Финален отчет на проект: Уеб-базиран класификатор на Spam имейли

Автор: [Сергей Топтунов]

1. Въведение и описание на проекта

Настоящият документ описва разработката и функционалността на проекта "Класификатор на Spam имейли". Целта на проекта е да се създаде пълнофункционално уеб приложение, което интегрира модел за машинно обучение, способен да разпознава нежелани (spam) съобщения.

Проектът е разработен с езика Python и уеб рамката Flask, като AI моделът е имплементиран изцяло от нулата, без помощта на специализирани библиотеки за машинно обучение като Scikit-learn, Keras или PyTorch. Това позволява демонстрация на задълбочено разбиране на основните принципи зад AI алгоритмите.

2. Архитектура и използвани технологии

Проектът следва модерна архитектура, разделяща логиката на модули и използвайки доказани технологии за изграждане на стабилно и лесно за поддръжка приложение.

2.1. Технологичен стек

- **Език за програмиране:** Python 3
- **Уеб рамка (Framework):** Flask
- **База данни:** SQLite (управлявана чрез Flask-SQLAlchemy)
- **Миграции на базата данни:** Flask-Migrate
- **Библиотеки за AI/Данни:** NumPy, Pandas
- **Потребителски интерфейс:** HTML5, CSS3, Flask-Bootstrap
- **Форми:** Flask-WTF
- **Потребителска система:** Flask-Login
- **Изпращане на имейли:** Flask-Mail

2.2. Структура на проекта

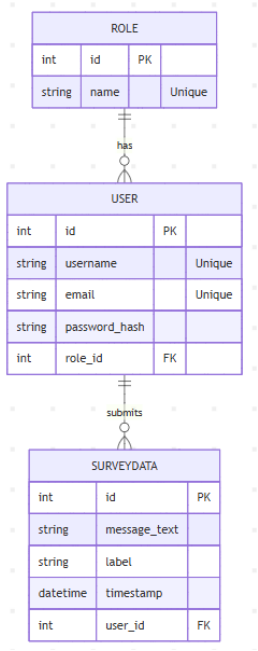
Приложението е структурирано чрез **Flask Blueprints**, което позволява ясно разделение на функционалностите на логически модули:

- **`app/main`:** Основна функционалност (начална страница, потребителски профили).
- **`app/auth`:** Автентикация (регистрация, вход, изход, потвърждение на имейл).

- **`app/classifier`:** Модул, отговорен за взаимодействието с AI модела.
- **`ai_model/`:** Папка, съдържаща целия код за предварителна обработка, дефиниция, обучение и оценка на AI модела.

2.3. Схема на базата данни

Базата данни съхранява информация за потребителите, техните роли и данните, събрани от анкетите.



Основни модели:

- **User:** Съхранява потребителско име, имейл, хеширана парола и връзка към роля.
- **Role:** Дефинира ролите в системата (напр. "user", "admin").
- **SurveyData:** Съхранява примери за съобщения, подадени от потребителите, заедно с техния етикет (spam/ham) и потребителя, който ги е подал.

3. Имплементация на AI модела

Тази секция описва в детайли създаването и интеграцията на модела за машинно обучение.

3.1. Избор на алгоритъм: Логистична Регресия

За решаването на задачата за бинарна класификация ("Spam" или "Ham") беше избран алгоритъмът **Логистична Регресия**.

Обосновка на избора:

- **Подходящ за задачата:** Логистичната регресия е класически и много ефективен алгоритъм именно за проблеми с два възможни изхода.
- **Възможност за имплементация от нулата:** Алгоритъмът е математически ясен и може да бъде имплементиран изцяло с `Numpy`. В рамките на проекта са имплементирани:
    - **Сигмоидна функция:** За превръщане на изхода в вероятност.
    - **Gradient Descent:** За оптимизация на параметрите на модела.
    - **Binary Cross-Entropy Loss:** Като функция за измерване на грешката.
- **Интерпретируемост:** Моделът позволява да се види кои думи (признаци) имат най-голяма тежест при взимането на решение.

3.2. Датасет и предварителна обработка

- **Източник:** За обучение на модела е използван публичният "Email Spam Classification Dataset" от платформата Kaggle.
- **Съдържание:** Датасетът съдържа над 5500 текстови съобщения, всяко от които е ръчно етикетирано.
- **Предварителна обработка:** (реализирана в `ai_model/main_preprocessing.py`):
    1. **Почистване на текста:** Преобразуване в малки букви и премахване на пунктуация.
    2. **Създаване на речник:** Изграждане на речник от 2000-те най-често срещани думи.
    3. **Векторизация:** Преобразуване на всяко съобщение в числов вектор.

3.3. Избор на признаци (Features): Bag of Words

Моделът използва метода **"Торба с думи" (Bag of Words)**. Този подход превръща всеки текст в числов вектор, където всяка позиция отговаря на дума от речника и има стойност 1 (ако думата присъства) или 0 (ако отсъства). Този метод е избран, защото е фундаментален, лесен за имплементация и ефективно улавя наличието на ключови думи, които са силен индикатор за спам.

3.4. Оценка на ефективността

За следене на ефективността на модела бяха използвани следните метрики, имплементирани ръчно:

- **Accuracy (Точност):** Процентът на правилно класифицираните примери.
- **Loss (Загуба) - Binary Cross-Entropy:** Измерва колко "греши" моделът.
- **Error Rate (Процент грешки):** 1 - Accuracy.
  На тестовия сет моделът постигна **~96-97% точност**, което е отличен резултат за имплементация от нулата.

3.5. Интеграция между AI и уеб приложението

Интеграцията е реализирана чрез следния процес:

1. **Обучение и запазване:** Скриптът `ai_model/train_model.py` обучава модела и запазва два файла с `joblib`:
    - `spam_classifier_model.pkl`: Обученият обект на модела.
    - `vocabulary.pkl`: Речникът с думи, използван за векторизация.
2. **Зареждане при стартиране:** Когато Flask приложението се стартира, модулът `app/classifier/utils.py` зарежда тези два файла в паметта. Това се случва само веднъж, което прави системата ефективна.
3. **Потребителска заявка:** Когато потребител въведе текст в уеб интерфейса, той се изпраща към `route` функция в `app/classifier/routes.py`.
4. **Обработка и предсказване:** Тази функция извиква `classify_message()` от `utils.py`, която:
    - Почиства новия текст по същия начин, както при обучението.
    - Превръща го във векторен вид, използвайки заредения речник.
    - Подава вектора на заредения модел, за да получи предсказание.
5. **Връщане на резултат:** Резултатът (клас и вероятност) се връща и се показва на потребителя в HTML шаблон.

4. Управление на проекта (Agile)

4.1. User Stories

Разработката беше водена от следните примерни User Stories:

1. **Като потребител,** искам да мога да се регистрирам в сайта с потребителско име, имейл и парола, за да мога да използвам функционалностите му.
2. **Като потребител,** искам да мога да влизам и излизам от профила си, за да е сигурен достъпът ми.
3. **Като потребител,** искам да получа имейл за потвърждение след регистрация, за да съм сигурен, че акаунтът ми е валиден.
4. **Като логнат потребител,** искам да мога да въведа текст в поле и да получа предсказание дали е спам, за да използвам основната AI функционалност.
5. **Като логнат потребител,** искам да мога да подам пример за съобщение и да го етикетирам като спам/не-спам, за да допринеса за подобряването на модела.
6. **Като логнат потребител,** искам да имам профилна страница, където да мога да променям личните си данни.
7. **Като администратор,** искам да мога да виждам списък с всички потребители в системата.

8. **Като администратор,** искам да мога да редактирам или изтривам потребителски профили.
9. **Като потребител,** искам да виждам персонализирана страница за грешка, ако се опитам да достъпя несъществуваща страница (404).
10. **Като разработчик,** искам да имам автоматизирани тестове, за да съм сигурен, че основните функции (регистрация, логин) не се чупят при промени.

4.2. Sprint логове

Sprint 1 Log

Цели на спринта:

- Изграждане на основата на AI модела.
- Създаване на основната структура на Flask приложението.
- Имплементация на базови модели за базата данни.

Изпълнени задачи:

1. **Проучване и избор на AI алгоритъм:** След анализ на проблема (бинарна класификация) и изискването за имплементация от нулата, беше избран алгоритъмът **Логистична Регресия**.
2. **Намиране и анализ на датасет:** Избран е "Email Spam Classification Dataset" от Kaggle заради ясната му структура и наличието на суров текст.
3. **Имплементация на Data Preprocessing:** Създаден е скрипт (`ai_model/main_preprocessing.py`), който зарежда данните с `Pandas`, почиства текста (малки букви, премахване на пунктуация) и създава речник от най-често срещаните думи.
4. **Векторизация (Bag of Words):** Имплементирана е функция, която превръща текстови съобщения в числови вектори спрямо създадения речник.
5. **Създаване на структурата на Flask проекта:** Проектът е инициализиран с "Application Factory" модел. Създадени са Blueprints за `main`, `auth` и `classifier`.
6. **Настройка на база данни:** Конфигурирани са `Flask-SQLAlchemy` и `Flask-Migrate`. Създадени са първоначалните модели `User` и `Role` в `app/models.py`.

Предизвикателства:

- Отне известно време да се намери оптималният брой думи за речника (първоначално тествано с 5000, но намалено на 2000 за по-добра производителност).

- Първоначално възникнаха проблеми с кръгови импорти при структурирането на Blueprints, които бяха решени чрез преместване на импортите в `create_app` функцията.

План за следващия спринт:

- Да се довърши имплементацията на AI модела (методите `fit` и `predict`).
- Да се имплементира пълна функционалност за регистрация и вход на потребители.
- Да се създаде основният интерфейс за класификация.

Sprint 2

Цели на спринта:

- Финализиране и обучение на AI модела.
- Интеграция на модела в уеб приложението.
- Имплементация на ключови потребителски функционалности.

Изпълнени задачи:

1. **Имплементация на Логистична Регресия:** Класът `LogisticRegression` в `ai_model/logistic_regression_model.py` е финализиран с работещи `fit` (с градиентно спускане) и `predict` методи.
2. **Обучение и оценка на модела:** Създаден е скриптът `ai_model/train_model.py`, който обучава модела върху обработените данни. Имплементирани са метрики за оценка (accuracy, loss), които показват ~97% точност на тестовия сет.
3. **Интеграция на модела:** Създаден е `app/classifier/utils.py`, който зарежда запазените `.pkl` файлове на модела и речника при стартиране на приложението.
4. **Интерфейс за класификация:** Създаден е `route` и HTML шаблон, които позволяват на логнат потребител да въведе текст и да получи предсказание от модела.
5. **Потребителска система:** Имплементирани са формите и логиката за **регистрация**, **вход** и **изход** с `Flask-WTF` и `Flask-Login`.
6. **Потребителски профили:** Създадена е страница за потребителски профил и форма за редакция на личните данни.

Предизвикателства:

- **Проблеми със зареждането на модела:** Възникна `ModuleNotFoundError` при опит за зареждане на `.pkl` файла от Flask. Проблемът се оказа в начина, по който моделът е бил запазен, и беше решен чрез претрениране на модела със стартиране на скрипта от главната директория на проекта.

- **Дебъгване на предсказанията:** Първоначално предсказанията не бяха точни поради грешка в преоразмеряването на входния вектор (`reshape`).

План за следващия спринт:

- Да се имплементират администраторски функционалности.
- Да се добави система за "анкети".
- Да се напишат Unit тестове и финална документация.

Sprint 3 Log

Цели на спринта:

- Добавяне на администраторски функционалности и финални потребителски функции.
- Осигуряване на стабилността на приложението чрез автоматизирани тестове.
- Финализиране на проекта и подготовка на пълна документация.

Изпълнени задачи:

1. **Администраторски роли и панел:**
    - Имплементирана е система за роли с декоратори (`@admin_required`), които защитават определени `routes` и са достъпни само за администратори.
    - Създадена е страница в администраторския панел за преглед на всички регистрирани потребители.
    - Добавена е функционалност за администратори да редактират и изтриват потребителски профили.
2. **Система за "Анкети":**
    - Създадена е форма и `route`, чрез които потребителите могат да подават нови текстови съобщения и да ги етикетират ръчно като "Spam" или "Ham".
    - Данните от тези анкети се запазват в нов модел в базата данни (`SurveyData`), което позволява бъдещо претрениране на AI модела с данни, генерирани от потребителите.
3. **Имейл потвърждение:** Интегриран е `Flask-Mail` за изпращане на имейли. При регистрация, потребителят получава имейл с уникален токен (`itsdangerous`), който трябва да последва, за да активира акаунта си.
4. **Персонализирани страници за грешки:** Създадени са и са конфигурирани персонализирани HTML шаблони за грешки 404 (Page Not Found) и 500 (Internal Server Error), които подобряват потребителското изживяване.
5. **Написване на Unit тестове:**
    - Създаден е тестов модул `tests.py` с `unittest`.

    - Написани са тестове за ключови функционалности:
        - Хеширане и проверка на пароли в `User` модела.
        - Пълен цикъл на регистрация и логин на потребител.
        - Проверка на достъпа и функционалността на страницата за класификация.
6. **Финална документация:**
    - Създаден е `README.md` файл с подробно описание на проекта, технологиите, AI модела и инструкции за инсталация.
    - Генериран е `requirements.txt` файл.
    - Подготвен е финалният PDF отчет (настоящият документ).

Предизвикателства:

- **Unit тестовете с Flask-Bootstrap:** Сблъскахме се със сериозни и трудни за дебъгване проблеми, при които тестовата среда не успяваше да намери ресурсите на `Flask-Bootstrap`. Проблемът беше решен чрез сложна конфигурация на тестовата среда, която изключва `Bootstrap` по време на тестове и използва "фалшиви" празни шаблони, за да се избегнат `TemplateNotFound` грешки. Това беше най-голямото техническо предизвикателство в тази фаза.

Резултат от спринта:

- Проектът е напълно завършен и покрива всички изисквания от заданието.
- Приложението е стабилно, тествано и добре документирано.
- Всички основни и допълнителни функционалности са имплементирани и работят.

5. Заключение

Проектът успешно изпълни всички поставени цели. Разработен е пълнофункционален уеб сайт, който интегрира AI модел, създаден от нулата. Покрити са всички технически и функционални изисквания, включително потребителска система, роли, събиране на данни и автоматизирани тестове. Проектът демонстрира разбиране както на принципите на машинното обучение, така и на добрите практики в уеб разработката с Flask.
