

Переход от бизнес-задачи к моделям машинного обучения: стратегии и методы для эффективного решения

Лапушкина Елена Павловна

Приамурский государственный университет им. Шолом-Алейхема

Студент

Языков Кирилл Александрович

Приамурский государственный университет им. Шолом-Алейхема

Студент

Аннотация

Целью данного исследования является применение различных методов машинного обучения для разработки модели, которая позволит предсказывать результаты, связанные с переходом от бизнес-задач к моделям машинного обучения. В результате исследования выбрана модель logistic regression, обеспечивающая лучшие результаты для текущей задачи. Сравнение моделей произведено на основании основных метрик: precision и recall.

Ключевые слова: jupyter notebook, precision, recall, python, anaconda, модель машинного обучения.

Moving from a business problem to machine learning models: strategies and methods for effective solutions

Lapushkina Elena Pavlovna

Sholom-Aleichem Priamursky State University

Student

Yazykov Kirill Alexandrovich

Sholom-Aleichem Priamursky State University

Student

Abstract

The purpose of this study is to apply various machine learning methods to develop a model that will predict the results associated with the transition from business tasks to machine learning models. As a result of the research, the logistic regression model was selected, which provides the best results for the current task. The model comparison is based on the main metrics: precision and recall.

Keywords: jupyter notebook, precision recall, python, anaconda, machine learning model.

1 Введение

1.1 Актуальность

Сегодня машинное обучение активно используется в различных сферах, включая финансы, производство и ритейл. Оно помогает прогнозировать риски, автоматизировать процессы и снижать затраты, что повышает производительность. Также технологии машинного обучения позволяют собирать данные о предпочтениях пользователей и создавать персонализированные предложения, что улучшает удовлетворенность клиентов. Основные методы включают регрессию, классификацию и глубокое обучение, которые позволяют эффективно обрабатывать данные, анализировать их и делать предсказания. ММ стало важным инструментом для оптимизации и инноваций в бизнесе.

1.2 Обзор исследований

И.А. Богомолов привел результаты аналитического исследования построения модели машинного обучения для задач классификации с применением Jupiter Notebook [1]. Р.Л. Астапов, Р.М. Мухамадеева в своей работе рассмотрели вопросы машинного обучения, описали метод GridSearchCV [2]. Э.А. Чельшев, М.В. Раскатова провели исследование, направленное на анализ основных метрик оценки качества классификации в задачах машинного обучения, включая такие важные понятия, как матрица ошибок, precision, recall [3]. Е. Бисонг в своей статье рассмотрел основы использования библиотеки Scikit-learn для построения моделей машинного обучения и глубокого обучения на облачной платформе Google [4]. К.Н. Гребнев в своей работе на примере задачи распознавания сорта вина на основании результатов химического анализа рассмотрел возможности библиотеки scikit-learn языка Python [5].

1.3 Цель исследования

Целью исследования является разработка и систематизация стратегий и методов перехода от формулировки бизнес-задач к построению эффективных моделей машинного обучения, что позволит улучшить принятие решений на основе анализа данных в различных отраслях.

Материалы и методы

Для реализации был использован язык программирования Python, инструмент управления пакетами и окружениями anaconda, Jupiter notebook. Данные для датасета были взяты с официального сайта <https://www.kaggle.com/datasets/blackmoon/russian-language-toxic-comments?resource=download>.

3 Результаты и дискуссии

Для того чтобы начать производить модель машинного обучения необходимо скачать anaconda, скачать можно на официальном сайте

<https://anaconda.com/app/>, далее система предложит открыть anaconda.navigators, открыв его запустить jupyter notebook (рис. 1).

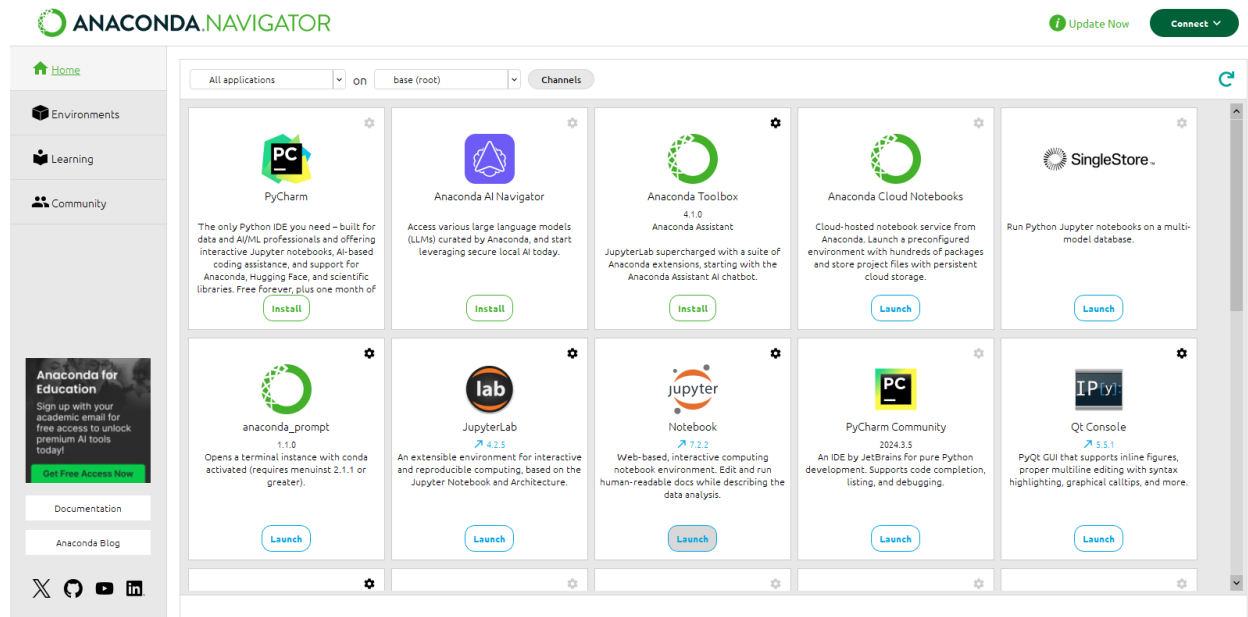


Рисунок 1 –Обзор главной страницы anaconda navigator

После запуска jupyter notebook, необходимо создать в нем новый ноутбук, и начать анализировать датасет, готовый датасет был скачан в kaggle на официальной странице <https://www.kaggle.com/datasets/blackmoon/russian-language-toxic-comments?resource=download> (рис.2).

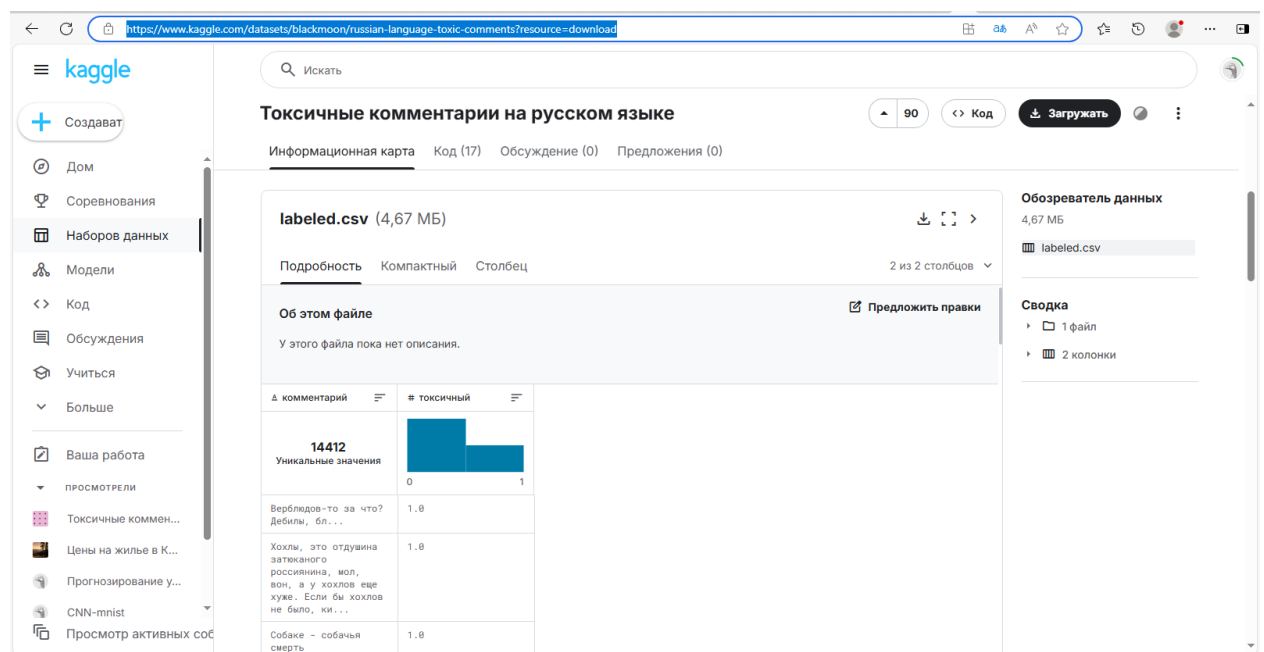


Рисунок 2- Обзор датасета на официальной странице в kaggle

Для того чтобы перейти к задаче машинного обучения, нужно разработать бинарный классификатор, который на вход будет принимать текст комментария, и на выходе выдавать два класса- это хороший или

плохой комментарий. После определения с типом задачи, определиться с метриками ограничений. Для того чтобы не удалять хорошие комментарии и удалять плохие, этому соответствуют две метрики, первая- recall-соотношение найденных плохих комментариев, то есть если эта метрика равна 1 значит найдены все плохие комментарии, вторая метрика- precision-вероятность того что если сказать, что комментарий будет плохой то он реально будет плохим, и в данном случае будут ограничения, порог должен быть не ниже 0.95.

В jupyter notebook необходимо импортировать библиотеки, под каждой из них будут размещены сообщения, подтверждающие успешный импорт (рис. 3).

```
[137]: import pandas as pd
from sklearn.model_selection import train_test_split
import nltk
import string
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import SnowballStemmer
nltk.download('punkt')
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LogisticRegression
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics import precision_score, recall_score, precision_recall_curve
from matplotlib import pyplot as plt
import numpy as np
from sklearn.metrics import PrecisionRecallDisplay
from sklearn.model_selection import GridSearchCV
nltk.download('punkt')
nltk.download('stopwords')
nltk.download('punkt_tab')
from sklearn.metrics import classification_report
```

```
[nltk_data] Downloading package punkt to
[nltk_data] C:\Users\hp\AppData\Roaming\nltk_data...
[nltk_data] Package punkt is already up-to-date!
[nltk_data] Downloading package punkt to
[nltk_data] C:\Users\hp\AppData\Roaming\nltk_data...
[nltk_data] Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to
[nltk_data] C:\Users\hp\AppData\Roaming\nltk_data...
[nltk_data] Package stopwords is already up-to-date!
[nltk_data] Downloading package punkt_tab to
[nltk_data] C:\Users\hp\AppData\Roaming\nltk_data...
[nltk_data] Package punkt_tab is already up-to-date!
```

Рисунок 3- Обзор библиотек

Далее нужно загрузить датасет, для этого используется библиотека — pandas, данные находятся в формате csv (рис. 4).

```
[138]: import pandas as pd

# Загрузка датасета из CSV файла
df = pd.read_csv('labeled.csv', sep=",")

# Вывод первых нескольких строк датасета
print(df.head())
```

```
      comment  toxic
0  Верблюд-то за что? Дебилы, бл...\n      1.0
1  Хохлы, это отдушина затканого россиянина, мол...  1.0
2  Собака - собачья смерть\n              1.0
3  Страницу обнови, дебил. Это тоже не оскорблени...  1.0
4  тебя не убедил 6-страничный pdf в том, что Скр...  1.0
```

```
[139]: df.shape
```

```
[139]: (14412, 2)
```

Рисунок 4- Обзор датасета

Следующий шаг, просмотр количества данных, первых пяти строчек, в первой колонке расположены комментарии во второй отмечены токсичны они или нет, для токсичных используется тип данных float, нужно привести его к int, для этого используется функция apply – которая применит к каждой строчке одну и ту же функцию, применив ее, тип данных будет integer в колонке toxic (рис. 5).

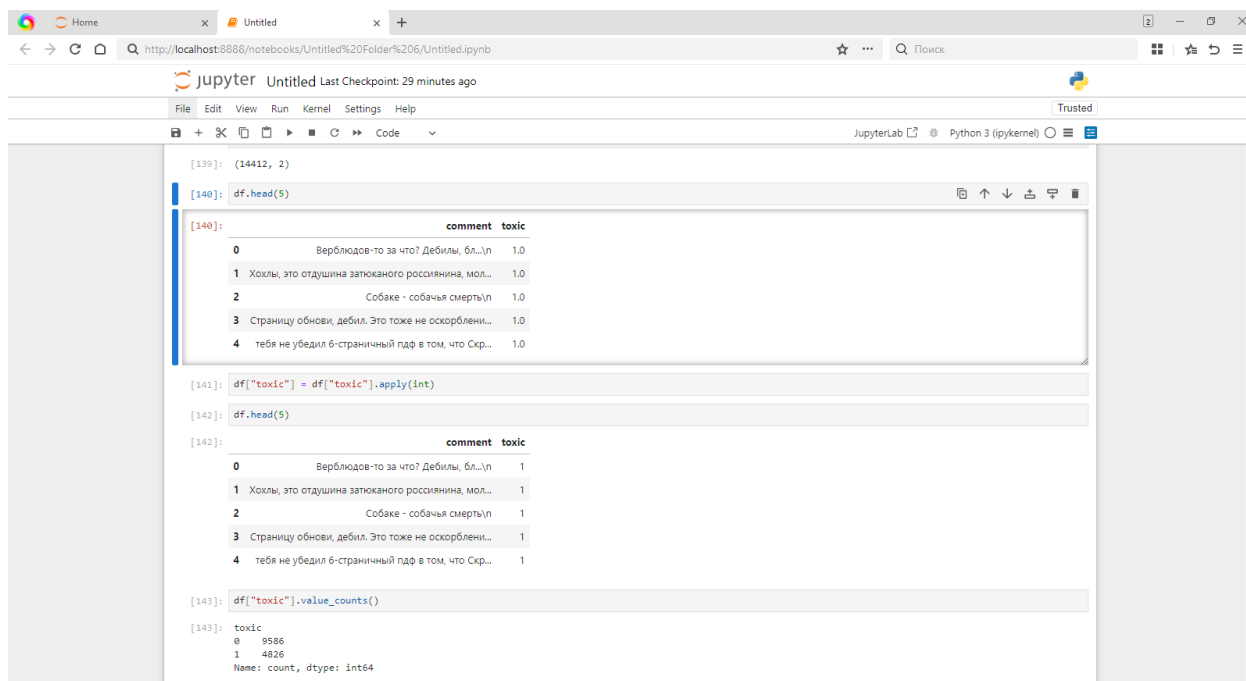


Рисунок 5- Обзор применения функции apply

Далее необходимо проанализировать распределение данных, то есть соотношение токсичных и нетоксичных комментариев. Видно, что положительных сообщений примерно в два раза больше, чем токсичных (рис.6).

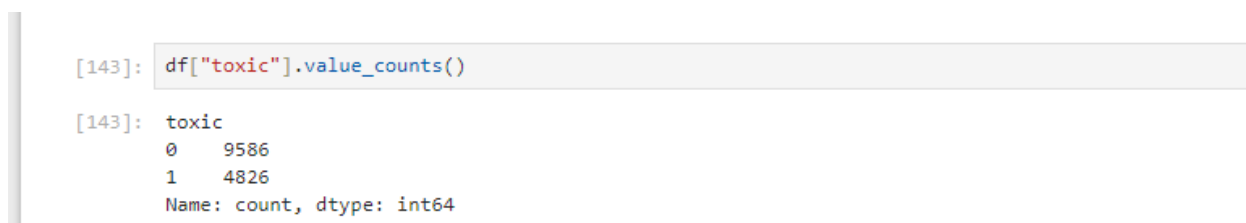


Рисунок 6- Обзор распределения данных

Вывести из датасета несколько комментариев токсичного и нетоксичного характера для оценки качества разметки (рис. 7), (рис. 8).

```
[144]: for c in df[df["toxic"] == 1]["comment"].head(5):
        print(c)
```

Верблюд-то за что? Дебилы, бл...

Хохлы, это отдушина затюканого россиянина, мол, вон, а у хохлов еще хуже. Если бы хохлов не было, кисель их бы придумал.

Собаке - собачья смерть

Страницу обнови, дебил. Это тоже не оскорбление, а доказанный факт - не-дебил про себя во множественном числе писать не будет. Или мы в тебя верим - это ты и твои воображаемые друзья?

тебя не убедил 6-страничный pdf в том, что Скрипалей отравила Россия? Анализировать и думать пытаешься? Ватник что ли?)

Рисунок 7- Обзор токсичных комментариев

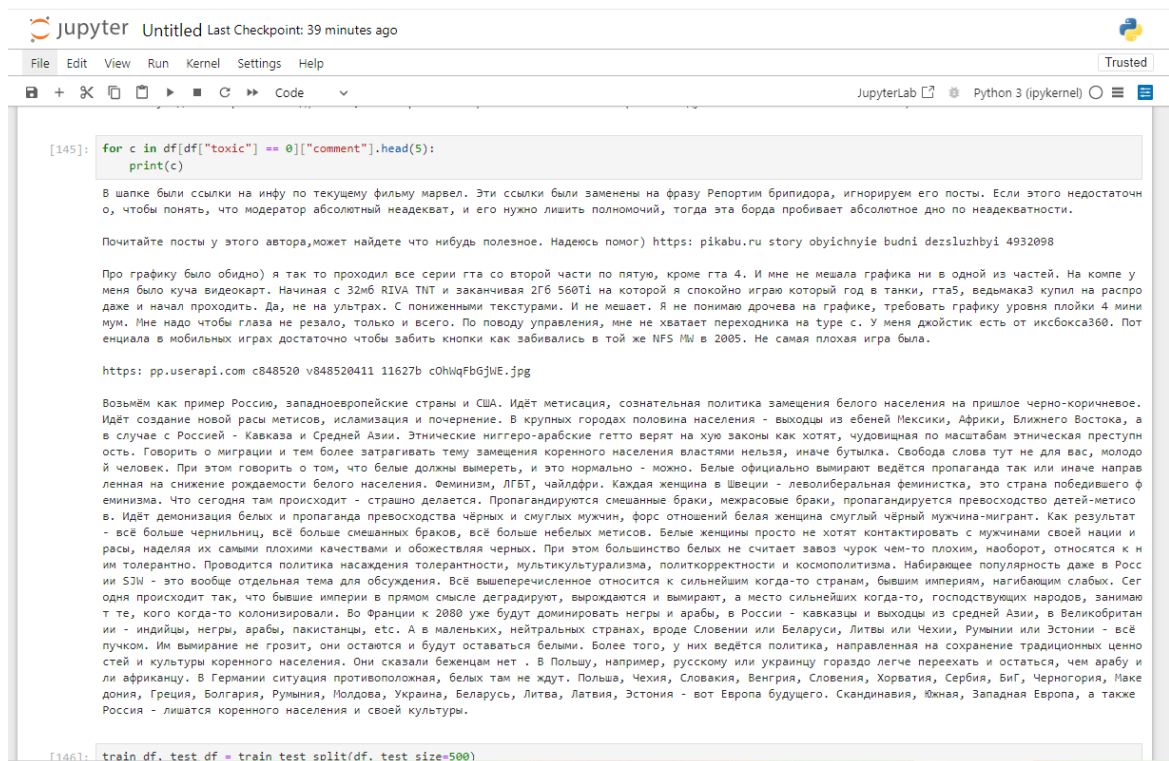


Рисунок 8- Обзор нетоксичных комментариев

Перед тем как использовать модель logistic regression, необходимо из текста сделать численные векторы, потому что сама модель на входе принимает вещественные векторы. Для обработки текста следует начать с вызова функции words tokenize, удалить знаки пунктуации с помощью стандартного модуля string, содержащего массив punctuation, после чего получить русские стоп-слова и удалить их, приведя весь текст к нижнему регистру и устранив окончания (рис.9).

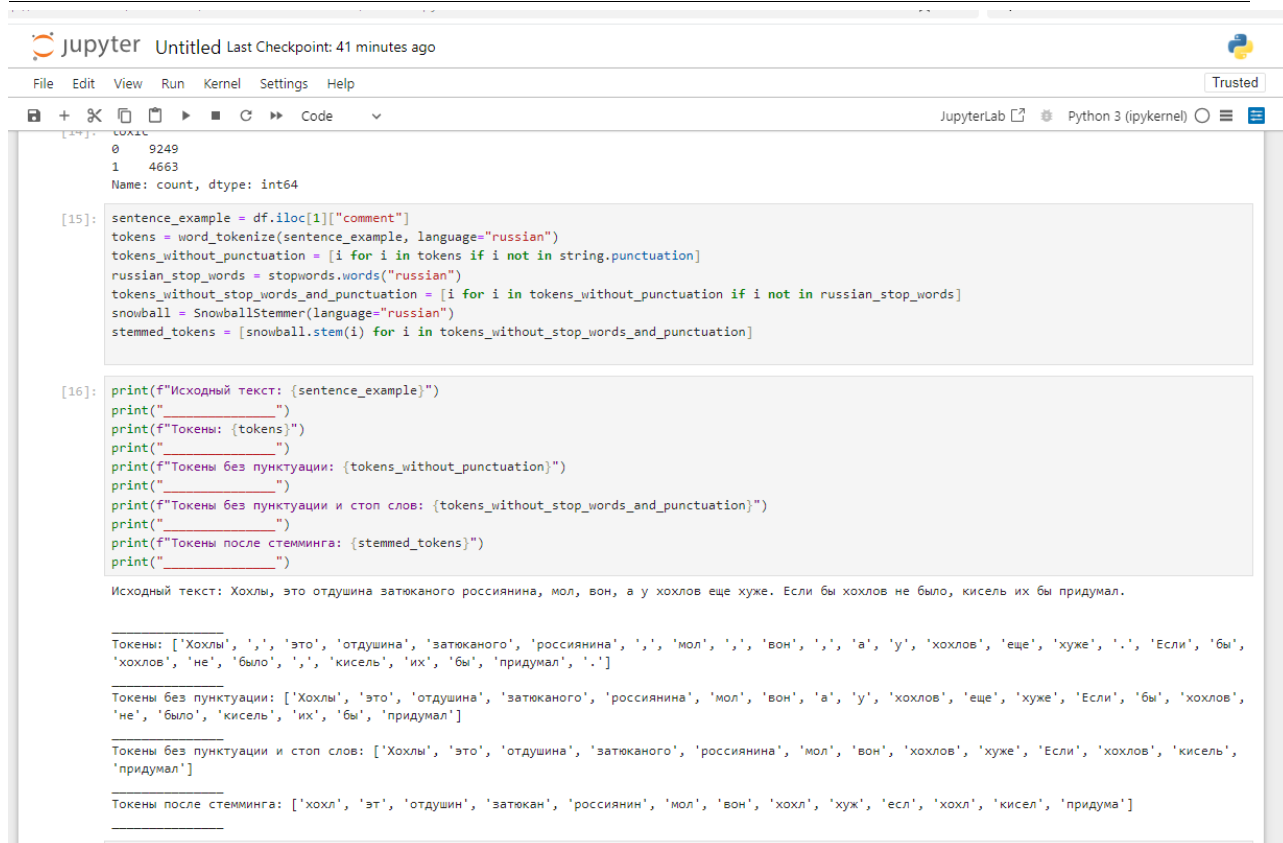


Рисунок 9- Обзор функции words tokenize

После того как готов массив токенов, применить `tfidfvectorizer`- это самый популярный способ для кодирования последовательности токенов. Вызвать функцию `fit_transform` и передать ему список предложений из датасета, функция его обучит и после этого сразу вернет готовые фичи которые можно передавать в модель машинного обучения (рис.10).

```
[19]: vectorizer = TfidfVectorizer(tokenizer=lambda x: tokenize_sentence(x, remove_stop_words=True))

[20]: features = vectorizer.fit_transform(train_df["comment"])
```

Рисунок 10- Обзор кодирования tfidfvectorizer

Далее приступить к обучению модели `logistic regression`, сначала создать класс, передать в неё `random_state=0` для того чтобы результаты были воспроизводимы, вызвать метод `fit` который уже используется для обучения куда передаются фичи (рис. 11).

```
[21]: model = LogisticRegression(random_state=0)
      model.fit(features, train_df["toxic"])

[21]: ▾      LogisticRegression      ⓘ ⓘ
      LogisticRegression(random_state=0)
```

Рисунок 11- Обзор класса logistic regression

После обучения модели проверить как она работает на тренировочном датасете, первый комментарий модель классифицировала как нормальный, исходя из содержания текста видно, что он нейтральный (рис.12).

```
[22]: model.predict(features[0])

[22]: array([0], dtype=int64)

[23]: train_df["comment"].iloc[0]

[23]: 'Блин, очень неплохо по тем временам пожил - если учесть, что правил он почти сорок лет, весьма успешный правитель - детям повезло меньше.\n'
```

Рисунок 12- Обзор нейтрального комментария

Далее применить класс pipeline, он соединит создание фичей и модель. В созданной модели прописать передачу tuple с названием компонентов и с классом, который выполняет этап предобработки (рис.13).

```
[24]: model_pipeline = Pipeline([
      ("vectorizer", TfidfVectorizer(tokenizer=lambda x: tokenize_sentence(x, remove_stop_words=True))),
      ("model", LogisticRegression(random_state=0))
    ])

[26]: model_pipeline.fit(train_df["comment"], train_df["toxic"])

C:\anaconda3\Lib\site-packages\sklearn\feature_extraction\text.py:521: UserWarning: The parameter 'token_pattern' will not be used since 'tokenizer' is not None
  warnings.warn(

[26]: ▾      Pipeline      ⓘ ⓘ
      Pipeline(steps=[('vectorizer',
      TfidfVectorizer(tokenizer=<function <lambda> at 0x0000016D0E9F77E0>)),
      ('model', LogisticRegression(random_state=0))])
      ▾      TfidfVectorizer      ⓘ
      TfidfVectorizer(tokenizer=<function <lambda> at 0x0000016D0E9F77E0>)
      ▾      LogisticRegression      ⓘ
      LogisticRegression(random_state=0)

[27]: pipeline = Pipeline(steps=[
      ('vectorizer', TfidfVectorizer(tokenizer=lambda x: tokenize_sentence(x, remove_stop_words=True))),
      ('model', LogisticRegression(random_state=0))
    ])
```

Рисунок 13- Обзор этапа предобработки данных

После обучения, придумать позитивный текст чтоб проверить, что модель вернет, array равна [0] значит комментарий прочитан как нормальный (рис. 14)

```
[28]: model_pipeline.predict(["Привет, у меня все нормально"])  
  
[28]: array([0], dtype=int64)
```

Рисунок 14 – Обзор массива array [0]

Написать агрессивный комментарий, array равна [1] значит комментарий прочитан как агрессивный (рис.15).

```
[29]: model_pipeline.predict(["Слушай не пойти ли тебе нафиг отсюда?"])  
  
[29]: array([1], dtype=int64)
```

Рисунок 15- Обзор массива array [1]

Данная модель работает хорошо, необходимо измерить метрики, в данном случае важны две метрики это- precision и recall, применить метрики на тренировочном датасете (рис.16).

```
[30]: precision_score(y_true=test_df["toxic"], y_pred=model_pipeline.predict(test_df["comment"]))  
  
[30]: 0.8682170542635659  
  
[31]: recall_score(y_true=test_df["toxic"], y_pred=model_pipeline.predict(test_df["comment"]))  
  
[31]: 0.6871165644171779
```

Рисунок 16- Обзор данных метрик precision и recall

Метрика precision < 0.95, поэтому такая модель не подходит. Применить функцию precision_recall_curve для precision и recall которая нарисует кривую, в ней меняется порог от 0 до 1 и в зависимости от нее считаются метрики, после этого рисуется график зависимости. На графике видно, что можно получить даже precision равный единице, но при этом очень низкий recall (рис. 17).

```
[37]: plt.figure(figsize=(8, 6))
plt.plot(recall, precision, marker='.')
plt.title('Precision-Recall Curve')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.grid()
plt.show()
```

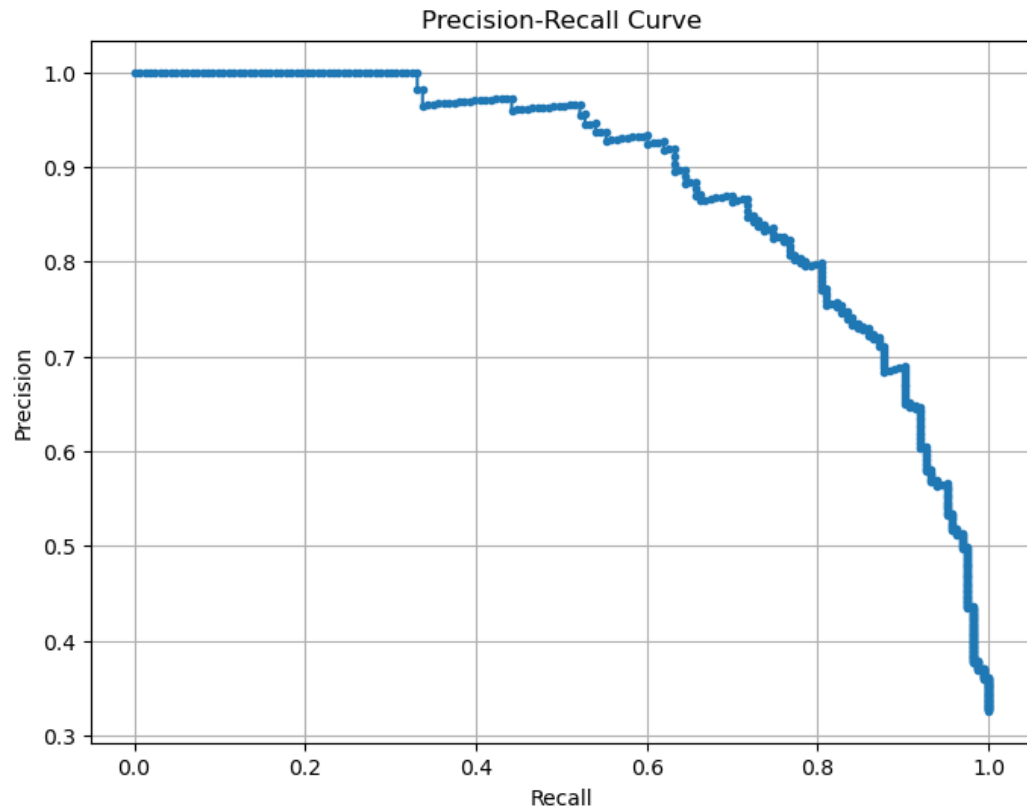


Рисунок 17- Обзор графика зависимости precision и recall

Исходя из данных графика, необходимо удалять комментарии только тогда, когда они очень негативные, то есть recall будет на уровне 0 1, около 10% негативных комментариев нужно будет удалить, поэтому нужно найти тот момент, когда precision > 0.95, использовать функцию np.where- которая вернет индексы тех элементов которые больше заданного порога, видно что с 410 значения порога precision удовлетворяют условиям (рис.18).

```
[38]: np.where(prec > 0.95)
```

```
[38]: (array([410, 411, 412, 413, 414, 415, 416, 417, 418, 419, 420, 421, 422,
423, 424, 425, 426, 427, 428, 429, 430, 431, 432, 433, 434, 435,
436, 437, 438, 439, 440, 441, 442, 443, 444, 445, 446, 447, 448,
449, 450, 451, 452, 453, 454, 455, 456, 457, 458, 459, 460, 461,
462, 463, 464, 465, 466, 467, 468, 469, 470, 471, 472, 473, 474,
475, 476, 477, 478, 479, 480, 481, 482, 483, 484, 485, 486, 487,
488, 489, 490, 491, 492, 493, 494, 495, 496, 497, 498, 499, 500],
dtype=int64),)
```

Рисунок 18- Обзор функции np.where

Далее вывести минимальный порог, с помощью функции `thresholds`, выведен итог 0.59, далее вызвать предсказания с ним, получив precision 0.95, recall 0.69 (рис. 19).

```
[40]: y_scores = model_pipeline.predict_proba(test_df["comment"])[:, 1]
      thresholds = thresholds[410]
      y_pred = (y_scores > thresholds).astype(int)
      precision = precision_score(test_df["toxic"], y_pred)

      print(f"Precision: {precision}")

Precision: 0.9550561797752809
```

```
[41]: from sklearn.metrics import roc_curve, recall_score
      y_scores = model_pipeline.predict_proba(test_df["comment"])[:, 1]
      fpr, tpr, thresholds = roc_curve(test_df["toxic"], y_scores)
      threshold = 0.5
      y_pred = (y_scores > threshold).astype(int)
      recall = recall_score(test_df["toxic"], y_pred)
      print(f"Recall: {recall}")

Recall: 0.6871165644171779
```

Рисунок 19- Обзор предсказаний precision и recall

Получив базовую модель, можно ее улучшить, применив класс `gridsearch`- который позволяет перебирать различные параметры с которыми модель будет выдавать лучшую метрику, изучить параметры можно в документации по logistic regression библиотеки `scikit learn` https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html, попробовав менять один параметр это- коэффициент регуляризации (рис. 20).

The screenshot shows the official documentation for `LogisticRegression` in the `scikit-learn` library. The left sidebar lists various modules, with `LogisticRegression` highlighted. The main content area provides a detailed overview of the class's parameters:

- dual** : *bool, default=False*
Двойственная (ограниченная) или простая (регуляризованная, см. также [это уравнение](#)) формулировка. Двойная формула реализован только для штрафа L2 с помощью Liblinear Solver. Предпочитайте `dual=False`, когда `n_samples > n_features`.
- TOL** : *float, no umolчанию = 1E-4*
Допуск по критериям остановки.
- C** : *float, no umolчанию = 1.0*
Обратная сила регуляризации; должно быть положительным плавающим положением. Как и в методах опорных векторов, меньшие значения указывают сильнее Регуляризации.
- fit_intercept** : *bool, default=True*
Указывает, должна ли константа (также известная как смещение или пересечение) быть Добавлено в функцию Decision.
- intercept_scaling** : *float, default=1*
Полезно только при использовании решателя `liblinear` и `self.fit_intercept` устанавливается в значение `True`. В этом случае `x` становится `[x, self.intercept_scaling]`, т.е. "синтетическая" характеристика с постоянным значением, равным `intercept_scaling` добавляется к вектору экземпляра. Перехват становится `.intercept_scaling * synthetic_feature_weight`.
Заметка! Вес синтетического признака подвержен регуляризации L1/L2 как и все остальные функции. Чтобы уменьшить влияние регуляризации на вес синтетического элемента (и поэтому на перехвате) `intercept_scaling` приходится увеличивать.

Рисунок 20- Обзор документации по logistic regression библиотеки `scikit learn`

Далее создать класс pipeline, преобразовать текстовые данные в числовую форму с помощью концепции vectorizer, применить gridsearchcv для оптимизации параметров модели logistic regression (рис.21).

```
[46]: grid_pipeline = Pipeline([
    ("vectorizer", TfidfVectorizer(tokenizer=lambda x: tokenize_sentence(x, remove_stop_words=True))),
    ("model",
     GridSearchCV(
        LogisticRegression(random_state=0),
        param_grid={'C': [0.1, 1, 10.0]},
        cv=3,
        verbose=4
     )
  ])

```

Рисунок 21- Обзор создания нового класса

Запустить обучение модели, обучение показало, что лучший score был получен при коэффициенте регуляризации равной 10 (рис.22).

```
[48]: grid_pipeline.fit(train_df["comment"], train_df["toxic"])

C:\anaconda3\Lib\site-packages\sklearn\feature_extraction\text.py:521: UserWarning: The parameter 'token_pattern' will not be used since 'tokenizer' is not None
  warnings.warn(
Fitting 3 folds for each of 3 candidates, totalling 9 fits
[CV 1/3] END .....C=0.1, score=0.686 total time= 0.3s
[CV 2/3] END .....C=0.1, score=0.687 total time= 0.3s
[CV 3/3] END .....C=0.1, score=0.686 total time= 0.1s
[CV 1/3] END .....C=1, score=0.834 total time= 0.4s
[CV 2/3] END .....C=1, score=0.849 total time= 0.4s
[CV 3/3] END .....C=1, score=0.841 total time= 0.2s
[CV 1/3] END .....C=10.0, score=0.858 total time= 0.4s
[CV 2/3] END .....C=10.0, score=0.874 total time= 0.5s
[CV 3/3] END .....C=10.0, score=0.870 total time= 0.6s

[48]: Pipeline
Pipeline(steps=[('vectorizer',
  TfidfVectorizer(tokenizer=<function <lambda> at 0x0000016D10C509A0>)),
  ('model',
   GridSearchCV(cv=3,
    estimator=LogisticRegression(random_state=0),
    param_grid={'C': [0.1, 1, 10.0]}, verbose=4))])
  TfidfVectorizer
  TfidfVectorizer(tokenizer=<function <lambda> at 0x0000016D10C509A0>)
    model: GridSearchCV
    GridSearchCV(cv=3, estimator=LogisticRegression(random_state=0),
      param_grid={'C': [0.1, 1, 10.0]}, verbose=4)
      best_estimator_: LogisticRegression
      LogisticRegression(C=10.0, random_state=0)
        LogisticRegression
        LogisticRegression(C=10.0, random_state=0)

```

Рисунок 22- Обзор обучения модели с лучшим показателем

Для проверки точности, проверить и обучить отдельно новую модель используя коэффициент регуляризации 10, сравнить чему будут равны метрики precision и recall. Взять код, который был прописан ранее для обучения модели и в нем указать коэффициент регуляризации равной 10, обучить её (рис. 23).

```
[49]: model_pipeline_c_10 = Pipeline([
      ("vectorizer", TfidfVectorizer(tokenizer=lambda x: tokenize_sentence(x, remove_stop_words=True))),
      ("model", LogisticRegression (random_state=0, C=10.))
    ])

[50]: model_pipeline_c_10.fit(train_df["comment"], train_df["toxic"])

C:\anaconda3\Lib\site-packages\sklearn\feature_extraction\text.py:521: UserWarning: The parameter 'token_pattern' will not be used since 'tokenizer' is not None
  warnings.warn(

[50]: ▶ Pipeline ① ②
      ▶ TfidfVectorizer ①
      ▶ LogisticRegression ①
```

Рисунок 23- Обзор обучения новой модели

После обучения модели, найти, когда значение precision будет больше >0.95 (рис. 24).

```
[49]: model_pipeline_c_10 = Pipeline([
      ("vectorizer", TfidfVectorizer(tokenizer=lambda x: tokenize_sentence(x, remove_stop_words=True))),
      ("model", LogisticRegression (random_state=0, C=10.))
    ])

[50]: model_pipeline_c_10.fit(train_df["comment"], train_df["toxic"])

C:\anaconda3\Lib\site-packages\sklearn\feature_extraction\text.py:521: UserWarning: The parameter 'token_pattern' will not be used since 'tokenizer' is not None
  warnings.warn(

[50]: Pipeline ① ②
      Pipeline(steps=[('vectorizer',
                       TfidfVectorizer(tokenizer=<function <lambda> at 0x0000016D10C52020>)),
                      ('model', LogisticRegression(C=10.0, random_state=0))])
      TfidfVectorizer
      TfidfVectorizer(tokenizer=<function <lambda> at 0x0000016D10C52020>)
      LogisticRegression
      LogisticRegression(C=10.0, random_state=0)

[51]: prec_c_10, rec_c_10, thresholds_c_10 = precision_recall_curve(y_true=test_df["toxic"], probas_pred=model_pipeline_c_10.predict_proba(test_df["comment"])):
```

Рисунок 24- Обзор обучения модели

Прописать `np.where (prec_c_10 > 0.95)` (рис. 25).

```
[56]: np.where(prec_c_10 > 0.95)

[56]: (array([416, 417, 418, 419, 420, 421, 422, 423, 424, 425, 426, 427, 428,
           429, 430, 431, 432, 433, 434, 435, 436, 437, 438, 439, 440, 441,
           442, 443, 444, 445, 446, 447, 448, 449, 450, 451, 452, 453, 454,
           455, 456, 457, 458, 459, 460, 461, 462, 463, 464, 465, 466, 467,
           468, 469, 470, 471, 472, 473, 474, 475, 476, 477, 478, 479, 480,
           481, 482, 483, 484, 485, 486, 487, 488, 489, 490, 491, 492, 493,
           494, 495, 496, 497, 498, 499, 500], dtype=int64),)
```

Рисунок 25- Обзор данных коэффициента регуляризации

После того как найдено первое значение, можно посчитать precision и recall. Исходя из данных, видно, что найдено негативных комментариев меньше, но зато сохранено и увеличено значение precision >0.97 (рис. 26).

```
[56]: np.where(prec_c_10 > 0.95)

[56]: (array([416, 417, 418, 419, 420, 421, 422, 423, 424, 425, 426, 427, 428,
         429, 430, 431, 432, 433, 434, 435, 436, 437, 438, 439, 440, 441,
         442, 443, 444, 445, 446, 447, 448, 449, 450, 451, 452, 453, 454,
         455, 456, 457, 458, 459, 460, 461, 462, 463, 464, 465, 466, 467,
         468, 469, 470, 471, 472, 473, 474, 475, 476, 477, 478, 479, 480,
         481, 482, 483, 484, 485, 486, 487, 488, 489, 490, 491, 492, 493,
         494, 495, 496, 497, 498, 499, 500], dtype=int64),)

[79]: precision_score(y_true=test_df["toxic"], y_pred=model_pipeline_c_10.predict_proba(test_df["comment"])[:, 1]>thresholds_c_10[416])

[79]: 0.9629629629629629

[80]: recall_score(y_true=test_df["toxic"], y_pred=model_pipeline_c_10.predict_proba(test_df["comment"])[:, 1]>thresholds_c_10[416])

[80]: 0.4785276073619632
```

Рисунок 26- Обзор предсказаний precision и recall

Данные по метрикам вывести в итоговую таблицу (рис. 27).

Model	Model- pipeline precision	Model- pipeline recall	Model- pipeline_c_10 precision	Model- pipeline_c_10 recall
Logistic Regression	0,95	0,69	0,97	0,48

Рисунок 27- Сравнение точности прогнозирования результата

4 Выводы

Анализ результатов показал, что модель Model_pipeline_c_10 precision достигла наивысшего уровня точности в предсказании положительных комментариев. Этот результат свидетельствует о высокой эффективности модели и открывает новые горизонты для дальнейших исследований в данной области.

Библиографический список

1. Богомолов И.А. Исследование построения модели машинного обучения для решения задач классификации // Информационные технологии и защита информации. 2024. С. 212-218. URL: <https://www.elibrary.ru/item.asp?id=74209962>

2. Астапов Р.Л., Мухамадеева Р.М. Автоматизация подбора параметров машинного обучения и обучение модели машинного обучения // Актуальные научные исследования в научном мире. - 2021. №5-2. С. 34-37. URL: <https://www.elibrary.ru/item.asp?id=46193354>

3. Челышев Э. А., Раскатова М. В. Метрики в задачах классификации // XXIV Всероссийская студенческая научно-практическая конференция Нижневартковского государственного университета. 2022. С. 242. URL:

https://www.elibrary.ru/download/elibrary_48328951_68589761.pdf#page=242

4. Бисонг Е. Введение в Scikit-learn // Построение моделей машинного обучения и глубокого обучения на облачной платформе Google: исчерпывающее руководство для начинающих. 2019. С. 215-229. URL: https://link.springer.com/chapter/10.1007/978-1-4842-4470-8_18
5. Гребнев К.Н. Машинное обучение с помощью библиотеки skicit learn языка python // Математический вестник педвузов и университетов волго-вятского региона. 2017. №19. С. 277-281. URL: <https://www.elibrary.ru/item.asp?id=29063789>