**МИНИСТЕРСТВО ТРАНСПОРТА РОССИЙСКОЙ ФЕДЕРАЦИИ**

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ**

**УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**

**«РОССИЙСКИЙ УНИВЕРСИТЕТ ТРАНСПОРТА»**

**(РУТ (МИИТ)**

Институт/факультет «Высшая Инженерная Школа, ВИШ»

Специальность/Направление подготовки Информатика и вычислительная техника

**КУРСОВАЯ РАБОТА**

**по дисциплине:** Анализ больших текстовых данных и текстовый поиск.

|  |  |
| --- | --- |
| **на тему:** | "Применение нейронной сети для прогноза канала Телеграмм, в котором опубликован пост" |

|  |  |
| --- | --- |
| **Студент группы ШАД-212** | **(Гинда Д. А.)** |
| **Научный руководитель** | **(Проневич О. Б.)** |
|  |  |

**Москва 2024 г.**

**Оглавление**

[**Основные термины** 3](#_Toc169202877)

[**Введение** 4](#_Toc169202878)

[**Основные шаги выполнения. Парсинг данных** 6](#_Toc169202879)

[**Основные шаги выполнения. Первичный анализ данных.** 9](#_Toc169202880)

[**Основные шаги выполнения. EDA** 14](#_Toc169202881)

[**Основные шаги выполнения. Построение модели МЛ.** 20](#_Toc169202882)

[**Основные шаги выполнения. Обучение нейронной сети.** 23](#_Toc169202883)

[**Основные шаги выполнения. Интерпретация результатов модели** 28](#_Toc169202884)

[**Основные шаги выполнения. Предсказание на новых данных** 30](#_Toc169202885)

[**Заключение** 31](#_Toc169202886)

[**Источники** 34](#_Toc169202887)

[**Приложение** 35](#_Toc169202888)

## **Основные термины**

1. **Tokenizer** (Токенизатор): Инструмент, разбивающий текст на отдельные токены, то есть на отдельные слова или фразы.
2. **LSTM** (Long Short-Term Memory): Тип рекуррентной нейронной сети, способный запоминать долгосрочные зависимости в последовательных данных.
3. **Токенизация (Tokenization)**: Процесс разделения текста на отдельные единицы, называемые токенами. Токены могут быть словами, числами, знаками препинания и другими элементами текста.
4. **Лемматизация (Lemmatization)**: Процесс приведения словоформы к его нормальной форме (лемме). Например, лемматизация слова "бежал" приведет к форме "бежать".
5. **Удаление стоп-слов (Stop Words Removal)**: Процесс исключения часто встречающихся слов (стоп-слов), которые не несут смысловой нагрузки в контексте задачи обработки текста. Примеры стоп-слов в английском языке: "the", "is", "and".
6. **Рекуррентная нейронная сеть (Recurrent Neural Network, RNN)**: Класс нейронных сетей, способных обрабатывать последовательности данных благодаря своей способности запоминать предыдущие состояния. LSTM (Long Short-Term Memory) является одним из видов рекуррентных нейронных сетей, способным эффективно работать с длинными последовательностями данных.
7. **Эмбеддинги (Embeddings)**: Представления слов или токенов в виде векторов с фиксированной размерностью. Эмбеддинги позволяют модели машинного обучения эффективно работать с текстовыми данными, представляя слова в векторном пространстве.

## **Введение**

**Описание задачи:** Цель нашего исследования заключается в изучении сообщений в каналах Telegram на основе их особенностей. Мы стремимся разработать модель нейронной сети, которая сможет определять, к какому конкретному каналу принадлежит сообщение, используя доступные характеристики текстовых постов.

**Характеристики набора данных:** Датасет состоит из 25 тысяч строки и 2 столбцов.Набор данных содержит информацию о постах в каналах телеграмм, включая следующие характеристики:

* Текст поста
* Группа

**Целевая переменная:** Целевая переменная - id канала, по которой можно определить принадлежность поста к определенному каналу.

**Оценка:** Для оценки качества модели используются следующие метрики:

* Accuracy
* ROC AUC Score
* F1 Score
* Recall Score
* Precision Score
* Матрица ошибок (confusion matrix)

**Задачи, выполненные в работе:**

1. Парсинг данных о постах в Telegram-каналах с использованием TelegramAPI и библиотеки Pyrogram.
2. Первичный анализ данных.
3. Очистка данных и использованием SpaCy и обработка смайликов
4. EDA
5. Подготовка данных:

* Обработка текста
* Токенизация
* Векторизация
* Кодирование категориальных переменных

1. Разделение данных на обучающую и тестовую выборки.
2. Построение модели МЛ.
3. Интерпретация результатов обучения модели МЛ.
4. Построение нейронной сети для определения принадлежности поста к определенному каналу на основе его характеристик.
5. Интерпретация модели и оценка ее качества.
6. Прогнозирование принадлежности постов на новых данных.

## **Основные шаги выполнения. Парсинг данных**

* **Цель**: В данном части целью является сбор текстовых сообщений из указанных каналов Telegram для последующего анализа и создания модели нейронной сети, способной предсказывать принадлежность сообщений к определенным каналам на основе их содержимого.
* **Используемый инструментарий**:
* Для работы с Telegram API использована библиотека Pyrogram.
* Для обработки текстовых данных и создания датафрейма использована библиотека Pandas.
* **Парсер**:
  + Создана функция extract\_messages, которая асинхронно извлекает текстовые сообщения из каналов Telegram. Для каждого канала указывается количество сообщений, которые необходимо извлечь.
  + Для каждого сообщения извлекается текст (message.text) и подпись (message.caption), если они присутствуют, и добавляются в список data вместе с указанием группы (group\_name), к которой относится канал.
  + Обработка исключения FloodWait для управления временными ограничениями Telegram API.
* **Список каналов и групп**:
  + Для каждого канала указаны его имя (name) и группа (group\_name), к которой он относится. Примеры групп: 'avia' (авиация), 'auto' (автомобили).
* **Количество сообщений**:
  + Установлено количество извлекаемых сообщений для каждого канала: message\_count = 6000.
* **Общий процесс выполнения**:
* В основной функции main() асинхронно создается клиент Pyrogram.
* Для каждого канала выполняется запрос на получение информации о канале (channel\_info = await app.get\_chat(channel\_name)) и извлекаются сообщения с помощью функции extract\_messages.
* Извлеченные данные сохраняются в датафрейм Pandas df, где каждая строка содержит текст сообщения и его принадлежность к группе.
* **Результаты**:
  + Собранные данные сохраняются в переменную df и выводятся на экран с помощью print(df) для проверки корректности парсинга и дальнейшего анализа.

Работа парсера показана на Рис (1).





Рис. 1 Парсер каналов.

## **Основные шаги выполнения. Первичный анализ данных.**

Далее был осуществлен вывод датасета, он состоит из 25 тысяч строк и 2 столбов. Рис. (2)

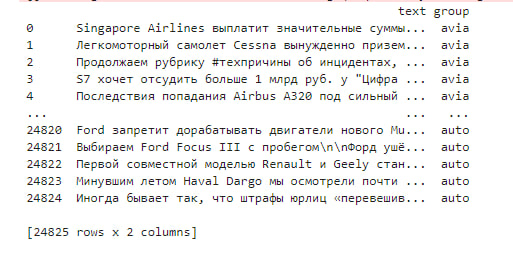


Рис. 2 Датасет до обработки.

Далее было произведено удаление дубликатов в датасете, удаление постов, которые содержат менее 7 элементов. Это позволит убрать посты несущие не информативный текст. Рис (3).

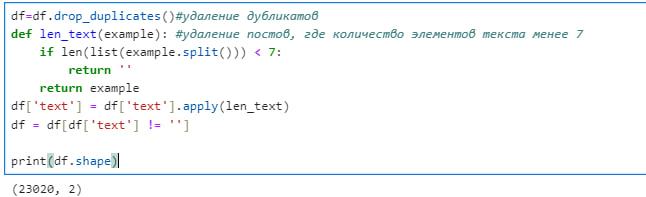


Рис. 3 Удаление дубликатов.

Далее был произведен вывод общей статистики по датасету. Рис. (4)

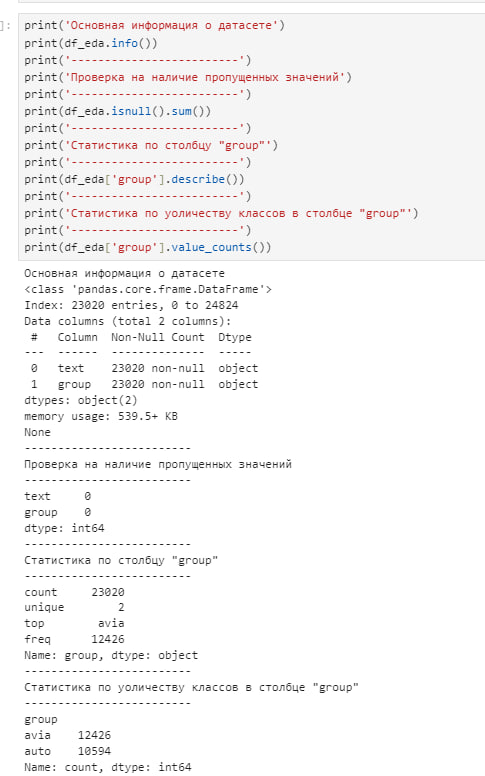


Рис. 4 Статистика по датасету

Далее была реализована загрузка моделей spaCy для русского и английского языков Рис. (5)



Рис. 5 Модели ru/en для обработки.

Функция для обработки текста и применение этой функции к новому столбцу Рис. (5)



Рис. 6 Функция обработки текста.

**Описание работы кода обработки текста:**

1. **Загрузка моделей spaCy**: В начале кода загружаются две модели spaCy для обработки текста на русском и английском языках:
   * nlp\_ru = spacy.load("ru\_core\_news\_sm"): Загружает модель spaCy для русского языка.
   * nlp\_en = spacy.load("en\_core\_web\_sm"): Загружает модель spaCy для английского языка.
2. **Функция translate\_emoji**:
   * Эта функция заменяет смайлики в тексте на их текстовые описания с помощью библиотеки emoji. Например, смайлик 😊 может быть заменен на ":smiling\_face\_with\_smiling\_eyes:".
3. **Функция preprocess\_text**:
   * Принимает на вход текстовую строку text, которую необходимо обработать.
   * Приводит все символы в строке к нижнему регистру с помощью text.lower().
   * Затем вызывает функцию translate\_emoji для замены смайликов на текстовые описания.
   * Удаляет все неалфавитные символы, кроме пробелов и символов подчеркивания, с помощью регулярного выражения re.sub(r'[^а-яА-Яa-zA-Z\s\_]', '', text).
   * Для английского текста (nlp\_en(text)):
     + Выполняет лемматизацию каждого токена (token.lemma\_) и исключает стоп-слова (if not token.is\_stop).
     + Фильтрует только те токены, которые начинаются с букв английского алфавита (re.match(r'[a-zA-Z]', token.text)).
     + Результаты лемматизации сохраняются в список tokens\_en.
   * Для русского текста (nlp\_ru(text)):
     + Проводит аналогичные операции лемматизации и удаления стоп-слов, но для русского языка.
     + Результаты лемматизации сохраняются в список tokens\_ru.
   * Возвращает объединенную строку из всех полученных токенов (tokens\_ru + tokens\_en), преобразованную в одну строку с пробелами между словами с помощью ' '.join().
4. **Применение функции к данным**:
   * Предполагается, что переменная df\_eda содержит DataFrame с колонкой 'text', содержащей текстовые данные, которые необходимо обработать.
   * Функция preprocess\_text применяется к каждой строке столбца 'text' DataFrame'а df\_eda.
   * Результаты обработки помещаются в новый столбец 'Processed\_Text' DataFrame'а df\_eda, содержащий обработанные текстовые данные.

Этот код выполняет комплексную предобработку текстовых данных, включая нормализацию текста, замену смайликов, удаление неалфавитных символов, лемматизацию и удаление стоп-слов для текстов на английском и русском языках. Полученный 'Processed\_Text' может использоваться далее для обучения моделей машинного обучения или для анализа текстовых данных.

Так выглядит датасет после обработки текста, в последнем столбце показано то, как изменился текст и именно этот столбец будет использован

Для дальнейшей обработки и обучения нейросети. Рис (7).

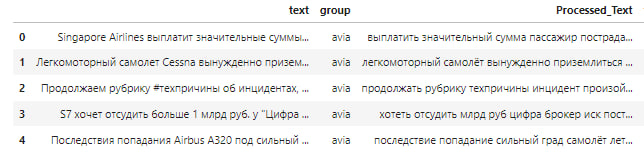


Рис. 7 Датасет после применения функции.

## **Основные шаги выполнения. EDA**

EDA, или Exploratory Data Analysis, представляет собой процесс анализа данных с целью выявления основных характеристик, закономерностей и аномалий в них. Этот этап критически важен перед приступлением к построению моделей машинного обучения и нейронных сетей, особенно в контексте задач NLP.

В качестве аналитики были построены следующие визуализации данных:

1. **Облако слов (Word Cloud)** Рис (8):
   * **Описание графика**: Облако слов визуализирует наиболее часто встречающиеся слова в общем корпусе текстов. Размер и расположение слов зависят от их частоты в текстах.
   * **Значимость**: Этот график помогает быстро оценить общие темы или ключевые слова в данных. Он полезен для первичного анализа, выявления доминирующих тематик и оценки общей направленности текстового материала.

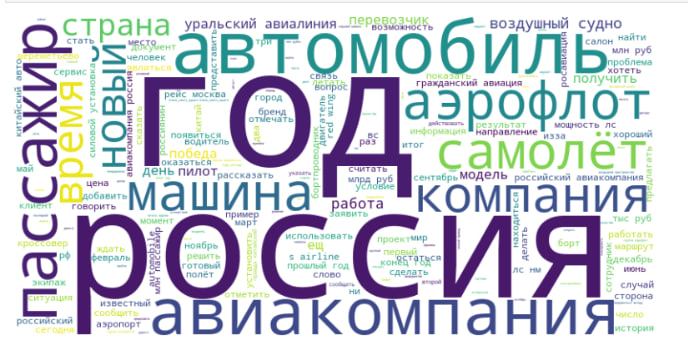


Рис. 8 Облако слов

1. **Частотность слов в каждой группе** Рис (9):
   * **Описание графика**: Для каждой группы текстов строится список наиболее часто встречающихся слов. Выводятся топ-N слов по частоте в каждой группе.
   * **Значимость**: Этот анализ помогает выявить специфические особенности каждой группы текстов. Использование CountVectorizer позволяет численно оценить частоту слов, что полезно для сравнения текстов по ключевым словам и определения их уникальных особенностей.



Рис. 9 Таблица с частотой слов по группам.

1. **Распределение длин текстов по группам** Рис (10):
   * **Описание графика**: Для каждой группы текстов показано распределение длины текстов. Используется гистограмма с возможностью добавления ядерной оценки плотности.
   * **Значимость**: Этот график позволяет сравнить распределение текстовой длины между разными группами. Он может указать на различия в структуре или стиле текстов в каждой группе, что важно для понимания их разнообразия и дальнейшей обработки.

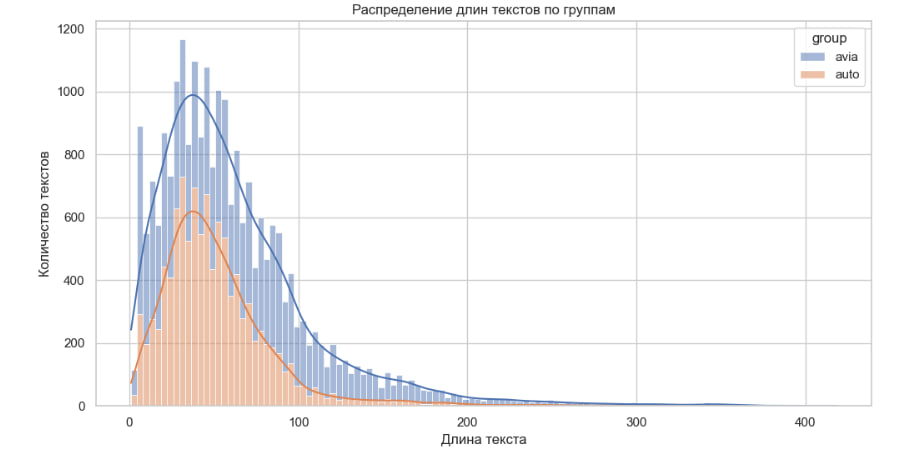


Рис. 10 Распределение длин текстов по группам.

1. **Визуализация частотности слов в каждой группе** Рис (11):
   * **Описание графика**: Для каждой группы текстов показывается столбчатая диаграмма, отображающая наиболее часто встречающиеся слова. Каждая диаграмма посвящена одной из групп текстов.
   * **Значимость**: Этот график предоставляет более детализированное представление о частоте слов в каждой группе. Он помогает сравнить ключевые слова между группами и выявить уникальные слова или темы, которые могут быть характерны для определенной категории текстов.

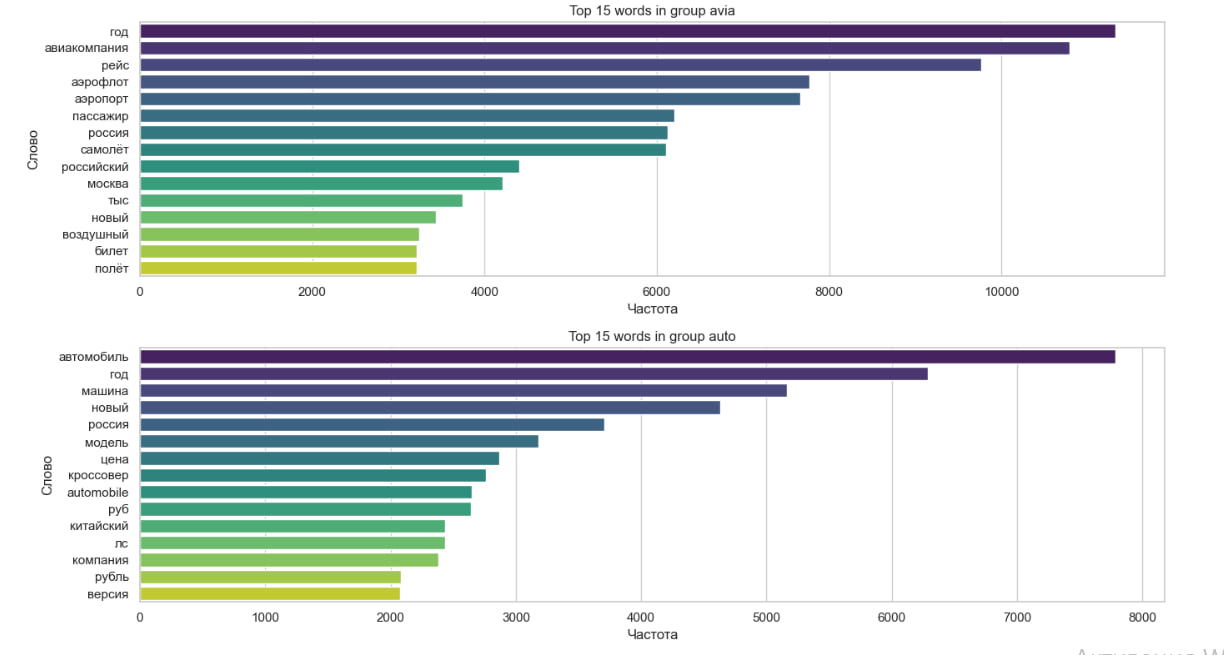


Рис. 11 Частота слов.

Каждый из этих графиков важен для осуществления EDA в задачах NLP, помогая исследователям и аналитикам глубже понять структуру и содержание текстовых данных перед применением более сложных моделей машинного обучения и нейронных сетей.

**Основные шаги выполнения. Разделение на выборки.**

1. **Разделение датасета**:
   * train\_test\_split(data\_ml['Processed\_Text'], data\_ml['group'], test\_size=0.2, random\_state=42): Делит исходные данные на обучающий и тестовый наборы в соотношении 80/20. X\_train и X\_test содержат предобработанные тексты, а y\_train и y\_test — соответствующие им метки групп.
2. **Преобразование меток классов**:
   * label\_encoder = LabelEncoder(): Создает объект LabelEncoder, который кодирует категориальные метки в числовые значения.
   * y\_train\_encoded = label\_encoder.fit\_transform(y\_train.tolist()): Преобразует метки обучающего набора в числовой формат.
   * y\_test\_encoded = label\_encoder.transform(y\_test.tolist()): Применяет ту же самую кодировку к тестовому набору, используя параметры, выученные на обучающем наборе.
3. **Токенизация и преобразование в последовательности**:
   * tokenizer = Tokenizer(num\_words=25000, oov\_token='<OOV>'): Создает объект Tokenizer, который разбивает тексты на слова (токены) и преобразует их в числовые последовательности. Параметр num\_words=25000 указывает максимальное количество слов, которые будут учтены при обучении токенизатора, а oov\_token='<OOV>' задает токен для неизвестных слов.
   * tokenizer.fit\_on\_texts(X\_train.tolist()): Обучает токенизатор на текстах обучающего набора.
   * X\_train\_seq = tokenizer.texts\_to\_sequences(X\_train.tolist()): Преобразует тексты обучающего набора в числовые последовательности на основе обученного токенизатора.
   * X\_test\_seq = tokenizer.texts\_to\_sequences(X\_test.tolist()): Применяет токенизатор к текстам тестового набора.
4. **Подготовка последовательностей к обучению**:
   * max\_len = max(len(seq) for seq in X\_train\_seq): Вычисляет максимальную длину последовательности в обучающем наборе.
   * X\_train\_pad = pad\_sequences(X\_train\_seq, maxlen=max\_len, padding='post'): Выравнивает числовые последовательности обучающего набора до максимальной длины max\_len, дополняя их нулями с конца (padding='post').
   * X\_test\_pad = pad\_sequences(X\_test\_seq, maxlen=max\_len, padding='post'): Аналогично обрабатывает тексты тестового набора.

Этот процесс подготовки данных необходим для того, чтобы модели машинного обучения и нейронные сети могли эффективно работать с текстовыми данными, представленными в числовом формате. Такой подход позволяет сохранить структуру и смысл текстов, упрощает работу с различными длинами текстов и обеспечивает совместимость с алгоритмами обучения.

Все вышеперечисленные обработки показаны на Рис. (12). Также на Рис. (12) показан вывод максимальной длины текста.



Рис. 12 Код с разделением на выборки и обработками.

## **Основные шаги выполнения. Построение модели МЛ.**

**О модели машинного обучения (ML)**

Модель, используемая здесь, это CatBoostClassifier Рис (13) из библиотеки CatBoost, которая представляет собой мощный алгоритм градиентного бустинга, оптимизированный для работы с категориальными признаками. Вот ключевые аспекты этой модели:

* **Параметры модели**:
  + iterations=1000: Количество итераций обучения.
  + learning\_rate=0.1: Скорость обучения, которая регулирует величину шага оптимизации на каждой итерации.
  + depth=6: Глубина деревьев в бустинге.
  + loss\_function='MultiClass': Функция потерь для мультиклассовой классификации.
  + eval\_metric='Accuracy': Метрика оценки качества модели — точность.
  + early\_stopping\_rounds=50: Количество итераций, после которого произойдет остановка обучения, если метрика на валидационном наборе перестает улучшаться.
  + random\_seed=42: Задает случайное начальное состояние для воспроизводимости результатов.
* **Обучение модели**:
  + mode\_cat.fit(X\_train\_pad, y\_train\_encoded, eval\_set=(X\_test\_pad, y\_test\_encoded), verbose=100): Здесь модель обучается на обучающем наборе X\_train\_pad с соответствующими закодированными метками y\_train\_encoded. Оценка производится на валидационном наборе X\_test\_pad и y\_test\_encoded. Параметр verbose=100 указывает на вывод информации о процессе обучения каждые 100 итераций.
* **Оценка качества**:
  + val\_predictions = mode\_cat.predict(X\_test\_pad): После обучения модели делаются предсказания на валидационном наборе.
  + accuracy = accuracy\_score(y\_test\_encoded, val\_predictions): Оценивается точность предсказаний с использованием метрики точности (accuracy\_score).

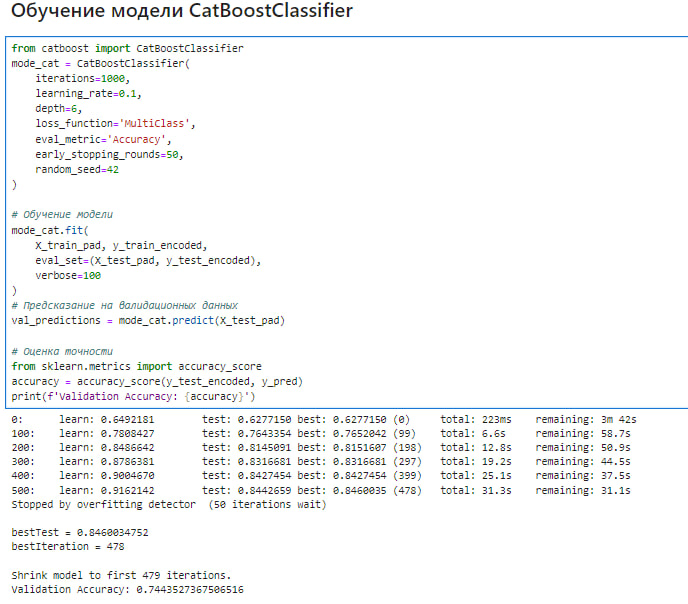


Рис. 13 Модель CatBoostClassifier.

**О визуализации поверхности вероятностей предсказаний модели**

Эта визуализация Рис (14) выполняет следующие шаги:

* **Уменьшение размерности с помощью PCA**:
  + PCA(n\_components=2): Уменьшает размерность данных до двух компонент для возможности их визуализации.
* **Определение сетки для поверхности**:
  + np.linspace и np.meshgrid определяют координатную сетку для построения поверхности.
* **Функция предсказания для сетки**:
  + predict\_mesh(x, y, model, pca): Вычисляет вероятности предсказаний модели mode\_cat на сетке, используя PCA для обратного преобразования данных.
* **Построение поверхности**:
  + plot\_surface строит 3D-поверхность, где оси x и y представляют уменьшенные компоненты PCA, а ось z представляет вероятности предсказаний модели.

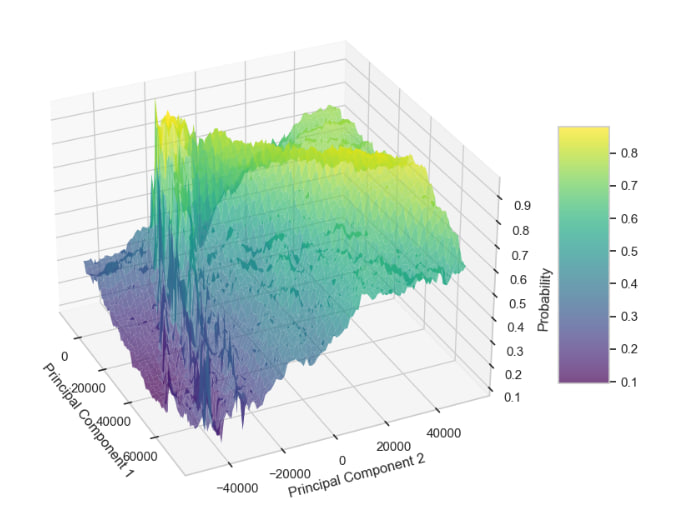


Рис. 14 Визуализация поверхности вероятностей предсказаний модели

**Зачем нужна визуализация?**

Визуализация поверхности вероятностей предсказаний модели помогает в понимании, как модель принимает решения в пространстве признаков. Она позволяет:

* **Визуально оценить структуру данных**: Уменьшение размерности с помощью PCA позволяет увидеть, как данные распределены в двухмерном пространстве.
* **Интерпретировать предсказания модели**: 3D-график показывает, какие области пространства признаков модель считает более вероятными для каждого класса, что может быть полезно для понимания ее работы.

Таким образом, эта визуализация не только делает модель понятной для интерпретации, но и может помочь в дальнейшем улучшении моделирования и ее оптимизации.

## **Основные шаги выполнения. Обучение нейронной сети.**

На Рис (15) показана архитектура и история обучения нейронной сети.

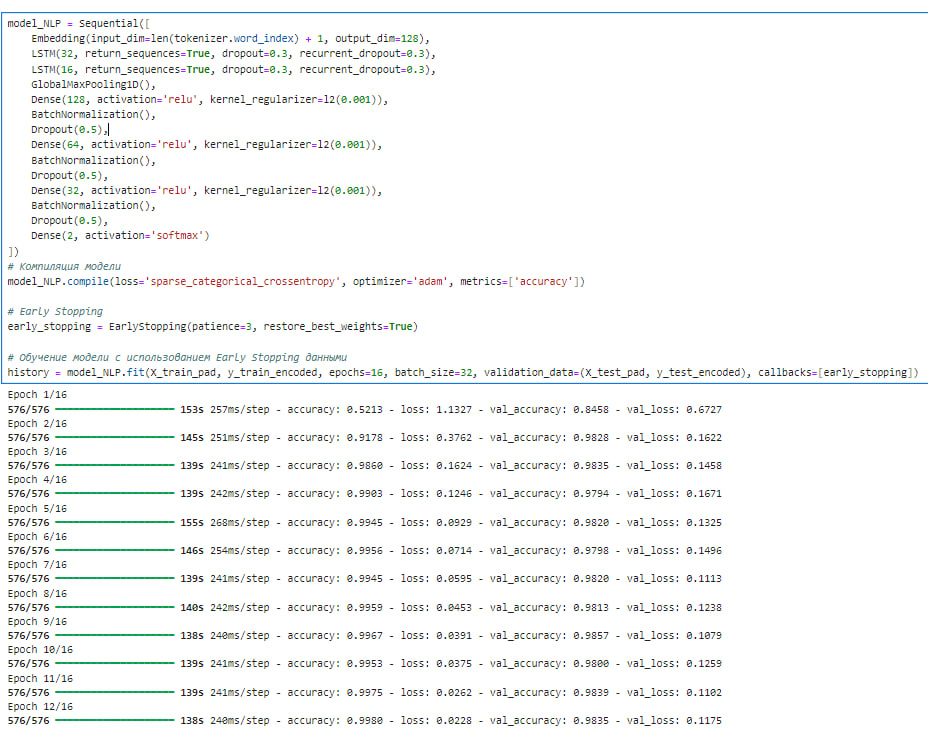


Рис. 15 Архитектура и обучение нейросети.

**Описание слоев**

1. **Embedding Layer**:
   * Embedding(input\_dim=len(tokenizer.word\_index) + 1, output\_dim=128):
     + **Задача**: Преобразует входные слова в плотные векторные представления фиксированной размерности.
     + **Параметры**:
       - input\_dim=len(tokenizer.word\_index) + 1: Размер словаря, плюс один для токена <OOV>.
       - output\_dim=128: Размерность векторного пространства слов.
2. **LSTM Layers**:
   * LSTM(32, return\_sequences=True, dropout=0.3, recurrent\_dropout=0.3):
     + **Задача**: Обрабатывает последовательные данные, запоминая долгосрочные зависимости.
     + **Параметры**:
       - 32: Количество нейронов в слое.
       - return\_sequences=True: Возвращает последовательность на каждом временном шаге.
       - dropout=0.3: Применяет дропаут к входным данным.
       - recurrent\_dropout=0.3: Применяет дропаут к рекуррентным состояниям.
   * Второй LSTM слой аналогичен первому, но с 16 нейронами.
3. **GlobalMaxPooling1D Layer**:
   * GlobalMaxPooling1D():
     + **Задача**: Применяет максимальное субвыборочное объединение по временной оси, уменьшая размерность выходных данных от LSTM слоев.
4. **Dense Layers**:
   * Dense(128, activation='relu', kernel\_regularizer=l2(0.001)):
     + **Задача**: Полносвязный слой с 128 нейронами.
     + **Параметры**:
       - activation='relu': Функция активации ReLU.
       - kernel\_regularizer=l2(0.001): Применяет L2-регуляризацию.
   * BatchNormalization(): Нормализует выходы предыдущего слоя.
   * Dropout(0.5): Применяет дропаут с вероятностью 0.5 для уменьшения переобучения.
   * Два следующих Dense слоя аналогичны, но с 64 и 32 нейронами соответственно.
5. **Output Layer**:
   * Dense(2, activation='softmax'):
     + **Задача**: Полносвязный выходной слой с функцией активации softmax для мультиклассовой классификации на два класса.

* **Компиляция**:
  + loss='sparse\_categorical\_crossentropy': Функция потерь для задачи мультиклассовой классификации.
  + optimizer='adam': Оптимизатор Adam.
  + metrics=['accuracy']: Метрика для оценки точности модели.
* **Early Stopping**:
  + patience=3: Останавливает обучение, если метрика на валидационном наборе не улучшается в течение 3 эпох.
  + restore\_best\_weights=True: Восстанавливает веса модели с наилучшей метрикой.

**Обучение модели**

* **Параметры**:
  + X\_train\_pad, y\_train\_encoded: Обучающие данные и метки.
  + epochs=16: Количество эпох обучения.
  + batch\_size=32: Размер мини-выборки.
  + validation\_data=(X\_test\_pad, y\_test\_encoded): Валидационные данные для оценки на каждой эпохе.
  + callbacks=[early\_stopping]: Использование ранней остановки.

**Значимость слоев и обучения**

Эта архитектура нейронной сети позволяет эффективно обрабатывать последовательные данные, такие как тексты, благодаря LSTM-слоям, которые хорошо работают с временными зависимостями. Эмбеддинги слов помогают модели понимать семантические отношения между словами. Полносвязные слои с нормализацией и дропаутом помогают избежать переобучения и улучшить обобщающую способность модели. Включение ранней остановки дополнительно предотвращает переобучение, сохраняя наилучшие веса модели.

Такой подход обеспечивает высокую производительность и устойчивость при работе с текстовыми данными в задачах классификации.

## **Основные шаги выполнения. Интерпретация результатов модели**

В качестве интерпретации обучения нейронной сети были построены 2 визуализации, показывающие значения функции потерь и точности на обучающем и валидационном наборах. Рис (16).

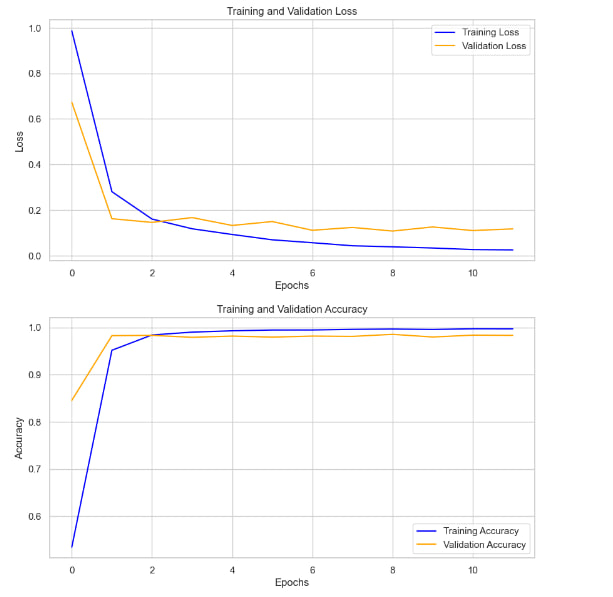


Рис. 16 Графики результатов обучения.

Далее была построена матрица ошибок, показанная на Рис. (17)

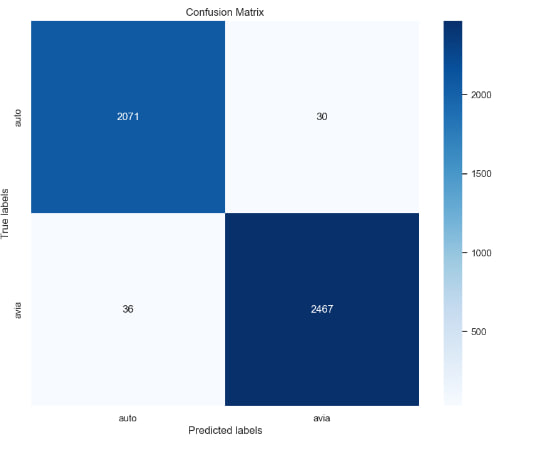


Рис. 17 Матрица ошибок

Матрица ошибок позволяет оценить, какие классы модель склонна путать между собой, и помогает выявить проблемы в качестве классификации.

## **Основные шаги выполнения. Предсказание на новых данных**

Для предсказания категории новых текстов была разработана функция Рис (18). На этом рисунке видно, что нейронная сеть правильно определила тематику поста и отнесла его к группе ‘auto’.

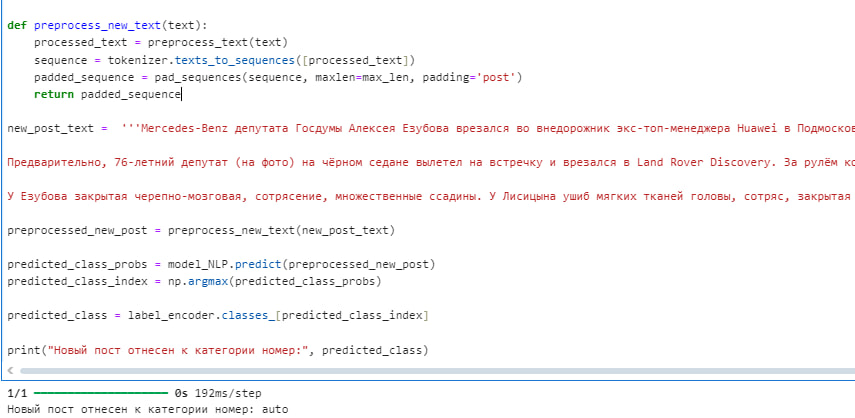


Рис. 18 Предсказание на новых данных.

Предобработка нового текста

1. **Функция предобработки**:
   * Очистка и нормализация текста.
   * Токенизация текста.
   * Дополнение последовательности до фиксированной длины.
2. **Применение функции предобработки** к новому тексту, чтобы подготовить его для модели.

Предсказание категории

1. **Получение вероятностей классов** для нового текста с помощью модели нейронной сети.
2. **Определение класса с наибольшей вероятностью**.
3. **Преобразование индекса в название класса** и вывод результата.

Объяснение

* **Предобработка**: Текст очищается, нормализуется и преобразуется в числовую последовательность, дополненную до максимальной длины.
* **Предсказание**: Модель предсказывает вероятности для каждого класса.
* **Интерпретация**: Находится категория с наибольшей вероятностью, и результат выводится.

Этот процесс позволяет классифицировать новые тексты, следуя аналогичным шагам предобработки и преобразования, что и для обучающих данных.

## **Заключение**

**Выводы**

В ходе нашего исследования мы успешно достигли поставленных целей, а именно разработали модель нейронной сети, которая способна определять принадлежность сообщения к конкретному Telegram-каналу, используя текстовые характеристики постов. Процесс включал множество этапов: от парсинга данных и первичного анализа до построения моделей машинного обучения и нейронной сети.

**Достигнутые цели**

1. **Парсинг данных**: С использованием Telegram API и библиотеки Pyrogram мы собрали и структурировали данные о постах из различных Telegram-каналов.
2. **Первичный анализ и очистка данных**: Мы провели EDA (разведочный анализ данных), очистили текстовые данные и обработали смайлики, используя SpaCy.
3. **Подготовка данных**: Проведена токенизация, векторизация и кодирование категориальных переменных.
4. **Моделирование**: Мы разделили данные на обучающую и тестовую выборки, построили модель машинного обучения с использованием CatBoostClassifier и нейронную сеть на базе LSTM.
5. **Оценка моделей**: Для оценки качества моделей использованы метрики Accuracy, ROC AUC Score, F1 Score, Recall Score, Precision Score и матрица ошибок.
6. **Предсказание на новых данных**: Модели успешно использованы для предсказания категории новых текстовых постов.

**Полезность работы**

Результаты нашего исследования имеют значительную практическую ценность. Разработанные модели могут быть применены для автоматической категоризации текстов, что полезно для автоматизации работы с контентом в Telegram-каналах. Это может включать фильтрацию, сортировку и анализ постов, что значительно улучшает эффективность управления информацией.

**Будущие планы**

1. **Улучшение модели**: Планируется продолжить работу над улучшением моделей, включая использование более сложных архитектур нейронных сетей и оптимизацию гиперпараметров.
2. **Расширение набора данных**: Включение большего количества данных из различных источников для повышения обобщающей способности модели.
3. **Интеграция с реальными системами**: Разработка приложений, использующих модель для реального времени анализа и категоризации постов.
4. **Исследование дополнительных характеристик**: Изучение влияния дополнительных характеристик, таких как метаданные постов, на качество классификации.

Таким образом, наше исследование не только достигло своих первичных целей, но и заложило основу для дальнейших улучшений и применения разработанных моделей в реальных сценариях.

## **Источники**

Исходные данные датафрэймов использовались из открытых каналов телеграмм, id групп были указаны в пункте ‘Парсинг данных’.

Версии библиотек указаны в файле requirements.txt на github.

## **Приложение**

Ссылка на github, с выложенным проектом: <https://github.com/Ho1ms/nlp-course-work>