Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Теоретическая информатика и компьютерные технологии»

Лабораторная работа Django

по курсу «Компьютерные сети»

«Анализ отзывов с использованием SVM»

Студент группы ИУ9-32Б Лавров Р. Д.

Преподаватель Посевин Д. П.

Москва 2025

1 Задание

Создать в models.py модель отзыва с полями: User (ForeignKey), Text (TextField или CharField), Sentiment (BooleanField) Авторизованный пользователь через веб интерфейс печатает отзыв. Текст отзыва отправляется на сервер. На сервере подгружается модель SVM (предварительно обученную) (сохранить и загрузить модель можно либо через joblib, либо через pickle), в модель подается текст отзыва (предварительно веректоризованный), на выходе она выдает либо 0 (негативная тональность, фильм не понравился) либо 1 (позитивная тональность, фильм понравился). Далее создается новый объект класса отзыв (с соответствующими значениями полей User, Text, Sentiment (сгенерирован SVM)) и сохраняется в базе данных.

1. Добавить дополнительно кнопку удаления отзыва из базы данных;

2 Результаты

Исходный код 1.

Листинг 1: models.py

```
from django.db import models
from django.contrib.auth.models import User
from django.core.validators import MaxLengthValidator
import secrets

class Review(models.Model):
    user = models.ForeignKey(User, on_delete=models.CASCADE, related_name="reviews")
    text = models.TextField()
    sentiment = models.BooleanField()  # True = позитивный, False = негативный
    created_at = models.DateTimeField(auto_now_add=True)

    def __str__(self):
        return f"{self.user.username} — {self.text[:20]}"

class UserToken(models.Model):
    user = models.ForeignKey(
        User,
        editable=False,
        on_delete=models.CASCADE,
        related_name="tokens",
        verbose_name="Пользователь",
```

```
    )
    name = models.CharField(
        max_length=100,
        verbose_name="Название токена",
        validators=[MaxLengthValidator(100)],
    )
    token = models.CharField(
        max_length=64, unique=True, editable=False, verbose_name="Значение токена"
    )
    created_at = models.DateTimeField(auto_now_add=True, verbose_name="Дата
    создания")


    class Meta:
        verbose_name = "Токен пользователя"
        verbose_name_plural = "Токены пользователей"
        ordering = ["-created_at"]
        unique_together = [["user", "name"]]

    def __str__(self):
        return f"{self.name} ({self.user.username})"

    def save(self, *args, **kwargs):
        if not self.token:
            self.token = self.generate_token()
        super().save(*args, **kwargs)

    @staticmethod
    def generate_token(length=32):
        return secrets.token_hex(length)
```

Листинг 2: views.py

```
import joblib
import os

# Пути к файлам предполагается(, что они лежат в корне проекта)
BASE_DIR = os.path.dirname(os.path.dirname(os.path.abspath(__file__)))
VECTORIZER_PATH = os.path.join(BASE_DIR, "ml/vectorizer.pkl")
MODEL_PATH = os.path.join(BASE_DIR, "ml/sentiment_model.pkl")

# Загружаем при старте один( раз)
vectorizer = joblib.load(VECTORIZER_PATH)
sentiment_model = joblib.load(MODEL_PATH)

import json
```

```
from django.contrib.auth.mixins import LoginRequiredMixin
from django.core.exceptions import ValidationError
from django.http import JsonResponse
from django.shortcuts import render, redirect
from django.views import View
from .models import UserToken, Review

class DeleteTokenView(LoginRequiredMixin, View):
    def post(self, request, token_id):
        try:
            token = UserToken.objects.get(id=token_id, user=request.user)
            token.delete()
            return JsonResponse({"status": "ok"})
        except UserToken.DoesNotExist:
            return JsonResponse({"status": "error", "message": "Token not found"}, status=404)


class DeleteReviewView(LoginRequiredMixin, View):
    def post(self, request, review_id):
        try:
            review = Review.objects.get(id=review_id, user=request.user)
            review.delete()
            return JsonResponse({"status": "ok"})
        except Review.DoesNotExist:
            return JsonResponse({"status": "error", "message": "Review not found"}, status
    =404)

from django.utils.functional import cached_property

class AddReviewView(View):
    def post(self, request):
        text = request.POST.get("text", "")
        sentiment = self.classify_text(text)

        Review.objects.create(
            text=text,
            sentiment=sentiment,
            user=request.user
        )

        return redirect("reviews")

    def classify_text(self, text):
        X = vectorizer.transform([text])
        prediction = sentiment_model.predict(X)[0]
        return bool(prediction)
```

```python

class ReviewPage(LoginRequiredMixin, View):
    def get(self, request):
        reviews = Review.objects.all().order_by("-id")
        return render(request, "myapp/reviews.html", {"reviews": reviews})

class Index(LoginRequiredMixin, View):
    def get(self, request):
        return redirect("reviews")
        user = request.user
        user_tokens = user.tokens.all()
        return render(request, "myapp/index.html", {"user_tokens": user_tokens})
    def post(self, request):
        data = json.loads(request.body.decode("utf-8"))
        if "token_name" not in data.keys():
            JsonResponse({"errors": "Bad request"}, status=400)
        token_name = data["token_name"]
        user = request.user
        new_token = UserToken(
            user=user,
            name=token_name,
        )
        try:
            new_token.full_clean()
            new_token.save()
            return JsonResponse(
                {
                    "token_name": new_token.name,
                    "token": new_token.token,
                },
                status=200,
            )
        except ValidationError as e:
            return JsonResponse({"detail": e.messages}, status=400)
```

Листинг 3: index.html

```
{% load static %}
<!DOCTYPE html>
<html lang="ru">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <link rel="stylesheet" href="{% static 'myapp/css/account.css' %}" />
    <title>My Site</title>
    <script src="{% static 'myapp/js/account.js' %}" type="module"></script>
```

```
</head>

<body>
  <header class="header">
    <div class="user-info">
      <span class="username">{{user.username}}</span>
    </div>
    <form method="post" action="{% url 'logout' %}">
      {% csrf_token %}
      <button class="logout-btn"Выйти></button>
    </form>
  </header>
  <div class="container">
    <div class="column">
      <h2Сгенерировать> токен</h2>

      <div class="form-group">
        <label for="token-select"Название> токена:</label>
        <input
          type="text"
          id="token-name"
          placeholder="Вставьте название сюда"
          maxlength="100"
        />
        <div class="error-message" id="token-select-error"Пусто></div>
      </div>

      <button class="submit-btn" id="apply-btn"Сгенерировать></button>
      <div class="token-list" id="firstTokenList">
        <h3 id="firstTokenListTitle"Токены>:</h3>
        {% for user_token in user_tokens %}
        <div class="token">
          <div class="token-info">
            <span class="token-name">{{user_token.name}}</span>
          </div>
          <div class="token-item">
            <span class="token-value">{{user_token.token}}</span>
            <button class="copy-btn"Копировать></button>
          </div>
        </div>
        {% endfor %}
      </div>
    </div>
  </div>
  <script>
    const token = "{{csrf_token}}",
```

```
56            index = "{% url 'index' %}";
57        </script>
58    </body>
59 </html>
```

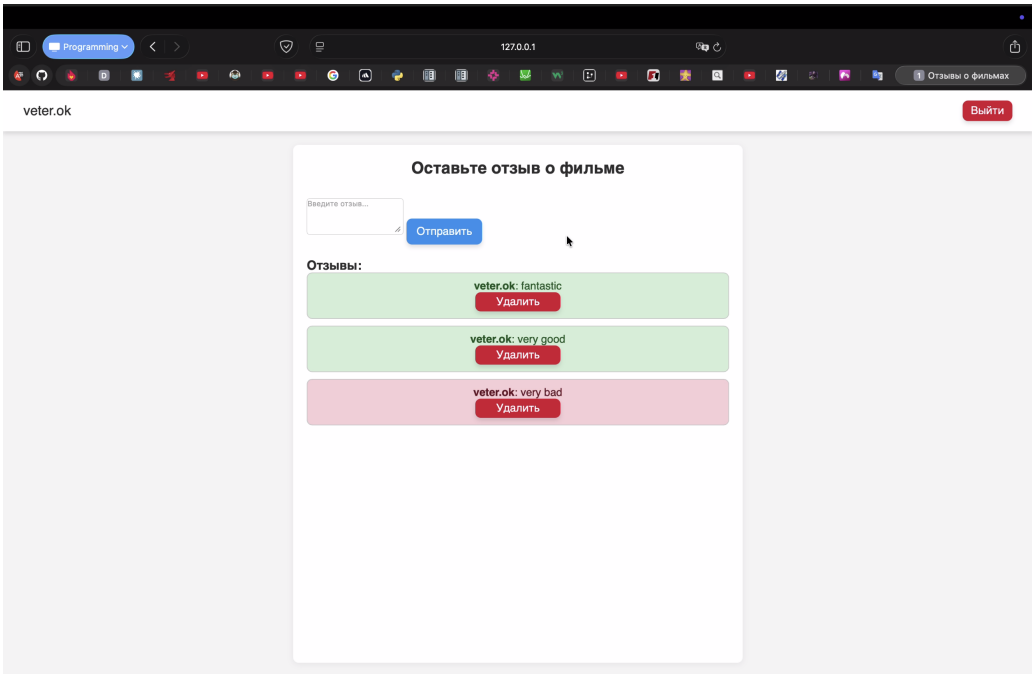

Рис. 1 — Результат работы

3 Вывод

В ходе выполнения лабораторной работы была создана модель отзыва в Django с необходимыми полями. Были изучены основы работы с моделями в Django и взаимодействие с базой данных.