Отчет о разработке тестового проекта Django

Введение

Проект представляет собой веб-приложение на базе фреймворка Django, которое выполняет обработку отзывов с помощью машинного обучения. Целью данного проекта является создание системы для предсказания настроения (положительное или отрицательное) по текстам отзывов на основе обученной модели машинного обучения. В данном отчете подробно описаны шаги разработки от первоначальной подготовки до настройки всех необходимых компонентов приложения, за исключением финального развертывания.

1. Подготовка проекта

1.1. Установка Django

На первом этапе был создан новый проект на Django, для чего потребовалось установить фреймворк Django и другие зависимости. Для этого был выполнен следующий набор команд:

Bash

```
pip install django
django-admin startproject movie_review
```

Данная команда создала структуру проекта с основными файлами: `settings.py`, `urls.py`, `wsgi.py`, а также файл `manage.py`, который используется для управления проектом.

1.2. Настройка виртуального окружения

Важным шагом является настройка виртуального окружения для изоляции зависимостей проекта. Было создано и активировано виртуальное окружение:

Bash

```
python3 -m venv myenv
source myenv/bin/activate
```

После активации окружения установлены все зависимости, необходимые для работы проекта.

2. Настройка машинного обучения

2.1. Подготовка данных и обучение модели

Используемый датасет содержит отзывы с их метками (положительные или отрицательные). На этом этапе была использована модель машинного обучения для классификации отзывов на основе текстов.

1. **Загрузка и предобработка данных**: датасет был загружен, и отзывы были преобразованы в числовые векторы с использованием TF-IDF (термино-частотного векторизатора). Было произведено деление данных на обучающую и тестовую выборки.

   python

   ```
   from sklearn.feature_extraction.text import TfidfVectorizer
   from sklearn.model_selection import train_test_split

   tfidf = TfidfVectorizer(max_features=5000)
   X_train, X_test, y_train, y_test = train_test_split(reviews,
   labels, test_size=0.2)
   X_train_tfidf = tfidf.fit_transform(X_train)
   ```

2. **Обучение модели**: для классификации отзывов была выбрана логистическая регрессия. Модель была обучена на тренировочных данных:

   python

   ```
   from sklearn.linear_model import LogisticRegression

   model = LogisticRegression()
   model.fit(X_train_tfidf, y_train)
   ```

3. **Сохранение модели и векторизатора**: после обучения модель и TF-IDF векторизатор были сохранены для дальнейшего использования в веб-приложении:

   python

   ```
   import joblib

   joblib.dump(model, 'models/logistic_model.pkl')
   joblib.dump(tfidf, 'models/tfidf.pkl')
   ```

2.2. Оценка точности модели

После обучения модель была протестирована на тестовой выборке для оценки точности и других метрик:

python

```
accuracy = model.score(X_test_tfidf, y_test)
print(f'Accuracy: {accuracy}')
```

Точность модели была достаточной для того, чтобы использовать её в продакшене.

3. Разработка веб-приложения на Django

3.1. Создание приложения Django

Для реализации функционала по обработке отзывов было создано новое приложение `reviews`:

bash

```
python manage.py startapp reviews
```

3.2. Модели и формы

Внутри приложения `reviews` была создана форма для ввода текста отзыва пользователем:

python

```
from django import forms

class ReviewForm(forms.Form):
    text = forms.CharField(widget=forms.Textarea)
```

3.3. Вьюшки и URL-маршруты

Создана вьюшка для обработки POST-запросов, отправляемых пользователем через форму. Эта вьюшка принимает текст отзыва, использует загруженную модель для предсказания и сохраняет результат в базу данных:

python

```
from django.shortcuts import render
from.forms import ReviewForm
from.models import Review
import joblib

model = joblib.load('models/logistic_model.pkl')
tfidf = joblib.load('models/tfidf.pkl')

def review_view(request):
    if request.method == 'POST':
        form = ReviewForm(request.POST)
        if form.is_valid():
            review_text = form.cleaned_data['text']
            review_vector = tfidf.transform([review_text])
            prediction = model.predict(review_vector)[0]
            sentiment = 'positive' if prediction == 1 else
'negative'
            Review.objects.create(text=review_text,
rating=prediction, sentiment=sentiment)
            return render(request, 'result.html', {'sentiment':
sentiment})
    else:
        form = ReviewForm()
    return render(request, 'review_form.html', {'form': form})
```

В файле `urls.py` было добавлено соответствие маршрутов для отображения формы и результатов классификации:

python

```
from django.urls import path
from.import views

urlpatterns = [
    path('', views.review_view, name='review'),
]
```

3.4. Сбор и обслуживание статических файлов

В проекте были настроены статические файлы (CSS и JavaScript). В файле `settings.py` были добавлены следующие параметры:

python

```
STATIC_URL = '/static/'
STATIC_ROOT = os.path.join(BASE_DIR, 'static')
```

Для их сбора была выполнена команда:

bash

```
python manage.py collectstatic
```

3.5. Тестирование приложения

Приложение было протестировано локально, чтобы убедиться, что предсказания модели и работа с базой данных выполняются корректно. В качестве веб-сервера для разработки был использован встроенный сервер Django:

bash

```
python manage.py runserver
```

Приложение было протестировано через браузер по адресу `http://127.0.0.1:8000`.

Заключение

Данный отчет описывает основные этапы разработки проекта: от подготовки модели машинного обучения до создания Django-приложения с формами, базой данных и веб-интерфейсом. Приложение было протестировано локально и готово к развертыванию на выбранном хостинге.