Міністерство освіти і науки України

Харківський національний університет радіоелектроніки

Факультет Комп’ютерних наук

Кафедра Програмної інженерії

**АТЕСТАЦІЙНА РОБОТА**

**Пояснювальна записка**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_бакалавр\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(освітньо-атестаційний рівень)

Бібліотека визначення мови коротких повідомлень, що складаються з літер кириличного алфавіту

Виконала: студент 4 курсу, групи ПІ-13-2

напряму підготовки

6.050103 «Програмна інженерія» (ПІ)

(шифр і назва напряму)

Сердюк Д.О.

(прізвище, ініціали)

Керівник Бабій А.С.

(прізвище, ініціали)

Рецензент Сокорчук І.П.

(прізвище, ініціали)

Допускається до захисту

Зав. кафедри \_\_\_\_\_\_\_\_\_\_\_ Дудар. З.В.

(підпис) (прізвище, ініціали)

2017 р.

Харківський національний університет радіоелектроніки

Факультет Комп’ютерних наук

Кафедра Програмної інженерії

Освітньо-атестаційний рівень бакалавр

Напрям підготовки 6.050103 «Програмна інженерія»

(шифр і назва)

ЗАТВЕРДЖУЮ:

Зав. кафедри \_\_\_\_\_\_\_\_\_\_\_\_\_\_

(підпис)

«\_\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 20 \_\_\_ р.

**ЗАВДАННЯ**

НА АТЕСТАЦІЙНУ РОБОТУ (ПРОЕКТ)

студентові\_\_\_\_\_\_\_\_\_\_\_\_Сердюк Дар’ї Олексіївні\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(прізвище, ім’я, по батькові)

1. Тема роботи (проекту)\_\_\_\_\_\_\_\_\_ Бібліотека визначення мови коротких повідомлень, що складаються з літер кириличного алфавіту\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

затверджена наказом по університету від "\_\_\_"\_\_\_\_\_\_\_\_\_\_\_\_\_\_20 \_\_ р. № \_\_\_\_\_\_\_\_\_\_\_\_

2. Термін подання студентом роботи (проекту) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

3. Вихідні дані до роботи (проекту)\_\_\_\_\_\_необхідно розробити бібліотеку розпізнавання мови коротких повідомлень; для розробки використовувати Python разом з бібліотекою Scikit-Learn\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

4. Зміст пояснювальної записки (перелік питань, що потрібно розробити) \_вступ, аналіз предметної галузі, формування вимог до програмної системи, архітектура та проектування програмного забезпечення, опис прийнятих програмних рішень, тестування розробленого програмного забезпечення, впровадження програмного забезпечення, висновки за роботою, перелік джерел\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(Зворотній бік бланку завдання)

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслеників, плакатів) діаграма пакетів, діаграма  розгортання,  діаграма послідовності, діаграма кооперацій, діаграма активності, структура бази даних, фрагменти коду програми\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

6. Консультанти розділів роботи (проекту)

|  |  |  |  |
| --- | --- | --- | --- |
| Найменування  розділу | Консультант  (посада, прізвище, ім’я, по батькові) | Позначка консультанта  про виконання розділу | |
| підпис | дата |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

7. Дата видачі завдання «\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_20\_\_\_ р.

**КАЛЕНДАРНИЙ ПЛАН**

|  |  |  |  |
| --- | --- | --- | --- |
| № | Назва етапів роботи (проекту) | Термін  виконання етапів проекту (роботи) | Позначка про виконання |
| 1 | Аналіз предметної галузі |  |  |
| 2 | Створення Специфікації ПЗ |  |  |
| 3 | Проектування та розробка ПЗ |  |  |
| 4 | Тестування та дослідна експлуатація ПЗ |  |  |
| 5 | Написання пояснювальної записки |  |  |
| 6 | Перевірка пояснювальної записки керівником та нормоконтролером |  |  |
| 7 | Оцінка роботи стороннім рецензентом |  |  |
| 8 | Здача роботи у електронний архів |  |  |
| 9 | Підпис завідувача кафедри |  |  |
| 10 | Захист випускної атестаційної роботи |  |  |

Студент \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(підпис)

Керівник роботи (проекту) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_ст.викл. Бабій А.С.\_\_\_\_

(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ / ABSTRACT

Пояснювальна записка до атестаційної роботи бакалавра: с., рис., табл., дод., 12 джерел.

Головним об’єктом дослідження є алгоритми розпізнавання мови друкованого тексту, а саме підготовчі етапи та методи класифікація.

Метою роботи є створення бібліотеки на мові програмування Python.

Методи розробки використовують мову програмування Python та допоміжні бібліотеки для обчислень, такі як NumPy, SciPy та бібліотека для машинного навчання (Machine Learning) і добування та аналізу даних (Data Mining, Data Analysis) Scikit-learn.

У результаті роботи були порівняні різноманітні методи розпізнавання мови: як тренування класифікатору, так і методи самої класифікації (Байесовська, метод опорних векторів (SVM) та інші); розроблена бібліотека розпізнавання мови коротких повідомлень (наприклад, повідомлень у соціальній мережі Twitter).

РОЗПІЗНАВАННЯ МОВИ, МАШИННЕ НАВЧАННЯ, КЛАСИФІКАЦІЯ, N-ГРАМИ, НОРМАЛІЗАЦІЯ, БАЙЕСОВСЬКА КЛАСИФІКАЦІЯ, PYTHON, SCIKIT-LEARN, NUMPY, FLASK, SQLITE

The main objective of the research is the language identification algorithms of the printed text, preprocessing steps and classification in particular.

The goal of this piece of work is to create a Python library.

The developed methods are based on Python programming language and additional computing libraries, e.g. NumPy, SciPy, and a machine learning, data analysis and data mining library Scikit-learn.

As a result, a number of methods for language identification were studied and compared: both training the classifier and the classification methods (Bayesian, Support Vector Machine etc.); a language identification library for short messages (tweets in Twitter) was created.

LANGUAGE IDENTIFICATION, MACHINE LEARNING, CLASSIFICATION, N-GRAMS, NORMALIZATION, BAYESIAN CLASSIFICATION, PYTHON, SCIKIT-LEARN, NUMPY, FLASK, SQLITE

ЗМІСТ

[ВСТУП 6](#_Toc484787026)

[1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ 7](#_Toc484787027)

[1.1 Аналіз предметної галузі 7](#_Toc484787028)

[1.2 Виявлення проблем та актуалізація рішень 8](#_Toc484787029)

[1.2.1 Байєсівська класифікація 9](#_Toc484787030)

[1.2.2 Метод опорних векторів 11](#_Toc484787031)

[1.2.3 Random forest 13](#_Toc484787032)

[1.2.4 Метод k-найближчих сусідів 14](#_Toc484787033)

[1.2.5 Огляд існуючих рішень 16](#_Toc484787034)

[1.3 Постановка задачі 17](#_Toc484787035)

[2 ФОРМУВАННЯ ВИМОГ ДО ПРОГРАМНОЇ СИСТЕМИ 18](#_Toc484787036)

[3 АРХІТЕКТУРА ТА ПРОЕКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ 21](#_Toc484787037)

[3.1 UML-проектування ПЗ 21](#_Toc484787038)

[3.2 Проектування архітектури ПЗ 25](#_Toc484787039)

[3.3 Проектування бази даних 27](#_Toc484787040)

[3.4 Створення REST API для системи 28](#_Toc484787041)

[4 ОПИС ПРИЙНЯТИХ ПРОГРАМНИХ РІШЕНЬ 31](#_Toc484787042)

[5 ТЕСТУВАННЯ РОЗРОБЛЕНОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ 37](#_Toc484787043)

[ВИСНОВКИ 39](#_Toc484787044)

[ПЕРЕЛІК ПОСИЛАНЬ 40](#_Toc484787045)

# ВСТУП

Розпізнавання мови є завданням автоматичного визначення мови друкованого тексту, виходячи з його змісту; це головна задача у дослідженнях та комерційних програмах з обробки природної мови (Natural Language Processing).

Людина може з високою ймовірністю визначити, чи на його рідній мові написано повідомлення десь с перших 2-3 слів. Або хоча б висловити припущення, що текст схожий на іспанську. Враховуючи, що у теперішній час існує велика кількість інформації, яку потрібно розпізнавати та опрацьовувати, це потребує багато ресурсів та, як наслідок, витрат. Власне саме тому і з’явилася необхідність автоматизації процесу розпізнавання.

Розпізнавання мови використовується, наприклад, для коректного і зрозумілого звертання онлайн-консультанта на сайті інтернет-магазину, або розсилки email-повідомлень з рекламою та різноманітними пропозиціями. Також такі сервіси як наприклад Google Translate дозволяють користувачу перекласти тест, не маючи і думки, якою мовою він написаний.

Завдання розпізнавання мов є виконуваною, так як природні мови мають власну структуру, алфавіт, найчастіше використовувані послідовності символів та інше. Наприклад, ймовірність використання літери «Е» в англійській мові дорівнює приблизно 13%, тоді як літери «Z» - всього 0.1%. До того ж, для тексту англійською мовою ймовірність використання послідовності «TH» є досить великою, тоді як в уривку такого ж розміру, написаного іспанською мовою, ця цифра наближається до нуля [1]. Саме з огляду на всі подібні дані, ми можемо розробляти різноманітні алгоритми для ідентифікації мови тексту.

Не дивлячись на це, залишається безліч відкритих питань і невирішених проблем в розпізнаванні друкованого тексту.

У цій роботі будуть розглянуті як вже застарілі (мало точні), так і використовувані в даний час методи розпізнавання мови тексту.

# 1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ

## 1.1 Аналіз предметної галузі

Більшість існуючих бібліотек розпізнавання мови використовують алгоритм, запропонований у роботі Кавнара та Тренкла [1]. Це була одна з перший статей на цю тему. Досить простий алгоритм порівнює 300 найбільш уживаних триграм (відсортованих за частотою) в кожній мови з 300-ми в тексті, який розпізнається. Далі, ці так звані профілі порівнюються, та який профіль є найбільш схожим з профілем тексту, то і є шуканою мовою. Цей метод добре працює для достатньо великих текстів (більше 300 символів). Великою перевагою цього методу також є можливість розпізнавання як малих, так і великих текстів, при цьому не зберігаючи багато інформації про моделі класифікатора.

Після цієї статті інші дослідники намагалися покращити існуючий метод. Наприклад, Прегер [2] порівнював результати роботи алгоритму для n-грам різної довжини, слів, комбінацій слів та іншого. N-грама – підрядок довжини N, виділена з більшого рядка. При цьому можуть використовуватися різні значення N. Для більшої точності в N-грами зазвичай додають додатковий символ в початок або кінець (тобто, відмітку початку або кінця) з метою зіставлення з початком або кінцем слова відповідно. Таким чином, для рядка довжиною n існує n + 1 біграм, n + 1 триграм, n + 1 квадграм і т.д. Слова або n-грами, які були найбільш унікальними (використовувались у найменшій кількості профілів), мали найвищий пріоритет.

Макнамара [3] намагався використовувати для цієї задачі рекурентні нейронній мережі, але результати були гіршими за метод n-грам.

Елворті [4] запропонував рішення для великих текстів: не опрацьовувати весь текст, а лише до тих пір поки не буде досягнуто певного відсотка точності. До того ж, щоб обрати результуючу мову, потрібно досягнути не тільки високого відсотка точності, а й бути набагато вищою ніж інші. Цей підхід працюватиме лише для документів, написанних одною мовою.

## 1.2 Виявлення проблем та актуалізація рішень

Сучасні алгоритми розпізнавання дозволяють визначити мову тексту з точністю до 99% (показник залежить від розміру, коректності написання тексту, наявності декількох мов і т.д.). Текст, який підлягає розпізнаванню, може бути написаний як на одній мові, так і на кількох. Обробка одномовних текстів - завдання набагато простіше, ніж багатомовних, так як для останнього необхідні знання більшої кількості мов [5].

Основні труднощі при ідентифікації мови:

а) розмір тексту - текст повинен бути такої довжини, щоб можна було коректно розпізнати мову написання, тобто, в довгому документі буде велика кількість різних слів;

б) шуми в тексті - тексти можуть містити різні шуми, наприклад, абревіатури, скорочені форми слів і т.д. Ця проблема особливо часто виникає при розпізнаванні коротких повідомлень (наприклад SMS-повідомлень, постів в соціальній мережі Twitter), коли текст скорочується максимально настільки, щоб він включав не більш допустимої кількості символів. Розробка узагальнений системи, яка враховувала б усі подібні фактори – надзвичайно складне завдання;

в) сегментація тексту – ця проблема стосується визначення мови багатомовного тексту; завдання полягає в тому, щоб розділити текст на області, які написані на одній мові, і визначити в подальшому мову цих відрізків (сегментів);

г) кодування тексту - тренувальні тексти і тестові дані повинні бути надані в одному кодуванні, що не завжди можливо проконтролювати;

д) близько пов’язані мови - схожі мови або мовні діалекти з близько-розташованих районів, які містять однакові слова і схожу граматичну структуру, важко розрізнити між собою;

е) підтримка мов меншин - в даний час існуючі системи визначення мови тексту обмежені до використання невеликої кількості мов (використовуваних великою кількістю людей (табл. 1.1) і для яких були створені мовні корпуси), які використовуються в тренувальних сетах, таким чином, визначення мови поза цієї групи є неелементарним завданням;

є) адаптація методів для розпізнавання мов, які використовують ієрогліфи на листі - більшість існуючих методів розпізнавання ґрунтується на розбиття слів на токени (наприклад, слова або послідовності символів), а в деяких азіатських мовах, наприклад, китайською та японською мовами, на листі не використовуються прогалини для поділу слів. Цей фактор значно ускладнює процес розпізнавання. При цьому, крім японської абетки, в мові використовується і китайська.

Таблиця 1.1 – Найчастіше використовувані мови на веб-сторінках (станом на 4 березня 2017) [6]

|  |  |
| --- | --- |
| Мова | Процент |
| Англійська | 51.7% |
| Російська | 6.5% |
| Японська | 5.6% |
| Німецька | 5.5% |
| Іспанська | 5.1% |
| Французька | 4.1% |
| Португальська | 2.5% |
| Італійська | 2.4% |
| Китайська | 2.0% |

На теперішній час існує велика кількість методів для розпізнавання мови, які будуть описані у наступних підрозділах.

### 1.2.1 Байєсівська класифікація

Байєсівська класифікація є ймовірностним та контрольованим методом навчання. В основі цього алгоритму знаходиться теорема Байєса. Ця теорема математично задається наступною формулою:

,

де P(C|D) – ймовірність того, що документ D належить до класу С,

P(D|C) – ймовірність зустріти документ D серед усіх документів класу С,

P(C) – безумовна ймовірність зустріти документ класу С у корпусі усіх документів,

P(D) – безумовна ймовірність документа D у корпусі усіх документів.

Теорема Байєса дозволяє змінити місцями подію та наслідок, таким чином, знаючи, з якою ймовірність певна подія приводить в певному наслідку, ця теорема дозволяє розрахувати ймовірність того, що саме ця причина призвела до нинішньої події.

Мета класифікації полягає в тому, щоб зрозуміти, до якого класу належить документ, тому нам потрібна не сама ймовірність, а найбільш ймовірний клас. байєсів класифікатор використовує оцінку апостеріорного максимуму для визначення найбільш вірогідного класу – класу з максимальною ймовірністю.

,

Враховуючи, что знаменник є константою, і не зможе вплинути на розподіл классів, тому його можно ігнорувати, і формула матиме наступний вигляд:

,

Ймовірність документа рахується як добуток умовних ймовірностей усіх слів (або n-грам) у документі (виходячи з підходу bag of words), коли поява одного слова не залежить від появи іншого:

,

Тоді виходить, що клас з максимальною ймовірністю рахується за формулою:

,

Існує також проблема, що перемноження багатьох ймовірностей з дуже малими значеннями може привести до зникнення порядку (arithmetic underflow). Уникнути такого результату можна завдяки використанню властивості логарифма добутку. Використовуючи логарифми (основа не має значення), формула буде виглядати наступним чином:

,

Оцінка ймовірності слова (n-грами) у класі може розраховуватися по-різному, наприклад, використовуючи поліноміальний розподіл:

,

де числівник описує кількість зустріч слова у документах класу, а знаменник – сумарну кількість слів в документах цього класу.

Але ж, якщо на етапі класифікації зустрінеться слово, якого не було у тренувальному сеті, то значення стане рівним нулю и такий документ не можливо буде класифікувати. Цю проблему можна вирішити наданням ненульової ймовірності для слів, які не зустрічались: наприклад, згладжуванням Лапласу. Тоді формула приймає вигляд:

,

де |V| – кількість слів у тренувальному сеті.

Тоді остаточна формула, яку використовує наївний байєсів класифікатор виглядає наступним чином:

,

Перевагою цього алгоритму класифікації є його відносна простота і невеликий об’єм тренувальних даних для оцінки параметрів для класифікації.

### 1.2.2 Метод опорних векторів

Цей метод найкраще працює для бінарної класифікація. Для даного завдання можливо застосувати його для відповіді на таке питання «Чи на мові N написано повідомлення?».

Модель опорно-векторної машини подає дані як точки у просторі (вимір простору визначається кількістю характеристик), на якій різні класи (зазвичай 2) розділені максимально віддаленою від границь гіперплощиною (тобто, точкою у одновимірному просторі, прямою у двовимірному і т.д.). Дані з тестового сету відображуються до того ж самого простору, та далі робиться передбачення про їх належність до того чи іншого класу даних. Головна мета цього алгоритму – знайти оптимальну гіперплощину, що відокремлює, тому що саме такий результат дозволить коректно працювати з раніше невідомими тестовими даними.

Якщо є набор даних з n точок таких, що

,

де y – це клас, яким може приймати значення -1 або 1, до якого належить , де є p-вимірним вектором. Треба знайти максимально віддалену точок , які належать класу у=-1 від точок , які належать класу у=1. Будь-яку гіперплозин можна записати як множину точок , які задовольняють:

,

де – вектор нормалі до гіперплощини.

Якщо дані є лінійно роздільними (рис. 1.1, рис 1.2), то можна обрати дві паралельні гіперплощини, які розділяють два класи так, що відстань між ними найбільша. Область, обмежена цими гіперплощинами називається гранню (margin), а максимально розділова площина – площина, яка лежить між ними. Найближчі точки до цієї площини лежать на її гранях. Вони саме і називаються опорними векторами.

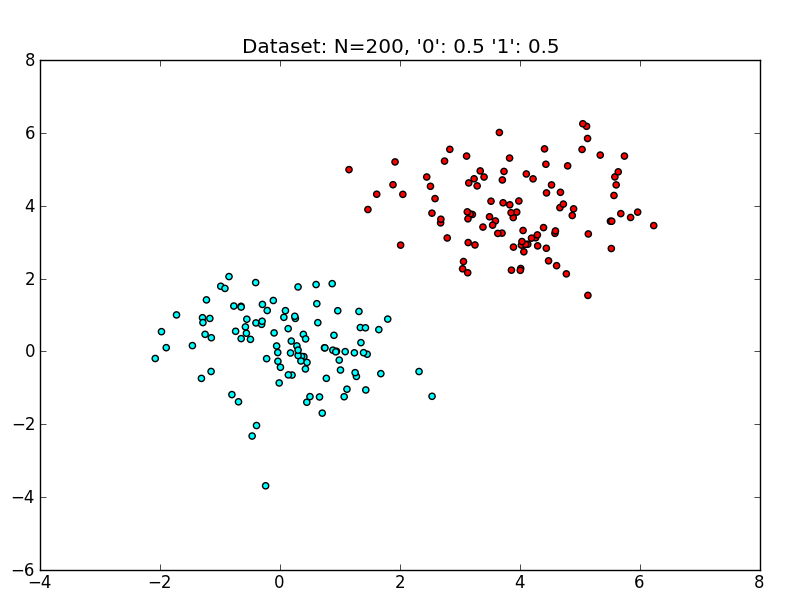


Рисунок 1.1 – Лінійно роздільні дані

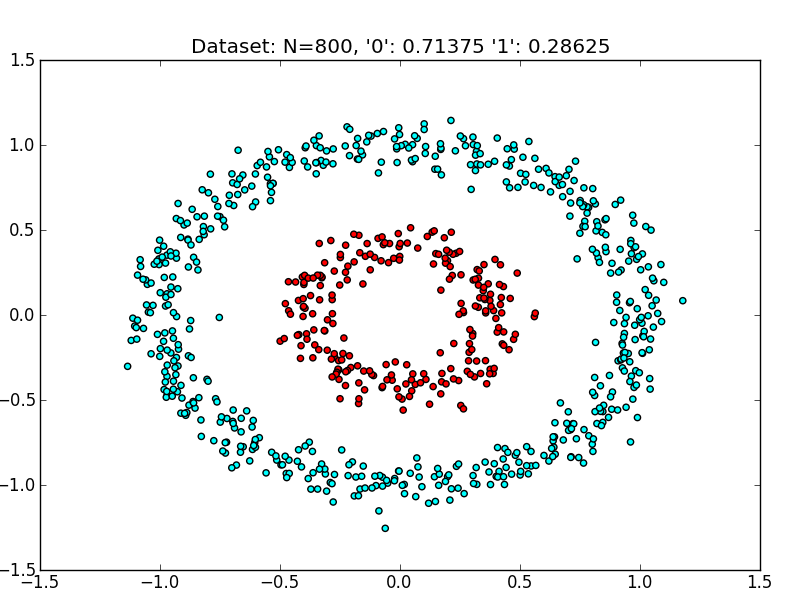


Рисунок 1.2 – Лінійно нероздільні дані

Ці гіперплощини описуються наступними рівняннями:

,

.

Відстань між цими площинами (а отже ширину розділюючої гіперплощини можна розрахувати як [7]. Тоді, відстань буде найбільшою при мінімальному значенні . Враховуючи, що точки не повинні потрапити усередину гіперплощини:

при y = 1,

при y=-1

Або:

для всіх .

Рішенням цієї проблемі оптимізації (значення та b) і є рівнянням площини, яке визначає класифікатор.

Якщо ж дані лінійно не роздільні, використовується алгоритм м’якого розділення.

Перевагами цього алгоритму є його стійкість до багатовимірних даних (з великою кількістю характеристик) та порівняно невелики затрати на пам’ять. Але метод є чутким до шумних даних.

### 1.2.3 Random forest

Random forest (випадковий ліс) – алгоритм класифікації, в основі якого лежить множина дерев ухвалення рішень (ансамбль) [8]. Такий ансамбль дозволяє об’єднати багато «слабкіших» учнів (класифікаторів) в одного «сильного» (рис. 1.3).

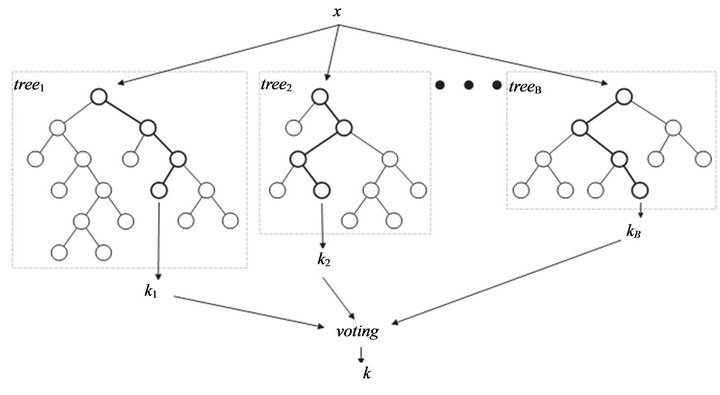


Рисунок 1.3 – Ліс дерев рішень

Отже, випадковий ліс починається з дерева ухвалення рішень. У дереві приняття рішень на вхід подаються вхідні дані, які проходять повз дерево до менших і менших, і у кінці до листа, на якому записані значення цільового класу.

Усі дерева ансамблю будуються незалежно один від одного:

1. генерується тренувальна вибірка з повторами (таким чином, що деякі приклади потраплять в неї декілька раз) розміру n;
2. при побудові дерева ухвалення рішень, яке класифікує вхідні дані, вибирається ознака не з усіх M ознак, а лише с m випадково вибраних. Ці ознаки мають бути оптимальними відповідно до обраної функції (наприклад, за критерієм Джині, який використовує Брейман [8]);
3. дерево будується до повного вичерпання тренувальної вибірки (до тих пір поки не закінчаться класи); гілки дерева при цьому не відсікаються.

Коли на вхід до випадкового лісу потрапляють дані, вони проходять через всі дочірні дерева і клас який буде вибрано найчастіше, і є результуючим.

Перевагами цього методу є його стійкість до незбалансованих даних та даних з великою кількістю ознак. Також, метод може бути розбитий на підзадачі, які можуть бути виконані паралельно и незалежно одна від одної Однак, алгоритм схильний до перенавчання при використанні даних з великою кількістю шумів. Також, для зберігання моделей потрібно багато пам’яті, тому потрібно за допомогою тестування для певних даних підібрати оптимальні кількість та глибину дерев.

### 1.2.4 Метод k-найближчих сусідів

Цей метод представляє дані як вектори у багатовимірному просторі, кожен з яких прив’язаний до класу. Зазвичай, таких класів два, але метод показує високі результати і на більшій кількості класів. Належність нового вхідного даного до класу визначається класами k-найближчих сусідів (рис. 1.4). Тобто, якщо k=1, то об’єкту буде наданий клас його найближчого сусіда.

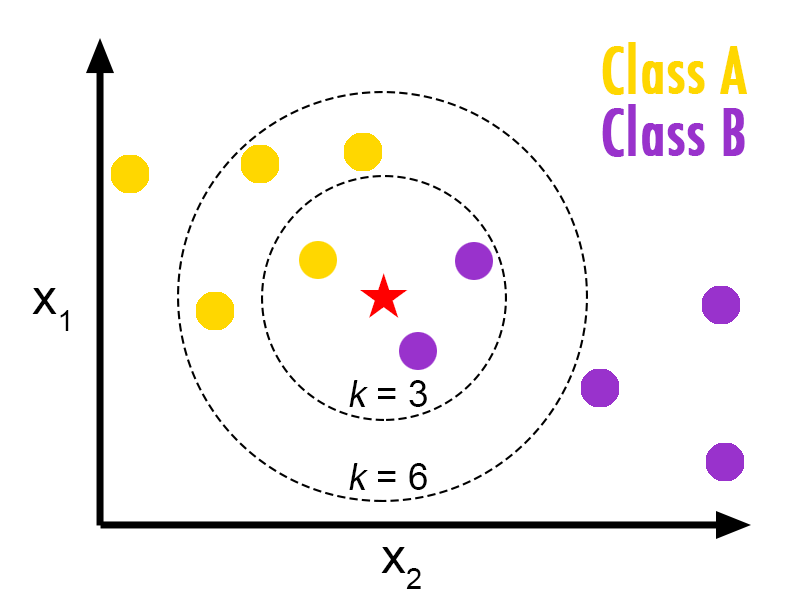


Рисунок 1.4 – Визначення результуючого класу в залежності від k

Для розрахунку відстані між точками А і В на площині ознак (feature space) можуть використовуватися різноманітні функції. Найчастіше у літературі згадується функція евклідової відстані:

,

де А і В представлені як вектори ознак A = (, B = (, а

m – розмірність багатовимірного простору.

Для визначення схожості текстових документів у задачах обробки текстових даних частіше обчислюється коефіцієнт Отіаі [9]:

*,*

де чисельник – скалярний добуток векторів A і В, знаменник – добуток їх евклідових довжин.

Метод k-найближчих сусідів є одним із найпростіших, та, не дивлячись на це, може показувати досить високі результати. Для найкращого результату потрібно вибрати оптимальне значення k, при якому рівень помилкової класифікації буде найнижчим. Мале значення k може призвести до високого впливу шуму на класифікацію, велике – розмиває границі класів.

### 1.2.5 Огляд існуючих рішень

На теперішній час існує велика кількість бібліотек, які сумісні з різними мовами програмування і використовують різні алгоритми для класифікації.

langid.py – самостійна система розпізнавання мови, написана на мові програмування Python та на теперішній момент підтримує 97 мов. Бібліотека не чутлива до XML/HTML розмітки. Класифікатор для бібліотеки бува натренований на різних ресурсах, тому результат роботи очікується найвищим. Для використання бібліотеки необхідно встановити мінімальну кількість залежностей та використовувати лише один головний .py файл. Система працює досить швидко завдяки розділення задач на різні потоки і процесори. Для класифікації бібліотека використовує наївний байєсів класифікатор, а в якості ознак – n-грами [10].

TextCat – імплементація алгоритму, описаного в [1]. Система підтримує 69 природних мов. Оригінальна імплементація написана мовою програмування Perl, хоча зараз також наявні адаптери до інших мов програмування (наприклад, NTextCat – додадок підтримуючий .NET Framework, пакет textcat для мови програмування R). Мовні моделі обмежені до найчастіше використаних 400 n-грам.

guess-language – бібліотека на мові програмування Python, яка використовує оптимізовані на C++ функції та має змогу розрізняти більше 60 мов. Алгоритм використовує триграми та текст у форматі UTF-8.

Google Translate API – бібліотека автоматичних перекладів, однією з функцій якої є також визначення мови вхідних даних, якщо вона невідома. Google Translate надає зручний для використання інтерфейс у вигляді REST API, який дозволяє відправляти не редагований текст (наприклад, HTML сторінку). Система підтримує найбільшу кількість мов з нині існуючих бібліотек. Але ж ця бібліотека є платною (близько 20$ за 1М символів).

Yandex Translate API – REST API бібліотека для автоматичних перекладів, яка підтримує більше 70-ти природніх мов та може перекладати як окремі слова, так і великі тексти. Бібліотека надає функцію визначення мови повідомлення. Має ліміт на використання – 1М символів на день. Довжина n-грам зазвичай у проміжку від 1 до 7, найчастіше використовується довжина 5.

## 1.3 Постановка задачі

Метою даної передатестаційної роботи є створення бібліотеки для розпізнавання мови коротких повідомлень. Одночасно зі створенням бібліотеки, необхідно порівняти ефективність різних методів розпізнавання (описані у розділі 1.2.1-1.2.4) і вибрати найефективніший з них, який буде використаний для реалізації системи (може бути декілька).

Для досягнення цієї мети необхідно реалізувати базові функції майбутньої системи. Далі під ці функції підібрати оптимальні параметри, виходячи з результатів тестувань. Необхідно провести тестування з різними розмірами тренувальної та тестової бази, з різними мовами тощо. Перед розробкою REST API необхідно переконатися, що бібліотека працює справно (за допомогою юніт- та мануальних тестів).

Далі, коли бібліотека розроблена, можна переходити до створення дизайну REST API. Необхідно продумати, які ресурси і методи будуть використані, як API буде обробляти користувацькі помилки. API повинен бути протестоване за допомогою тестового клієнта (модуль flaskapi).

Критеріями успішного виконання задачі є:

1. високі результати точності визначення мови повідомлення;
2. стійкість до неправильно введених даних;
3. коректно спроектована архітектура, за якою кожен модуль матиме свою відповідальність;
4. REST API має надавати зручний інтерфейс для роботи з верхнім модулем бібліотеки;
5. стандарти кодування повинні бути консистентними крізь усю систему.

# 2 ФОРМУВАННЯ ВИМОГ ДО ПРОГРАМНОЇ СИСТЕМИ

З метою уникнення різних інтерпретацій майбутнього продукту необхідно сформувати конкретні та деталізовані вимоги до системи. В даному документі будуть розглянені функціональні та нефункціональні вимоги.

Бібліотека повинна надавати зручний інтерфейс (API), а саме надавати функції завантаження як тренувальних, так і тестових даних заданою мовою з різних ресурсів; функції визначення мови заданого повідомлення.

Бібліотека повинна бути написана з використанням мови програмування Python та бібліотеки для машинного навчання Scikit-Learn. Бібліотека повинна використовувати SQLite у якості СУБД. Використовувати SQLite можна за допомогою вбудованої у Python бібліотеки та до того ж для СУБД зберігає дані на диску і не потребує тривалого налаштування.

Так, функціональні вимоги описують поведінку функцій системи, деталі внутрішньої реалізації. Тоді, система повинна забезпечувати виконання наступних функціональних вимог користувача:

1. завантаження тренувальних даних;
2. завантаження тестових даних;
3. видалення усіх даних;
4. опціональне використання методу крос-валідації;
5. вибрати класифікатор, на основі якого буде проводитися класифікація та визначення мови повідомлення;
6. визначити мову короткого повідомлення.

Нефункціональні вимоги відображають власне роботу користувача з системою. Отже, бібліотека повинна задовольняти наступні не функціональні вимоги:

1. бути стійкою до некоректно введених даних користувачем;
2. коректно обробляти помилки сторонніх сервісів;
3. має гарантувати коректність зберігання даних;
4. має надавати лише необхідні функції відкритими, залишаючи внутрішні деталі реалізації прихованими;
5. відкритий API повинен бути детально задокументований;
6. код бібліотеки повинен мати коментарі у разі, якщо вибране рішення не є явним;
7. не повинна використовувати зайві бібліотеки, тим самим не створюючи надлишкових залежностей;
8. не вимагати багато зусиль для встановлення (як самої бібліотеки, так і залежностей);
9. має бути ретельно протестована;
10. не працює при відсутності підключення до Інтернету;
11. повинна підтримуватися різними програмними платформами (Windows, Linux та ін.), що забезпечується встановленням інтерпретатору мови програмування Python.

Також повинен бути реалізований REST API клієнт, написаний на мові програмування Python з використанням веб-фреймфорку Flask. Клієнт повинен відповідати принципам RESTful веб-сервісів, взаємодія с клієнтом повинна проходити за допомогою HTTP-методів та JSON.

Клієнт повинен задовольняти наступним функціональним вимогам:

а) надавати функції завантаження тренувальних та тестових даних з вказуванням кількості ;

1. надавати функції перегляду тренувальних та тестових даних за мовою;
2. надавати функції визначення мови тексту з можливістю вибору методу класифікації;
3. надавати функції очищення усіх збережених даних;
4. перевіряти ефективність роботи класифікатора на основі тестових даних, заданих користувачем.

Клієнт повинен задовольняти наступним нефункціональним вимогам:

1. бути стійким до неправильно введених користувачем даних (за допомогою повідомлень з помилкою та HTTP-статусів);
2. має бути ретельно протестований;
3. автоматичне опрацювання помилок у фреймворку Flask повинні бути замінені на користувацькі – повертати коректний статус і повідомлення у форматі JSON;
4. бути максимально простим для використання;
5. має дотримуватись стандартів HTTP – використовувати HTTP-статуси, заголовки тощо;
6. повинен мати документацію для точок входу. Як засіб для документації може бути використаний Swagger – додадок для створення специфікації API, який надає можливість відправляти запроси на API безпосередньо чрез специфікацію та створювати документацію у форматі PDF.

Для тестування і підтримки бібліотеки повинні бути написані unit-тести. Unit-тести дозволяють перевірити роботу окремих модулів бібліотеки. Рекомендується практикувати TDD підхід для написання коду. Головним правилом TDD є написання тестів ще до коду та рух якнайменшими ітераціями (мінімально необхідними діями, щоб зробити тест зеленим).

Для unit-тестування необхідно використовувати unittest та flaskapi для Flask-додатку у Python. Тести повинні охоплювати весь критичний функціонал та пункти, які описані у наведених вище вимогах. API-тести (тести за допомогою тестового клієнта у flaskapi) повинні покривати роботу API: повернення коректних статусів, опрацювання помилок та роботу з бібліотекою.

Окрім unit-тестування, вже після того як система готова працювати, треба провести інтегральне (комплексне) тестування. У результаті цього можна побачити як різні модулі взаємодіють в цілому в реальних умовах, побачити можливі помилки та виправити їх.

# 3 АРХІТЕКТУРА ТА ПРОЕКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

## 3.1 UML-проектування ПЗ

Під час моделювання було побудовано 5 діаграм, які цілковито відображають роботу та функціонал майбутньої системи.

На рис.3.1 відображена високорівнева структура системи. Коли дані приходять на розпізнавання, вони підлягають первинній обробці, тобто, видаленню шумів з тексту (таких як гіперпосилань, згадувань інших користувачів та ін.), видаленню дуже коротких повідомлень (які не представляють цінності для класифікації) приведенню кодування символів до загальноприйнятого у системі.

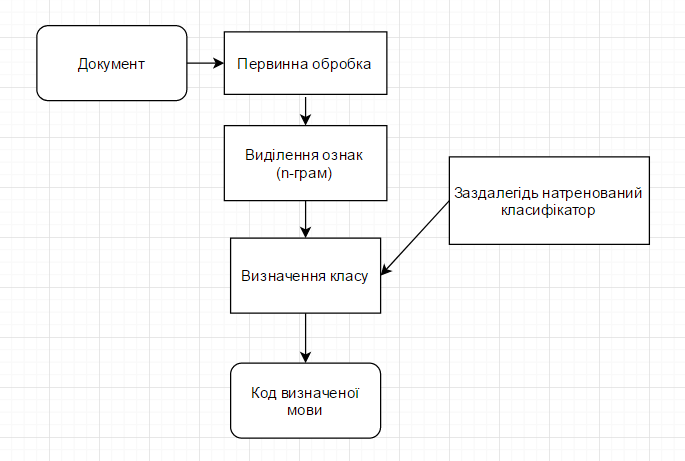


Рисунок 3.1 – Високорівнева структура системи

Далі проводить виділення ознак (feature extraction). Це може бути розбиття тексту на n-грами, або просте розбиття на слова. N-грами можуть бути задовільного розміру, який залежить від конкретних даних. Далі, припускаючи, що класифікатор вже натренований, треба виконати операцію класифікації і на виході буде отримана передбачена мова повідомлення.

На рисунку 3.2 зображена діаграма активності системи. На даній діаграмі відображений процес користування користувачем бібліотекою и опрацювання помилок у разі, якщо користувач намагається для розпізнавання використати класифікатор, який не підтримується системою.

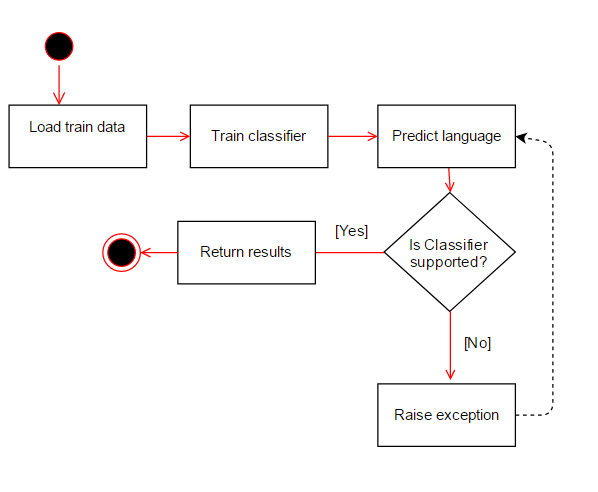


Рисунок 3.2 – Діаграма активності

На рисунку 3.3 зображена діаграма пакетів. На цій діаграмі відображені як залежності між модулями усередині системи, так і залежності модулів від сторонніх бібліотек та систем (scikit-learn, twitter, SQLite). З діаграми видно, що модулем верхнього рівня (власне, API) є модуль ‘functions.py’, який використовує модулі ‘data.py’, для завантаження тренувальних та тестових даних, та ‘language\_detection.py’, у якому міститься логіка для визначення мови повідомлення – попередня обробка та класифікатори. Ці модулі більш детально описані у розділі 3.2.

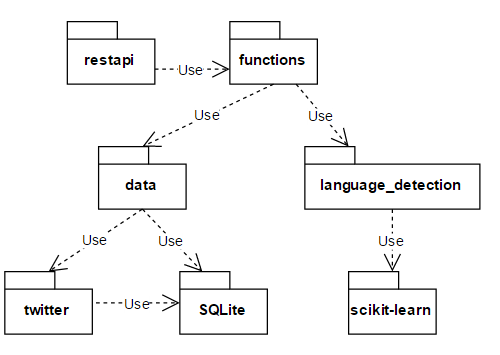


Рисунок 3.3 – Діаграма пакетів

Також на діаграмі присутній модуль restapi, імплементований за допомогою веб-фрейморку Flask, оснований на принципах RESTful веб-сервісів. Він звертається виключно до модуля верхнього рівня ‘functions.py’ і не знає про дочірні модулі. Призначення модуля restapi детально описане у розділі 3.5.

На діаграмі послідовності (рис. 3.4) зображено актора (користувача) і послідовність дій, яку потрібно виконати, аби розпізнати мову повідомлення, використовуючи бібліотеку. Також на діаграмі показано відповідальність кожного модуля і основні їх функції. Варто відзначити, що функцію load\_train\_data() треба виконати не менше 2 разів для різних мов. Наприклад, у такій послідовності: load\_train\_data(‘en’, 200), load\_train\_data(‘ru’, 200). І далі викликати функцію detect\_language(message), яка поверне передбачувану мову.

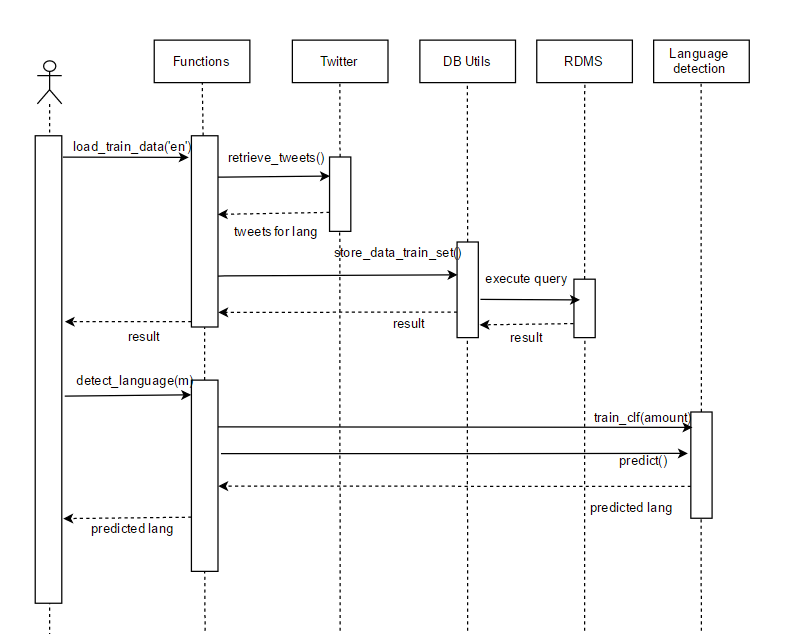


Рисунок 3.4 ­ Діаграма послідовності

На рис. 3.5 зображена високорівнева діаграма послідовності для використання REST API ендпоінтів. Тут відображені лише два компонента – головний модуль бібіліотеки та власне API, – бо усі внутрішні функції представлені на рис. 3.4. Модуль `restapi` звертається виключно до модуля верхнього рівня `functions`. На діаграмі представлена мінімальна послідовність дій, які необхідно виконати для досягнення бажаного результату.

Також діаграма відображає URL та методи для запитів, статуси успішного виконання запитів та грубе співвідношення між запитами до REST API та функціями головного модуля бібліотеки (functions). Варто зауважити, що на даній діаграмі відображені не усі доступні ендпоінти. Більш детально ця функціональність буде розглянута у розділі 3.4.

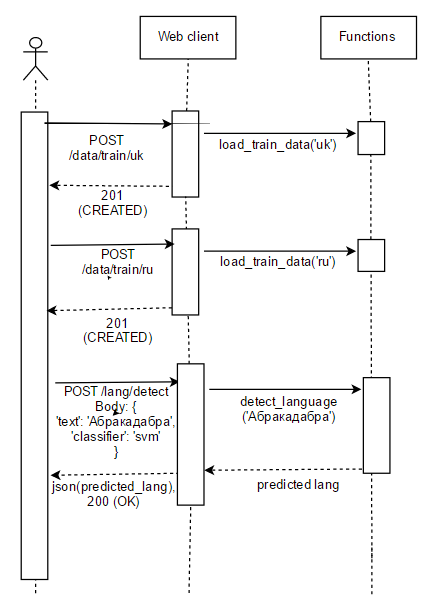


Рисунок 3.5 – Діаграма послідовностей для веб-клієнта

На рис. 3.6 знаходиться діаграма розгортання, яка відображає взаємодію обчислювальних вузлів під час роботи програми. Так, коли виконується запит до REST API за протоколом HTTP на порт 5000 , головний модуль звертається до дочірнього, який у свою чергу робить запит до Twitter Search API, який надає короткі повідомлення, які надалі використовуються як тренувальні або тестові дані. Запит виконується за протоколом HTTPS з попереднім отриманням токену для виконання подальших запитів до API.

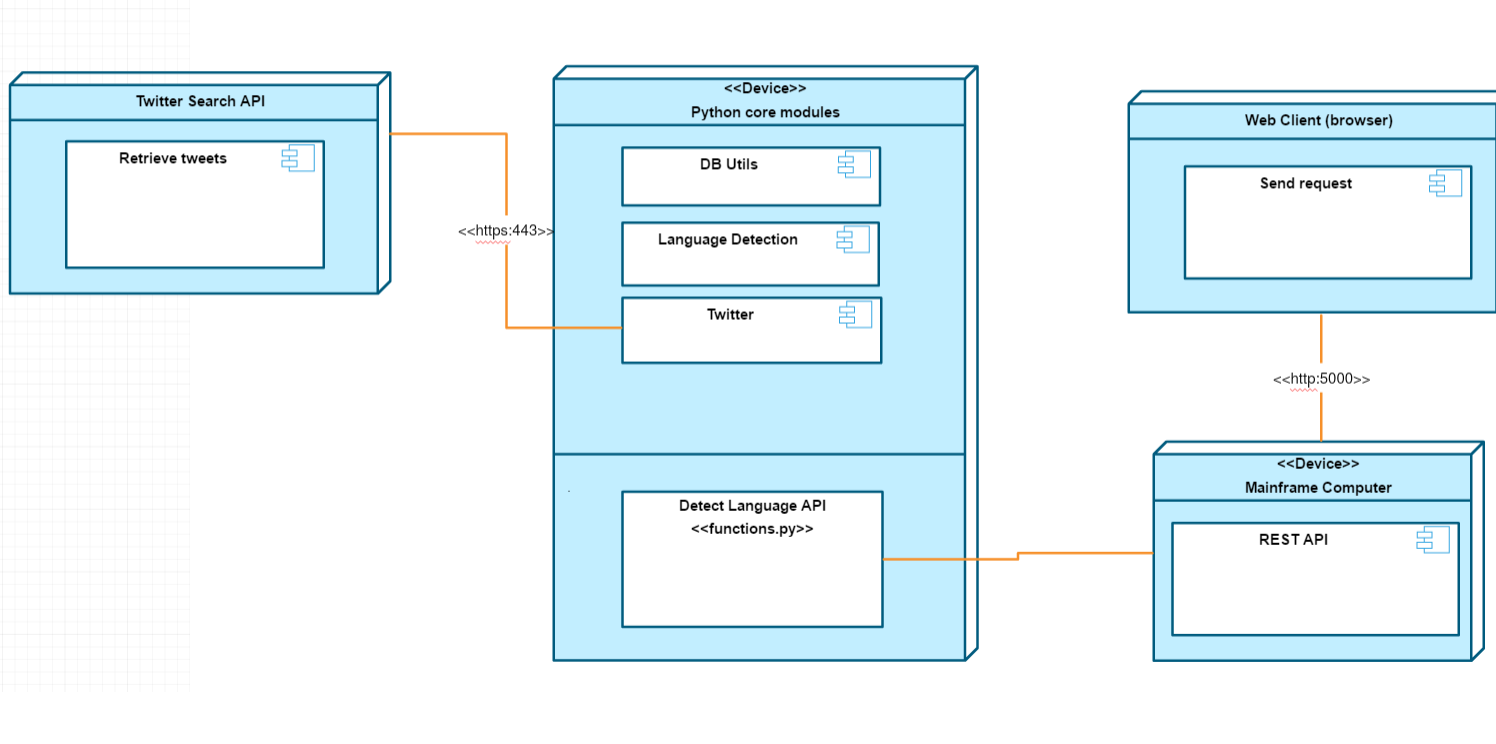


Рисунок 3.6 – Діаграма розгортання

Усі представлені діаграми цілком описують роботу системи та її взаємодію зі сторонніми компонентами.

## 3.2 Проектування архітектури ПЗ

Для імплементації бібліотеки були вибрані наступні технології та фреймворки:

1. Python – об'єктно-орієнтована мова програмування високого рівню з динамічною типізацією, підтримує пакети та пакети модулів, що сприяє повторному використанню коду; для Python реалізована велика кількість бібліотек для машинного начання;
2. Scikit-learn – бібліотека, написана для мови програмування Python, яка імплементує різноманітні алгоритми машинного навчання – класифікація, регресія, кластерізація, первинна обробка та інші. Вміщує вбудовані набори даних, які є надзвичайно корисними на початку роботи з бібліотекою. Містить також модуль для оцінки результатів роботи, використовуючи тренувальні та тестові дані або метод крос-валідації (sklearn.metrics). Працює за допомогою чисельних та наукових бібліотек NumPy і SciPy. Деякі алгоритми написані на Cython – адаптер мови Python до C/C++, дозволяє визвати функції на C/C++ напряму з коду на Python – для кращої продуктивності алгоритмів;
3. Flask – відкритий веб-фреймворк для мови програмування Python, призначений для розробки веб-систем; не потребує спеціальних засобів чи бібліотек, тому вважається легковаговим; є одним з найпопулярніших веб-фреймворків (на ресурсі GitHub) для Python; має вбудований сервер та відладчик, можливості інтегрувального тестування, підтримку RESTful запитів, підтримку безпечних кукіс (дані, які сервер зберігає про клієнта на стороні клієнта);
4. SQLite – одна з найпоширеніших систем керування базами даних, що надає багатий набір функціональних можливостей, які підтримують безпечне середовище для зберігання, обслуговування і отримання даних. SQLite — характеризується великою швидкістю, стійкістю і простотою використання, була розроблена для підвищення швидкодії обробки великих баз даних; бібліотека для роботи з СУБД є вбудованою у мову програмування Python.

Середою розробки бібліотеки був вибраний PyCharm від JetBrains. Ця середа розробки має багато переваг, серед яких зручне підсвічування синтаксису, зручна навігація по файлам проекту та відображення проекту, вбудовані елементи для юніт-тестування та форматування коду згідно стандартів, підтримка системи контролю версій та інше.

Система буде організована як декілька незалежних модулів, тобто кожен модуль може працювати окремо. Також, буде окремий головний модуль верхнього рівня, який надаватиме зручний спосіб використання бібліотеки: максимально зрозуміли функції з невеликою кількістю параметрів. Цей модуль повинен бути задокументований. Функції, які будуть у ньому доступні, повинні бути функціями лише верхнього рівня, а внутрішня реалізація системи повинна бути прихована.

Система не буде кожен раз знову тренувати класифікатор. Замісто цього система зберігатиме параметри моделей та інші необхідні дані у базі даних. Тренувальні та тестові дані (корпуса, тексти повідомлень у соціальній мережі Twitter) також будуть зберігатися у базі даних. Варто зауважити, що тренувальні і тестові дані не повинні перетинатися, бо в іншому випадку ми не зможемо перевірити як натренована модель. Це повинно бути гарантовано модулями пов’язаними з викачуванням даних та зберіганнях їх у базу даних. Також треба регулярно перевіряти модель на перенавчання. Це можна робити за допомогою метода крос-валідації.

Крос-валідація – це ефективний метод оцінки моделі та її продуктивності на незалежних даних. Для такої оцінки вхідні дані розбиваються на k частин. Далі на k-1 частинах проводять тренування моделі, а на інших – тестування. Це проводиться k раз, і в результаті ми отримуємо оцінку ефективності моделі з найбільш рівномірним розподілом даних. Для крос-валідації необхідною умовою є різнорідність даних, тобто дані не повинні співпадати та повторюватися.

Тестування – важливий етап розробки бібліотеки та REST API клієнта. Тож, для тестування функцій бібліотеки написані юніт-тести, використовуючи бібліотеку unittest для Python. Для тестування коректної поведінки REST API ендпоінтів написані юніт-тести, які використовують flaskapi як тестовий клієнт. Також написані інтеграційні тести, які перевіряють взаємодію різних модулів і особливо бази даних. Більш детально тестування додатка буде описано у розділі 5.

## 3.3 Проектування бази даних

У якості системи керування базами даних була вибрана реляційна система SQLite. Важливою перевагою цієї системи є простота її реалізації: у порівнянні з більшістю інших СУБД, SQLite не використовує відому парадигму клієнт-сервер, а лише зберігає усю інформацію у файлі на диску (зазвичай, розширення \*.db), на якому виконується додаток. Це не потребує додаткових зусиль від користувача. Варто відмітити, що система є крос- платформеною.

До того ж, SQLite надзвичайно легко використовувати разом з Python, бо бібліотека для роботи з цією СУБД (sqlite3) є вбудованою у мову програмування Python та веб-фреймворк Flask, так що для використання бібліотеки не потребується встановлювати додаткові драйвери бази даних та інші залежності. Саме це і робить її надзвичайно простою до використання.

Для виконання завдання використовувалося дві ідентичні таблиці з однаковими колонками: одна для зберігання тренувальних даних (`train\_set`), інша – для тестових (`test\_set`) (рис 3.7).

Для обох таблиць встановлено обмеження, що значення поля `data` – тобто, тексту, повинно бути унікальним в рамках таблиці, що гарантується бібліотекою, а саме модулем dbutils.py та twitter.py. Саме тому, унікальний ідентифікатор в якості первинного ключа для кожного запису був би зайвим.

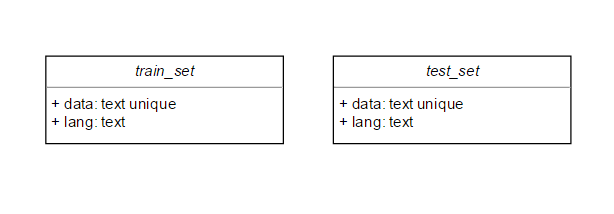


Рисунок 3.7 – Структура бази даних

Таблиця train\_set необхідна для збереження даних для подальшого тренування класифікатора. Зазвичай, класифікатор тренують на даних того самого типу, що і будуть розпізнаватися. Наприклад, повідомлення у соціальній мережі Twitter є досить специфічними (містять не літературну мову, скорочення тощо) у порівнянні зі звичайними текстами (наприклад, мовними корпусами, які стандартизовані), тому класифікатор необхідно тренувати також на таких повідомленнях. Перед зберіганням тексту тренувальної бази, він піддається обробці: з рядків видаляються «шумні» символи (HTTP посилання, згадування інших користувачів за допомогою «@» та інші), та рядки, довжина який менша за 20 символів видаляються.

Таблиця test\_set необхідна для оцінки роботи класифікатора. Вона є ідентичною до таблиці train\_set. Перед збереженням тестових даних вони проходять таку ж саму обробку як і тренувальні.

Для коректної оцінки необхідно, щоб дані у тренувальній і тестовій таблицях не збігалися. Таким чином можна провести оцінки декількох класифікаторів на однакових даних. Варто зауважити, що оцінку також можна провести методом крос-валідації.

## 3.4 Створення REST API для системи

Під час дизайну REST API усі рішення покладалися на специфікацію HTTP та принципи RESTful веб-сервісів. Передача даних на веб-клієнті ґрунтується на протоколі передачі даних – в цьому випадку на HTTP, а тому і на HTTP-методах. API має певну кількість ресурсів (URI), операції над якими визначаються використовуваним HTTP-методом. Так, для отримання даних використовується метод GET, для створення – POST, для видалення – DELETE тощо.

Також, API повідомляє про успішне/неуспішне виконання запиту за допомогою статусу відповіді (response status). З успішних статусів використовуються наступні значення: 200 – OK (повертається відповідь, зазвичай на GET), 201 – CREATED (ресурс створений, зазвичай на POST), 204 – NO\_CONTENT (пуста відповідь, зазвичай на DELETE).

Спроектовані методи, URI та успішні (очікувані) статусі наявні у таблиці 3.1. Вони відповідають функціям у головному модулі functions.py.

Таблица 3.1 – URI та методи, які вони підтримують

|  |  |  |  |
| --- | --- | --- | --- |
| URI | Метод | Статус | Опис |
| /data/train/{lang} | POST | 201 | Завантажити та зберегти тренувальні дані. Об’єм тренувальних даних передається у JSON-форматі запиту. |
| GET | 200 | Отримати тренувальні дані за заданою мовою. У відповіді знаходиться загальна кількість записів та власне записи |
| /data/test/{lang} | POST | 201 | Завантажити та зберегти тестові дані. Об’єм тренувальних даних передається у JSON-форматі запиту |
| GET | 200 | Отримати тестові дані за заданою мовою. У відповіді знаходиться загальна кількість записів та власне записи |
| /data | DELETE | 204 | Знищити усі попередньо завантажені дані |
| /lang/detect | POST | 200 | Визначити мову повідомлення, яке передається у JSON-запиті |

Для головної функції системи, яка для REST API відповідає ендпоінту POST /lang/detect був вибраний метод POST (хоча сам запит відповідає методу GET). Такий вибір полягає в тому, що розмір тексту, мову якого треба ідентифікувати, може бути досить великого розміру і повідомлення не поміститься в параметр запиту метода GET (в URL), тоді як запит переданий у тілі запиту (у форматі JSON) може мати необмежений розмір.

Текст, який необхідно розпізнати, повинен бути переданий у JSON-запиті до ресурсу. Також, опціонально можна передати назву класифікатора, за якою буде проводитися класифікація. Якщо назва класифікатора не зазначена (не передана у запиті), то буде використаний встановлений за замовчуванням – наївний байєсів.

Для більш детальної документації та відправлення тестових записів можна звернутися до відкритої специфікації Swagger.

# 4 ОПИС ПРИЙНЯТИХ ПРОГРАМНИХ РІШЕНЬ

Для імплементації бібліотеки була вибрана мова програмування Python. Python добре підходить для задач машинного навчання та обробки природньої мови. Адже машинне навчання власне складається з двох етапів: будування моделі (тренування) та передбачення результату. Тренування моделі – це обчислювально важка операція, а скрипти на Python працюють досить швидко.

Також, для Python розроблена велика кількість бібліотек, пов’язана з машинним навчання та математичними обчисленнями – наприклад, SciPy, NumPy, Scikit Learn, Pandas, які є доволі специфічними. Scikit Learn надає зручний інтерфейс для роботи з якісними імплементаціями популярних алгоритмів – класифікації, кластеризації, регресії та інше. Для використання бібліотеки є дуже детальна документація, яка розміщена на сайті бібліотеки [11].

Python має вбудований модуль для роботи с системою керування базами даних SQLite – sqlite3. Цієї СКБД достатньо для невеликої бази даних для обробки природньої мови.

У таблиці 4.1 наведене порівняння точності прогнозу класифікаторів для різних мов. Варто зауважити, що розміри тренувальної і тестової бази були однакові для кожної мови і містили по 400 коротких повідомлень. Також, усі параметри класифікаторів задані за замовчуванням. Ці параметри будуть підібрані пізніше після проведення подальшого тестування цих класифікаторів.

Таблиця 4.1 – Порівняння роботи різних класифікаторів

|  |  |  |
| --- | --- | --- |
| Класифікатор | Мова | Результат |
| Метод опорних векторів (SVM) | ru | 0.92 |
| uk | 0.63 |
| sr | 0.71 |
| bg | 0.46 |
| середнє | 0.68 |
| Метод найближчих сусідів | ru | 0.67 |
| uk | 0.89 |
| sr | 0.53 |
| bg | 0.51 |
| середнє | 0.65 |

Продовження таблиці 4.1

|  |  |  |
| --- | --- | --- |
| Класифікатор | Мова | Результат |
| Наївний байєсів | ru | 0.85 |
| uk | 0.9 |
| sr | 0.83 |
| bg | 0.53 |
| середнє | 0.78 |

Варто згадати, що тренувальні дані не чисті, адже як було виявлено при мануальних запитів до Twitter API, використовуючи українську та російську мови, що самі повідомлення, які повертаються не запитом на певну мову, не завжди дійсно написані тією мовою. Отже, тренувальна база є шумною і можна очікувати невисокі результати для певних мов.

Також були проведені окремі тести для оцінки кожного з класифікаторів. Наприклад, для методу найближчих сусідів використовувалися різні кількості сусідів (від 2 до 6), результати зі значенням 5 були найбільш високими. Саме це значення і задає Scikit-Learn за замовчуванням. При використанні різної функції зважування (‘distance’, ‘uniform’) при прогнозі не змінило загальний показник, були отримані різні результати лише для певних мов.

При тестуванні методі опорних векторів використовувалися різні функції ядра: лінійне розділення, многочленна функція розділення, поліноміальна функція, Ґаусова радіально-базисна функція. Остання власне використовується за замовченням. Виконавши експерименти, було з’ясовано, що класифікатор краще працює з функцією лінійного розділення, аніж з Ґаусовою радіально-базисною функцією. Результати наведені у таблиці 4.2.

Таблиця 4.2 – Порівняння методу опорних векторів з різними функціями ядра

|  |  |  |
| --- | --- | --- |
| Мова | Ґаусова радіально-базисна | Лінійна |
| ru | 0.92 | 0.88 |
| uk | 0.63 | 0.86 |
| sr | 0.71 | 0.88 |
| bg | 0.46 | 0.48 |
| середнє | 0.68 | 0.77 |

З результатів можна відмітити значне покращення середнього результату, хоча при цьому впала точність визначення повідомлень російською мовою. Отже, оптимальною функцією ядра буде лінійна.

Для аналізу ефективності наївного байєсівського класифікатора змінювався параметр ‘alpha’, який є коефіцієнтом згладжування Лапласу. За замовчуванням приймає значення 1.0. Результати дослідження наведені у таблиці 4.3.

Таблиця 4.3 –Повірняння різного параметра ‘alpha’ для байєсівського класифікатора

|  |  |  |  |
| --- | --- | --- | --- |
| Мова | alpha=0.02 | alpha=0.2 | alpha=1.0 |
| ru | 0.90 | 0.88 | 0.85 |
| uk | 0.92 | 0.92 | 0.90 |
| sr | 0.88 | 0.85 | 0.83 |
| bg | 0.55 | 0.53 | 0.53 |
| середнє | 0.81 | 0.80 | 0.78 |

З результатів зрозуміло, що хоча для середнього значення зміни невеликі, то якщо подивитися детальніше, то можна побачити помітні зміни у окремих мовах, аж до різниця у 5 відсотків. Отже, оптимальним параметром наразі є alpha=0.02, тобто параметр адитивного згладжування дорівнює 0.02.

Можна підвести підсумок, що у середньому найвищі результати показує Байєсівська класифікація, хоча її алгоритм є найпростішим.

Усі вище названі класифікатори тестувалися на попередньо перетворених на n-грами повідомленнях. Початкова довжина n-грам становить від 1 до 3. Тепер потрібно визначити оптимальні довжини n-грам, при який результати будуть найвищими. Отримані результати були майже однакові: середня точність у межах від 0.72 (для великих n-грам) до 0.82 (до максимально коротких). Деякі результати представлені у таблиці 4.4. Варто зауважити, що для цих тестів використовувався наївний байєсів класифікатор з оптимальними параметрами.

Таблиця 4.4 – Повірняння ефективності байєсівогу класифікатора з n-грамами різних довжин

|  |  |  |  |
| --- | --- | --- | --- |
| Мова | (1, 2) | (1, 3) | (5, 8) |
| ru | 0.94 | 0.90 | 0.64 |
| uk | 0.96 | 0.92 | 0.82 |
| sr | 0.89 | 0.88 | 0.75 |
| bg | 0.48 | 0.55 | 0.64 |

Продовження таблиці 4.4

|  |  |  |  |
| --- | --- | --- | --- |
| середнє | 0.82 | 0.81 | 0.72 |

Хоча середній результат для n-грам довжиною від 1 до 2 символів трохи кращий, можна побачити помітне погіршення результатів для визначення болгарської мови. За тим же принципом відкидається останній результат, у якому бачимо значне покращення розпізнавання для болгарської мови, але ж усім іншим притаманне погіршення більше ніж на 10 відсотків. Тому оптимальна довжина n-грам становить від 1 до 3-х символів.

Так як користувач має змогу вибрати класифікатор, з яким працювати, визначимо мапу класифікаторів (рис. 4.1), об’єкт, який розбиватиме повідомлення на n-грами (CountVectorizer), та об’єкт, який перетворює кількості токенів в їх частоти (TfidfTransformer).

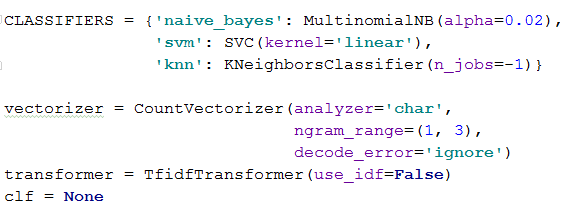


Рисунок 4.1 – Множина класифікаторів

І далі створимо конвеєр (pipeline), з яким будемо надалі працювати (рис. 4.2). Конвейєр включає в себе попередньо описані векторизатор, трансформер та класифікатор.

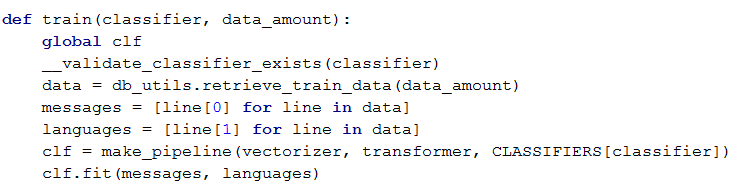


Рисунок 4.2 – Тренування класифікатору

Конвейєри використовуються, щоб зібрати декілька послідовних операцій докупи та використовувати як один об’єкт. Також при створенні конвеєру йому можна передати параметри, які будуть використані для усіх етапів конвеєру. Усі елементи конвеєру повинні бути трансформаторами, тобто мати метод transform(), який і буде викликаний конвеєром під час виконання.

Для прогнозування класу повідомлення потрібно викликати метод predict() (рис. 4.3).

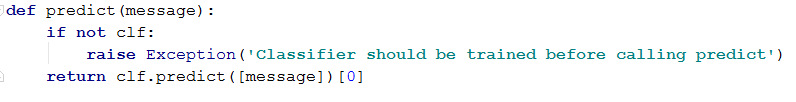


Рисунок 4.3 – Виклик методі конвеєра для передбачення

def predict(message):

…

return clf.predict([message])[0]

Конвейєр при цьому викликає метод predict() на його останньому етапі, тобто на об’єкті класифікатора.

Веб-фреймворк Flask є мікрофреймворком, тобто підходить до розробки невеликих додатків. Адже REST API клієнт містить логіку, зв’язану виключно з валідацією та роботою з HTTP, та загалом використовує API модуля functions.py, то достатньо такого простого фреймфорка. До того ж, для Flask створені модулі, які дозволяють налаштувати розмежування доступу, тобто аутентифікацію по HTTP.

Для завантаження тренувальних та тестових даних використовується метод POST, адже саме метод POST означає створення нових ресурсів. При успішному виконанні сервер повертає статус 201 (CREATED), що означає успішне завантаження тренувальних або тестових даних (рис. 4.4).

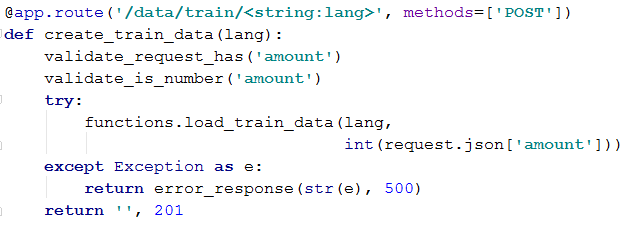


Рисунок 4.4 – Ендпоінт для завантаження тренувальних данних

Для отримання тренувальних чи тестових даних використовується метод GET, який загалом використовується для отримання даних з серверу. Сервер повертає список тренувальних чи тестових даних за запитаною мовою та статус 200 (OK), якщо запит був успішним, при цьому сам фрейворк надає можливості налаштувати валідацію даних, які передаються як параметр шляху (рис. 4.5).

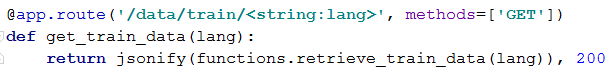


Рисунок 4.5 – Ендпоінт для отримання тренувальних даних

Також створений ресурс для видалення даних, який підтримує метод DELETE (рис. 4.6). Цей метод використовується для видалення ресурсів з сервера. У випадку вдалого виконання повертається статус 204 (NO\_CONTENT).

Якщо користувач у запиті не передає дані, необхідні для виконання запиту, або ж ці дані є некоректними, запит не буде виконаний та сервер поверне статус 400 (BAD REQUEST) з описанням помилки. Якщо на сервері виникла помилка, то клієнту повернуться статус 500 (INTERNAL SERVER ERROR) з описанням помилки.

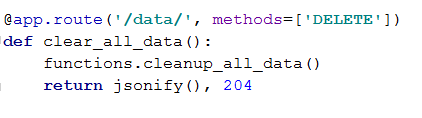


Рисунок 4.6 – Ендпоінт для видалення даних

Найголовніший ресурс додатку – визначення мови повідомлення підтримує метод POST (рис. 4.7). За визначенням цей ендпоінт повинен відповідати на метод GET, але у методі GET неможливо передати велике повідомлення, його розмір обмежений максимальною довжиною URL, яка становить 2083 символи. Саме тому для цього ресурсу використовується метод POST, у тілі якого можна передати довге повідомлення, і у тому ж JSON передати назву класифікатора.

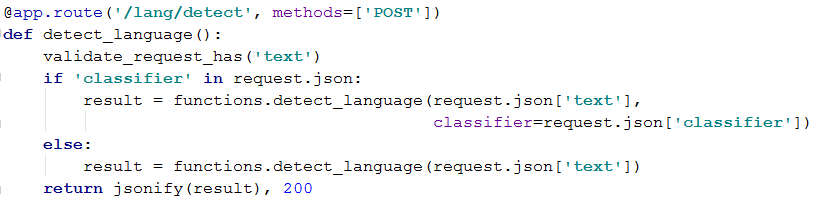


Рисунок 4.7 – Ендпоінт для визначення мови тексту

У відповідь на запит сервер повертає двозначний код мови та успішний статус 200.

# 5 ТЕСТУВАННЯ РОЗРОБЛЕНОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Тестування – невід’ємна частина розробки програмного забезпечення. Тестування дозволяє безпечно для існуючого функціоналу розробляти новий, так як тести вкажуть на можливі помилки. За наявності тестів, які покривають весь основний функціонал додатку, можна завжди бути впевненим у коректній роботі програмного забезпечення. Важливо проводити тестування регулярно після будь-яких змін. Саме тому для цього зручно використовувати автоматичні тести, які можна швидко запустити і перевірити коректність роботи системи.

Юніт-тестування (або модульне тестування) – такий метод тестування програмного забезпечення, коли тестуванню підлягає лише окремий модуль або клас. Залежні модулі/класи тоді замінюються на прототипи (test doubles).

У якості інструментів тестування для даного додатку використовувалась бібліотека для Python unittest – версія бібліотеки JUnit адаптована під Python. Ця бібліотека містить методи для написання тестів і їх запуску через наслідування. Unittest має методи, які виконуються перед або після виконанням кожного тесту, що надає змогу створювати та видаляти ресурси. Містить різноманітні методи для перевірки результатів виконання (assertions). Ця бібліотека використовувалась для написання юніт- та інтеграційних тестів.

Для використання бібліотеки unittest необхідно:

1. унаслідувати клас ‘unittest.TestCase';
2. починати імена тестових методів з `test\_`;
3. усередині тестів використовувати методи попередника assertEqual(), assertRaises() та інші;
4. запустити тести за допомогою виклику unittest.main().

Юніт-тестами покривалися як публічні функції, так і внутрішні. Внутрішні функції необхідно тестувати, коли вони є частиною досить великої публічної функції, а результат їх роботи є критичним.

Розробка та тестування модуля `restapi` виконувалась вже після розробки і тестування головного модуля системи. Власне, модуль `restapi` був розроблений за технологією TDD (test driven development) – процес, коли розробка програмного забезпечення керується написаними тестами. Тобто, тести пишуться ще до початку розробки. TDD містить в собі три фази: червона, зелена, рефакторинг (red-green-refactor). Під часи червоної фази відбувається написання тесту, який не проходить (тобро залишається «червоним»). Далі необхідно зробити тест «зеленим»: починається фаза розробки. І нарешті відбувається рефакторинг – підчищення коду після розробки. І завдяки ретельним тестам можна бути впевненим, що після рефакторингу цілісність системи не буде порушена. До того ж, тести також підлягають рефакторингу. Це все сприяє написанню «чистого» коду [12]. WIP

Мануальне (ручне) тестування – метод тестування, під час якого всі сценарії системи вручну перевіряються на помилки. Дуже важливо протягом мануального тестування перевірити якомога більше можливих сценаріїв. Мануальне тестування зазвичай використовується для тестування інтерфейсу користувача. У цій системі мануальне тестування проводилося для усіх моделей системи та окремо для модуля `restapi`.

Для мануального тестування бібліотеки використовувалась консоль мови програмування Python. Таким чином відбувалося інтеграційне тестування методом знизу-вгору, тобто, спочатку біли протестовані модулі нижнього рівня (`dbutils`, `twitter`), а далі все вищого і вищого рівня поки не дісталися до `functions`. На етапі тестування головного модуля системи припускається, що усі інші модулі вже працюють вірно.

Для мануального тестування REST API був використаний додаток до Google Chrome – Postman. Postman має зручний інтерфейс для відправлення HTTP-запитів та аналізу відповідей з сервера. До того ж у додатку можна створювати колекції з запитів, які надалі автоматично запускати. Однак, перевіряти результати роботи також треба мануально.

Мануальне тестування є досить затратним процесом, потребує багато часу, але у той же час є найбільш ефективним, бо тестувальник на момент тестування стає реальним користувачем системи.

WIP

https://coverage.readthedocs.io/en/coverage-4.4.1/

У результаті проведення своєчасного (тобто регулярного) тестування була виявлена певна кількість помилок у різних модулях та на різних рівнях абстракції. Завдяки цьому ці помилки можна було вчасно виправити. Саме тому під час розробки REST API при інтеграції не виникло значних проблем.

# ВИСНОВКИ

У результаті виконання атестаційної роботи бакалавра був проведений аналіз предметної галузі стосовно теми дипломної роботи. Були поставлені вимоги для проектованої системи.

Протягом аналізу предметної галузі були вивчені наступні методи класифікації: метод опорних векторів, наївний байєсів, метод найближчих сусідів; були вивчені аналоги на ринку. Надалі ефективність роботи цих алгоритмів для релевантних даних була порівняна, були вибрані найефективніші коефіцієнти для кожного класифікатору. Найефективнішим класифікатором визначений наївний байєсів.

Було вирішено, що бібліотека визначення мови коротких повідомлень буде реалізована на мові програмування Python з використанням бібліотеки машинного навчання Scikit-Learn. Середовищем розробки був вибраний PyCharm.

REST API клієнт реалізований за допомогою веб-фреймворку для Python – Flask. Клієнт повинен дотримується принципів проектування RESTful додатків.

У якості СУБД вибрана SQLite, бо вона досить легковажна для мети цієї роботи, і модуль для роботи з цією бібліотекою вже вбудований у Python.

Таким чином, в ході роботи була розроблена система розпізнавання мови коротких повідомлень. Система досить вузькоспеціалізована, хоча може бути використана не тільки для розпізнавання текстів, написаних кириличними символами.

# ПЕРЕЛІК ПОСИЛАНЬ

1. Cavnar W.B., N-Gram-based Text Categorization [Текст] / W.B. Cavnar, J.M. Trenkle // UNLV Publications/Reprographics. – 1994.
2. Prager J.M., Language identification for multilingual documents [Текст] / J.M. Prager // Journal of Management Information Systems, 16(3):71-101, – 1999.
3. MacNamara S., Neural networks for language identification: a comparative study. Information Processing and Management [Текст] / S. MacNamara, P. Cunningham, and J. Byrne // Information Processing and Management: an International Journal. – 1998.
4. Elworthy D., Language identification with confidence limits. [Текст] / D. Elworthy // In Proceedings of the 6th Annual Workshop on Very Large Corpora –1998.
5. Shashirekha H.L., Automatic Language Identification from Written Texts – An Overview [Текст] / H.L. Shashirekha // International Journal of Innovative Research in Computer and Communication Engineering. – 2014. - №5
6. Usage of content languages for websites. [Електронний ресурс] / W3Techs – [https://w3techs.com/technologies/overview/content\_language/all 8.5.2017](https://w3techs.com/technologies/overview/content_language/all%208.5.2017) р. – Заголовок з екрана
7. Воронов К.М., Лекции по методу опорных векторов [Текст] / К.М. Воронов – 2007.
8. Breiman L., Random Forests [Текст] / L.Breiman // Machine Learning – 2001. – №45.
9. Manning C., An introduction to information retrieval [Текст] / Cambridge: Cambridge University Press. – 2008.
10. Boldwin L., Cross-domain Feature Selection for Language Identification [Текст] / L. Boldwin, M. Boldwin, T. Baldwin // IJCNLP, Thailand – 2011.
11. User guide: contents [Електронний ресурс] / Scikit learn – [http://scikit-learn.org/stable/user\_guide.html 28.5.2017](http://scikit-learn.org/stable/user_guide.html%2028.5.2017) р. – Заголовок з екрана
12. Martin R. C., Clean Code: A Handbook of Agile Software Craftsmanship [Текст] / R.C. Martin // Prentice Hall; 1 edition – 2008.