Федеральное государственное автономное

образовательное учреждение высшего образования

«Пермский государственный национальный исследовательский университет» (ПГНИУ)

Региональный институт непрерывного образования (РИНО ПГНИУ)

Цифровая кафедра

Выпускная аттестационная (квалификационная) работа

по курсу профессиональной переподготовки

«Анализ данных и машинное обучение»

**ОБНАРУЖЕНИЕ САРКАЗМА В ТЕКСТАХ**

**С ПОМОЩЬЮ МАШИННОГО ОБУЧЕНИЯ**

Разработчики проекта:

студентки ИТХ-1,2-2021

Ахматова Ульяна Александровна

Воробьева Дарья Юрьевна

Уткина Александра Владимировна

Пермь, 2024

**ПАСПОРТ ПРОЕКТА**

**Название проекта:**

«Обнаружение сарказма в текстах с помощью машинного обучения».

**Сведения об авторах:** Ахматова Ульяна Александровна;Воробьева Дарья Юрьевна; Уткина Александра Владимировна.

**Цель:**

Построить классификатор распознавания сарказма в комментариях пользователей.

**Задачи:**

1. Найти объёмный и сложный для обработки датасет.
2. Придумать решаемую задачу, поставить цель.
3. Изучить предметную область.
4. Выполнить анализ датасета: его загрузку и предварительную обработку перед началом решения поставленной задачи.
5. Осуществить моделирование зависимости целевого признака от факторных с помощью моделей МО, в том числе, выбрать лучшую модель, оценить её качество, выполнить прогнозирование.
6. Выполнить интерпретацию полученных результатов и сделать выводы о достижении цели.

**Краткое описание проекта:**

Требуется проанализировать данные и определить, возможно ли описать зависимость сарказма в комментарии исходя из имеющихся признаков. Дать интерпретацию полученным результатам, сделать выводы.

**Конкретные ожидаемые результаты:**

Обработка естественного языка (NLP) и обученная модель для определения выражений с сарказмом и без.

Содержание

[1 Содержание проекта 4](#_Toc185435914)

[2 Анализ предметной области и описание исходных данных 5](#_Toc185435915)

[3 Предварительная обработка исходного датасета 8](#_Toc185435916)

[4 Обучение модели 11](#_Toc185435917)

[5 Совершенствование модели, предсказание 18](#_Toc185435918)

[Заключение 22](#_Toc185435919)

[Библиографический список 24](#_Toc185435920)

[ПРИЛОЖЕНИЕ А программный код проекта 25](#_Toc185435921)

1 Содержание проекта

В последние годы социальные сети и онлайн-платформы стали важными инструментами общения и обмена мнениями для миллионов пользователей по всему миру. Однако с увеличением объёмов пользовательских комментариев также возросла актуальность проблемы обнаружения нежелательного контента, включая оскорбления, спам и сарказм.

Сарказм представляет собой сложную форму коммуникации, которая затрудняет работу автоматических систем модерации. Его понимание часто требует глубокого анализа контекста, интонации и культурных особенностей. Например, одна и та же фраза может быть воспринята совершенно по-разному в зависимости от ситуации или культурного опыта собеседников. Это делает задачу автоматического распознавания сарказма особенно сложной, так как традиционные алгоритмы обработки текста не всегда способны уловить тонкие нюансы языка.

Создание модели машинного обучения для автоматического определения сарказма в комментариях пользователей может значительно улучшить процесс модерации контента. Это позволит разработчикам внедрить более эффективные решения, способные автоматически фильтровать и классифицировать комментарии, помогая сохранить продуктивную и безопасную атмосферу в онлайн-пространстве. Более того, автоматизация обнаружения сарказма может снизить нагрузку на команды модераторов, позволяя им сосредоточиться на более сложных случаях, требующих человеческого вмешательства и более глубокого анализа. В результате, работа модераторов станет ещё эффективной и целенаправленней.

Актуальность данной работы заключается в том, что разработка и обучение моделей, способных распознавать сарказм, открывают новые горизонты для улучшения качества взаимодействия пользователей с цифровыми платформами.

В ходе данной работы была реализована модель для обнаружения сарказма в комментариях. Работа осуществляется на языке программирования Python [1] в облачной среде разработки Google Colaboratory (Colab) [2].

Целью данной работы является построение классификатора распознавания сарказма в комментариях пользователей.

Для достижения цели выделены следующие задачи:

1. Найти объёмный и сложный для обработки датасет.
2. Придумать решаемую задачу, поставить цель.
3. Изучить предметную область.
4. Выполнить анализ датасета: его загрузку и предварительную обработку перед началом решения поставленной задачи.
5. Осуществить моделирование зависимости целевого признака от факторных с помощью моделей МО, в том числе, выбрать лучшую модель, оценить её качество, выполнить прогнозирование.
6. Выполнить интерпретацию полученных результатов и сделать выводы о достижении цели.

2 Анализ предметной области и описание исходных данных

Reddit – сайт, который предназначен для обсуждения актуальных тем разных областей жизни, новостей и вопросов [3]. Пользователь Reddit имеет возможность публиковать посты по определённой тематике, комментировать других пользователей, оценивать посты как положительно, так и отрицательно. В зависимости от обсуждаемости темы и от количества оценок публикации могут попадать в топ или становиться заблокированными.

Рассматриваемый сайт представляет собой платформу, где пользователи могут свободно делиться своими мыслями и идеями на самые разные темы, без ограничения в тематиках. Это создаёт атмосферу, в которой каждый может выразить своё мнение так, будто ведёт разговор с реальным собеседником. Такой подход способствует более естественному и непринуждённому общению, что, в свою очередь, позволяет пользователям чувствовать себя комфортно и раскрепощённо.

Одной из особенностей такого формата общения является близость стиля к разговорному языку. Это значит, что участники обсуждений могут использовать более неформальные выражения, прибегать к бытовым примерам и делиться личными историями, что делает диалог более живым и увлекательным. В этом контексте литературные приёмы, такие как метафоры, сравнения и сарказм, становятся важными инструментами для передачи эмоций и создания ярких образов.

Например, метафоры могут помочь пользователям выразить сложные идеи более доступным языком, делая их мысли более понятными и запоминающимися. Сравнения позволяют проводить параллели между различными ситуациями, что помогает читателям лучше понять точку зрения автора. Сарказм же, как форма выражения иронии, может добавить элемент юмора в обсуждение, но одновременно требует от читателя способности уловить тонкие нюансы и контекст.

Однако использование таких литературных приёмов также создаёт определенные сложности для автоматических систем модерации контента. Сарказм, в частности, может быть труднораспознаваемым даже для человека, не говоря уже о машинах, поскольку он часто зависит от контекста и интонации. Это подчёркивает важность разработки эффективных инструментов для анализа и понимания комментариев пользователей на платформе, чтобы сохранить её открытость и поддерживать конструктивный диалог.

Своё мнение на сайте Reddit пользователь может высказывать как в постах, так и в комментариях. Для того, чтобы начать работу над моделью машинного обучения, рассмотрим структуру поста.

Структура поста на сайте Reddit:

— заголовок – первое, что видит пользователь. Именно заголовок играет решающую роль в привлечении внимания. Часто пользователи используют вопросы или провокационные утверждения, чтобы вызвать осуждение;

— тело поста – здесь автор делится основной информацией и/или своей точкой зрения. Структура текста может варьироваться от краткого предложения до длинного лонгрида;

— автор – никнейм создателя поста на сайте;

— дата публикации;

— тематика публикации – автор может выбрать соответствующие теги и/или категории, что поможет другим пользователям найти интересующий их контент в разы быстрее;

— оценка сообщества – аналогия лайков и дизлайков. Данная метрика позволяет анализировать, как пользователи реагируют на конкретные комментарии. Исходя из этой оценки составляется рейтинг комментария на сайте;

— комментарии – после публикации поста может начаться активное обсуждение в комментариях, где пользователи могут делиться своими мнениями, задавать вопросы и добавлять контекст к обсуждаемой теме. Комментарии варьируются от конструктивных до критических, не относящихся даже к тематике поста.

Таким образом, структура поста на Reddit не только определяет его содержание, но и влияет на то, как он будет воспринят сообществом. Некоторые ключевые слова способны значительно повысить шансы на успешное обсуждение и обмен мнениями.

Нами был выбран датасет системы организации конкурсов по исследованию данных Kaggle [4], который содержит комментарии пользователей на разные тематики.

Рассмотрим каждый столбец-признак (feature):

– label – метка, указывающая на то, содержит ли комментарий саркастическое выражение или нет. "1" – сарказм, "0" – отсутствие сарказма;

– comment – текст комментария, оставленного пользователем;

– author – автор комментария (имя пользователя, который оставил комментарий);

– subreddit – название тематики, в которой был оставлен комментарий. Сабреддиты могут иметь разные темы и культурные особенности;

– score – общая оценка комментария, рассчитываемая на основе голосований пользователей. Это может помочь понять, насколько комментарий был воспринят позитивно или негативно. Как раз-таки та самая оценка сообщества, о которой мы говорили ранее.;

– ups – лайки;

– downs – дизлайки;

– date – дата и время, когда был оставлен комментарий;

– created\_utc – время создания комментария в формате UTC (координированное всемирное время), точная временная метка;

– parent\_comment – родительский комментарий. Он позволяет отслеживать, к каким именно комментариям относятся ответы, т.е. помогает вникнуть в контекст.

3 Предварительная обработка исходного датасета

В начале подключим все необходимые библиотеки.

import os

import numpy as np

import pandas as pd

from sklearn.feature\_extraction.text import TfidfVectorizer

from sklearn.linear\_model import LogisticRegression

from sklearn.pipeline import Pipeline

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

from sklearn.metrics import accuracy\_score, confusion\_matrix

import seaborn as sns

from matplotlib import pyplot as plt

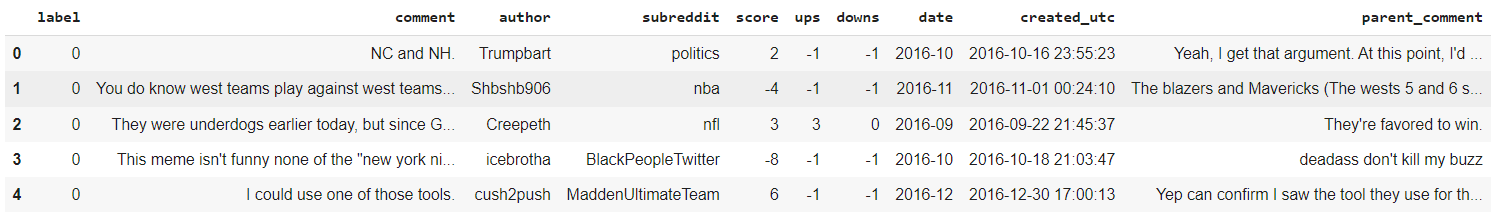
Далее подключаем google-диск и загружаем данные.

from google.colab import drive

drive.mount('/content/drive')

train\_df = pd.read\_csv(PATH\_TO\_DATA)

train\_df.head()

Рисунок 1. Исходный датафрейм

Запрашиваем общую информацию о датасете.

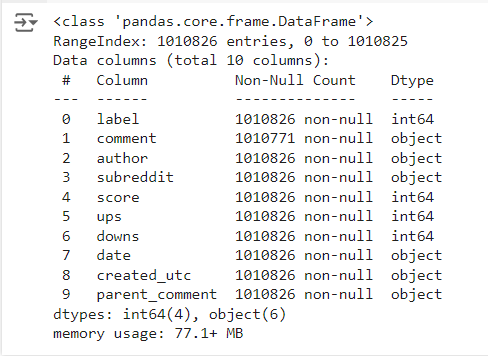
train\_df.info()

Рисунок 2. Общая информация о данных

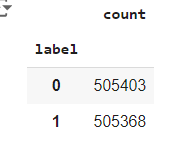
В датасете присутствуют как числовые, так и строковые данные. Также можем заметить, что в датасете присутствуют пустые значения в столбце «comment». Так как количество пустых значений мало, произведём их удаление.

train\_df.dropna(subset=['comment'], inplace=True)

Иные дополнительные методы не требуются.

Далее проверяем сбалансированность классов. Необходимо знать, соблюдён ли баланс в классах исследуемого значения, чтобы исключить вариант преобладания примеров одного типа над другими.

train\_df['label'].value\_counts()

Рисунок 3. Сбалансированность классов

Видим, что классы сбалансированы, дополнительные методы не применяем.

Визуализируем балансировку классов.

label\_counts = train\_df['label'].value\_counts()

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

plt.pie(label\_counts,

labels=label\_counts.index,

autopct='%1.1f%%', #форматирование процентов

colors=sns.color\_palette('Pastel2'))

# Добавление заголовка

plt.title('Наличие сарказма', fontsize=11)

plt.axis('equal')

plt.show()

Рисунок 4. Визуализация классов

4 Обучение модели

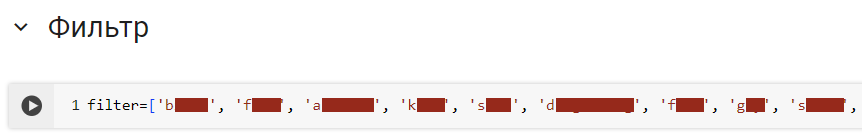
Как мы описывали ранее, пользователи могут оставлять совершенно различные комментарии, в том числе и нецензурные. Для того, чтобы модель также учитывала этот фактор, мы не очищали исходный датасет, но сделали так называемый «фильтр». Фильтр представлен списком самых популярных оскорбительных и неуместных выражений и будет использоваться при демонстрации работы модели.

Рисунок 5. Фильтр

В дальнейшем при выводе информации мы будем «фильтровать» комментарии. К примеру, если выбранный комментарий содержит запрещённое слово, то необходимо выбрать иной комментарий.



Рисунок 6. Фильтрация

В начале обучения разделим датасет на обучающую и тестовую выборки.

train\_texts, valid\_texts, y\_train, y\_valid = \

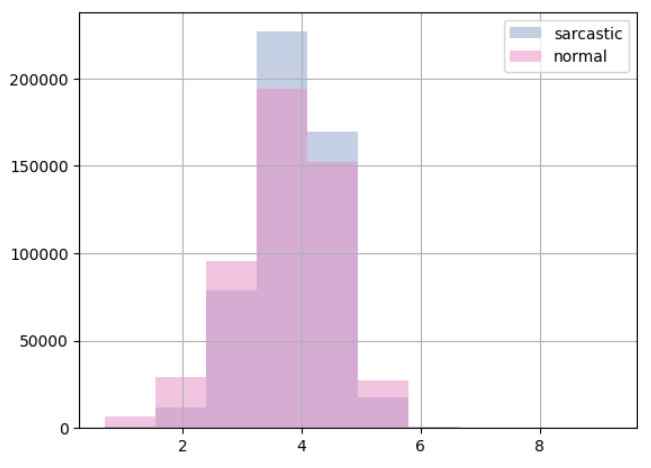
train\_test\_split(train\_df['comment'], train\_df['label'], random\_state=17)

Визуализируем их распределение.

train\_df.loc[train\_df['label'] == 1, 'comment'].str.len().apply(np.log1p).hist(label='sarcastic', alpha=.5, color = sns.color\_palette('Set2')[2])

train\_df.loc[train\_df['label'] == 0, 'comment'].str.len().apply(np.log1p).hist(label='normal', alpha=.5, color = sns.color\_palette('Set2')[3])

plt.legend();

Рисунок 7. Визуализация распределения выборок

Также посмотрим карты слов высказываний с сарказмом и без. Данные графики наглядно представят самые используемые слова в каждом классе. В дальнейшем это поможет модели определить зависимости.

from wordcloud import WordCloud, STOPWORDS

wordcloud = WordCloud(background\_color='white', stopwords = STOPWORDS,

max\_words = 200, max\_font\_size = 100,

random\_state = 17, width=800, height=400, colormap='Set2')

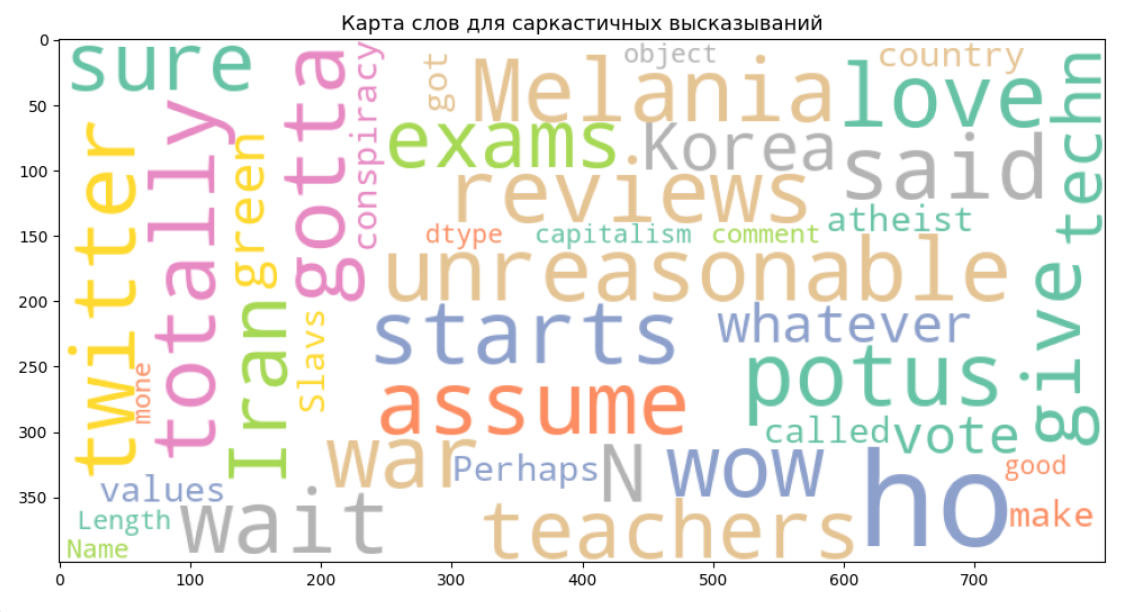
# Для 1

plt.figure(figsize=(12, 10))

plt.title('Карта слов для саркастичных высказываний', fontsize=13)

wordcloud.generate(str(train\_df.loc[train\_df['label'] == 1, 'comment']))

plt.imshow(wordcloud);

Рисунок 8. Карта слов для саркастических высказываний

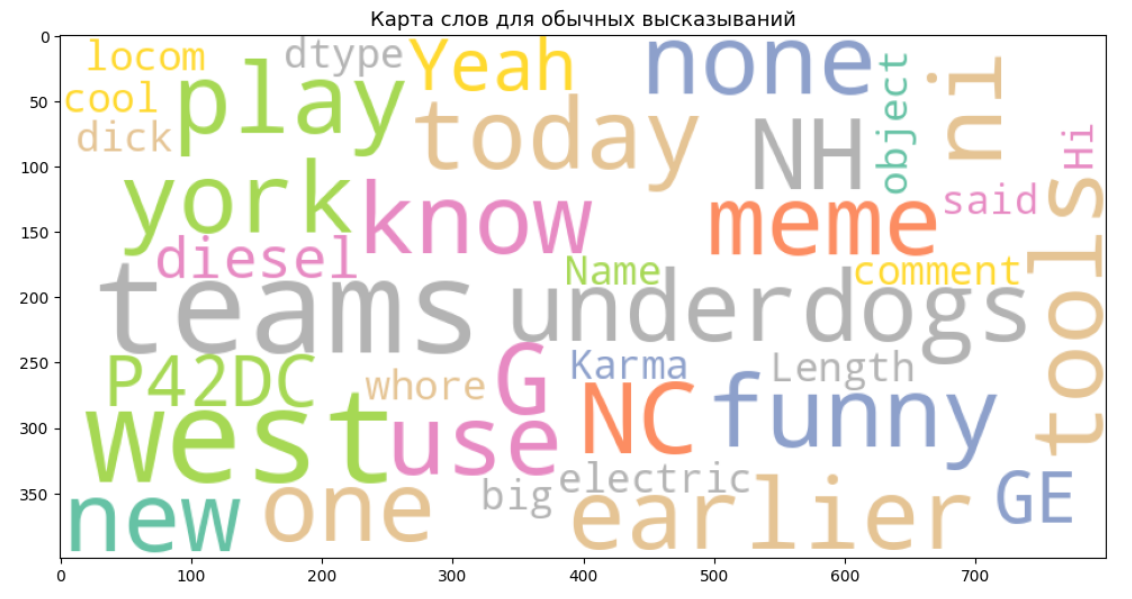


Рисунок 9. Карта слов для обычных высказываний

В карте слов саркастических высказываний можем заметить такие выражения, как «sure» («да-да, конечно»), «totally» («абсолютно»), «love» («любовь», «с любовью»). Подобные слова могут быть маркерами саркастических фраз и в английском, и в русском языках. В то же время в обычных высказываниях мы наблюдаем более разрозненные и эмоционально неокрашенные слова: «tools» («инструменты»), «one» («один»), «said» («сказала»), «today» («сегодня»).

Далее проверим, являются ли некоторые темы более «саркастичными», чем другие? То есть, есть ли какие-то тематики, в которых больше всего содержится выражений с сарказмом.

sub\_df = train\_df.groupby('subreddit')['label'].agg(['size', 'mean', 'sum'])

sub\_df.sort\_values(by='sum', ascending=False).head(10)

sub\_df[sub\_df['size'] > 1000].sort\_values(by='mean', ascending=False).head(10)

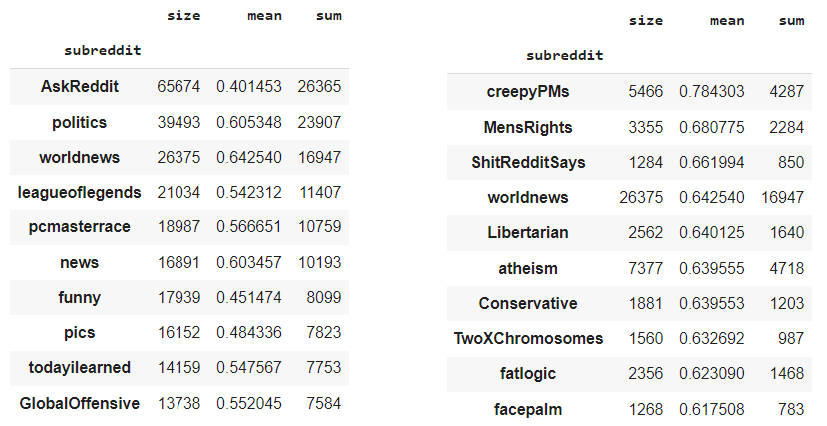


Рисунок 10. Метрики тематик

Метрики не дали особого понимания, какие из тематик содержат больше всего сарказма. Посмотрим, как влияет авторство на наличие сарказма в комментарии.

sub\_df = train\_df.groupby('author')['label'].agg(['size', 'mean', 'sum'])

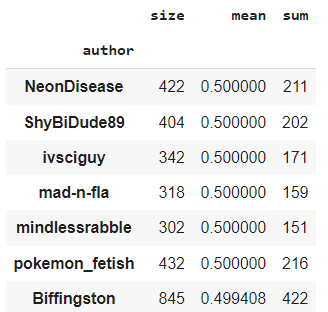
sub\_df[sub\_df['size'] > 300].sort\_values(by='mean', ascending=False).head(10)

Рисунок 11. Метрики тематик по авторам

Для авторов то же самое, метрика не даёт особого понимания. За исключением того факта, что были выбраны чьи-то комментарии – мы можем видеть одинаковое количество саркастических и не саркастических комментариев.

Определим значимость слов в тексте. Для этого используется метод TF-IDF (term frequency — inverse document frequency). Значимость определяется за счёт частоты появления слова относительно всех слов, встречаемых в предложениях. В текущем случае мы установили ограничение на максимальное количество объектов и минимальную частоту слов, а также определили n-граммы: униграммы (для анализа каждого слова по отдельности) и биграммы (для анализа слов по словосочетаниям).

tf\_idf = TfidfVectorizer(ngram\_range=(1, 2), max\_features=50000, min\_df=2)

Создадим модель мультиномиальной логистической регрессии (классификатор softmax).

logit = LogisticRegression(C=1, n\_jobs=4, solver='lbfgs',

                           random\_state=17, verbose=1)

Объединим метод и модель в пайплайн для дальнейшей работы и запустим обучение с выводом затраченного времени.

%%time

tfidf\_logit\_pipeline.fit(train\_texts, y\_train)

Выполним предсказание обученной модели с выводом потраченного времени и выведем оценку модели.

%%time

valid\_pred = tfidf\_logit\_pipeline.predict(valid\_texts)



Рисунок 12. Оценка модели

Учитывая объём датасета, значение оценки точности работы модели удовлетворительно.

Визуализируем матрицу ошибок.

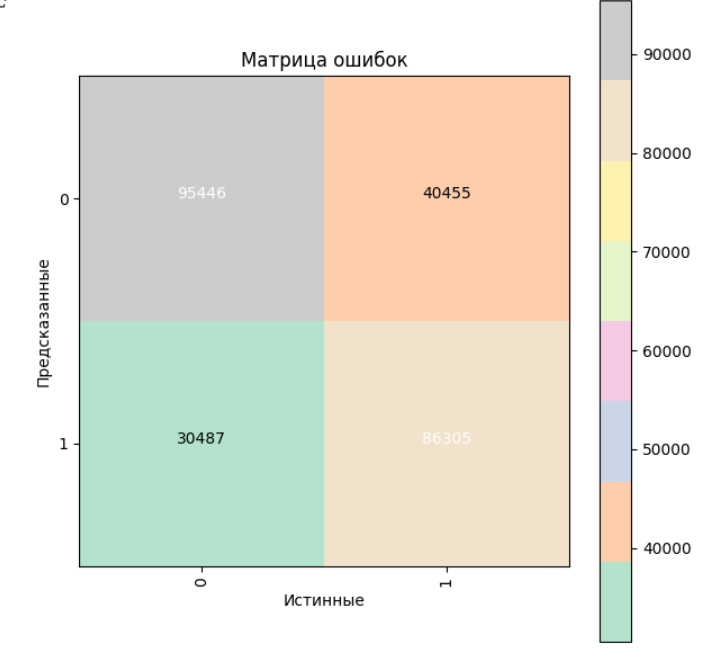


Рисунок 13. Матрица ошибок

Глядя на матрицу, можем сказать, что большинство комментариев было правильно классифицировано. Модель уже может уверенно распознавать сарказм в некоторых фразах, например, «да конечно».

Обратимся к интерпретатору lime для получения представления о работе модели.

# Импорт необходимых библиотек

from lime import lime\_text

from lime.lime\_text import LimeTextExplainer

# Создание экземпляра объяснителя

explainer = LimeTextExplainer(class\_names=['Not Sarcastic', 'Sarcastic'])

# Выбор примера для объяснения (например, первый комментарий в валидационном наборе)

idx = 43210 # Вы можете изменить индекс на любой другой, чтобы объяснить другой комментарий

text\_to\_explain = valid\_texts.iloc[idx]

# Получение предсказаний вероятностей

predict\_proba = tfidf\_logit\_pipeline.predict\_proba

# Объяснение примера

exp = explainer.explain\_instance(text\_to\_explain, predict\_proba, num\_features=10)

# Визуализация объяснения

exp.show\_in\_notebook(text=True)

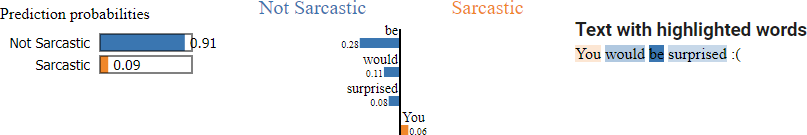


Рисунок 14. Метод lime

С помощью данного метода мы можем увидеть, как классифицирует наша модель. На рисунке выше представлен один из комментариев, lime определил какие слова больше всего повлияли на отнесение этого высказывания к классу «саркастичные».

5 Совершенствование модели, предсказание

Для совершенствования модели отделим имеющийся столбец тематики в датасет и произведём деление на тестовую и обучающую выборки.

subreddits = train\_df['subreddit']

train\_subreddits, valid\_subreddits = train\_test\_split(subreddits, random\_state=17)

Далее создадим отдельные векторизаторы Tf-Idf. Воспользуемся TfidfVectorizer – инструментом, который преобразует текстовые данные в числовые векторы с использованием метода, называемого TF-IDF. Это позволит нам эффективно подготовить данные к обучению модели, используя числовые представления текстовых данных и, как следствие, улучшить метрики модели.

tf\_idf\_texts = TfidfVectorizer(ngram\_range=(1, 2), max\_features=50000, min\_df=2)

tf\_idf\_subreddits = TfidfVectorizer(ngram\_range=(1, 1))

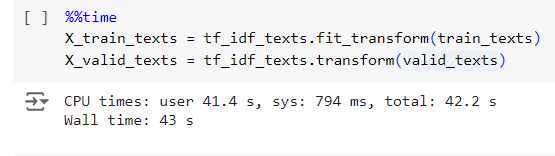
Выполняем преобразования отдельно для комментариев и отдельно для подтематик.

Рисунок 15. Векторизация

После этого объединяем датасеты в один: и для тренировки, и для тестов – и обучаем модель ещё раз.

from scipy.sparse import hstack

X\_train = hstack([X\_train\_texts, X\_train\_subreddits])

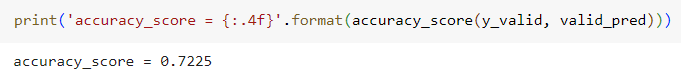
X\_valid = hstack([X\_valid\_texts, X\_valid\_subreddits])

logit.fit(X\_train, y\_train)

%%time

valid\_pred = logit.predict(X\_valid)

Далее смотрим, как изменились метрики модели.

Рисунок 16. Метрики после применения векторизации

При сравнении оценок точности (71.9% и 72.6%) можно заметить, что точность растёт, но достаточно слабо. Так как даже человек не всегда может определить наличие сарказма в том или ином высказывании, текущую метрику стоит считать достойной.

В итоге можем посмотреть, как классифицирует комментарии наша модель.

# функция, чтобы достать предложение и значение его сарказма

def WordsAndSarcasm(index):

if not any(word.lower() in valid\_texts.iloc[index] for word in filter):

print(f"{valid\_subreddits.iloc[index]} пишет: '{valid\_texts.iloc[index]}'. Это сказано с сарказмом?")

if (valid\_pred[index] == 0):

sarcastic\_cowsay(random.randint(1, 17))

else:

print('Сарказм не был обнаружен, а значит ругаться некому.')

else:

WordsAndSarcasm(random.randint(0, valid\_texts.shape[0]))

WordsAndSarcasm(random.randint(0, valid\_texts.shape[0]))

Рисунок 17. Предсказание без сарказма

Так как данная работа проделана в учебных целях, авторы приняли решение демонстрировать саркастичные высказывания саркастичным способом – в случае, если выражение определяется моделью как саркастичное, на экран будет выводиться животное и раскрашенный текст о том, что он заметил использование сарказма пользователем.

Загружаем необходимый модуль для отображения говорящих зверей.

pip install cowsay

Добавляем строчки кода по определению используемых цветов текста, а также подчёркивания, жирности и окончания предложения.

#раскраска животных

class Colors:

    bold = '\033[1m'  # Bold

    uline = '\033[4m'  # Underline

    endc = '\033[0m'  # End of color line

colors = [

    "\033[31m",  # Красный

    "\033[32m",  # Зеленый

    "\033[33m",  # Желтый

    "\033[34m",  # Синий

    "\033[35m",  # Магента

    "\033[36m",  # Циан

    "\033[95m",  # Purple color

    "\033[94m",  # Blue color

    "\033[35m"

]

Напишем функцию по выводу случайного животного, текст которого будет раскрашен в случайный цвет.

import cowsay

import random

def sarcastic\_cowsay(option):

    """Функция, выбирающая персонажа по номеру и выводящая сарказм."""

    # Словарь доступных персонажей

    characters = {

        1: cowsay.cheese,

        2: cowsay.cow,

        3: cowsay.daemon,

        4: cowsay.dragon,

        5: cowsay.fox,

        6: cowsay.ghostbusters,

        7: cowsay.kitty,

        8: cowsay.meow,

        9: cowsay.miki,

        10: cowsay.milk,

        11: cowsay.octopus,

        12: cowsay.pig,

        13: cowsay.stegosaurus,

        14: cowsay.trex,

        15: cowsay.turkey,

        16: cowsay.turtle,

        17: cowsay.tux

    }

    # Словарь с русскими именами персонажей

    character\_names = {

        1: "сыр",

        2: "корова",

        3: "демон",

        4: "дракон",

        5: "лисица",

        6: "призраки",

        7: "котенок",

        8: "кот",

        9: "мики",

        10: "молоко",

        11: "осьминог",

        12: "свинья",

        13: "стегозавр",

        14: "ти-рекс",

        15: "индейка",

        16: "черепаха",

        17: "пингвин"

    }

    if option in characters:

        character = characters[option]

        character\_name = character\_names[option]

        random\_color = colors[random.randint(0, len(colors) - 1)]

        character(f"{random\_color}{Colors.bold}Даже я, {character\_name}, вижу, что это сарказм. {Colors.endc}")

    else:

        cowsay.cow("Упс! Неправильный номер. Просто идеально.")

# Генерируем случайное число от 1 до 17

random\_number = random.randint(1, 17)

При вызове функции WordsAndSarcasm можем увидеть, что модель посчитала фразу саркастичной, а значит и её вывод произошёл с использованием поддерживающего животного.

Рисунок 18. Предсказание с сарказмом

Заключение

На основе проведённой классификации можно сделать вывод, что наличие сарказма в предложении зависит от контекста слов и выражений. Использованная векторизация текста позволила не только выявить частоту и редкость появления тех или иных высказываний, но и определить слова, которые влияют на формирование саркастичного комментария. Кроме этого, сама тема разговора может слегка влиять на наличие сарказма в тексте, так как в разговорной речи контекст зависит не только от рядом стоящих слов, но и от обсуждаемого предмета.

Таким образом, поставленная цель была достигнута. Был выполнен анализ данных и построена модель классификации комментариев по наличию сарказма в них, которая может определять с неплохой точностью.

Для достижения цели были решены следующие задачи:

* выполнен анализ проблемы, обоснована её актуальность;
* осуществлена загрузка данных и подготовка их для дальнейшего обучения;
* осуществлено моделирование зависимости целевого признака от факторных;
* выполнена интерпретация полученных результатов.

Построенная модель классифицирует комментарии и может применяться для социальных и лингвистических целей. К примеру, для изучения особенностей языка и применения тех или иных оборотов для выразительности речи, для настройки более чувствительных фильтров модерации, для обучения иных моделей определения стиля и предназначения текста. Все перечисленные применения преследуют одну идею – увеличить безопасность взаимодействия в социальных сетях и уменьшить конфликтность тех или иных сообществ, так как проявляемая агрессия часто начинается с сарказма и заканчивается оскорблениями.

Библиографический список

1. Welcome to Python / [Электронный ресурс] // Python: [сайт]. — URL: https://www.python.org (дата обращения: 15.12.2024).
2. Google Colab / [Электронный ресурс] // Google Colab: [сайт]. — URL: https://colab.research.google.com (дата обращения: 15.12.2024).
3. Reddit - Dive into anything / [Электронный ресурс] // Reddit: [сайт]. — URL: https://www.reddit.com (дата обращения: 15.12.2024).
4. Sarcasm on Reddit / [Электронный ресурс] // Kaggle: [сайт]. — URL: https://www.kaggle.com/datasets/danofer/sarcasm (дата обращения: 18.12.2024).
5. Барский, А. Б. Введение в нейронные сети : учебное пособие / А. Б. Барский. — 4-е изд. — Москва : Интернет-Университет Информационных Технологий (ИНТУИТ), Ай Пи Ар Медиа, 2024. — 357 c. — ISBN 978-5-4497-2381-9. — Текст : электронный // Цифровой образовательный ресурс IPR SMART : [сайт]. — URL: https://www.iprbookshop.ru/133929.html (дата обращения: 15.12.2024)
6. Бессмертный, И. А.  Интеллектуальные системы : учебник и практикум для академического бакалавриата / И. А. Бессмертный, А. Б. Нугуманова, А. В. Платонов. — Москва : Издательство Юрайт, 2019. — 243 с. — (Бакалавр. Академический курс). — ISBN 978-5-534-01042-8. — Текст : электронный // Образовательная платформа Юрайт [сайт]. — URL: <https://www.urait.ru/bcode/433716> (дата обращения: 15.12.2024)

ПРИЛОЖЕНИЕ А  
программный код проекта

pip install cowsay

class Colors:

    bold = '\033[1m'  # Bold

    uline = '\033[4m'  # Underline

    endc = '\033[0m'  # End of color line

colors = [

    "\033[31m",  # Красный

    "\033[32m",  # Зеленый

    "\033[33m",  # Желтый

    "\033[34m",  # Синий

    "\033[35m",  # Магента

    "\033[36m",  # Циан

    "\033[95m",  # Purple color

    "\033[94m",  # Blue color

    "\033[35m"

]

import cowsay

import random

def sarcastic\_cowsay(option):

    # Словарь доступных персонажей

    characters = {

        1: cowsay.cheese,

        2: cowsay.cow,

        3: cowsay.daemon,

        4: cowsay.dragon,

        5: cowsay.fox,

        6: cowsay.ghostbusters,

        7: cowsay.kitty,

        8: cowsay.meow,

        9: cowsay.miki,

        10: cowsay.milk,

        11: cowsay.octopus,

        12: cowsay.pig,

        13: cowsay.stegosaurus,

        14: cowsay.trex,

        15: cowsay.turkey,

        16: cowsay.turtle,

        17: cowsay.tux

    }

    # Словарь с русскими именами персонажей

    character\_names = {

        1: "сыр",

        2: "корова",

        3: "демон",

        4: "дракон",

        5: "лисица",

        6: "призраки",

        7: "котенок",

        8: "кот",

        9: "мики",

        10: "молоко",

        11: "осьминог",

        12: "свинья",

        13: "стегозавр",

        14: "ти-рекс",

        15: "индейка",

        16: "черепаха",

        17: "пингвин"

    }

    if option in characters:

        character = characters[option]

        character\_name = character\_names[option]

        #character(f"Даже я, {character\_name}, вижу, что это сарказм. {random.choice(jokes)}")

        random\_color = colors[random.randint(0, len(colors) - 1)]

        character(f"{random\_color}{Colors.bold}Даже я, {character\_name}, вижу, что это сарказм. {random.choice(jokes)}{Colors.endc}")

    else:

        cowsay.cow("Упс! Неправильный номер. Просто идеально.")

# Генерируем случайное число от 1 до 17

random\_number = random.randint(1, 17)

pip install lime

import os

import numpy as np

import pandas as pd

from sklearn.feature\_extraction.text import TfidfVectorizer

from sklearn.linear\_model import LogisticRegression

from sklearn.pipeline import Pipeline

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

from sklearn.metrics import accuracy\_score, confusion\_matrix

import seaborn as sns

from matplotlib import pyplot as plt

PATH\_TO\_DATA = '/content/drive/MyDrive/2trim/4 курс/Нейронки/Итоговое/train-balanced-sarcasm.csv'

from google.colab import drive

drive.mount('/content/drive')

train\_df = pd.read\_csv(PATH\_TO\_DATA)

train\_df.head()

train\_df.info()

#проверим баланс

train\_df['label'].value\_counts()

#визуализация соотношений классов

label\_counts = train\_df['label'].value\_counts()

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

plt.pie(label\_counts,

        labels=label\_counts.index,

        autopct='%1.3f%%',          #форматирование процентов

        colors=sns.color\_palette('Pastel2'))

# Добавление заголовка

plt.title('Наличие сарказма', fontsize=11)

plt.axis('equal')

plt.show()

#делим на обучающую и тестовую выборки

train\_texts, valid\_texts, y\_train, y\_valid = \

        train\_test\_split(train\_df['comment'], train\_df['label'], random\_state=17)

# Посмотрим график

train\_df.loc[train\_df['label'] == 1, 'comment'].str.len().apply(np.log1p).hist(label='sarcastic', alpha=.5, color = sns.color\_palette('Set2')[2])

train\_df.loc[train\_df['label'] == 0, 'comment'].str.len().apply(np.log1p).hist(label='normal', alpha=.5, color = sns.color\_palette('Set2')[3])

plt.legend();

# Выведем карту слов

from wordcloud import WordCloud, STOPWORDS

wordcloud = WordCloud(background\_color='white', stopwords = STOPWORDS,

                max\_words = 200, max\_font\_size = 100,

                random\_state = 17, width=800, height=400, colormap='Set2')

# Для 1

plt.figure(figsize=(12, 10))

plt.title('Карта слов для саркастичных высказываний', fontsize=13)

wordcloud.generate(str(train\_df.loc[train\_df['label'] == 1, 'comment']))

plt.imshow(wordcloud);

# Для 0

plt.figure(figsize=(12, 10))

plt.title('Карта слов для обычных высказываний', fontsize=13)

wordcloud.generate(str(train\_df.loc[train\_df['label'] == 0, 'comment']))

plt.imshow(wordcloud);

sub\_df = train\_df.groupby('subreddit')['label'].agg(['size', 'mean', 'sum'])

sub\_df.sort\_values(by='sum', ascending=False).head(10)

sub\_df[sub\_df['size'] > 1000].sort\_values(by='mean', ascending=False).head(10)

sub\_df = train\_df.groupby('author')['label'].agg(['size', 'mean', 'sum'])

sub\_df[sub\_df['size'] > 300].sort\_values(by='mean', ascending=False).head(10)

sub\_df = train\_df[train\_df['score'] >= 0].groupby('score')['label'].agg(['size', 'mean', 'sum'])

sub\_df[sub\_df['size'] > 300].sort\_values(by='mean', ascending=False).head(10)

sub\_df = train\_df[train\_df['score'] < 0].groupby('score')['label'].agg(['size', 'mean', 'sum'])

sub\_df[sub\_df['size'] > 300].sort\_values(by='mean', ascending=False).head(10)

# создаём биграммы, устанавливаем ограничение на максимальное количество объектов

# и минимальную частоту слов

tf\_idf = TfidfVectorizer(ngram\_range=(1, 2), max\_features=50000, min\_df=2)

# мультиномиальная логистическая регрессия, также известная как классификатор softmax

logit = LogisticRegression(C=1, n\_jobs=4, solver='lbfgs',

                           random\_state=17, verbose=1)

# sklearn's конвейер

tfidf\_logit\_pipeline = Pipeline([('tf\_idf', tf\_idf),

                                 ('logit', logit)])

%%time

tfidf\_logit\_pipeline.fit(train\_texts, y\_train)

%%time

valid\_pred = tfidf\_logit\_pipeline.predict(valid\_texts)

print('accuracy\_score = {:.4f}'.format(accuracy\_score(y\_valid, valid\_pred)))

def plot\_confusion\_matrix(actual, predicted, classes,

                          normalize=False,

                          title='Матрица ошибок', figsize=(7,7),

                          cmap="Pastel2", path\_to\_save\_fig=None):

    import itertools

    from sklearn.metrics import confusion\_matrix

    cm = confusion\_matrix(actual, predicted).T

    if normalize:

        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]

    plt.figure(figsize=figsize)

    plt.imshow(cm, interpolation='nearest', cmap="Pastel2")

    plt.title(title)

    plt.colorbar()

    tick\_marks = np.arange(len(classes))

    plt.xticks(tick\_marks, classes, rotation=90)

    plt.yticks(tick\_marks, classes)

    fmt = '.2f' if normalize else 'd'

    thresh = cm.max() / 2.

    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):

        plt.text(j, i, format(cm[i, j], fmt),

                 horizontalalignment='center',

                 color='white' if cm[i, j] > thresh else 'black')

    plt.tight\_layout()

    plt.ylabel('Предсказанные')

    plt.xlabel('Истинные')

    if path\_to\_save\_fig:

        plt.savefig(path\_to\_save\_fig, dpi=300, bbox\_inches='tight')

plot\_confusion\_matrix(y\_valid, valid\_pred,

                      tfidf\_logit\_pipeline.named\_steps['logit'].classes\_, figsize=(6, 6))

# Импорт необходимых библиотек

from lime import lime\_text

from lime.lime\_text import LimeTextExplainer

# Создание экземпляра объяснителя

explainer = LimeTextExplainer(class\_names=['Not Sarcastic', 'Sarcastic'])

# Выбор примера для объяснения (например, первый комментарий в валидационном наборе)

idx = 43210 # Вы можете изменить индекс на любой другой, чтобы объяснить другой комментарий

text\_to\_explain = valid\_texts.iloc[idx]

# Получение предсказаний вероятностей

predict\_proba = tfidf\_logit\_pipeline.predict\_proba

# Объяснение примера

exp = explainer.explain\_instance(text\_to\_explain, predict\_proba, num\_features=10)

# Визуализация объяснения

exp.show\_in\_notebook(text=True)

subreddits = train\_df['subreddit']

train\_subreddits, valid\_subreddits = train\_test\_split(subreddits, random\_state=17)

tf\_idf\_texts = TfidfVectorizer(ngram\_range=(1, 2), max\_features=50000, min\_df=2)

tf\_idf\_subreddits = TfidfVectorizer(ngram\_range=(1, 1))

%%time

X\_train\_texts = tf\_idf\_texts.fit\_transform(train\_texts)

X\_valid\_texts = tf\_idf\_texts.transform(valid\_texts)

X\_train\_texts.shape, X\_valid\_texts.shape

%%time

X\_train\_subreddits = tf\_idf\_subreddits.fit\_transform(train\_subreddits)

X\_valid\_subreddits = tf\_idf\_subreddits.transform(valid\_subreddits)

X\_train\_subreddits.shape, X\_valid\_subreddits.shape

from scipy.sparse import hstack

X\_train = hstack([X\_train\_texts, X\_train\_subreddits])

X\_valid = hstack([X\_valid\_texts, X\_valid\_subreddits])

X\_train.shape, X\_valid.shape

logit.fit(X\_train, y\_train)

%%time

valid\_pred = logit.predict(X\_valid)

print('accuracy\_score = {:.4f}'.format(accuracy\_score(y\_valid, valid\_pred)))

#тут функция, чтобы достать рандомное предложение и значение его сарказма

#return words, issarcasm

def WordsAndSarcasm(index):

  if not any(word.lower() in valid\_texts.iloc[index] for word in filter):

    print(f"{valid\_subreddits.iloc[index]} пишет: '{valid\_texts.iloc[index]}'. Это сказано с сарказмом?")

    if (valid\_pred[index] == 0):

      sarcastic\_cowsay(random.randint(1, 17))

    else:

      print('Сарказм не был обнаружен, а значит ругаться некому.')

  else:

    WordsAndSarcasm(random.randint(0, valid\_texts.shape[0]))

WordsAndSarcasm(random.randint(0, valid\_texts.shape[0]))

#тестирование

def WordsAndSarcasm(index):

    current\_text = valid\_texts.iloc[index]

    current\_subreddit = valid\_subreddits.iloc[index]

    # Проверяем, содержит ли текст слова из фильтра

    if not any(word.lower() in current\_text for word in filter):

        print(f"{current\_subreddit} пишет: '{current\_text}'. Это сказано с сарказмом?")

        # Проверяем предсказание

        if (valid\_pred[index] == 0):

            sarcastic\_cowsay(random.randint(1, 17))  # Используется ваша функция для сарказма

        else:

            print('Сарказм не был обнаружен, а значит ругаться некому.')

    else:

        # Если текст содержит слова из фильтра, выводим случайное сообщение

        # print("Не удается определить сарказм. Попробуйте другое сообщение.")

        new\_index = random.randint(0, valid\_texts.shape[0] - 1)  # Новый случайный индекс

        WordsAndSarcasm(new\_index)  # Рекурсивно вызываем функцию для нового индекса

# Вызов функции

WordsAndSarcasm(random.randint(0, valid\_texts.shape[0] - 1))