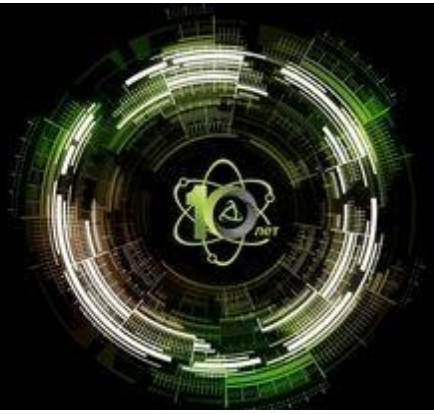

АО «Гринатом»

Отчет

“Тестовое задание для кандидатов на предстажировку Case Lab ML”

Выполнила Клинова Мария

Структура отчета

О задании

Цель работы: обучить модель для распознавания эмоциональной окраски отзыва о фильме.

Датасет представлен Стенфордской лабораторией изучения искусственного интеллекта по ссылке: https://ai.stanford.edu/~amaas/data/sentiment/

Рейтинг фильма определяется значением после символа «_».

Этапы работы:

1. Обучение модели классификации отзывов;
2. Создание веб-интерфейса для классификации отзыва о фильме на базе фреймворка Django;
3. Деплой сервиса на платформе ‘render.com’;
4. Написание отчета.

О решении

Выполнила: Клинова Мария

Ссылка на прототип сервиса: https://case-lab-ml-klinova.onrender.com/

Ссылка на GitHub: https://github.com/TyupTupa/Case_Lab_ML

Обучение модели

Выбор и обучение модели представлено в файле model_learning.ipynb.

Процесс работы с данными и обучения модели представлял собой последовательность этапов:

1. Импорт необходимых модулей и библиотек для предобработки

Для предобработки были использованы следующие модули и ресурсы библиотек**: re, os, pandas** и **nltk.**

2. Создание датафреймов

Данные для тестирования были загружены в два датасета test и train с помощью функции creating_df(type). На вход функции подается тип датафрейма, который необходимо сформировать из датасета aclImdb, – train (для тренировки) или test (для тестирования). Выход – датафрейм со столбцами name, text, sentiment, rate.

Столбец	Предназначение
Name	Название .txt файла, откуда был загружен отзыв.
Text	Текст отзыва.
Sentiment	Окраска текста – positive (позитивная) или negative (негативная).
Rate	Рейтинг фильма.

Так же после создания датафрейма его строки были перемешаны для удаления зависимостей порядка и улучшения обучающей способности.

```
[8]: def creating_df(type):
         """
         Создает датафрейм с колонками: 'name' (название файла),
             'text' (текст файла) и 'sentiment' (окраска текста).
         """

         df = pd.DataFrame()
         def extract_rate(file_name):
             return int(file_name.split('_')[1].split('.')[0])

         for name in os.listdir(f'aclImdb/{type}/neg'):
             with open(f'aclImdb/{type}/neg/{name}', 'r', encoding='utf-8') as file:
                 text = file.read()
                 rate = extract_rate(name)
                 new_row = pd.DataFrame({'name': [name], 'text': [text], 'sentiment': ['negative'], 'rate': [rate]})
                 df = pd.concat([df, new_row], ignore_index=True)
         for name in os.listdir(f'aclImdb/{type}/pos'):
             with open(f'aclImdb/{type}/pos/{name}', 'r', encoding='utf-8') as file:
                 text = file.read()
                 rate = extract_rate(name)
                 new_row = pd.DataFrame({'name': [name], 'text': [text], 'sentiment': ['positive'], 'rate': [rate]})
                 df = pd.concat([df, new_row], ignore_index=True)
         df = df.sample(frac=1).reset_index(drop=True)  #перемешивает строки датафрейма
         return df
```

Рис. 1. Функция creating_df(type)

```
[10]: print(train)
```

```
              name                                               text  \
0        9927_9.txt  Domino is a great movie. It's about a young wo...
1        1439_3.txt  I give 3 stars only for the beautiful pictures...
2        8674_2.txt  The first few minutes of "The Bodyguard" do ha...
3         548_3.txt  I saw the 7.5 IMDb rating on this movie and on...
4      12145_10.txt  I saw this movie once on late night t.v. and k...
...             ...                                                ...
24995    2998_7.txt  Enjoyed viewing this film on TCM and watching ...
24996   10178_3.txt  How did I ever appreciate this dud of a sequel...
24997    4205_8.txt  Samuel Fuller is hardly one of America's great...
24998    3008_7.txt  The unflappable William Powell. He is a joy to...
24999   1545_10.txt  Excellent film. Suzy Kendall will hold your in...

      sentiment  rate
0      positive     9
1      negative     3
2      negative     2
3      negative     3
4      positive    10
...         ...   ...
24995  positive     7
24996  negative     3
24997  positive     8
24998  positive     7
24999  positive    10

[25000 rows x 4 columns]
```

Рис. 2. Пример датафрейма до обработки

3. Предобработка (очистка и подготовка целевой переменной)

Очистка представляет собой подготовку и преобразование необработанных текстовых данных в более чистый, структурированный формат для обучения модели.

Очистка текстов отзывов состоит из следующих шагов:

1. Приведение к нижнему регистру;
2. Удаление тегов </br>, найденных в тексте;
3. Удаление всех вспомогательных символов и пунктуации;
4. Замена всех чисел или слов, содержащих числа (например '50s'), на 'digit';
5. Удаление коротких слов из 1-2 букв;
6. Токенизация текста;
7. Лемматизация токенов;
8. Удаление стоп-слов;
9. Объединение слов обратно в строку;
10. Перемешивание строк датафрейма.

Целевая переменняая sentiment_bin необходима для сравнения методов обучения модели, выбора способа классификации и предсказания рейтинга фильма.

```
[15]: def text_cleaner(df):
          """
          Очищает текстовые данные в датафрейме
          """

          df['text'] = df['text'].str.lower()
          df['text'] = df['text'].str.replace('</br>', '', regex=False)
          df['text'] = df['text'].str.replace(r'[^a-zA-Za-я0-9\s]', '', regex=True)
          df['text'] = df['text'].str.replace(r'\b\d+\b|\b\d+\w*', 'digit', regex=True)
          df['text'] = df['text'].str.replace(r'\b\w{1,2}\b', '', regex=True)
          df['text'] = df['text'].apply(word_tokenize)
          lemmatizer = WordNetLemmatizer()
          df['text'] = df['text'].apply(lambda words: [lemmatizer.lemmatize(word) for word in words])
          df['text'] = df['text'].apply(lambda words: [word for word in words if word not in en_stops])
          df['text'] = df['text'].apply(lambda tokens: ' '.join(tokens))

          df['sentiment_bin'] = df['sentiment'].map({'positive': 1, 'negative': 0}) #создание новой целевой переменной

          return df
```

Рис. 3. Функция подготовки (очистки) текстов для обучения модели

```
[16]: train = text_cleaner(train)
      print(train)
```

```
              name                                               text  \
0        9927_9.txt  domino great movie young woman name domino har...
1        1439_3.txt  give digit star beautiful picture africa rest ...
2        8674_2.txt  first minute bodyguard campy charm open crawli...
3         548_3.txt  saw digit imdb rating movie basis decided watc...
4       12145_10.txt  saw movie late night knew wa best movie ever o...
...              ...                                                ...
24995    2998_7.txt  enjoyed viewing film tcm watching young willia...
24996   10178_3.txt  ever appreciate dud sequel doe throw ball wors...
24997    4205_8.txt  samuel fuller hardly one america great directo...
24998    3008_7.txt  unflappable william powell joy watch screen ma...
24999    1545_10.txt  excellent film suzy kendall hold interest thro...

      sentiment  rate  sentiment_bin
0      positive     9              1
1      negative     3              0
2      negative     2              0
3      negative     3              0
4      positive    10              1
...         ...   ...            ...
24995  positive     7              1
24996  negative     3              0
24997  positive     8              1
24998  positive     7              1
24999  positive    10              1

[25000 rows x 5 columns]
```

Рис. 4. Пример датафрейма с очищенным текстом

4. Импорт библиотеки для обучения модели

Используемая для выбора модели и обучения модели библиотека – **sklearn**, а также ее компоненты, такие как:

- Векторизация текста: CountVectorizer, TfidfVectorizer;
- Методы снижения размерности: TruncatedSVD, PCA;
- Алгоритмы классификации: RandomForestClassifier, GradientBoostingClassifier, Logistic Regression;
- accuracy_score – метрика для оценки точности модели;
- Pipeline – используется для создания конвейера, объединяющего несколько этапов предобработки данных и классификации в единую модель.

5. Обучение модели и проверка точности предсказаний (в том числе выбор модели)

Для предсказания рейтинга фильма рассмотрим два способа:

1. Мультиклассовая классификация с целью предсказания рейтинга фильма, целевые значения - train['rate'];

2. Бинарная классификация отзывов на положительные и негативные, рейтинг фильма определяется как вероятность принадлежности к классу "positive", с округление в большую сторону, целевые значения - train['sentiment_bin'].

Тестировались различные методы преобразования текстовых данных в числовые векторы, такие как CountVectorizer и TfidfVectorizer. Кроме того, были проведены эксперименты с различными алгоритмами классификации: RandomForestClassifier, GradientBoostingClassifier и LogisticRegression.

Это позволило выбрать наиболее подходящую модель для данной задачи на основе метрики точности (accuracy_score).

Мультиклассовая модель

- предсказание рейтинга фильма.

Наиболее точные модели:

[39]:
```
#пайплайн последовательно векторизует числа, после этого применяет алгорим классификации
pipe_text_multiclass_classfication = Pipeline([
    ('vectorizer', TfidfVectorizer(ngram_range=(1, 3), max_features=10000)),
    ('classifier', LogisticRegression(max_iter=5000))
])

#разделение данных
features_train, features_test, target_train, target_test = train['text'], test['text'], train['rate'], test['rate']

pipe_text_multiclass_classfication.fit(features_train, target_train)
prediction = pipe_text_multiclass_classfication.predict(features_test)

#проверка точности
accuracy_multiclass = accuracy_score(target_test, prediction)
```

[40]:
```
print(accuracy_multiclass)
```

```
0.41972
```

Рис. 5. Векторизация TfidfVectorizer и логистическая регрессия

[55]:
```
#пайплайн последовательно векторизует числа, после этого применяет алгорим классификации
pipe_text_multiclass_classfication = Pipeline([
    ('vectorizer', TfidfVectorizer(ngram_range=(1, 3),  max_features=10000)),
    ('classifier', RandomForestClassifier(n_estimators=100, random_state=42))
])

#разделение данных
features_train, features_test, target_train, target_test = train['text'], test['text'], train['rate'], test['rate']

pipe_text_multiclass_classfication.fit(features_train, target_train)
prediction = pipe_text_multiclass_classfication.predict(features_test)

#проверка точности
accuracy_multiclass = accuracy_score(target_test, prediction)
```

[56]:
```
print(accuracy_multiclass)
```

```
0.3758
```

Рис. 6. Векторизация TfidfVectorizer и алгоритм случайного леса (100 деревьев)

[63]:
```
#пайплайн последовательно векторизует числа, после этого применяет алгорим классификации
pipe_text_multiclass_classfication = Pipeline([
    ('vectorizer', TfidfVectorizer(ngram_range=(1, 3),  max_features=10000)),
    ('classifier', RandomForestClassifier(n_estimators=300, random_state=42))
])

#разделение данных
features_train, features_test, target_train, target_test = train['text'], test['text'], train['rate'], test['rate']

pipe_text_multiclass_classfication.fit(features_train, target_train)
prediction = pipe_text_multiclass_classfication.predict(features_test)

#проверка точности
accuracy_multiclass = accuracy_score(target_test, prediction)
```

[64]:
```
print(accuracy_multiclass)
```

```
0.37996
```

Рис. 7. Векторизация TfidfVectorizer и алгоритм случайного леса (300 деревьев)

[74]:
```
#пайплайн последовательно векторизует числа, после этого применяет алгорим классификации
pipe_text_multiclass_classfication = Pipeline([
    ('vectorizer', TfidfVectorizer(ngram_range=(1, 3), max_features=10000)),
    ('pca', PCA(n_components=500)),
    ('classifier', LogisticRegression(max_iter=1000, random_state = 42))
])

#разделение данных
features_train, features_test, target_train, target_test = train['text'], test['text'], train['rate'], test['rate']

pipe_text_multiclass_classfication.fit(features_train, target_train)
prediction = pipe_text_multiclass_classfication.predict(features_test)

#проверка точности
accuracy_multiclass = accuracy_score(target_test, prediction)
```

[75]:
```
print(accuracy_multiclass)
```

```
0.42084
```

Рис. 8. Векторизация TfidfVectorizer, снижение размерности PCA (500 элементов) и логистическая регрессия

[88]:
```
#пайплайн последовательно векторизует числа, после этого применяет алгорим классификации
pipe_text_multiclass_classfication = Pipeline([
    ('vectorizer', TfidfVectorizer(ngram_range=(1, 3), max_features=10000)),
    ('pca', PCA(n_components=1000)),
    ('classifier', LogisticRegression(max_iter=1000, random_state = 42))
])

#разделение данных
features_train, features_test, target_train, target_test = train['text'], test['text'], train['rate'], test['rate']

pipe_text_multiclass_classfication.fit(features_train, target_train)
prediction = pipe_text_multiclass_classfication.predict(features_test)

#проверка точности
accuracy_multiclass = accuracy_score(target_test, prediction)
```

[89]:
```
print(accuracy_multiclass)
```

```
0.42136
```

Рис. 9. Векторизация TfidfVectorizer, снижение размерности PCA (1000 элементов) и логистическая регрессия

Выбранная модель:

[148]:
```
#пайплайн последовательно векторизует числа, после этого применяет алгорим классификации
pipe_text_multiclass_classfication = Pipeline([
    ('vectorizer', TfidfVectorizer(ngram_range=(1, 3), max_features=20000)),
    ('pca', PCA(n_components=1000)),
    ('classifier', LogisticRegression(max_iter=1000, random_state = 42))
])

#разделение данных
features_train, features_test, target_train, target_test = train['text'], test['text'], train['rate'], test['rate']

pipe_text_multiclass_classfication.fit(features_train, target_train)
prediction = pipe_text_multiclass_classfication.predict(features_test)

#проверка точности
accuracy_multiclass = accuracy_score(target_test, prediction)
```

[149]:
```
print(accuracy_multiclass)
```

```
0.42272
```

Рис. 10. Векторизация TfidfVectorizer (увеличено максимальное число фич), снижение размерности PCA (1000 элементов) и логистическая регрессия

Бинарная

- предсказание окраски отзыва.

Наиболее точные модели:

[41]:
```
#пайплайн последовательно векторизует числа, после этого применяет алгорим классификации
pipe_text_binary_classification = Pipeline([
    ('vectorizer', CountVectorizer(ngram_range=(1, 3))),
    ('classifier', LogisticRegression(max_iter=5000))
])

#разделение данных
features_train, features_test, target_train, target_test = train['text'], test['text'], train['sentiment_bin'], test['sentiment_bin']

pipe_text_binary_classification.fit(features_train, target_train)
prediction = pipe_text_binary_classification.predict(features_test)

#проверка точности
accuracy_binary = accuracy_score(target_test, prediction)
```

[42]:
```
print(accuracy_binary)
```

```
0.8832
```

Рис. 11. Векторизация CountVectorizer и логистическая регрессия

[57]:
```
#пайплайн последовательно векторизует числа, после этого применяет алгорим классификации
pipe_text_binary_classification = Pipeline([
    ('vectorizer', TfidfVectorizer(ngram_range=(1, 3),  max_features=10000)),
    ('classifier', RandomForestClassifier(n_estimators=100, random_state=42))
])

#разделение данных
features_train, features_test, target_train, target_test = train['text'], test['text'], train['sentiment_bin'], test['sentiment_bin']

pipe_text_binary_classification.fit(features_train, target_train)
prediction = pipe_text_binary_classification.predict(features_test)

#проверка точности
accuracy_binary = accuracy_score(target_test, prediction)
```

[58]:
```
print(accuracy_binary)
```

```
0.8446
```

Рис. 12. Векторизация TfidfVectorizer и алгоритм случайного леса (100 деревьев)

[65]:
```
#пайплайн последовательно векторизует числа, после этого применяет алгорим классификации
pipe_text_binary_classification = Pipeline([
    ('vectorizer', TfidfVectorizer(ngram_range=(1, 3),  max_features=10000)),
    ('classifier', RandomForestClassifier(n_estimators=300, random_state=42))
])

#разделение данных
features_train, features_test, target_train, target_test = train['text'], test['text'], train['sentiment_bin'], test['sentiment_bin']

pipe_text_binary_classification.fit(features_train, target_train)
prediction = pipe_text_binary_classification.predict(features_test)

#проверка точности
accuracy_binary = accuracy_score(target_test, prediction)
```

[66]:
```
print(accuracy_binary)
```

```
0.85104
```

Рис. 13. Векторизация TfidfVectorizer и алгоритм случайного леса (300 деревьев)

```
[71]: #пайплайн последовательно векторизует числа, после этого применяет алгорим классификации
pipe_text_binary_classification = Pipeline([
    ('vectorizer', TfidfVectorizer(ngram_range=(1, 3), max_features=10000)),
    ('pca', PCA(n_components=500)),
    ('classifier', LogisticRegression(max_iter=5000))
])

#разделение данных
features_train, features_test, target_train, target_test = train['text'], test['text'], train['sentiment_bin'], test['sentiment_bin']

pipe_text_binary_classification.fit(features_train, target_train)
prediction = pipe_text_binary_classification.predict(features_test)

#проверка точности
accuracy_binary = accuracy_score(target_test, prediction)
```

```
[72]: print(accuracy_binary)
```

```
0.87316
```

Рис. 14. Векторизация TfidfVectorizer, снижение размерности PCA (500 элементов) и логистическая регрессия

Выбранная модель:

```
[49]: #пайплайн последовательно векторизует числа, после этого применяет алгорим классификации
pipe_text_binary_classification = Pipeline([
    ('vectorizer', TfidfVectorizer(ngram_range=(1, 3),  max_features=10000)),
    ('classifier', LogisticRegression(max_iter=5000))
])

#разделение данных
features_train, features_test, target_train, target_test = train['text'], test['text'], train['sentiment_bin'], test['sentiment_bin']

pipe_text_binary_classification.fit(features_train, target_train)
prediction = pipe_text_binary_classification.predict(features_test)

#проверка точности
accuracy_binary = accuracy_score(target_test, prediction)
```

```
[50]: print(accuracy_binary)
```

```
0.88472
```

Рис. 15. Векторизация TfidfVectorizer (ограничение фич = 10000) и логистическая регрессия

6. Тестирование

Предсказаны значения выбранных моделей. Рейтинга фильма – мультиклассовой моделью и окраски отзыва – бинарной моделью.

```
[37]: test['predicted_rate_multiclass'] = pipe_text_multiclass_classfication.predict(test['text'])
test['predicted_semtiment_binary'] = pipe_text_binary_classification.predict(test['text'])
test['probability_rate'] = np.ceil(pipe_text_binary_classification.predict_proba(test['text'])[:, 1] * 10)
```

Рис. 16. Предсказания моделей на тестовых данных

К датафрейму добавлены три столбца:

Столбец	Предназначение
Predicted_rate_multiclass	Рейтинг, пересказанный мультиклассовой моделью pipe_text_multiclass_classification
Predicted_sentiment_binary	Эмоциональная окраска, предсказанная моделью pipe_text_binary_classification
Probability_rate	Рейтинг фильма, полученный из вероятности принадлежности к классу «позитивный» модели pipe_text_binary_classification

```
[39]: print('Процент соответствия реального рейтинга с предсказанным мультиклассовой моделью:',
            (test['rate'] == test['predicted_rate_multiclass']).mean() * 100)
      print('Процент соответствия реального рейтинга вероятности принадлежности классу "positive"',
            (test['rate'] == test['probability_rate']).mean() * 100)

      Процент соответствия реального рейтинга с предсказанным мультиклассовой моделью: 42.272
      Процент соответствия реального рейтинга вероятности принадлежности классу "positive" 28.951999999999998

[40]: print('Процент соответствия реальной окраски текста на соснове предсказаний мультиклассовой модели:',
            (test['sentiment_bin'] == np.where(test['predicted_rate_multiclass'] >= 5, 1, 0)).mean() * 100)
      print('Процент соответствия реальной окраски текста и пресказанной бинарной модельюю:',
            (test['sentiment_bin'] == test['predicted_semtiment_binary']).mean() * 100)

      Процент соответствия реальной окраски текста на соснове предсказаний мультиклассовой модели: 87.20400000000001
      Процент соответствия реальной окраски текста и пресказанной бинарной модельюю: 88.428
```

Рис. 17. Оценка точности моделей в предсказании окраски фильма и рейтинга фильма

Маленькая точность мультклассовой модели обусловлена большим числом классов (классы от 1 до 10) и субъективностью оценок. По отзыву точно определить рейтинг фильма невозможно, так, например, негативный отзыв может иметь оценку 2, а модель может предсказать любое достаточно близкое значение (1 или 3).

Определение рейтинга фильма в зависимости от вероятности принадлежности к классу «позитивный» оказалось **неэффективным.**

Алгоритм определения окраски по рейтингу фильма: если рейтинг фильма от 1 до 5 включительно, то отзыв негативный, от 6 до 10 включительно – положительный.

Для создания сервиса выбрана **мультиклассовая модель** с лучшим показателем точности – **42.272%**. Эмоциональную окраску обе модели предсказывают одинаково.

7. Экспорт модели

Экспорт выполнен с помощью библиотеки **joblib**.

Создание веб-сервиса

Сервис создан на фреймворке Django для языка Python. Сервис работает на главной странице проекта. Для сервиса создано приложение main. Структура проекта:

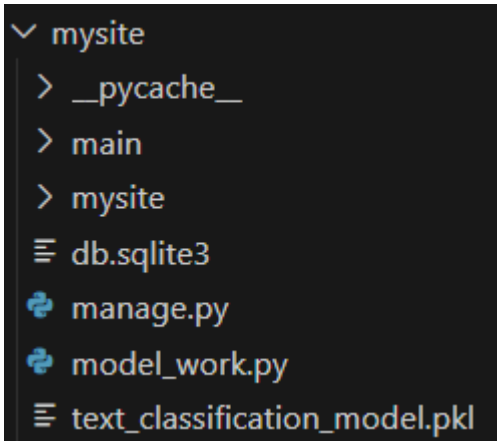

Рис. 18. Структура веб-сервиса

Для импорта модели и взаимодействия с небыли разработаны две функции model_load() и model_answer():

В model_load() описывается процесс загрузки модели с помощью библиотеки joblib.

В model_answer() функция предсказывает оценку текста и вычисляет бинарную окраску, исходя из значения предсказанного .

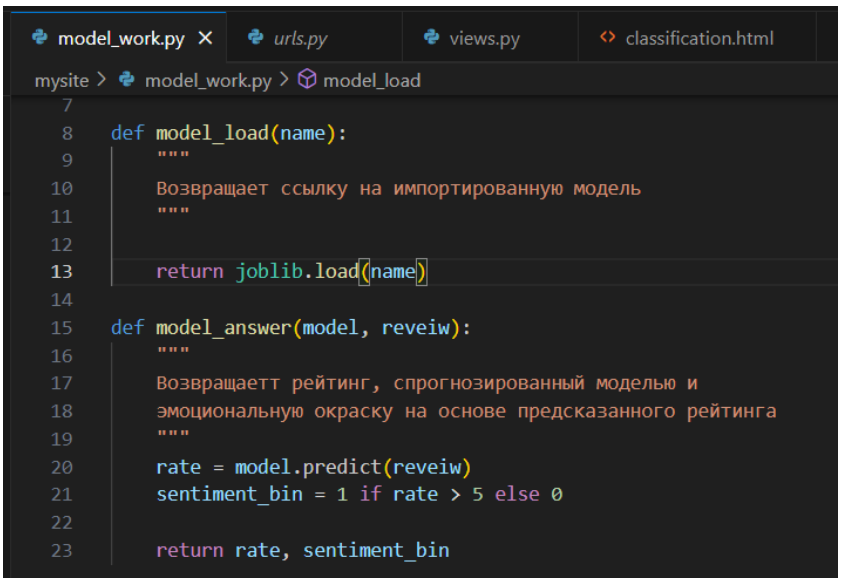

Рис 19. Функции model_load() и model_answer()

Функция classification_form(request), обрабатывающая GET и POST запросы, обращается к функции, которая загружает модель единожды get_model(). Располагаются в main/views.py.

```python
MODEL = None #глобальная переменная, чтобы загружать модель единожды при запуске приложения

def get_model():
    """
    Изменяет глобаьную переменную MODEL, содержащую модель sklearn.
    Если модель еще не загружена в память, она загружается из
    сериализованного файла (в формате pickle).
    """

    global MODEL
    if MODEL is None:
        name = 'text_classification_model.pkl'
        MODEL = model_load(name)

    return MODEL
```

Рис. 20. Функция get_model()

```python
def classification_form(request):

    """
     Обрабатывает POST и GET запросы для формы классификации текста
    """

    model = get_model()
    rate = None
    sentiment_bin = None

    if request.method == 'POST':
        form = TextForm(request.POST)
        if form.is_valid():
            text = form.cleaned_data['text']
            rate, sentiment_bin = model_answer(model, [text])
            rate = rate[0]

    else:
        form = TextForm()

    return render(request, 'main/classification.html', {
        'form': form,
        'rate': rate,
        'sentiment_bin': sentiment_bin
    })
```

Рис. 21. Функция-обработчик classification_form(request)

Примеры работы сервиса на тестовых данных:

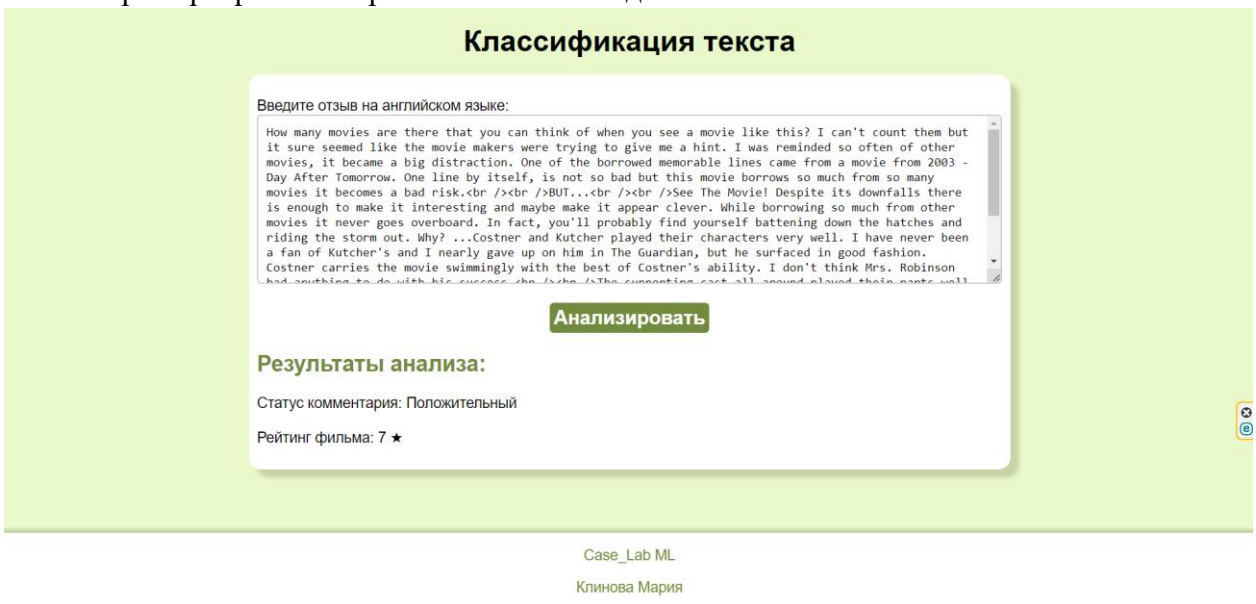

Рис. 22. Классификация отзыва из директории test/pos отзыв 3_7.txt

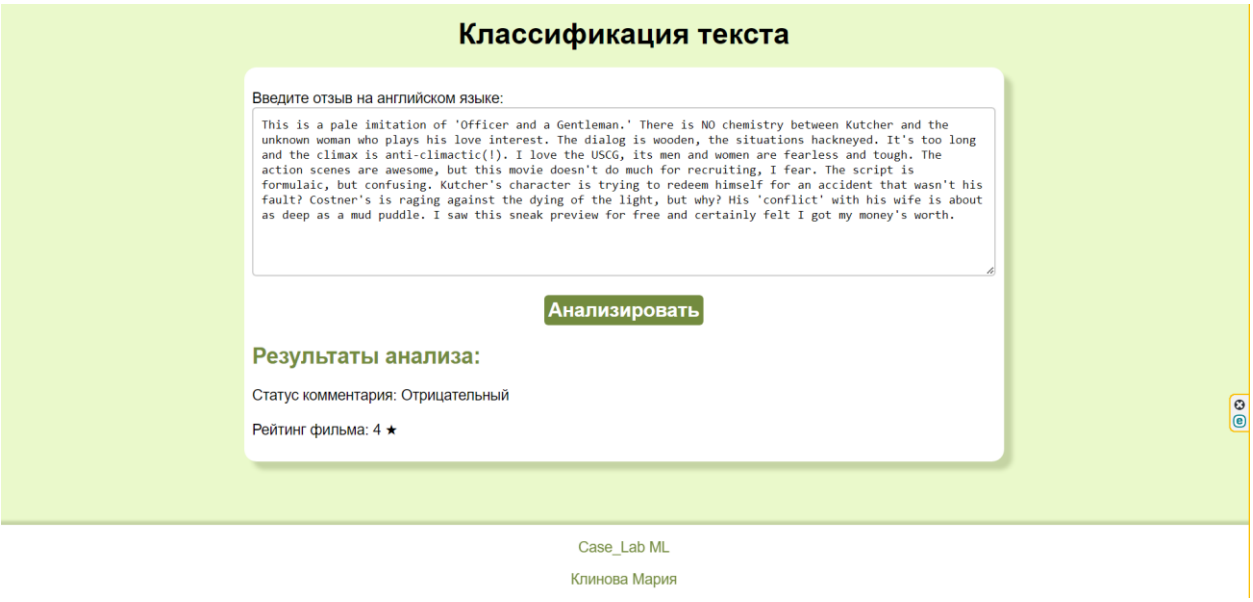

Рис. 23. Классификация отзыва из директории test/neg отзыв 1_3.txt

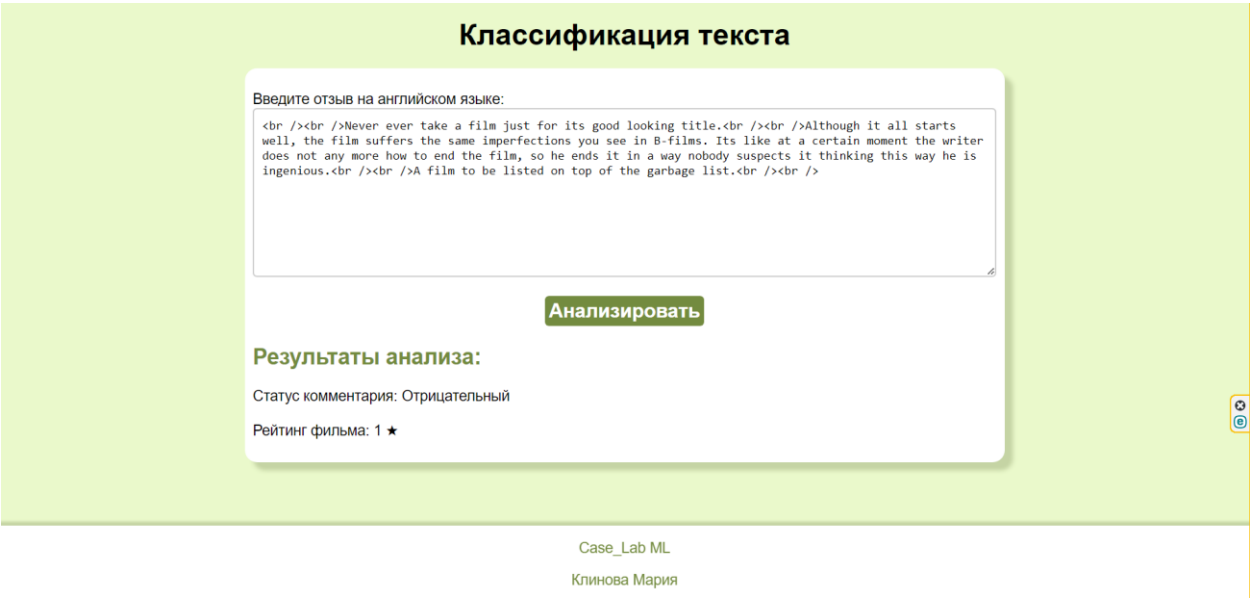

Рис. 24. Классификация отзыва из директории test/neg отзыв 98_1.txt

Личный отзыв о фильме:

«I liked this movie, good acting. The main character is played by my favorite actress. The movie is funny, as intended, but there are big gaps in the plot. For example, at the end of the movie, the heroine comes to a big city and builds a career, and it is completely unclear how she achieved success, only the result is shown.»

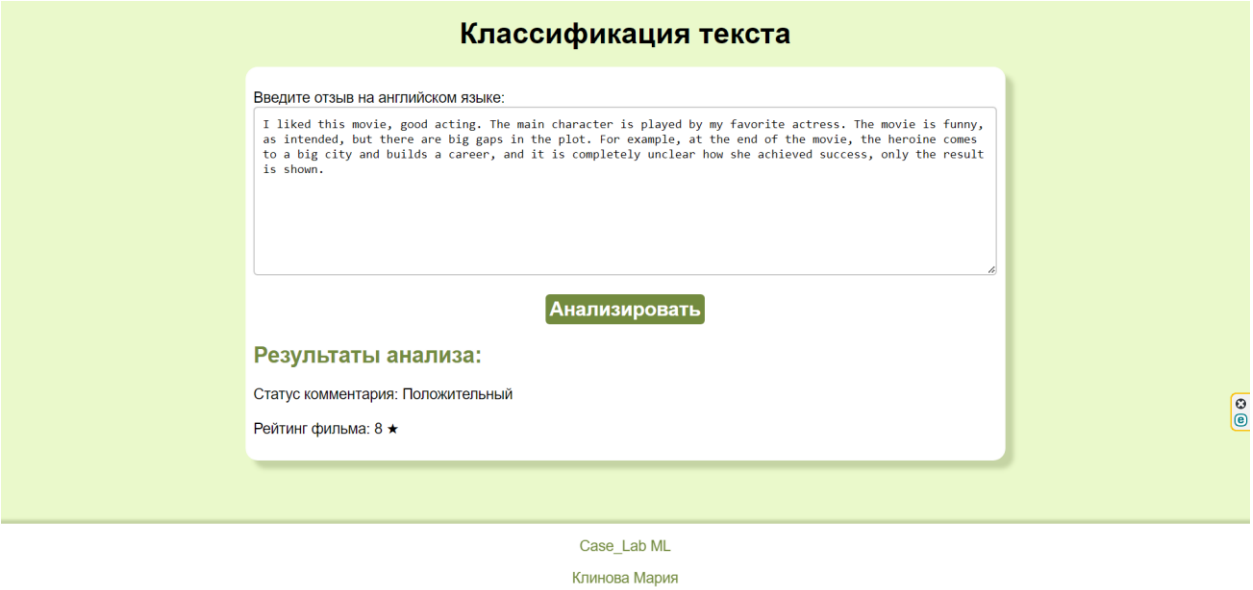

Рис. 25. Классификация личного отзыва

Результаты работы:

Отзыв	Реальный рейтинг фильма	Предсказанный рейтинг
test/pos/3_7.txt	7	7
test/neg/1_3.txt	3	4
test/neg/98_1.txt	1	1
Личный	-	8

Веб – сервис работает корректно.

Деплой проекта на Render.com

Ссылка на рабочий проект: https://case-lab-ml-klinova.onrender.com/

ВНИМАНИЕ!

Первый запуск странички может быть долгим из-за ограничений бесплатного тарифа render.com.

В целях экономии ресурсов площадка отключает проект, если по ссылке не переходят определенное количество времени. Затем при первом же переходе сервис запускает проект заново. Пожалуйста, дождитесь загрузки странички, чтобы проверить работоспособность сервиса.

Для запуска отключен режим отладки и в доступные хосты добавлен адрес для хостинга на площадке render.com.

```
27  DEBUG = False
28
29  ALLOWED_HOSTS = ['case-lab-ml-klinova.onrender.com', 'localhost', '127.0.0.1']
30
```

Рис. 26. Файл settings.py