Н И Т У « М И С И С »

Кафедра инженерной кибернетики

ОТЧЕТ

ПО

ЛАБОРАТОРНОЙ РАБОТЕ №1

«Разработка демонстрационного прототипа программного приложения для решения специализированной задачи интеллектуальной обработки и анализа информации с использованием API современных ИИ-сервисов»

учебная дисциплина «Экспертные и рекомендательные системы»

Студент: Гаврилюк А.В. гр. БПМ-22-ПО-3

Преподаватель: Хонер П. Д.

Оценка: _____

Дата защиты: 29.09.2025

2025 г.

Введение

Цель данной лабораторной работы заключалась в использовании двух API, основанных на методах ИИ, для решения типовой задачи анализа информации. В процессе работы требовалось сравнить функциональные возможности этих сервисов, оценить, как каждое API справилось с поставленной задачей и сравнить полученные результаты между собой.

1. Поставленная задача

Суть задачи заключалась в сравнении двух API для анализа тональности текста (сентимент анализа). В качестве тестовой выборки было решено взять данные из датасета с отзывами о приложениях.

Задача заключалась в сравнении двух API для генерации рецепта коктейля, с целью оценки полноты описания необходимых ингредиентов, пошаговых действий по приготовлению и дополнительной информации.

В качестве результата работы приложение предоставляет пользователю тон переданного API текста.

2. Выбранные средства разработки программного обеспечения

Для разработки приложения были использованы два API, доступ к которым был получен посредством использования платформы RapidAPI:

- Sentiment Analysis API : Данное API позволяет определить тональность текста. Оно использует ИИ-методы, которые позволяют опираться на контекст текста в целом, а не на определенные слова. На выходе может выдать три лейбла: Positive, Neutral и Negative.
- Sentiment by API-Ninjas: Это инструмент, позволяющий быстро и довольно качественно определять тональность текста. На выходе также может выдать три лейбла: Positive, Neutral и Negative.

В качестве языка разработки было решено использовать Python. Для написания пользовательского интерфейса было решено использовать библиотеку PyQt5.

3. Описание и примеры исходных данных

В качестве тестовой выборки было решено использовать несколько записей из датасета с отзывами о приложениях, основной датасет можно найти по следующей ссылке: https://huggingface.co/datasets/sealuzh/app_reviews .

Ниже представлены исходные данные, на которых приложение было протестировано:

1. Faster & nicer The new version compatible with Nougat looks awesome! It's also incredibly much faster on connection (I wish the Ubuntu client connected that quickly!). Really nice job on the update thanks for updating it to work on Nougat too :) [Label: Positive]
2. Could handle disconnects better This is a well made application in general and works great when you have a solid connection. But iffy connections can be very frustrating when the screen goes back to login mid-sentence. Also highlighting should never use the same color as a username. [Label: Neutral]
3. Good app. This app is really good. I only gave it 4 stars cause the menu options blur badly while scrolling. Other than that it's perfect and very helpful. [Label: Positive]

Рассмотрим результаты работы двух API на каждом их этих примеров.

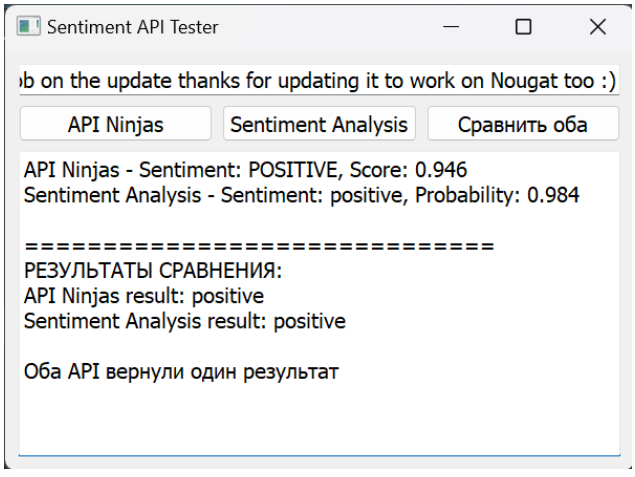

Рис. 1 – Отзыв под номером 1.

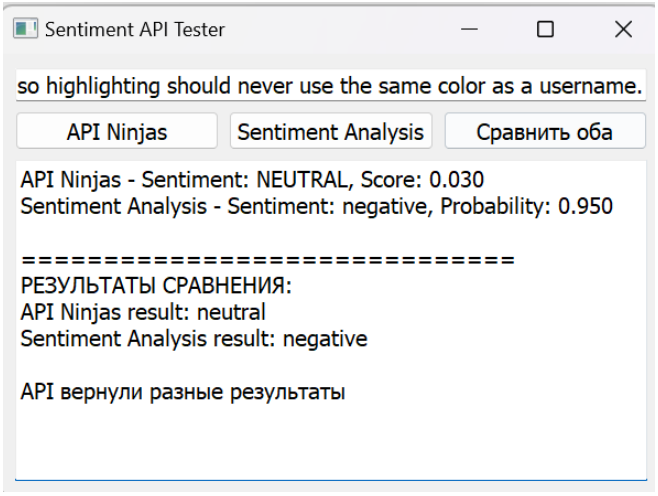

Рис. 2 – Отзыв под номером 2.

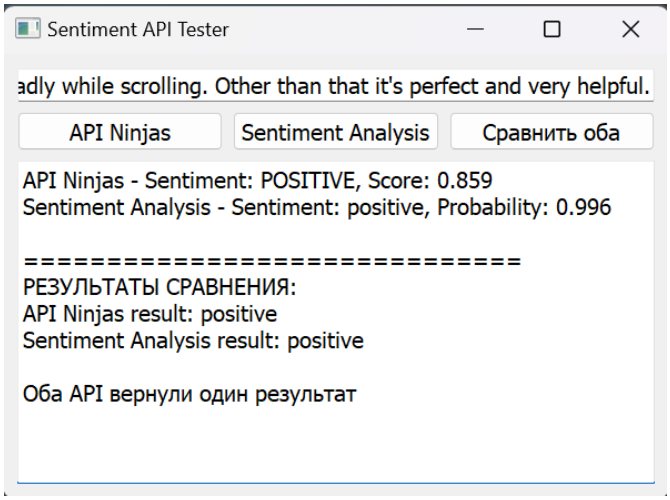

Рис. 3 – Отзыв под номером 3.

4. Результаты

В целом, два сравниваемых API показали хорошие результаты, основное их отличие заключается в возвращаемых результатах – API by API-Ninjas вместе с лейблом возвращает метрику Score, а Sentiment Analisys API возвращает лейбл и метрику Probability, что немного затрудняет их сравнение.

На тестовых данных API by API-Ninjas показало результат 3/3 правильно определенных тональностей, в то время как Sentiment Analisys API ошиблось на 2 отзыве, показав результат 2/3.

Таким образом, можно сказать что более точно определяет тональность API by API-Ninjas.

5. Исходный код

Api_controller.py

```
import requests
import json
from data_handler import get_api_config

# Получаем конфигурацию API
config = get_api_config()


def make_api_request_get(url: str, headers: dict, params: dict = None) -> dict:
    """
    Функция для выполнения GET запроса к API и возврата ответа в формате JSON
    :url: Эндпойнт
    :headers: Заголовки запроса
    :params: Параметры запроса
    :return: Словарь с ответом
    """
    try:
        response = requests.get(url, headers=headers, params=params)
        response.raise_for_status()
        return response.json()
    except requests.exceptions.HTTPError as err:
        print(f"HTTP-ошибка: {err}")
        return None


def make_api_request_post(url: str, headers: dict, payload: dict = None) ->
dict:
    """
    Функция для выполнения POST-запроса к API и возврата JSON-ответа
    :url: Эндпойнт
    :headers: Заголовки запроса
    :payload: Данные для запроса
    :return: Словарь с ответом
    """
    try:
        response = requests.post(url, headers=headers, json=payload)
        response.raise_for_status()
        return response.json()
    except requests.exceptions.HTTPError as err:
        print(f"HTTP-ошибка: {err}")
        return None


def analyze_ninjas_api(text: str) -> dict:
    """
    Функция для анализа отзыва через API Ninjas Sentiment API.
    :text: Текст отзыва
    :return: Словарь с результатами
    """
    headers = {
        "X-RapidAPI-Key": config["rapidapi_key"],
        "X-RapidAPI-Host": config["ninjas_host"],
    }

    params = {"text": text}
    result = make_api_request_get(config["ninjas_url"], headers, params)
```

```
    if result:
        return {
            'success': True,
            'data': result,
            'api_name': 'API Ninjas'
        }
    else:
        return {
            'success': False,
            'error': 'API request failed',
            'api_name': 'API Ninjas'
        }


def analyze_sentiment_analysis_api(text: str) -> dict:
    """
    Функция для анализа отзыва через Sentiment Analysis API.
    :text: Текст отзыва
    :return: Словарь с результатами
    """
    headers = {
        "X-RapidAPI-Key": config["rapidapi_key"],
        "X-RapidAPI-Host": config["sentiment_host"],
        "Content-Type": "application/json",
        "Accept": "application/json"
    }

    payload = [{
        "id": "1",
        "language": "en",
        "text": text
    }]

    result = make_api_request_post(config["sentiment_url"], headers, payload)

    if result:
        return {
            'success': True,
            'data': result,
            'api_name': 'Sentiment Analysis'
        }
    else:
        return {
            'success': False,
            'error': 'Ошибка запроса к API',
            'api_name': 'Sentiment Analysis'
        }
```

data_handler.py

```
import os
from dotenv import load_dotenv


def get_api_config():
    """
    Функция для загрузки конфигурации API из переменных окружения
    :return: Словарь с настройками API
    """
    # Загружаем переменные из .env файла
    load_dotenv()

    config = {
        # RapidAPI ключ (общий для всех API)
        "rapidapi_key": os.getenv("RAPIDAPI_KEY"),
```

```
        # Pizza API (из вашего примера)
        "pizza_url": os.getenv("PIZZA_API_URL"),
        "pizza_host": os.getenv("PIZZA_API_HOST"),

        # Wizzard API (из вашего примера)
        "wizzard_url": os.getenv("WIZZARD_API_URL"),
        "wizzard_host": os.getenv("WIZZARD_API_HOST"),

        # API Ninjas
        "ninjas_url": os.getenv("NINJAS_API_URL"),
        "ninjas_host": os.getenv("NINJAS_API_HOST"),

        # Sentiment Analysis API
        "sentiment_url": os.getenv("SENTIMENT_API_URL"),
        "sentiment_host": os.getenv("SENTIMENT_API_HOST"),
    }

    return config
```

response_comparer.py

```
from typing import Dict, Optional


def normalize_sentiment_result(api_result: dict, api_name: str) -> Optional[str]:
    """
    Извлекает тональность и нормализует к единому формату
    :api_result: Результат работы API
    :api_name: Название API ('API Ninjas' или 'Sentiment Analysis')
    :return: Нормализованная тональность ('positive', 'negative', 'neutral') или None
   в случае ошибки
    """
    if not api_result or not api_result.get('success'):
        return None

    try:
        if api_name == 'API Ninjas':
            # API Ninjas возвращает: {"score": 0.123, "text": переданный текст, "sen-
   timent": "POSITIVE"}
            label = api_result['data'].get('sentiment', '').lower()
            return label

        elif api_name == 'Sentiment Analysis':
            # Sentiment Analysis возвращает массив с результатами
            data = api_result['data']
            if isinstance(data, list) and len(data) > 0:
                result = data[0]
                predictions = result.get('predictions', [])
                if predictions:
                    # Находим предсказание с максимальной вероятностью
                    best_prediction = max(predictions, key=lambda x: x.get('proba-
   bility', 0))
                    sentiment = best_prediction.get('prediction', '').lower()
                    return sentiment

        return None

    except (KeyError, TypeError, IndexError):
        return None


def compare_api_results(ninjas_result: dict, sentiment_result: dict) -> dict:
    """
    Сравнивает результаты двух API и определяет, совпадают ли они
    :ninjas_result: Результат API Ninjas
```

```
    :sentiment_result: Результат Sentiment Analysis API
    :return: Словарь с результатами сравнения
    """
    # Нормализуем результаты
    ninjas_sentiment = normalize_sentiment_result(ninjas_result, 'API Ninjas')
    sentiment_analysis_sentiment   =   normalize_sentiment_result(sentiment_result,
   'Sentiment Analysis')

    # Проверяем, получены ли результаты от обеих API
    both_successful = ninjas_sentiment is not None and sentiment_analysis_sentiment
   is not None

    # Сравниваем результаты
    results_match = False
    if both_successful:
        results_match = ninjas_sentiment == sentiment_analysis_sentiment

    return {
        'ninjas_sentiment': ninjas_sentiment,
        'sentiment_analysis_sentiment': sentiment_analysis_sentiment,
        'both_api_successful': both_successful,
        'results_match': results_match,
        'comparison_status':     get_comparison_status(ninjas_sentiment,     senti-
   ment_analysis_sentiment)
    }


def get_comparison_status(sentiment1: Optional[str], sentiment2: Optional[str]) ->
   str:
    """
    Определяет статус сравнения результатов
    :sentiment1: Результат первого API
    :sentiment2: Результат второго API
    :return: Статус сравнения
    """
    if sentiment1 is None and sentiment2 is None:
        return "both_failed"
    elif sentiment1 is None:
        return "ninjas_failed"
    elif sentiment2 is None:
        return "sentiment_analysis_failed"
    elif sentiment1 == sentiment2:
        return "match"
    else:
            return "mismatch"
```

user_interface.py

```
import sys
from PyQt5.QtWidgets import (
    QApplication,
    QMainWindow,
    QWidget,
    QVBoxLayout,
    QHBoxLayout,
    QLineEdit,
    QPushButton,
    QTextEdit
)

from api_controller import analyze_ninjas_api, analyze_sentiment_analysis_api
from response_comparer import compare_api_results


class SentimentAnalyzerGUI(QMainWindow):
    def __init__(self):
```

```
        super().__init__()
        self.init_ui()

    def init_ui(self):
        """Инициализация интерфейса"""
        self.setWindowTitle("Sentiment API Tester")
        self.setGeometry(100, 100, 600, 400)

        app_font = QApplication.font()
        app_font.setPointSize(10)
        QApplication.setFont(app_font)

        # Основной виджет
        main_widget = QWidget()
        self.setCentralWidget(main_widget)

        # Основной layout
        layout = QVBoxLayout()
        main_widget.setLayout(layout)

        # Поле ввода текста
        self.text_input = QLineEdit()
        self.text_input.setPlaceholderText("Введите текст")
        layout.addWidget(self.text_input)

        # Layout для кнопок
        button_layout = QHBoxLayout()

        self.btn_ninjas = QPushButton("API Ninjas")
        self.btn_ninjas.clicked.connect(self.call_ninjas_api)
        button_layout.addWidget(self.btn_ninjas)

        self.btn_sentiment = QPushButton("Sentiment Analysis")
        self.btn_sentiment.clicked.connect(self.call_sentiment_api)
        button_layout.addWidget(self.btn_sentiment)

        self.btn_compare = QPushButton("Сравнить оба")
        self.btn_compare.clicked.connect(self.compare_apis)
        button_layout.addWidget(self.btn_compare)

        layout.addLayout(button_layout)

        # Область для вывода результатов
        self.output_area = QTextEdit()
        self.output_area.setReadOnly(True)
        layout.addWidget(self.output_area)

    def append_output(self, text):
        """Добавление текста в вывод"""
        current_text = self.output_area.toPlainText()
        if current_text:
            self.output_area.setText(current_text + "\n" + text)
        else:
            self.output_area.setText(text)

    def format_api_result(self, result: dict, api_name: str) -> str:
        """Форматирует результат API для вывода"""
        if not result or not result.get('success'):
            error_msg = result.get('error', 'Unknown error') if result else 'No re-
sponse'
            return f"{api_name} - ERROR: {error_msg}"

        try:
            if api_name == "API Ninjas":
                data = result['data']
                sentiment = data.get('sentiment', 'Unknown')
```

```
                score = data.get('score', 0)
                return f"{api_name} - Sentiment: {sentiment}, Score: {score:.3f}"

            elif api_name == "Sentiment Analysis":
                data = result['data']
                if isinstance(data, list) and len(data) > 0:
                    predictions = data[0].get('predictions', [])
                    if predictions:
                        best = max(predictions, key=lambda x: x.get('probability',
0))
                        sentiment = best.get('prediction', 'Unknown')
                        probability = best.get('probability', 0)
                        return f"{api_name} - Sentiment: {sentiment}, Probability:
{probability:.3f}"

            return f"{api_name} - Невозможно прочесть ответ"

        except (KeyError, TypeError, IndexError) as e:
            return f"{api_name} - Parse error: {str(e)}"

    def call_ninjas_api(self):
        """Вызов API Ninjas"""
        text = self.text_input.text().strip()
        if not text:
            self.output_area.setText("Ошибка: Введите текст")
            return

        self.btn_ninjas.setEnabled(False)
        self.output_area.clear()

        # Выполняем запрос
        result = analyze_ninjas_api(text)
        output = self.format_api_result(result, "API Ninjas")
        self.output_area.setText(output)

        self.btn_ninjas.setEnabled(True)

    def call_sentiment_api(self):
        """Вызов Sentiment Analysis API"""
        text = self.text_input.text().strip()
        if not text:
            self.output_area.setText("Ошибка: Введите текст")
            return

        self.btn_sentiment.setEnabled(False)
        self.output_area.clear()

        # Выполняем запрос
        result = analyze_sentiment_analysis_api(text)
        output = self.format_api_result(result, "Sentiment Analysis")
        self.output_area.setText(output)

        self.btn_sentiment.setEnabled(True)

    def compare_apis(self):
        """Сравниваем результаты двух API"""
        text = self.text_input.text().strip()
        if not text:
            self.output_area.setText("Ошибка: Введите текст")
            return

        # На время выполнения запросов выключаем кнопки
        self.disable_all_buttons()
        self.output_area.clear()

        # Вызываем API Ninjas
```

```
        ninjas_result = analyze_ninjas_api(text)
        output = self.format_api_result(ninjas_result, "API Ninjas")
        self.append_output(output)

        # Вызываем Sentiment Analysis API
        sentiment_result = analyze_sentiment_analysis_api(text)
        output = self.format_api_result(sentiment_result, "Sentiment Analysis")
        self.append_output(output)

        # Выполняем сравнение
        comparison = compare_api_results(ninjas_result, sentiment_result)
        comparison_text = self.format_comparison_result(comparison)
        self.append_output(comparison_text)

        # Включаем кнопки обратно
        self.enable_all_buttons()

    def disable_all_buttons(self):
        """Отключить все кнопки"""
        self.btn_ninjas.setEnabled(False)
        self.btn_sentiment.setEnabled(False)
        self.btn_compare.setEnabled(False)

    def enable_all_buttons(self):
        """Включить все кнопки"""
        self.btn_ninjas.setEnabled(True)
        self.btn_sentiment.setEnabled(True)
        self.btn_compare.setEnabled(True)

    def format_comparison_result(self, comparison: dict) -> str:
        """Форматирует результат сравнения API для его вывода"""
        separator = "\n" + "=" * 30
        result_text = separator + "\nРЕЗУЛЬТАТЫ СРАВНЕНИЯ:"

        if comparison['both_api_successful']:
            ninjas_sent = comparison['ninjas_sentiment']
            sentiment_sent = comparison['sentiment_analysis_sentiment']

            result_text += f"\nAPI Ninjas result: {ninjas_sent}"
            result_text += f"\nSentiment Analysis result: {sentiment_sent}"

            if comparison['results_match']:
                result_text += "\n\nОба API вернули один результат"
            else:
                result_text += "\n\nAPI вернули разные результаты"
        else:
            result_text += f"\nСтатус сравнения: {comparison['comparison_status']}"
            if comparison['ninjas_sentiment']:
                result_text += f"\nAPI Ninjas: {comparison['ninjas_sentiment']}"
            if comparison['sentiment_analysis_sentiment']:
                result_text += f"\nSentiment Analysis: {comparison['sentiment_anal-
   ysis_sentiment']}"

        return result_text


def main():
    app = QApplication(sys.argv)
    window = SentimentAnalyzerGUI()
    window.show()
    sys.exit(app.exec_())


if __name__ == "__main__":
    main()
```