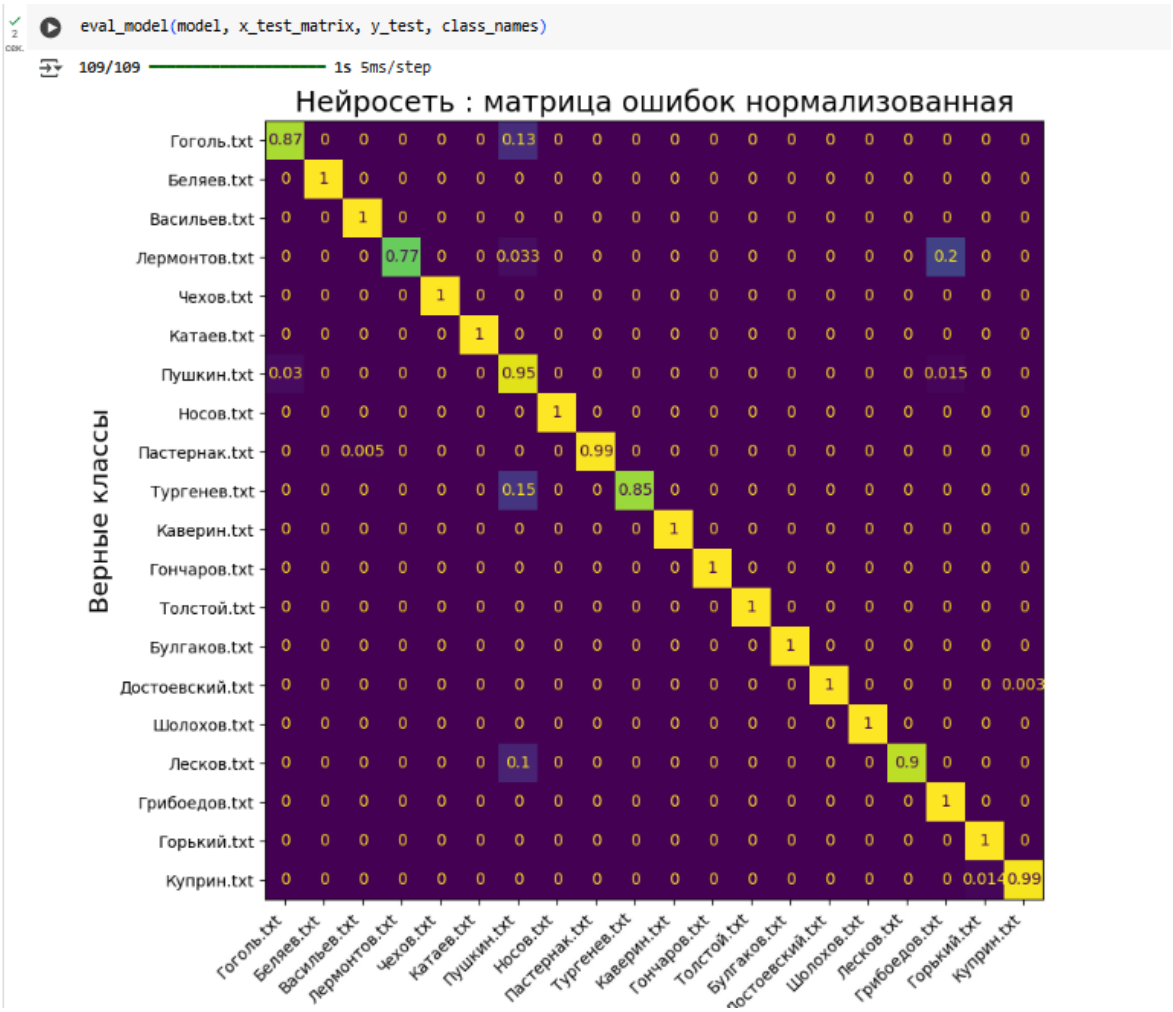


Рисунок 25. Матрица ошибок

Результат: точность 97%

Структура модели с использованием слоя «Embedding»

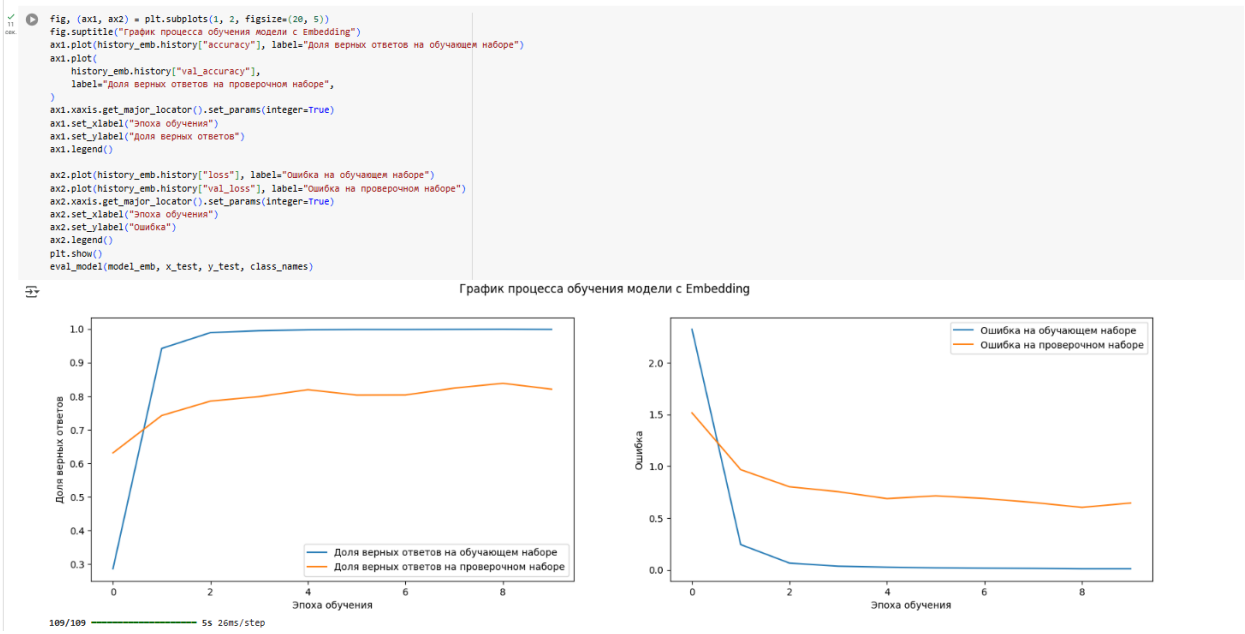


Рисунок 26. График процесса обучения модели с Embedding

Результат: точность 77%

Вывод: в ходе лабораторной работы были рассмотрены разные подходы к обработке текстов для задач классификации. Исследовано влияние архитектур (BoW, Embedding) и параметров подготовки данных на качество моделей. Было выполнено 3 уровня индивидуальных заданий с практическими результатами и анализом точности моделей.