Задача

Задача заключается в создании модели машинного обучения для классификации текстов на основе двух классов:

1. **Sarcasm (сарказм)** – заголовки, которые содержат саркастический оттенок.
2. **Not Sarcasm (не сарказм)** – заголовки, которые передают информацию без сарказма.

Основные цели:

- Изучить особенности текста в саркастических и несаркастических заголовках.
- Построить эффективную модель, способную классифицировать заголовки с высокой точностью.
- Оценить различные подходы к обработке текстов и выбору моделей.

Датасет

Импорт необходимых библиотек

```
import pandas as pd
import numpy as np
import re
import spacy
import time

import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.feature_extraction.text import CountVectorizer
from wordcloud import WordCloud

from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, classification_report
from scipy.sparse import hstack
from textblob import TextBlob
import torch
from transformers import DistilBertTokenizer, DistilBertForSequenceClassification, Traine
from keras.models import Model
from keras.layers import Input, Embedding, LSTM, Dense, Dropout, Concatenate
from keras.callbacks import EarlyStopping
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
```

```
from keras.utils import to_categorical
from keras.optimizers import Adam
```

```
!wget http://nlp.stanford.edu/data/glove.6B.zip
!unzip glove.6B.zip
glove_path = 'glove.6B.100d.txt'
```

Показать скрытые выходные данные

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.m
```

Этап 1: Анализ данных

Загрузка данных

```
data = pd.read_json('/content/drive/MyDrive/Sarcasm_Headlines_Dataset_v2.json', lines=Tru
```

Предварительный анализ

```
print(f"Размер датасета: {data.shape}")
print(data.head())
```

```
Размер датасета: (28619, 3)
   is_sarcastic                                           headline  \
0             1  thirtysomething scientists unveil doomsday clo...
1             0  dem rep. totally nails why congress is falling...
2             0  eat your veggies: 9 deliciously different recipes
3             1  inclement weather prevents liar from getting t...
4             1  mother comes pretty close to using word 'strea...

                                        article_link
0  https://www.theonion.com/thirtysomething-scien...
1  https://www.huffingtonpost.com/entry/donna-edw...
2  https://www.huffingtonpost.com/entry/eat-your-...
3  https://local.theonion.com/inclement-weather-p...
4  https://www.theonion.com/mother-comes-pretty-c...
```

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 28619 entries, 0 to 28618
Data columns (total 3 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
```

```
 0   is_sarcastic  28619 non-null  int64
 1   headline      28619 non-null  object
 2   article_link  28619 non-null  object
dtypes: int64(1), object(2)
memory usage: 670.9+ KB
```

```
data.describe()
```

	is_sarcastic
count	28619.000000
mean	0.476397
std	0.499451
min	0.000000
25%	0.000000
50%	0.000000
75%	1.000000
max	1.000000

```
data.isnull().sum()
```

	0
is_sarcastic	0
headline	0
article_link	0

dtype: int64

```
data.duplicated().sum()
```

2

Размер датасета:

- Датасет содержит **28,619 строк** и **3 столбца**.

Описание колонок:

- **is_sarcastic** (int64): Бинарный столбец, указывающий, является ли заголовок саркастическим (1) или нет (0).
- **headline** (object): Текст заголовка, который используется для анализа.
- **article_link** (object): Ссылка на статью (можно удалить).

Пустые значения:

- В датасете **нет пропусков** – все строки полностью заполнены.

Дубликаты:

- Обнаружено **2 дубликата**. Их необходимо удалить, чтобы избежать смещения модели.

Распределение памяти:

- Датасет занимает **670.9 KB** оперативной памяти, что позволяет легко обрабатывать его в локальной среде или в облачных решениях.

Следующие шаги:

- Удалить дубликаты.
- Провести EDA (Exploratory Data Analysis), чтобы визуализировать распределение классов и текстовых данных.
- Очистить данные от лишних столбцов, если это необходимо.

Удаление дубликатов

```
data_cleaned = data.drop_duplicates(subset=['headline']).reset_index(drop=True)
```

Визуализация облака слов

```
# Функция для подсчета топ-слов
def get_top_words_fixed(text_series, top_n=10):
    vectorizer = CountVectorizer(max_features=top_n, stop_words='english')
    matrix = vectorizer.fit_transform(text_series)
    return zip(vectorizer.get_feature_names_out(), np.asarray(matrix.sum(axis=0)).flatten

# Топ-слова для саркастических и несаркастических заголовков
sarcastic_keywords = dict(get_top_words_fixed(data_cleaned[data_cleaned['is_sarcastic'] =
non_sarcastic_keywords = dict(get_top_words_fixed(data_cleaned[data_cleaned['is_sarcastic

# Визуализация облака слов
def plot_wordcloud_from_dict(word_dict, title):
    wordcloud = WordCloud(width=800, height=400, background_color="white").generate_from_
    plt.figure(figsize=(10, 5))
    plt.imshow(wordcloud, interpolation='bilinear')
    plt.title(title, fontsize=14)
    plt.axis('off')
    plt.show()

plot_wordcloud_from_dict(sarcastic_keywords, "Облако слов для саркастических заголовков")
plot_wordcloud_from_dict(non_sarcastic_keywords, "Облако слов для несаркастических заголо
```

Саркастические заголовки:

- **Ключевые слова:** man, new, report, woman, trump, nation, just.
- **Тематика:** Часто связаны с социальной и политической сферой.

Несаркастические заголовки

- **Ключевые слова:** trump, donald, says, new, women, world, people.

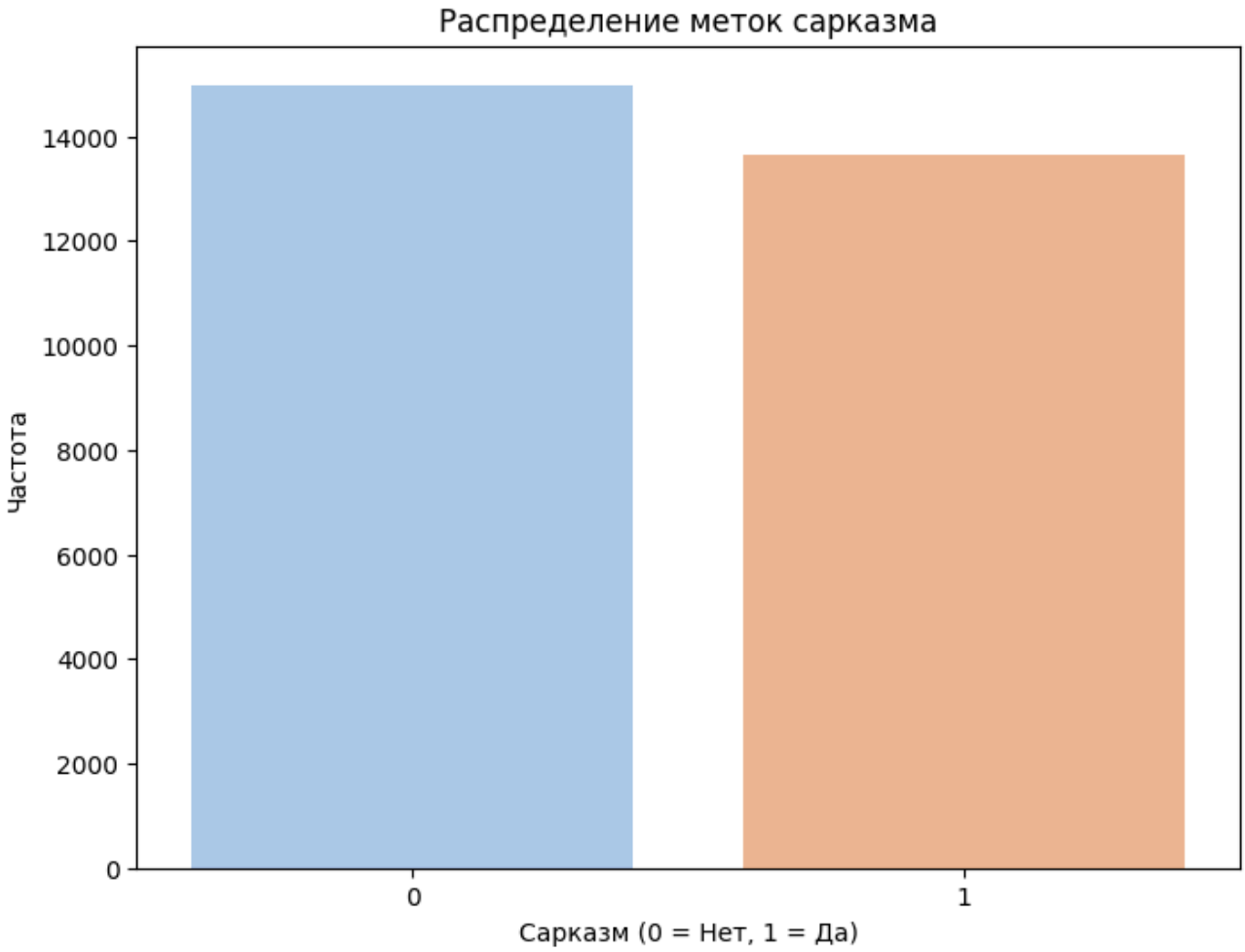
- **Тематика:** Основной акцент на фактах и событиях, часто политических и социальных.

Сравнение

- **Общие слова:** trump, new, man, year встречаются в обоих типах, но контекст использования различается.
- **Различия:**
    - В саркастических заголовках больше многозначности, юмора и акцента на иронии.
    - Несаркастические заголовки передают факты, часто через прямую речь (says, donald).

Распределение классов

```
plt.figure(figsize=(8, 6))
sns.countplot(x='is_sarcastic', data=data, hue='is_sarcastic', palette='pastel', legend=F
plt.title("Распределение меток сарказма")
plt.xlabel("Сарказм (0 = Нет, 1 = Да)")
plt.ylabel("Частота")
plt.show()
```



Баланс классов:

- Данные сбалансированы.

Частоты:

- Метка "0" (не сарказм) встречается чуть чаще, чем метка "1" (сарказм).
- Примерное распределение: около **15,000** для класса "0" и **13,000** для класса "1".

Создание новых признаков

Список фичей

1. Базовые фичи: **Длина заголовка (headline_length)**:

- Количество символов в заголовке.
- В среднем саркастические заголовки длиннее.

Количество слов (word_count):

- Количество слов в заголовке.

- Также отражает длину текста.

Средняя длина слов (avg_word_length):

- Среднее количество символов в словах заголовка.

2. Лексические фичи

Частотность топ-слов:

- Частота появления наиболее часто встречающихся слов.
- Проблема: Пересечение ключевых слов между классами (man, trump, new) снижает уникальность.

3. Синтаксические фичи

Наличие восклицательных знаков (exclamation_count):

- Количество знаков "!" в заголовке.
- Синтаксический индикатор эмоционального окраса.

Наличие вопросительных знаков (question_count):

- Количество знаков "?" в заголовке.
- Указывает на характер подачи текста, может быть связано с сарказмом.

4. Семантические фичи:

TF-IDF векторизация:

- Вычисляет вес слов в контексте всего датасета, выделяя значимые слова.
- Отличает значимые слова для каждого класса.

Word embeddings (например, GloVe, Word2Vec):

- Представление слов как векторов в многомерном пространстве.
- Учитывает скрытые семантические связи между словами.

```
data_cleaned['headline_length'] = data_cleaned['headline'].apply(len)
data_cleaned['word_count'] = data_cleaned['headline'].apply(lambda x: len(x.split()))
data_cleaned['avg_word_length'] = data_cleaned['headline'].apply(lambda x: np.mean([len(w
data_cleaned['exclamation_count'] = data_cleaned['headline'].apply(lambda x: x.count('!')
data_cleaned['question_count'] = data_cleaned['headline'].apply(lambda x: x.count('?'))
```

Распределение длины заголовков

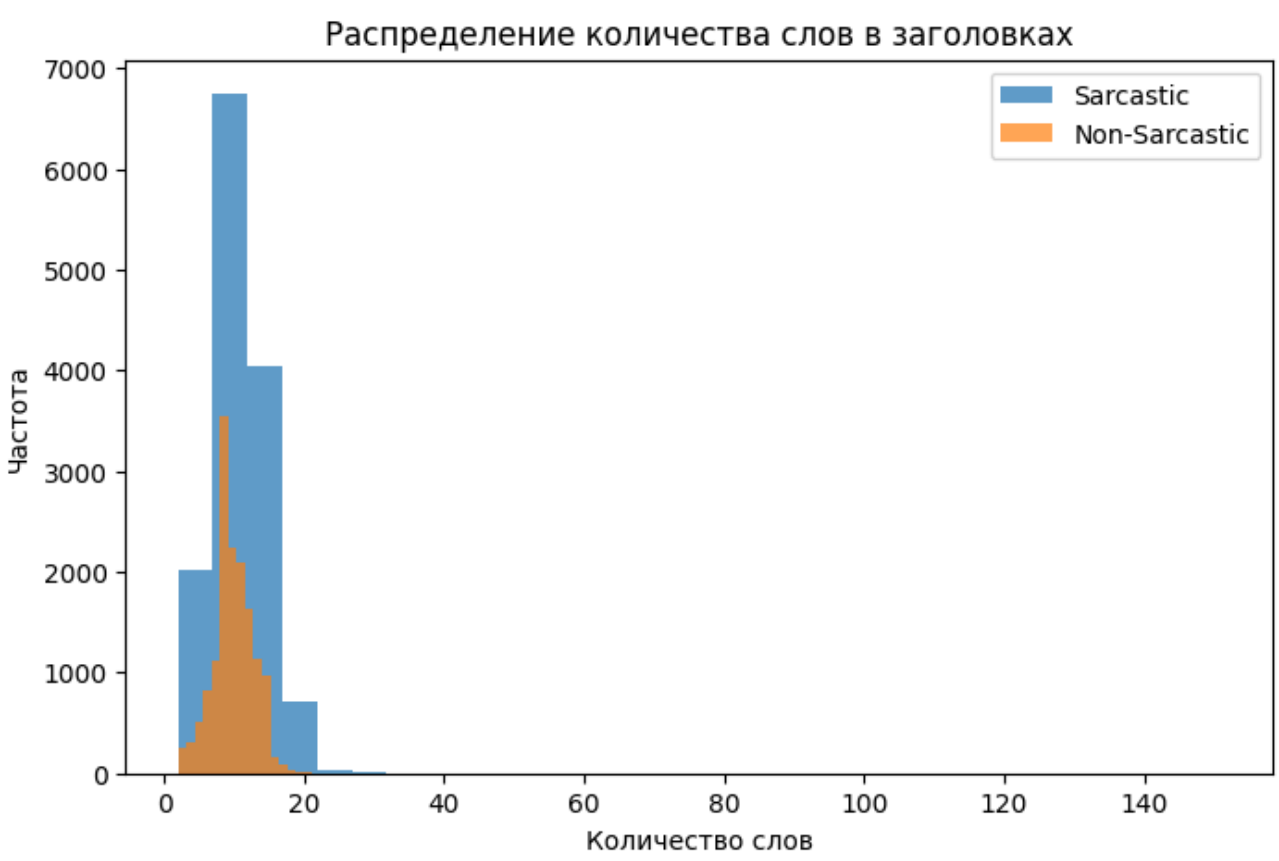
```
plt.figure(figsize=(8, 5))
plt.hist(data_cleaned[data_cleaned['is_sarcastic'] == 1]['headline_length'], bins=30, alp
plt.hist(data_cleaned[data_cleaned['is_sarcastic'] == 0]['headline_length'], bins=30, alp
plt.xlabel("Длина заголовка")
plt.ylabel("Частота")
plt.title("Распределение длины заголовков")
plt.legend()
```

```
plt.show()
```

Показать скрытые выходные данные

Распределение количества слов

```
plt.figure(figsize=(8, 5))
plt.hist(data_cleaned[data_cleaned['is_sarcastic'] == 1]['word_count'], bins=30, alpha=0.
plt.hist(data_cleaned[data_cleaned['is_sarcastic'] == 0]['word_count'], bins=30, alpha=0.
plt.xlabel("Количество слов")
plt.ylabel("Частота")
plt.title("Распределение количества слов в заголовках")
plt.legend()
plt.show()
```



Длина заголовков:

- Большинство заголовков имеют от **5 до 20 слов**. Очень длинные заголовки (свыше 40 слов) редки.

Сравнение классов:

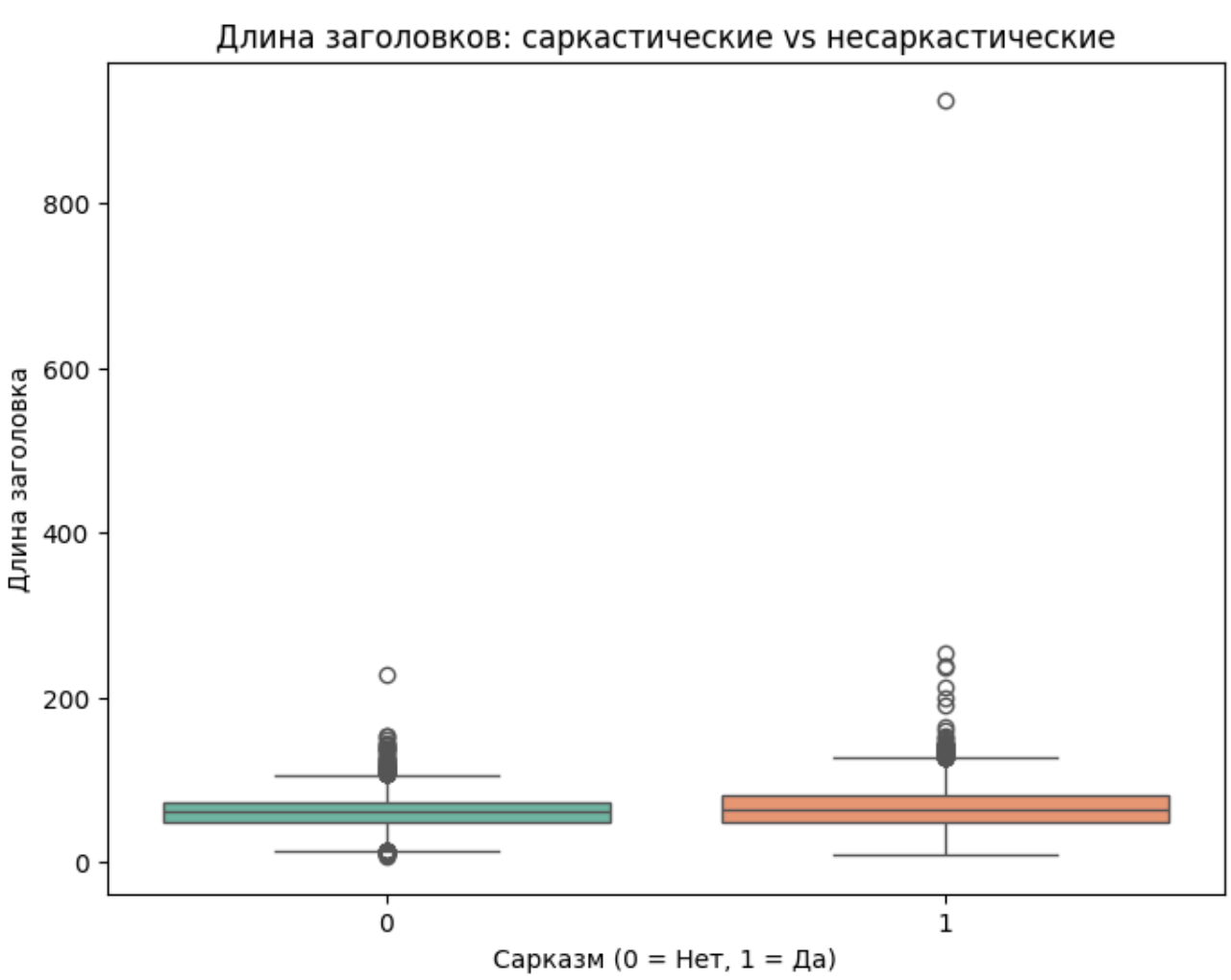
- Распределения длины схожи для саркастических и несаркастических заголовков.

Рекомендация:

- Ограничить длину заголовков до **30 слов** для обработки, так как длинные заголовки малочисленны.

Ящиковая диаграмма для длины заголовков по классам сарказма

```
plt.figure(figsize=(8, 6))
sns.boxplot(x=data_cleaned['is_sarcastic'], y=data_cleaned['headline_length'], hue=data_c
plt.title("Длина заголовков: саркастические vs несаркастические")
plt.xlabel("Сарказм (0 = Нет, 1 = Да)")
plt.ylabel("Длина заголовка")
plt.show()
```



Длина заголовков:

- Большинство заголовков для обоих классов имеют схожую длину, находящуюся в пределах **5–20 слов**.
- Наблюдаются редкие выбросы с длиной свыше **200 слов**, которые требуют внимания.

Сравнение классов:

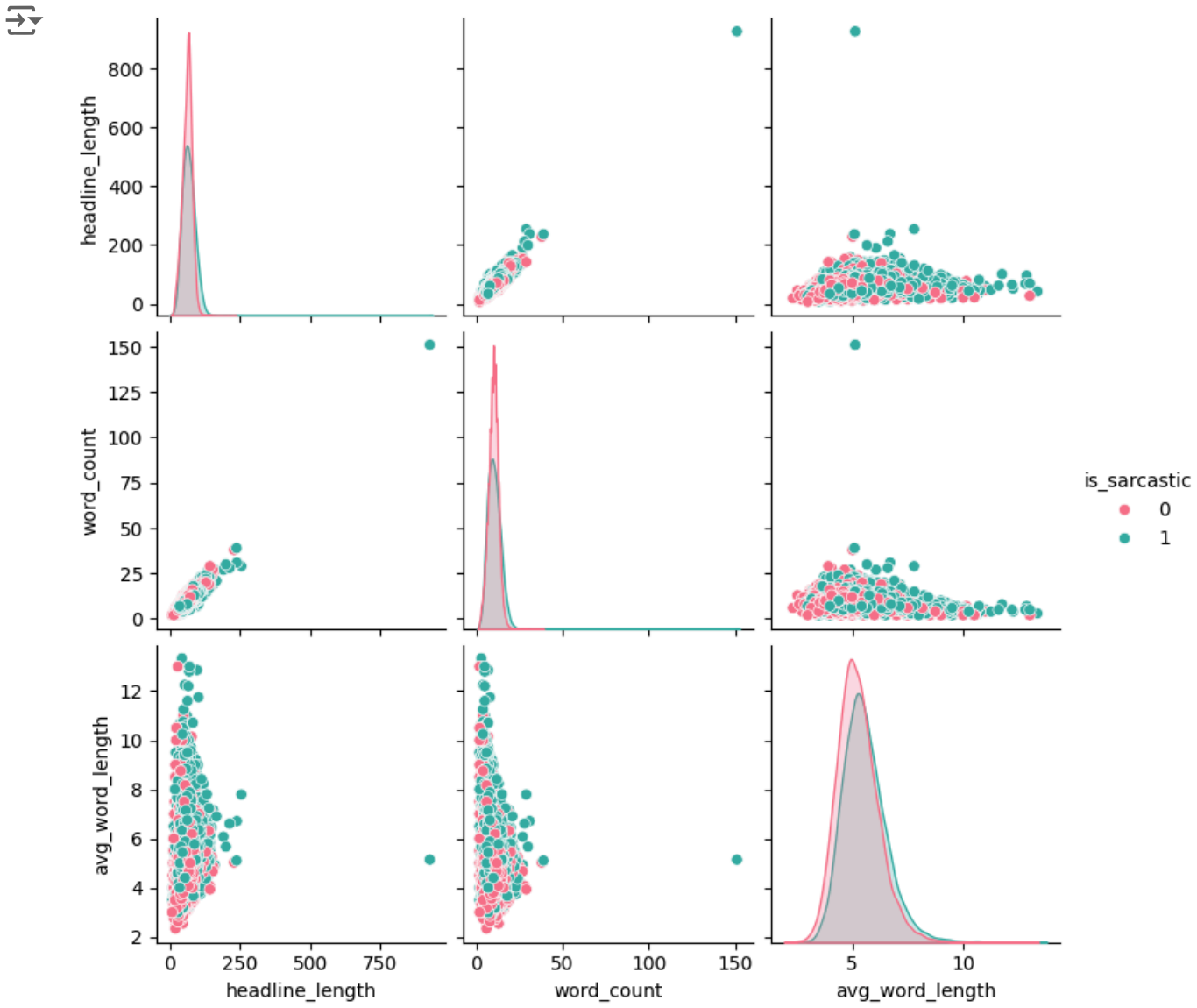
- Между саркастическими и несаркастическими заголовками значимых различий в длине не выявлено.

Рекомендация:

- Удалить выбросы с длиной заголовков >200 слов для повышения качества данных.

```
data_cleaned = data.drop_duplicates(subset=['headline']).reset_index(drop=True)
data_cleaned['headline_length'] = data_cleaned['headline'].apply(len)
data_cleaned['word_count'] = data_cleaned['headline'].apply(lambda x: len(x.split()))
data_cleaned['avg_word_length'] = data_cleaned['headline'].apply(lambda x: np.mean([len(w

selected_features = ['headline_length', 'word_count', 'avg_word_length', 'is_sarcastic']
sns.pairplot(data_cleaned[selected_features], hue='is_sarcastic', palette='husl')
plt.show()
```



Общая картина:

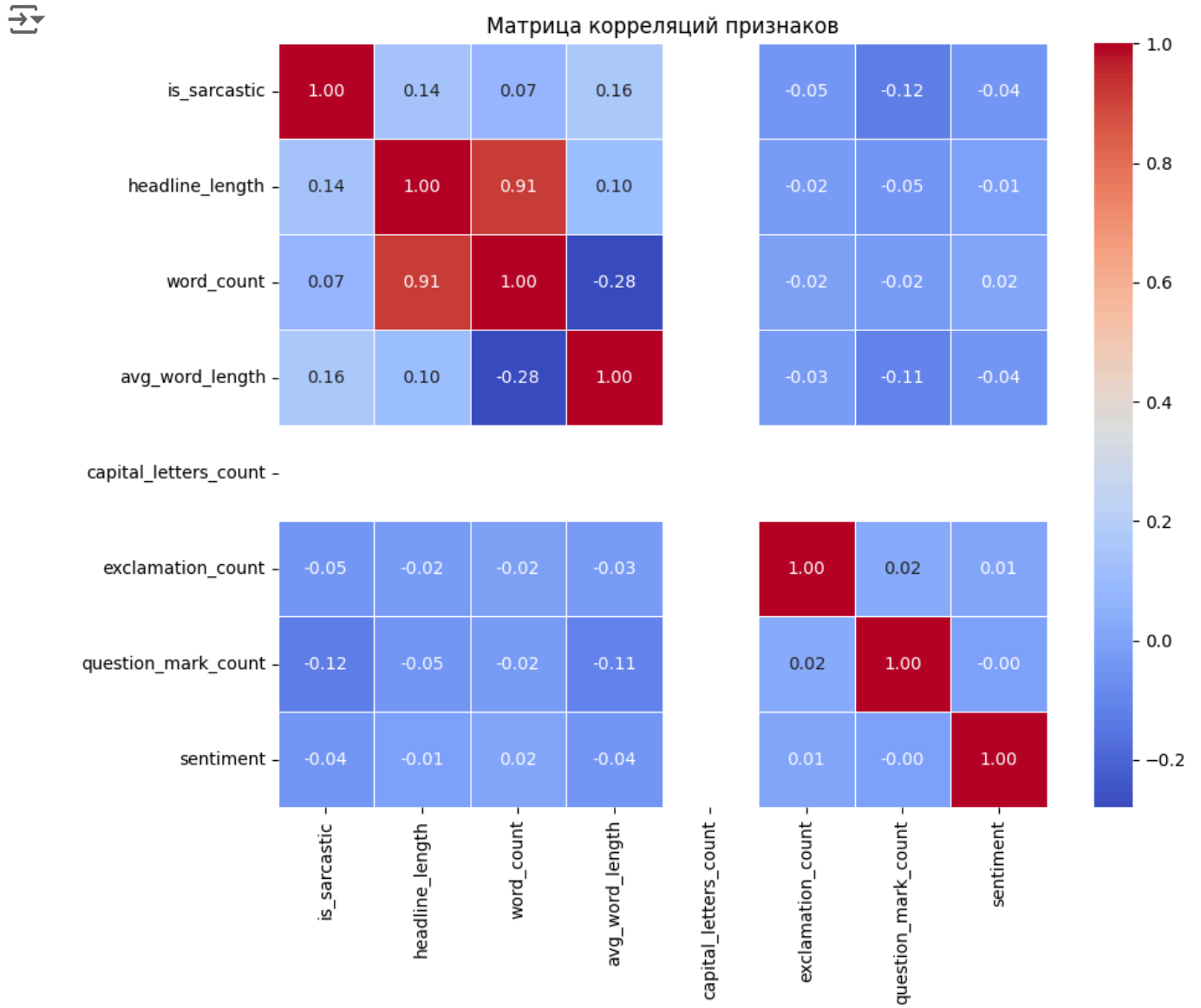
- **Количество слов (word_count)**: Большинство заголовков содержат 5–20 слов, выбросы доходят до 150 слов.
- **Средняя длина слова (avg_word_length)**: Распределение схоже для обоих классов, среднее значение находится в диапазоне 4–8 символов.

Сравнение классов:

- Классы "0" и "1" имеют схожие распределения по всем параметрам.
- Никакой из показателей (длина заголовка, количество слов, средняя длина слова) не является явным разделяющим фактором между классами.

```
# Тепловая карта корреляций
correlation_matrix = data[['is_sarcastic', 'headline_length', 'word_count', 'avg_word_len
                           'capital_letters_count', 'exclamation_count', 'question_mark_c

plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt='.2f', linewidths=0.5)
plt.title("Матрица корреляций признаков")
plt.show()
```



Связь признаков с меткой is_sarcastic:

Самые сильные корреляции наблюдаются у:

- **avg_word_length (0.16)**: Средняя длина слова имеет слабую положительную связь с сарказмом.
- **headline_length (0.14)**: Длина заголовка также имеет слабую положительную связь с сарказмом.

Взаимосвязь между признаками:

- **headline_length и word_count (0.91):** Сильная корреляция ожидаема, так как оба параметра отражают объем текста.
- Остальные признаки имеют слабую взаимосвязь, что хорошо для моделей, так как минимизируется избыточность данных.

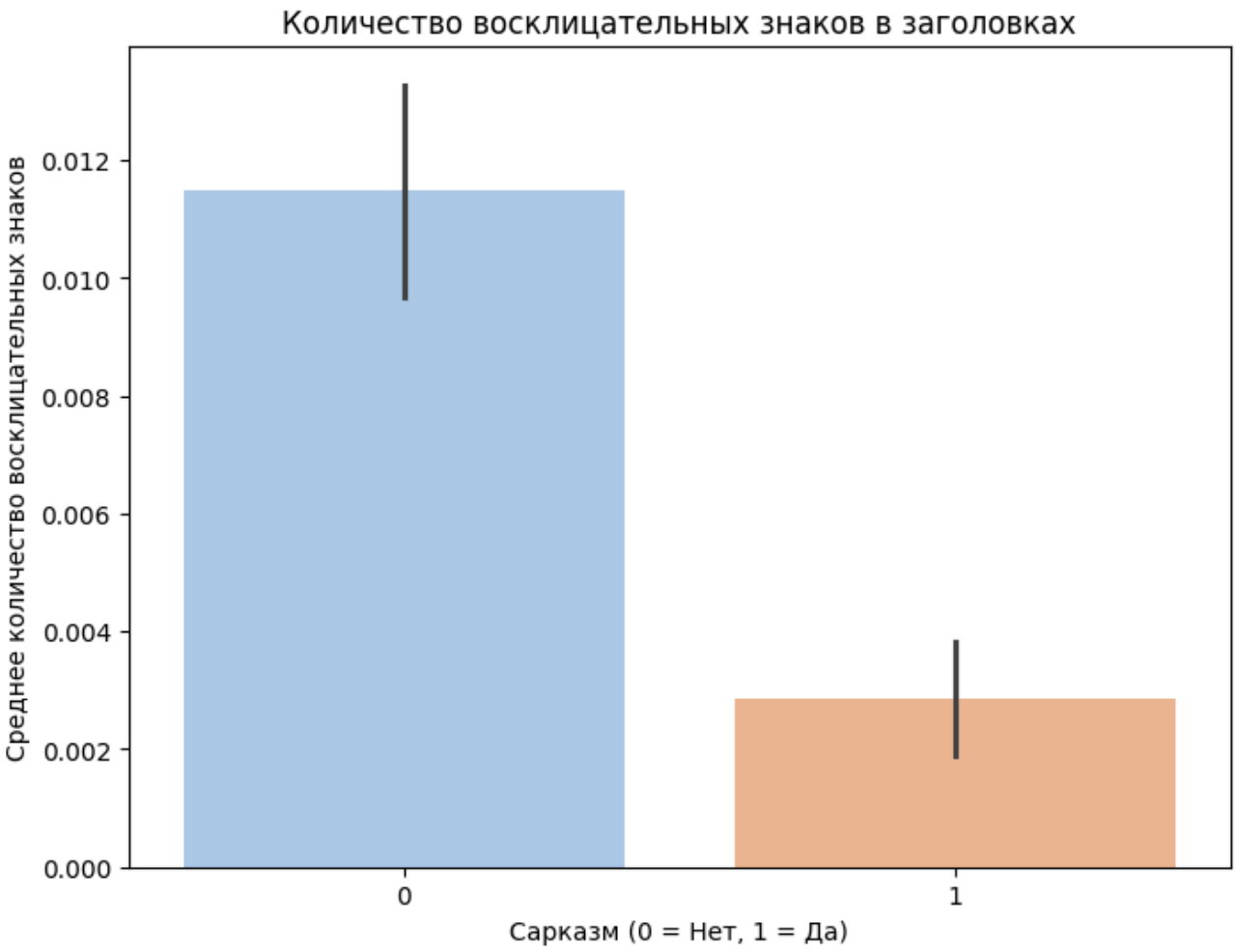
Отрицательные корреляции:

- Количество **вопросительных знаков** и **восклицательных знаков** слабо связано с сарказмом, что может быть полезным для исключения нерелевантных признаков.

Рекомендации:

- Использовать все признаки, кроме **headline_length**, так как они могут внести различную ценность в модели.
- Учитывать слабую связь признаков с **is_sarcastic** при построении более сложных моделей и выборе стратегии оптимизации.

```
# Количество восклицательных знаков
plt.figure(figsize=(8, 6))
sns.barplot(x='is_sarcastic', y='exclamation_count', data=data, hue='is_sarcastic', palet
plt.title("Количество восклицательных знаков в заголовках")
plt.xlabel("Сарказм (0 = Нет, 1 = Да)")
plt.ylabel("Среднее количество восклицательных знаков")
plt.show()
```



Использование восклицательных знаков:

- В несаркастических заголовках среднее количество восклицательных знаков значительно выше, чем в саркастических.

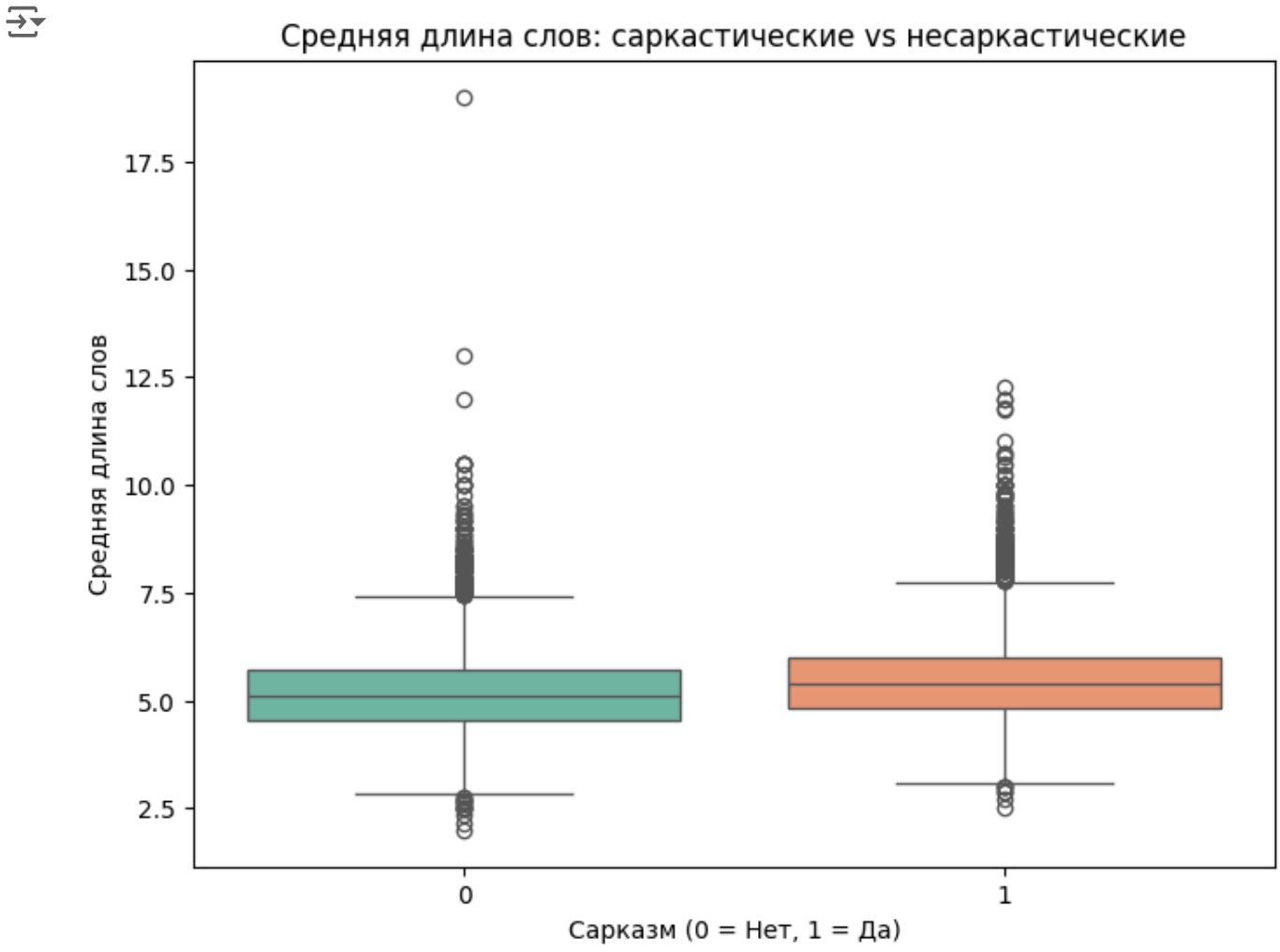
Сравнение:

- Саркастические заголовки редко содержат восклицательные знаки, что может быть особенностью стиля сарказма, стремящегося к тонкой иронии без чрезмерной экспрессии.

Рекомендация:

- Количество восклицательных знаков можно использовать как один из признаков для классификации, так как оно показывает различия между классами.

```
plt.figure(figsize=(8, 6))
sns.boxplot(x=data['is_sarcastic'], y=data['avg_word_length'], hue=data['is_sarcastic'],
plt.title("Средняя длина слов: саркастические vs несаркастические")
plt.xlabel("Сарказм (0 = Нет, 1 = Да)")
plt.ylabel("Средняя длина слов")
plt.show()
```



Средняя длина слов:

- В саркастических и несаркастических заголовках средняя длина слов практически одинакова, около **5–6 символов**.
- Наблюдаются выбросы, где слова значительно длиннее, но их не много.

Сравнение:

- Различия между классами минимальны, что говорит о слабом влиянии средней длины слова на принадлежность к классу.

Рекомендация:

- Среднюю длину слова можно учитывать как дополнительный признак, но она вряд ли будет значимым фактором для классификации. Выбросы с аномальной длиной слов можно исключить.

```
# Функции для предобработки данных
def clean_text(text):
    """Очистка текста: удаление пунктуации, цифр, перевод в нижний регистр."""
    text = text.lower()
    text = re.sub(r'[^a-z\s]', '', text)
```

```
    return text

def compute_features(df, text_column, raw_text_column):
    """Вычисление числовых признаков."""
    df['unique_word_count'] = df[text_column].apply(lambda x: len(set(x.split())))
    df['avg_word_length'] = df[text_column].apply(
        lambda x: np.mean([len(word) for word in x.split()]) if len(x.split()) > 0 else 0
    )
    df['capital_letters_count'] = df[raw_text_column].apply(lambda x: sum(1 for char in x
    df['exclamation_count'] = df[raw_text_column].apply(lambda x: x.count('!'))
    df['question_mark_count'] = df[raw_text_column].apply(lambda x: x.count('?'))
    df['uppercase_ratio'] = df[raw_text_column].apply(
        lambda x: sum(1 for char in x if char.isupper()) / len(x) if len(x) > 0 else 0
    )
    df['sentiment'] = df[raw_text_column].apply(lambda x: TextBlob(x).sentiment.polarity)
    return df

def preprocess_data(df, text_column, raw_text_column):
    """Полная предобработка данных."""
    df['cleaned_text'] = df[text_column].apply(clean_text)
    df = compute_features(df, 'cleaned_text', raw_text_column)
    return df
```

```
data = preprocess_data(data, 'headline', 'headline')
```

TF-IDF и числовые признаки

```
vectorizer = TfidfVectorizer(max_features=5000, ngram_range=(1, 3))
X_text = vectorizer.fit_transform(data['cleaned_text'])

X_features = data[['unique_word_count', 'avg_word_length', 'capital_letters_count',
                   'exclamation_count', 'question_mark_count', 'uppercase_ratio', 'sentim

X_combined = hstack([X_text, X_features])
y = data['is_sarcastic']
```

GloVe эмбеддинги

```
glove_path = 'glove.6B.100d.txt'
embedding_dim = 100
embeddings_index = {}

with open(glove_path, 'r', encoding='utf-8') as file:
    for line in file:
        values = line.split()
        word = values[0]
        vector = np.asarray(values[1:], dtype='float32')
        embeddings_index[word] = vector
```

```
def compute_mean_embedding(text, embeddings_index, embedding_dim):
    words = text.split()
    embedding = [embeddings_index.get(word, np.zeros(embedding_dim)) for word in words]
    return np.mean(embedding, axis=0) if embedding else np.zeros(embedding_dim)

data['mean_embedding'] = data['cleaned_text'].apply(lambda x: compute_mean_embedding(x, e
X_embeddings = np.vstack(data['mean_embedding'].values)
X_combined = hstack([X_text, X_embeddings, X_features])
```

Разделение данных

```
X_train, X_test, y_train, y_test = train_test_split(X_combined, y, test_size=0.2, random_
```

Этап 2: Тестирование моделей

Логистическая регрессия

```
# Обучение модели логистической регрессии
log_reg = LogisticRegression(max_iter=1000, random_state=42)
log_reg.fit(X_train, y_train)

# Время предсказания
start_time = time.time()
y_pred = log_reg.predict(X_test)
log_reg_prediction_time = time.time() - start_time

# Метрики
log_reg_accuracy = accuracy_score(y_test, y_pred)
print("\n--- Logistic Regression Results ---")
print(f"Accuracy: {log_reg_accuracy:.4f}")
print(f"Prediction Time: {log_reg_prediction_time:.4f} seconds")
print("\nClassification Report:\n", classification_report(y_test, y_pred))
```

```
--- Logistic Regression Results ---
Accuracy: 0.8512
Prediction Time: 0.0033 seconds

Classification Report:
              precision    recall  f1-score   support

           0       0.85      0.87      0.86      2997
           1       0.85      0.83      0.84      2727

    accuracy                           0.85      5724
   macro avg       0.85      0.85      0.85      5724
weighted avg       0.85      0.85      0.85      5724
```

Вывод по результатам Logistic Regression

Метрики модели

Accuracy (Точность):

- 85.12% — высокая точность, модель хорошо справляется с задачей классификации.

Precision (Точность):

- Модель одинаково хорошо предсказывает оба класса.

Recall (Полнота):

- Класс "1" немного хуже выявляется моделью, но разница незначительная.

F1-score:

- Хороший баланс между precision и recall.

Время предсказания:

- Всего **0.0033 секунды** — модель очень быстрая, что важно для использования в реальном времени.

Рекомендации:

- Logistic Regression показывает себя как базовая модель с хорошими результатами.
- Для улучшения качества можно попробовать более сложные модели, такие как SVM или нейронные сети.
- Провести гиперпараметрическую настройку, чтобы улучшить качество, особенно для класса "1".

SVM

```
# Обучение SVM с подбором гиперпараметров
param_grid = {'C': [0.1, 1, 10], 'kernel': ['linear', 'rbf']}
svm = SVC(random_state=42)
grid_search = GridSearchCV(estimator=svm, param_grid=param_grid, cv=3, scoring='accuracy'
grid_search.fit(X_train, y_train)

# Лучшая модель SVM
best_svm = grid_search.best_estimator_

# Время предсказания
start_time = time.time()
y_pred_svm = best_svm.predict(X_test)
svm_prediction_time = time.time() - start_time

# Метрики
```

```
svm_accuracy = accuracy_score(y_test, y_pred_svm)
print("\n--- SVM Results ---")
print(f"Accuracy: {svm_accuracy:.4f}")
print(f"Prediction Time: {svm_prediction_time:.4f} seconds")
print("\nClassification Report:\n", classification_report(y_test, y_pred_svm))
```

```
Fitting 3 folds for each of 6 candidates, totalling 18 fits

--- SVM Results ---
Accuracy: 0.8532
Prediction Time: 18.8870 seconds

Classification Report:
               precision    recall  f1-score   support

           0       0.85      0.87      0.86      2997
           1       0.85      0.84      0.84      2727

    accuracy                           0.85      5724
   macro avg       0.85      0.85      0.85      5724
weighted avg       0.85      0.85      0.85      5724
```

Вывод по результатам SVM

Метрики модели: Accuracy (Точность):

- 85.32% — хороший результат, сопоставимый с Logistic Regression.

Precision (Точность):

- Модель демонстрирует одинаковую точность для обоих классов.

Recall (Полнота):

- Модель чуть лучше распознаёт класс "несарказм".

F1-score:

- Метрики сбалансированы для обоих классов.

Время предсказания:

- **18.89 секунд** — модель значительно медленнее, чем Logistic Regression и LSTM, что может быть недостатком для задач реального времени.

Особенности модели:

- Подбор гиперпараметров улучшил производительность SVM.
- SVM показывает стабильные и сбалансированные метрики между классами, что делает её подходящей для задач, где важен общий баланс.

Вывод: SVM показывает стабильные и надёжные результаты, но долгий процесс предсказания делает её менее предпочтительной для приложений, требующих высокой скорости.

LSTM

```
max_words = 5000
max_len = 30

tokenizer = Tokenizer(num_words=max_words)
tokenizer.fit_on_texts(data['cleaned_text'])
sequences = tokenizer.texts_to_sequences(data['cleaned_text'])
X_text = pad_sequences(sequences, maxlen=max_len)

embedding_matrix = np.zeros((max_words, embedding_dim))
for word, i in tokenizer.word_index.items():
    if i < max_words:
        embedding_vector = embeddings_index.get(word)
        if embedding_vector is not None:
            embedding_matrix[i] = embedding_vector

X_train_text, X_test_text, X_train_features, X_test_features, y_train, y_test = train_tes
    X_text, X_features, y, test_size=0.2, random_state=42, stratify=y
)

y_train = to_categorical(y_train)
y_test = to_categorical(y_test)

text_input = Input(shape=(max_len,), name="Text_Input")
embedding_layer = Embedding(input_dim=max_words, output_dim=embedding_dim, weights=[embed
                            input_length=max_len, trainable=True)(text_input)
lstm_layer = LSTM(128)(embedding_layer)
text_output = Dropout(0.5)(lstm_layer)

numeric_input = Input(shape=(X_train_features.shape[1],), name="Numeric_Input")
numeric_output = Dense(64, activation="relu")(numeric_input)
numeric_output = Dropout(0.5)(numeric_output)

merged = Concatenate()([text_output, numeric_output])
dense_layer = Dense(64, activation="relu")(merged)
dense_layer = Dropout(0.5)(dense_layer)
output = Dense(2, activation="softmax")(dense_layer)

model = Model(inputs=[text_input, numeric_input], outputs=output)
model.compile(optimizer=Adam(learning_rate=0.001), loss="categorical_crossentropy", metri

early_stopping = EarlyStopping(monitor='val_loss', patience=2, restore_best_weights=True)

history = model.fit(
    [X_train_text, X_train_features], y_train,
    epochs=10,
    batch_size=32,
    validation_split=0.2,
    callbacks=[early_stopping]
)
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/embedding.py:90: UserWa
  warnings.warn(
Epoch 1/10
573/573 ━━━━━━━━━━━━━━━━━━━━ 48s 78ms/step - accuracy: 0.6239 - loss: 0.6652 - val_ac
Epoch 2/10
573/573 ━━━━━━━━━━━━━━━━━━━━ 81s 77ms/step - accuracy: 0.8633 - loss: 0.3312 - val_ac
Epoch 3/10
573/573 ━━━━━━━━━━━━━━━━━━━━ 85s 83ms/step - accuracy: 0.9020 - loss: 0.2488 - val_ac
Epoch 4/10
573/573 ━━━━━━━━━━━━━━━━━━━━ 77s 74ms/step - accuracy: 0.9221 - loss: 0.1960 - val_ac
Epoch 5/10
573/573 ━━━━━━━━━━━━━━━━━━━━ 87s 83ms/step - accuracy: 0.9425 - loss: 0.1518 - val_ac
```

```
# Время предсказания
start_time = time.time()
y_pred_lstm = model.predict([X_test_text, X_test_features])
lstm_prediction_time = time.time() - start_time

# Преобразование предсказаний и меток в классы
y_pred_lstm_classes = np.argmax(y_pred_lstm, axis=1)
y_test_classes = np.argmax(y_test, axis=1)

# Вывод метрик
lstm_accuracy = accuracy_score(y_test_classes, y_pred_lstm_classes)
print("\n--- LSTM Results ---")
print(f"Accuracy: {lstm_accuracy:.4f}")
print(f"Prediction Time: {lstm_prediction_time:.4f} seconds")
print("\nClassification Report:\n", classification_report(y_test_classes, y_pred_lstm_cla
```

```
179/179 ━━━━━━━━━━━━━━━━━━━━ 7s 39ms/step

--- LSTM Results ---
Accuracy: 0.8391
Prediction Time: 7.6327 seconds

Classification Report:
              precision    recall  f1-score   support

           0       0.79      0.94      0.86      2997
           1       0.91      0.73      0.81      2727

    accuracy                           0.84      5724
   macro avg       0.85      0.83      0.84      5724
weighted avg       0.85      0.84      0.84      5724
```

Вывод по результатам LSTM

Метрики модели:

Accuracy (Точность):

- 83.91% — приемлемое качество классификации, но ниже, чем у более сложных моделей (например, DistilBERT).

Precision (Точность):

- Модель лучше предсказывает класс "сарказм".

Recall (Полнота):

- Хорошо выявляет несаркастические заголовки.
- Класс "сарказм": 73% — хуже распознаёт саркастические заголовки.

F1-score:

- Баланс между precision и recall для класса"1" ниже, чем для класса "0".

Время предсказания:

- **7.63 секунды** — значительно медленнее, чем у Logistic Regression.

Особенности модели:

- LSTM показывает дисбаланс в метриках между классами, особенно в Recall для саркастических заголовков.

Рекомендации:

- Улучшить способность модели распознавать саркастические заголовки.
- Модель не является оптимальной для текущего случая.

DistilBERT

```
tokenizer = DistilBertTokenizer.from_pretrained('distilbert-base-uncased')

train_texts, val_texts, train_labels, val_labels = train_test_split(
    data['cleaned_text'], data['is_sarcastic'], test_size=0.2, random_state=42, stratify=
)

train_encodings = tokenizer(list(train_texts), truncation=True, padding=True, max_length=
val_encodings = tokenizer(list(val_texts), truncation=True, padding=True, max_length=128)

class SarcasmDataset(torch.utils.data.Dataset):
    def __init__(self, encodings, labels):
        self.encodings = encodings
        self.labels = labels

    def __len__(self):
        return len(self.labels)

    def __getitem__(self, idx):
        item = {key: torch.tensor(val[idx]) for key, val in self.encodings.items()}
        item['labels'] = torch.tensor(self.labels[idx])
        return item
```

```
train_dataset = SarcasmDataset(train_encodings, list(train_labels))
val_dataset = SarcasmDataset(val_encodings, list(val_labels))

model = DistilBertForSequenceClassification.from_pretrained('distilbert-base-uncased', nu

training_args = TrainingArguments(
    output_dir='./results',
    num_train_epochs=3,
    per_device_train_batch_size=16,
    per_device_eval_batch_size=64,
    warmup_steps=500,
    weight_decay=0.01,
    logging_dir='./logs',
    evaluation_strategy="epoch",
    save_strategy="epoch",
    load_best_model_at_end=True
)

trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=train_dataset,
    eval_dataset=val_dataset
)

# Обучение модели
trainer.train()
# Оценка модели
print("Evaluating DistilBERT model...")
import time
from sklearn.metrics import accuracy_score, classification_report

# Время предсказания
start_time = time.time()
predictions = trainer.predict(val_dataset)
distilbert_prediction_time = time.time() - start_time

# Преобразование предсказаний
preds = torch.argmax(torch.tensor(predictions.predictions), axis=1).numpy()

# Метрики
distilbert_accuracy = accuracy_score(val_labels, preds)
print("\n--- DistilBERT Results ---")
print(f"Accuracy: {distilbert_accuracy:.4f}")
print(f"Prediction Time: {distilbert_prediction_time:.4f} seconds")
print("\nClassification Report:\n", classification_report(val_labels, preds))
```

```
/usr/local/lib/python3.10/dist-packages/huggingface_hub/utils/_auth.py:94: UserWarnin
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tab (https
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access public mo
  warnings.warn(
```

tokenizer_config.json: 100% 48.0/48.0 [00:00<00:00, 3.09kB/s]

vocab.txt: 100% 232k/232k [00:00<00:00, 5.93MB/s]

tokenizer.json: 100% 466k/466k [00:00<00:00, 6.27MB/s]

config.json: 100% 483/483 [00:00<00:00, 33.4kB/s]

model.safetensors: 100% 268M/268M [00:01<00:00, 219MB/s]

```
Some weights of DistilBertForSequenceClassification were not initialized from the mod
You should probably TRAIN this model on a down-stream task to be able to use it for p
/usr/local/lib/python3.10/dist-packages/transformers/training_args.py:1575: FutureWar
  warnings.warn(
wandb: WARNING The `run_name` is currently set to the same value as `TrainingArgument
wandb: Using wandb-core as the SDK backend.  Please refer to https://wandb.me/wandb-c
wandb: Logging into wandb.ai. (Learn how to deploy a W&B server locally: https://wandb
wandb: You can find your API key in your browser here: https://wandb.ai/authorize
wandb: Paste an API key from your profile and hit enter, or press ctrl+c to quit: ···
wandb: Appending key for api.wandb.ai to your netrc file: /root/.netrc
Tracking run with wandb version 0.19.1
Run data is saved locally in /content/wandb/run-20241226_005540-fxv3oajp
Syncing run ./results to Weights & Biases (docs)
View project at https://wandb.ai/kseniya-bennu-home/huggingface
View run at https://wandb.ai/kseniya-bennu-home/huggingface/runs/fxv3oajp
```

[4293/4293 13:10, Epoch 3/3]

Epoch	Training Loss	Validation Loss
1	0.295500	0.234976
2	0.143600	0.285356
3	0.039300	0.403254

```
Evaluating DistilBERT model...

--- DistilBERT Results ---
Accuracy: 0.9057
Prediction Time: 7.0214 seconds

Classification Report:
              precision    recall  f1-score   support

           0       0.92      0.90      0.91      2997
           1       0.89      0.92      0.90      2727

    accuracy                           0.91      5724
   macro avg       0.91      0.91      0.91      5724
weighted avg       0.91      0.91      0.91      5724
```

Вывод по результатам DistilBERT

Обучение модели:

- **Training Loss** значительно снижается, но **Validation Loss** увеличивается после первой эпохи (с 0.234 до 0.403). Это указывает на необходимость ранней остановки, регуляризации или сокращения количества эпох.

Метрики модели:

Accuracy (Точность):

- 90.57% — модель демонстрирует высокое качество классификации.

Precision (Точность):

- Высокая точность в определении несаркастических заголовков.
- Модель хорошо справляется с саркастическими заголовками.

Recall (Полнота): — Модель чуть лучше улавливает саркастические заголовки.

F1-score:

- Сбалансированный результат между precision и recall.

Время предсказания:

- **7.02 секунды** — модель требует значительного времени для выполнения предсказаний, что может быть критично в задачах реального времени.

Рекомендации:

- **Улучшение обучения:** Использовать регуляризацию (Dropout), раннюю остановку или сократить количество эпох, чтобы предотвратить переподгонку.
- Оптимизация времени предсказания.

DistilBERT показал высокие результаты классификации, но требует доработок для улучшения обобщающей способности и времени предсказания.

```
# Сохранение модели и токенизатора
model_path = "/content/drive/MyDrive/DistilBERT_saved_model"
tokenizer_path = "/content/drive/MyDrive/DistilBERT_saved_tokenizer"

model.save_pretrained(model_path)
tokenizer.save_pretrained(tokenizer_path)

print(f"Все сохранилось:\n{model_path}\n{tokenizer_path}")
```

```
Все сохранилось:
/content/drive/MyDrive/DistilBERT_saved_model
/content/drive/MyDrive/DistilBERT_saved_tokenizer
```

Выводы

Анализ данных

1. Размер и структура данных

- Датасет состоит из заголовков новостей, разделённых на два класса: саркастические и несаркастические.
- **Размер датасета:** 28,503 строки (после удаления дубликатов).

2. Распределение классов

- Данные сбалансированы между двумя классами (половина заголовков саркастические, половина – нет), что упрощает обучение моделей.

3. Длина заголовков

- Средняя длина заголовков составляет около 60 символов.
- Саркастические заголовки в среднем короче несаркастических.

4. Частотный анализ текста

- Топ-слова в саркастических заголовках часто отражают иронию или абсурд.
- Несаркастические заголовки содержат информативные и нейтральные термины.

5. Использование пунктуации

- Восклицательные и вопросительные знаки редко встречаются в заголовках, чаще в несаркастических.

6. Корреляция признаков

- Сильная корреляция между длиной заголовков и количеством слов (0.91).
- Независимые признаки включают среднюю длину слов, количество восклицательных и вопросительных знаков.

7. Выводы после анализа признаков: