МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра програмної інженерії

КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА

на тему: **«ВИКОРИСТАННЯ ТЕХНОЛОГІЙ МАШИННОГО НАВЧАННЯ У ВЕБРОЗРОБЦІ»**

Виконав: студент 2 курсу, групи 8.1212-іпз-1
спеціальності 121 інженерія програмного забезпечення
(шифр і назва спеціальності)
освітньої програми інженерія програмного забезпечення
(назва освітньої програми)
П.В. Нетреба
(ініціали та прізвище)

Керівник доцент кафедри програмної інженерії, доцент, к.ф.-м.н. Мильцев О.М.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Рецензент завідувач кафедри фундаментальної та прикладної математики, професор, д.т.н. Гребенюк С.М.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя – 2023

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет математичний

Кафедра програмної інженерії

Рівень вищої освіти магістр

Спеціальність 121 інженерія програмного забезпечення
(шифр і назва)

Освітня програма інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри програмної інженерії, к.ф.-м.н., доцент

_____ Лісняк А.О.
(підпис)

“ ____ ” _____ 2023 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Нетребі Павлу Володимировичу
(прізвище, ім’я та по-батькові)

1. Тема роботи Використання технологій машинного навчання у веброзробці

керівник роботи Мильцев Олександр Михайлович, к.ф.-м.н., доцент
(прізвище, ім’я та по-батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від « 01 » травня 2023 року № 642-с

2. Строк подання студентом роботи 27.11.2023 р.

3. Вихідні дані до роботи 1. Постановка задачі.
2. Перелік літератури.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Технічне завдання.
2. Проєктування та теоретичне обґрунтування системи.
3. Реалізація системи.

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)
презентація

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання 12.05.2023 р.

К А Л Е Н Д А Р Н И Й П Л А Н

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Розробка плану роботи.	26.05.2023	
2.	Збір вихідних даних.	09.06.2023	
3.	Обробка методичних та теоретичних	30.06.2023	
	джерел.		
4.	Розробка першого та другого розділу.	01.09.2023	
5.	Розробка третього розділу.	27.10.2023	
6.	Оформлення та нормоконтроль	20.11.2023	
	кваліфікаційної роботи магістра.		
7.	Захист кваліфікаційної роботи.	15.12.2023	

Студент _____ П.В. Нетреба
(підпис) (ініціали та прізвище)

Керівник роботи _____ О.М. Мильцев
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

Нормоконтролер _____ А.В. Столярова
(підпис) (ініціали та прізвище)

РЕФЕРАТ

Кваліфікаційна робота магістра «Використання технологій машинного навчання у веброзробці»: 89 с., 33 рис., 5 табл., 42 джерела, 1 додаток.

КЛАСИФІКАЦІЯ ВІДГУКІВ, МАШИННЕ НАВЧАННЯ, ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ, NLTK, PANDAS, PYTHON, SCIKIT-LEARN.

Об’єкт дослідження: бінарний автоматичний класифікатор тональності текстів.

Метою дослідження є створення автоматичного класифікатора із застосуванням методів машинного навчання на основі відгуків про програмні продукти.

Методи дослідження: аналіз текстової інформації, машинне навчання, статистичні методи, валідація та оцінка результатів.

При розробці системи було проведено аналіз предметної області, обрана архітектура і платформа реалізації, розроблена функціональна модель системи, зібрані навчальні та тестові корпуси даних для класифікації відгуків про програмне забезпечення на негативні та позитивні; написано код класифікатора мовою програмування *Python* з застосування методів машинного навчання.

Практична значимість полягає у навчальному відпрацюванні класичного завдання класифікації відгуків на програмне забезпечення. Надалі на відпрацьованих алгоритмам можна буде застосовувати навчальну та тестову вибірки з різною тематикою.

SUMMARY

Master’s qualifying paper «Using Machine Learning Technologies in Web Development»: 89 pages, 33 figures, 5 tables, 42 references, 1 supplement.

REVIEW CLASSIFICATION, MACHINE LEARNING, SOFTWARE, NLTK, PANDAS, PYTHON, SCIKIT-LEARN.

The object of the study is binary automatic text tone classifier.

The aim of the study is to create an automatic classifier using machine learning methods based on software product reviews.

The methods of research are text information analysis, machine learning, statistical methods, validation and evaluation of results.

In developing the system, we analyzed the subject area, selected the architecture and implementation platform, developed a functional model of the system, collected training and test data sets to classify software reviews into negative and positive; wrote the classifier code in Python using machine learning methods.

The practical significance lies in the training of the classical task of classifying software reviews. In the future, the developed algorithms can be used for training and test samples with different topics.

ЗМІСТ

ВСТУП

У сучасному діловому світі одним із важливих завдань є отримання зворотного зв’язку від споживача. Якщо таких коментарів надходить занадто багато, виникає завдання їх автоматичного сортування.

У даному дослідженні зроблено спробу автоматичного виявлення корисних коментарів на програмні продукти, які пропонуються для завантаження на спеціалізованих майданчиках, таких як *App Store* та *Google Play*. Переглядати повідомлення, залишені користувачами необхідно, оскільки на їхній основі можна зробити висновок про якість програмного продукту, сайту, вебзастосунку. Виконувати цю операцію вручну затратно за ресурсами, тому і з’являється необхідність автоматичним чином здійснити сортування коментарів так, щоб з високою точністю відкидати тексти, що не несуть корисної інформації для розробників, і зберігати більшу частину актуального зворотного зв’язку, який допоможе вдосконалити якість програмних продуктів.

Об’єкт дослідження: бінарний автоматичний класифікатор тональності текстів.

Предмет дослідження: автоматичний класифікатор текстів із застосуванням методів машинного навчання.

Метою дослідження є створення автоматичного класифікатора із застосуванням методів машинного навчання на основі відгуків про програмні продукти.

Для досягнення мети були поставлені та вирішені такі **завдання**:

– проведено аналіз предметної області, обрана архітектура і платформа реалізації;

– розроблена функціональна модель системи;

– зібрані навчальні та тестові корпуси даних для класифікації відгуків про програмне забезпечення на негативні та позитивні;

– написано код класифікатора мовою програмування *Python* з

застосування методів машинного навчання.

Методи дослідження: обробка природної мови *(NLP)* – для розуміння текстових даних відгуків, включає токенізацію, лематизацію та визначення сутностей; класифікаційні моделі – для визначення категорій відгуків; навчання з учителем – використання навчальних даних з відомими мітками для тренування моделей та підвищення їх точності; аналіз емоцій – для визначення емоційного тону відгуків; аналіз частоти – визначення частоти вживання певних слів або фраз у відгуках для виявлення тенденцій та особливостей; вивчення взаємозв’язків між різними характеристиками відгуків та їх вплив на функціональність продукту; перехресна перевірка – розділення даних на тренувальний та тестовий набори для об’єктивної оцінки точності моделей; метрики ефективності – використання метрик для кількісної оцінки результатів класифікації.

1 ТЕХНІЧНЕ ЗАВДАННЯ

1.1 Постановка завдання

У наші дні з’являється величезна кількість платформ для створення і запуску програмних продуктів і додатків. За останні 5 років було завантажено близько сотень мільйонів додатків [1], і це тільки враховуючи мобільні платформи. Недавні дослідження свідчать про значний вплив відгуків користувачів на успіх програмного продукту [2]. Відгуки допомагають іншим користувачам зорієнтуватися під час вибору програмного продукту. Також, величезне число відгуків містять інформацію про помилки, запити наполіпшення [3, 4]. Ця інформація може виявитися корисною при покращенні програмного продукту, його супроводі. Кількість таких відгуків на популярний продукт може досягати сотень тисяч [5], серед яких є неінформативні та повторювані. Маючи таку кількість відгуків, завдання фільтрації і розбору корисної інформації стає важкий для розробників та аналітиків.

Завдання даної роботи складається в дослідженні можливості створення ефективної моделі обробки відгуків користувачів з автоматизованим зворотним зв’язком та створенням актуальних та унікальних завдань розробникам.

Мета дослідження полягає у зіставленні низки існуючих рішень по класифікації відгуків на заздалегідь задані категорії, а також дослідженні рішень по автоматичному визначенню подібних відгуків з наступним вибором найбільш ефективних методів класифікації.

При рішенні поставленого завдання не останню роль грає час роботи алгоритму. Для популярних програмних продуктів щодня з’являються кілька сотень нових відгуків [6]. Алгоритм повинен бути здатний обробити вхідний потік даних за розумний час.

Рішення даного завдання можна звести до рішення його складових частин, таких як:

– класифікація всіх відгуків від користувачів на категорії: позитивні відгуки, негативні відгуки(звіти про помилки, запити функціональності продукту, інші судження);

– визначення в кожній з категорій груп подібних відгуків.

Завдання класифікації відгуків сформулюється так: дана множина об'єктів – відгуків $X = \{x_1, x_2, \dots, x_n, \dots\}$ і множина можливих відповідей – категорії $Y = \{1, 2, \dots, k\}$. Існує невідома цільова залежність – відображення $y^*: X \longrightarrow Y$, значення якої відомі тільки на об'єктах кінцевої вибірки: (x_1, y_1), $(x_2, y_2), \dots, (x_m, y_m)$. Потрібно побудувати алгоритм $a: X \longrightarrow Y$, який наближав би цільову залежність, як на елементах вибірки, так і на всій множині *X*. Визначимо категорії:

– негативний відгук: звіти про помилки, описують проблеми з програмним продуктом, які мають бути виправлені в майбутньому, такі, наприклад, як падіння продукту, невірна поведінка, проблеми, пов'язані з продуктивністю;

– позитивний відгук: запити функціональностей продукту, запити на додавання функціональностей, наприклад, існуючих в інших продуктах, недолік змісту, контенту або ідеї, як зробити додаток краще;

– неінформативні відгуки, категорія комбінує неінформативні судження, такі як опис вже існуючих можливостей продукту, досвід використання в конкретних ситуаціях, захоплення, невдоволення. Такі відгуки відображають інформацію, яка може бути прочитана з реклами або документації до програмного продукту або є необґрунтованою критикою, похвалою. Дані категорії виділено в відповідно з [6].

Правильна класифікація визначається людиною-асесором, яка надає відгуку мітку *j* відповідно до вище описаного визначення категорій.

Завдання визначення у категоріях груп подібних відгуків можна зарахувати до задачі виявлення дублікатів [7, 8]. При рішенні даного завдання маємо певні категорії. Завдання можна віднести до класу завдань навчання з вчителем. Маємо опис (ознаки) множини об'єктів(відгуків), потрібно виявити

внутрішні взаємозв'язки, закономірності, існуючі між об'єктами (відгуками). Часто дане завдання вирішується графовими алгоритмами [7].

Поділ головного завдання на кілька етапів має очевидну мету поліпшення точності класифікації, а після, створення кластерів схожих по змісту відгуків.

Існує ряд найбільш близьких робіт [8]-[13], в яких досліджується можливість створення модуля для системи відстеження помилок щодо пошуку найбільш близьких даному відгуку звітів про помилки.

1.2 Задача класифікації відгуків

Класичним підходом до вирішення цього завдання є розгляд кожного документа (відгука) в вигляді набору слів (*bag of words*). Нехай n – кількість документів $(d_1, d_2, \dots, d_n)$, а m – кількість унікальних термінів $(w_1, w_2, \dots, w_m)$ (слів, або, більш у широкому сенсі, токенів). Тоді кожен документ d_i можемо представити в вигляді рядка $(x_1, x_2, \dots, x_j)$, де x_j – кількість входження терміну w_m у документ d_n . Як правило, більшість значень x_j дорівнюють 0 (матриця є розрідженою). Такий підхід дозволяє оперувати з векторним поданням документів. Але в такому поданні не враховується важливість окремих слів. Якщо слово вдеякому документі зустрічається часто, а в інших документах рідко, то воно представляє більший інтерес, так як з більшою мірою характеризує цей документ. Для обліку ваги слова самим популярним способом зважування для моделі термін-документ – це застосування *TF-IDF* формули, за якою вага слова розраховується по формулі:

$$w_{ik} = \frac{\left(1 + \log(N_{ik}) * \log\left(\frac{|D|}{N_k}\right)\right)}{\sqrt{\sum_{s \neq k}(\log(N_{is}) + 1)}}, \tag{1.1}$$

де N_{ik} – кількість появи *k*-ого терміну в *i*-ому документі,

N_k – кількість появи k-ого терміна у всіх документах,

$|D|$ – кількість документів у колекції.

Таким чином, рядок $(x_1, x_2, \ldots, x_m)$, характеризує документ, замінюється рядком ваг розрахованих по формулі *TF-IDF*.

Найбільш популярні методи машинного навчання для класифікації відгуків це наївний байєсовський класифікатор, метод опорних векторів, дерева рішень, метод максимальної ентропії.

У роботі [6] було розглянуто методи машинного навчання у застосуванні до класифікації відгуків. У роботі проводиться порівняння таких алгоритмів, як наївний байєсовський класифікатор, дерева рішень, і метод максимальною ентропії. Кожен відгук представлявся як набір слів. Показано, що наївний класифікатор Байєса є найбільш точним серед представлених.

Велику роль приділено передобробці відгуків. Було досліджено залежність якості класифікації від варіантів передобробки. Розглянуто такі варіанти попередньої обробки, як: видалення стоп-слів (слова, які зустрічаються у багатьох документах і не несуть смислового навантаження, такі, як, *a*, *has*, *once* і так далі), стеммінг (знаходження основи слова для заданого вихідного слова: *cats*, *catty* → *cat*; *stems*, *stemmer* → *stem* і так далі), лематизація (приведення словоформи до її словникової форми: *better* → *good*; walking → walk і так далі). Наприклад, видалення стоп-слів збільшує точність, тоді як застосування лематизації її зменшує при визначенні запитів функціональності. Загалом, потрібно акуратно відноситися до видалення стоп-слів і лематизації. При видаленні стоп-слів, не враховуємослова – *want*, *please*, *can*, що може негативно зашкодити класифікації. Облік метаданих може покращити класифікатор. Були розглянуті такі метадані: довжина відгуку, рейтинг, поставлений користувачем продукту (1-5), часи дієслів в відгуки. Додавання метаданих позитивно позначалося на класифікатор.

У роботі [9] було розглянуто метод опорних векторів (*SVM*) стосовно до задачі класифікації текстів. Документи також були перетворені на набір слів, але враховувалися слова, які були присутні в документах більше, чим 2 рази і не були

стоп-словами. Метод опорних векторів чутливий до розмірності матриці документів і облік великого числа слів сильно позначатиметься на продуктивності. Один із способів боротьби з великим числом ознак документа – використання методу головних компонент (*PCA*) для зменшення розмірності, що неминуче призводить до зменшення інформації, що аналізується. Застосований метод *TF-IDF* з наступною нормалізацією вектора. Експерименти порівнювали *SVM*класифікатори з різними ядрами (поліноміальне, радіальна базисна функція) з іншими класифікаторами на медичних документах про хвороби і схожих документах. Метод опорних векторів з радіальної базисною функцією показав найкращий результат.

Робота [10] присвячена використанню *n*-грам у задачах класифікації текстів (*n*-грами – послідовності з *n* елементів, символів). Даний підхід позбавляє створення систем обробки помилок у відгуках, документах.Підхід стійкий до помилок в словах, неправильно поставленим знакам пунктуації, ефективний на невеликих документах. Для того щоб класифікатор був чутливий до початку і кінця слів до слів на початку та наприкінці додається певна кількість «_» символів. Тобто, 2-грами слова *TEXT: _T, TE, EX, XT, T_*. Документи були очищені від цифр та розділових знаків, розбиті на токени. Токени утворили *n*-грами з $n = 1,2,3,4,5$. У роботі було показано, що використання *n*-грам успішно вирішує завдання класифікації текстів.

Зовсім інший підхід до класифікації текстів представлений в роботах [15]-[18]. Ідея полягає в перетворенні кожного слова в вектор однакової довжини.У роботах наводиться приклад того, як може бути враховано порядок слів. Кожен відгук розглядається не як набір слів та їх кількість. Кожне слово у відгуку має свій контекст. Наприклад, для i -го слова w_i у відгуку, в залежно від параметра k (розмір вікна), контекст можна представити як послідовність: $C = w_{i-k}, w_{i-k+1}, \dots, w_{i-1}, \dots, w_{i+1}, \dots, w_{i+k}$. Очевидно, для слова w_i можливо взяти набір таких контекстів в залежності від розміру вікна в деяких межах. Далі, використовується припущення про те, що слова, які мають схожі контексти мають схожі значення. На основі цього будується такий простір, що слова, які

мають схожі значення, видаються «близькими» ($|x_i - x_j| < \varepsilon$) векторами у цьому просторі. Для перетворення слів у вектора будується нейронна мережа з одним прихованим шаром, використовуючи *skip-gram* модель, яка представлена в [19] (обчислення векторів слів контексту за даним *i*-м словом і зіставлення з існуючим контекстом).Налаштування мережі здійснюється шляхом зворотного поширення помилки. Для уникнення обчислювальної складності використовується *Hierarchical softmax* [20], суть якого полягає у побудові бінарного дерева для представлення всіх слів в словнику. Дана модель дозволяє зменшити обчислювальну складність оновлення ваг нейронної мережі з *O(V)* до *O(log(V))*, де *V* – розмірність вектора слова.

Після навчання моделі по представленню слів векторами однакової розмірності потрібно вирішити проблему класифікації наборів, отриманих векторів (представленню окремого відгуку) на категорії. Для вирішення цієї проблеми часто використовується наступний підхід. Увесь простір векторів, що отримані, зі слів розбивається алгоритмом *k*-найближчих сусідів (*kNearest Neighbor – kNN*) на групи близьких векторів. Нехай розбили простір векторів на m груп. Тоді кожен відгук можна уявити вектором $(g_1, g_2, \dots, g_m)$, де g_i – кількість слів у відгуку, що належать *i*-групі. Після такого представлення відгуків у вигляді векторів однакової розмірності можемо застосувати до них один з відомих класифікаторів.

1.3 Задача пошуку груп подібних відгуків

Дане завдання відноситься до класу завдань навчання без вчителя. Відомі лише описи множини об’єктів. Класичним підходом для рішення цього завдання є подання об’єкта (відкликання) в вигляді набору характеристик (ознак) та облік відстаней між об’єктами при створенні кластерів (матриця відстаней). Тому часто дане завдання вирішується графовими алгоритмами кластеризації.

У роботі [7] робиться спроба створення автоматичного пошуку дублікатів

відгуків для системи відстеження помилок *Bugzilla*. У роботі проводиться передобробка вихідного набору документів та зведення завдання до вирішення завдання кластеризації на графах, вирішення якої, у свою чергу взято з [12], де графи застосовуються для дослідження груп у соціальних мережах. Дана система відстеження помилок використовується для таких великих проєктів, як *Mozilla*, *Eclipse*. Дана система відстеження помилок є відкритою базою даних звітів про помилки, в якій розмічені вручну дублікати помилок. Для перетворення звітів про помилки на вектор ознак була використана модель набір слів (*bag of words*). Використані такі інструменти передобробки документа, як стемінг, лематизація, видаленнястоп-слів. Варто відзначити, що було сформовано два набору текстів – заголовки та текст звіту враховувалися окремо, оскільки мають різний внесок у звіт. Кожен документ представлявся вектором $(w_1, w_2, \dots, w_m)$, де $w_i = 3 + 2 * \log_2$ (кількість *i*-го слова у документі). Дані коефіцієнти було знайдено на основі набору документів, важлива лише логарифмічна залежність. Далі, розраховується відстань між словами (схожість двох векторів ознак) за допомогою популярної формули знаходження косинуса кута між ними:

$$\text{схожість} = \cos\theta = \frac{v_1 * v_2}{|v_1| * |v_2|}. \quad (1.2)$$

Далі застосовується підхід до кластеризації графа, взятий із [12]. Припояві нового звіту про помилку, обчислюється вектор його заголовка і тіла і залежно від близькості до кластерів, приймається рішення про те, чи є він дублікатом чи ні. У роботі [16] робляться спроби покращити якості класифікатора, використовуючи деякі розширення і припущення, але ідея залишається тією ж самої.

У роботі [8] проблема вирішується іншим чином. Кожен звіт про помилки представляється набором наступних ознак:

– заголовок;
– тіло звіту (опис відгуку);

– назва продукту;

– компонент продукту (може бути вказано при складанні звіту про помилці);

– тип;

– пріоритет;

– версія продукту.

На основі даного набору ознак для будь-яких двох звітів про помилки обчислюються «порівняння» по наступним формулам:

$$\text{порівняння}_1 = BM25F(d_{1x}, d_{2x}), x = 1{,}2 \text{ (уніграми)}, \quad (1.3)$$

$$\text{порівняння}_2 = BM25F(d_{1x}, d_{2x}), x = 1{,}2 \text{ (біграми)}, \quad (1.4)$$

$$\text{порівняння}_3 = \begin{cases} 1 \text{ якщо } d_{13} = d_{23} \\ 0 \text{ інакше} \end{cases}, \quad (1.5)$$

$$\text{порівняння}_4 = \begin{cases} 1 \text{ якщо } d_{14} = d_{24} \\ 0 \text{ інакше} \end{cases}, \quad (1.6)$$

$$\text{порівняння}_5 = \begin{cases} 1 \text{ якщо } d_{15} = d_{25} \\ 0 \text{ інакше} \end{cases}, \quad (1.7)$$

$$\text{порівняння}_6 = \frac{1}{1 + |d_{16} - d_{26}|}, \quad (1.8)$$

$$\text{порівняння}_7 = \frac{1}{1 + |d_{17} - d_{27}|}. \quad (1.9)$$

Для даних формул d_{ij} – означає j ознаку i документа.

В перших двох формулах обчислюється близькість між заголовками і тілами звітів двох документів по формулі [15]:

$$BM25F(d, q) = \sum_{t \in d \cap q} IDF(t) * \frac{TF_D(d, t)}{k_1 + TF_D(d, t)}, \quad (1.10)$$

де

$$IDF(t) = \log \frac{N}{N_d}, \quad (1.11)$$

$$TF_d(d,t) = \sum_{f=1}^{K} \frac{w_f Occur(d[f],t)}{1 - b_f + \frac{b_f * l_f}{al_f}}, \quad (1.12)$$

де t – терм, присутній в обох документах d, q;

k_1 – параметр, що підбирається;

D – набір документів;

N – кількість всіх документів;

N_d – кількість документів, що містять терм t;

K – однакова для всіх документів кількість полів, на які розбивається документ (наприклад: заголовок та тіло звіту);

w_f – вага поля f;

$Occur(d[f],t)$ – кількість появи терму t в полі f;

l_f – розмір поля f в термах;

al_f – середній розмір поля f серед всіх документів;

b_f – налаштований параметр, який визначає масштабування довжини поля.

Загалом, ця формула є удосконаленою формулою *TF-IDF*.

У результаті, для будь-яких двох документів маємо 7 ознак, що обчислюються. Далі додаються додаткові ознаки – порядкові номери звітів про помилки і схожість по формулі косинусів (на основі вектора з 7 обчислюваних раніше ознак). Використовуються кілька відомих алгоритмів, таких, як логістична регресія, наївний байєс для навчання на основі вже проведеною класифікації (дублікат або ні) серед всіх документів з наступним порівнянням якості класифікації.

2 ПРОЄКТУВАННЯ ТА ТЕОРИТИЧНЕ ОБҐРУНТУВАННЯ СИСТЕМИ

2.1 Обробка природного мови

Мова – це неструктуровані дані, що використовуються людьми для спілкування між собою. Але комп’ютеру така мова незрозуміла,він, в свою чергу, сприймає тільки структуровані або напівструктуровані дані, що включають поля або розмітку, які дозволяють комп’ютерній програмі аналізувати їх. Неструктуровані дані, попри відсутність машиночитаної структури, не є випадковими. Навпаки, вони підкоряються лінгвістичним правилам, які роблять ці дані зрозумілими для людей [21].

Природні мови визначаються контекстом використання, а не її правилами які доводиться реконструювати для комп’ютерної обробки. Частіше всього люди самі визначають значення використовуваних слів, хоча і спільно з іншими учасниками розмови. Наприклад, словом *crab* в англійській мові може позначати морську тварину або вічно незадоволену, дратівливу людина або рух боком, але при цьому обидва – автор усного чи письмового тексту та його слухач чи читач – повинні погодитися із загальним розумінням цього слова під час діалогу. Саме тому мова зазвичай обмежується суспільством і територіальним місце розташування – спілкуватися і передавати сенс висловлювання зазвичай набагато простіше людям, які мають схожий життєвий досвід.

На відміну від формальних мов, у яких однакові поєднання знаків завжди мають однаковий сенс і які завжди є предметними, природні мови набагато універсальніші. Але з цього витікає проблема надмірності природних мов, необхідною для підтримки безлічі смислів. Інформаційна надмірність є властивістю будь-якого тексту, яке забезпечує можливість успішного сприйняття мови. Також варто зазначити, що з цього погляду будь-який текст природною мовою характеризується інформаційною надмірністю, в протилежному випадку він не може бути сприйнятий і зрозумілий адресатом, як,

наприклад, не буде зрозумілий людині текст програми, написаний на одній з мов програмування.

Однак надмірність представляє собою серйозну перешкоду,тому що не можемо вказати буквальний зміст для кожного символу, так як він за замовчуванням є неоднозначним. Структурна і лексична неоднозначність є основною характеристикою людської мови; дана особливість не тільки дає можливість генерувати нові ідеї, але також дозволяє спілкуватися людям з різним досвідом і культурою, хоча нерідко виникають випадкові непорозуміння при такій взаємодії.

У то же час якість, яка робить природню мову такимбагатим інструментом спілкування між людьми, ускладнює її аналіз з застосуванням детермінованих правил. Людина переважає комп'ютер в миттєвому розумінні мови, оскільки люди мають гнучкістю інтерпретації. Отже, в програмному середовищі потрібні настільки ж нечіткі й обчислювальні методи, які пристосовуються до контексту. І методи машинного навчання можуть в цьому сприяти – їх додавання до сфери обробки природної мови забезпечило певну гнучкість.

Насправді додатки, які використовують прийоми обробки природної мови для аналізу текстових та аудіоданих, вже давно стали невід'ємною частиною нашого життя, завдяки інноваціям в області машинного навчання і епосі надлишку даних, коли сам Інтернет представляє собою колосальний граф знань, який, серед всього іншого, містить велику гіпертекстову енциклопедію, спеціалізовані бази даних про фільми, музику, спортивні результати та про нескінченну кількість інших речей [22]. Ці продукти настільки поширені, що люди цілковито не помічають широкий спектр закулісних інструментів: від спам-фільтрів, що збирають інформацію про наш поштовий трафік, до пошукових систем, які видають саме те, що ми шукали, і віртуальних помічників, завжди готових відповісти на будь-яке питання.

Як простий приклад впровадження аналізу природної мови можна привести підтримку «тегів рекомендацій», реалізовану в інформаційних продуктах таких компаній, як *Amazon*, *Netflix*, *YouTube* та інших. У тегах

зберігається метаінформація про фрагменти контенту, вони визначають властивості описуваного ними змісту та можуть використовуватися для угруповання подібних елементів. Дана метаінформація важлива для рекомендацій і пошуку, вона грає велику роль в визначенні того, зацікавить чи контент конкретного користувача.

Існує безліч інших цікавих прикладів, мабуть, найбільш звичним для нас є функція в *iMessage,* технологія обміну миттєвими повідомленнями від компанії *Apple*, яка намагається передбачити, що користувач надрукує далі, спираючись на нещодавно введений текст, а функція автокорекції виправляє орфографічні помилки. Також є ряд голосових віртуальних помічників: *Alexa* від *Amazon, Siri* від *Apple,* – здатних аналізувати мову і давати більш або менш осмислені відповіді.

Хоча це напрямок активно розвивається в останнє десятиліття, все ж багато проблеми ще залишилися невирішеними і далеко не всі можливі перспективи реалізовано. Основна складність в науці про дані (*data science*), яку відзначають фахівці в цій галузі, пов’язана з тим, що процес дослідження даних не завжди сумісний з практикою розробки програмного забезпечення. Дані можуть бути непередбачуваними, тому завжди складно передбачити, чи буде результат позитивним. Хілларі Мейсон (*Hilary Mason*) одного разу сказала про розробку додатків для обробки даних, «наука про дані не завжди відрізняється гнучкістю» [23].

Більше того, більшість публікацій про машинне навчання і обробку природньої мови носять дослідницький характер, щоускладнює процес розробки реальних програм. Наприклад, незважаючи на наявність великої кількості відмінних інструментів для машинного навчання на текстових даних, наявних у відкритому доступі джерела інформації – документації, посібники та статті в Мережі – як правило, спираються на штучно підібрані масиви даних і дослідницькі інструменти. Досить рідко можна знайти пояснення на конкретних прикладах, зокрема того, як створити корпус, достатньо великий для підтримки програми, як керувати його структурою і розміром або як трансформувати

вихідні документи в дані, придатні до використання. Але саме ці аспекти є ключовою частиною практики створення масштабованих додатків по обробці даних, заснованих на аналізі природного мови.

Метою прикладного аналізу тексту є не що інше, як створення таких програм – вони приймають на вході текстові дані, розбирають їх на складові, виконують обчислення на основі цих частин і знову збирають їх, повертаючи осмислений результат. Таким чином, додатки з обробки даних отримують цінні відомості з вихідних текстових матеріалів і, в свою чергу, генерують нові дані [24].

На даний момент базова методологія, яка використовується при створенні додатків, заснованих на аналізі природної мови, – це машинне навчання з учителем чи без. З його допомогою можна розбити на значущі кластери тексти по схожості або класифікувати тексти з застосуванням конкретних міток.

Стандартний конвеєр (*pipeline*) такої програми реалізує ітеративний процес, що складається з двох етапів: складання та розгортання – та є відображенням конвеєра машинного навчання [25]. На першому етапі збірки вихідні дані перетворюються на форму, придатну для передачі в модель машинного навчання і експериментів з ними на заключному етапі.

На етапі розгортання відбувається остаточний вибір моделей, які надалі використовуватимуться для оцінок та прогнозів, безпосередньо, що стосуються користувача. На рис. 2.1 показаний шлях даних в додатку обробки природної мови.

У наведеній діаграмі можна виділити для кожного з двох етапів чотири стадії: взаємодія, дані, накопичення і обчислення. Розглянемо їх детальніше.

У процесі взаємодії необхідна допоміжна програма для введення даних, а користувачеві – прикладний інтерфейс.

Під етапом даних розуміються внутрішні компоненти, які діють як проміжна ланка перед переходом до стадії накопичення. Дані перетворюються, щоб вони були доступні для використання або за допомогою моделі, або за допомогою будь-якого іншого формату.

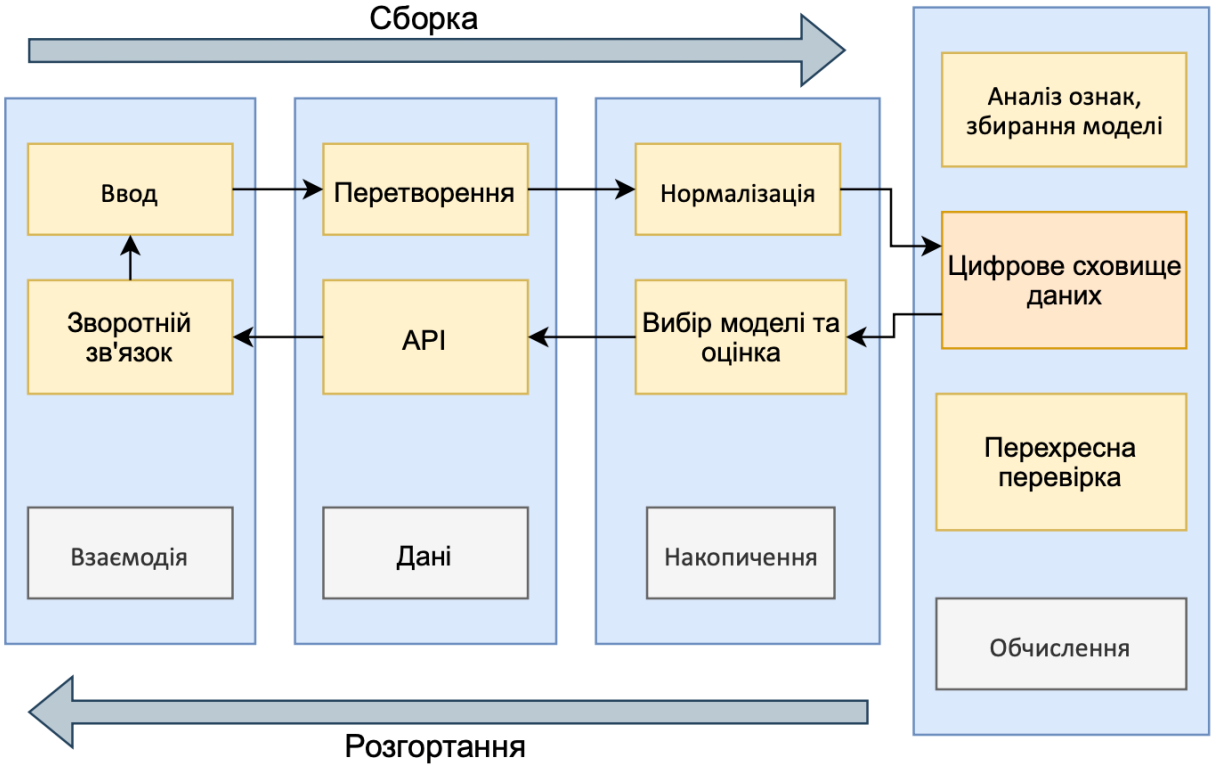


Рисунок 2.1 – Конвеєр програми обробки природної мови

Накопичення – функція зберігання, яку зазвичай виконує база даних. На цій стадії відбувається збереження великих обсягів даних і параметрів, необхідних для роботи.

Зрештою, стадія обчислень або обробки відповідальна за навчання моделей і управління сховищем обчислювальних даних.

Варто зазначити, що етап сборки додатків вимагає особливої уваги – тим більше в випадку аналізу тексту. Реалізуючи програми, обробки природної мови, створюємо додаткові лексичні ресурси (такі як словники, перекладачі, регулярні вирази і т.д.), від яких залежить робота продукту.

На рис. 2.2 показано більш розгорнуте уявлення стадії сборки додатків машинного навчання, заснованих на аналізі природної мови. Процес переходу від вихідних даних до розгорнутої моделі, відповідно, складається з послідовних перетворень даних. Спочатку отримані матеріали трансформуються в вхідний корпус, накопичуються та зберігаються у сховищі даних. Далі вхідні дані групуються, проходять очищення та нормалізацію, потім перетворюються на вектори для подальшої обробки. У кінцевому перетворенні одна чи кілька

моделей навчаються на векторизованому корпусі. У результаті всіх вищеописаних операцій створюється узагальнене подання оригінальних даних, яке потім використовується готовим додатком.

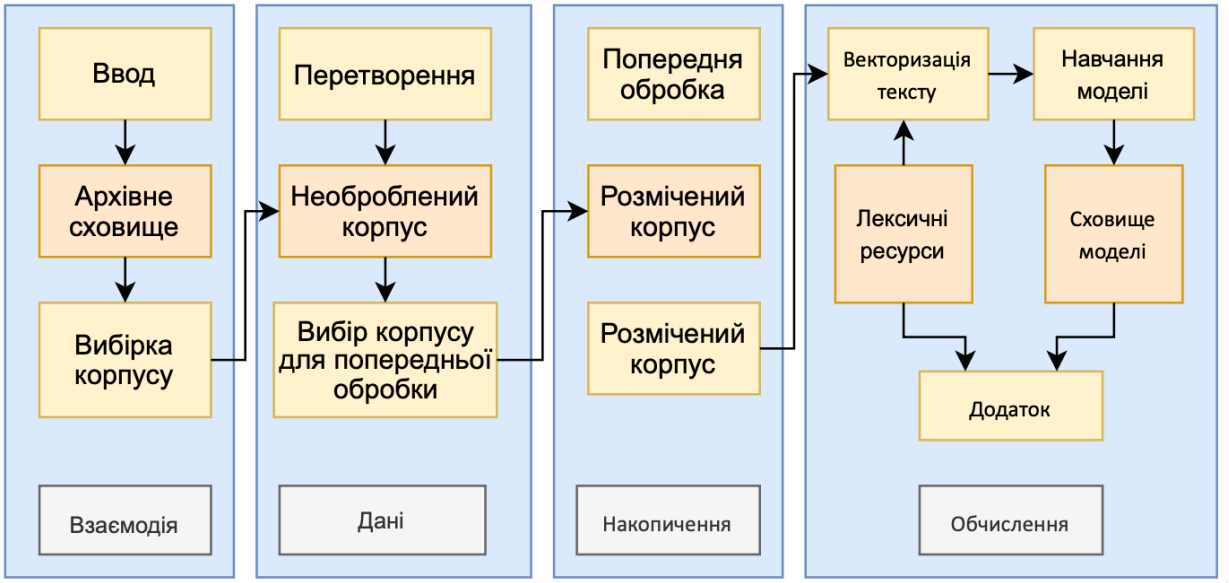


Рисунок 2.2 – Етап сборки в додатках обробки природної мови

Розгортання, крім вибору і використання моделі, не сильно відрізняється від прямолінійної розробки програмного забезпечення. Часто при створенні таких програми розробляється API (*Application Programming Interface* – програмний інтерфейс додатку), який використовується іншими програмами або користувальницькими інтерфейсами.

У кінці користувачі дають зворотній зв’язок на вихідні дані моделей, а їхня відповідь, у свою чергу, знову подається на вхід і використовується для адаптації моделей.

Наприкінці можна сказати, що неструктуровані дані важкопідігнати під конкретну модель даних, тому що їх вміст залежить від контексту або має змінний характер. Тексти на природній мові становлять особливий різновид такого типу даних. Обробка такої інформації досить складна, тому що вона потребує знання як лінгвістики, так і спеціальних методів *data science*. Є суттєвий прогрес у напрямку обробки даних на природній мові, спільнота

досягла успіху в сфері розпізнавання сутностей і тематичних областей, узагальнення, генерації тексту та аналізі тональності, але моделі, адаптовані для однієї предметної області, погано узагальнюються для інших. Навіть самі сучасні методи не зможуть розшифрувати сенс довільного фрагмента тексту. І цей факт не дивує, оскільки у людей також виникають проблеми зі сприйняттям природної мови. Він неоднозначний за своєю природою. Якщо дві людини слухають одну і ту ж розмову, то чи винесуть вони однаковий зміст з неї? Крім того, сенс окремих слів може змінюватись в залежності від настрою людини, що говорить. Отже, сама концепція виглядає спірно [26].

2.2 Машинне навчання як метод обробки даних

З давніх-давен людина виконувала різного роду роботу у великій кількості, але не завжди є можливість, час і ресурси, щоб виконувати певні завдання вручну. Тому не можна сказати, що машинне навчання (МН), або *machine learning*, є абсолютно новою технологією, навпаки, закладка фундаменту в цій галузі почалася в XVII столітті з появою першого механічного калькулятора (1642) і розробкою Готфрідом Лейбницем двійкової системи (1679).

Вже в XX століття поступово почала поширюватися теорія когнітивного навчання, розроблена швейцарським психологом Жаном Піаже. Дана теорія будується на основі припущення, що людина має здатність до навчання і може структурувати та зберігати накопичену інформацію. Як бачимо по швидкості розвитку сучасних технологій, когнітивне навчання було адаптовано і для комп’ютерів.

Одним із піонерів у цій галузі став Артур Самуель, який створив програму *Checkers-playing* (1956) – інноваційна програма в світі, вона вміла, як випливає з назви, грати в шашки. Хоч правила ігри дещо прості, але мають певну стратегію, тому було вигідно відпрацювати на ній алгоритм, відповідних проблем штучного інтелекту (ШІ). Комп’ютер навчався на гідах по грі, в яких були прописані партії

з гарними та поганими ходами. У1959 Самуель дав таке визначення машинному навчанню: «Це процес, в результаті якого машина (комп'ютер) здатна показувати поведінку, яку в її не було явно закладено (запрограмовано)» [8].

Також варто відзначити роботу Френка Розенблатта, наприкінці 1950-х Корнельському університеті він побудує систему *Mark I Perceptron*, яку умовно можна визнати першим нейрокомп'ютером. Ця машина заснована на принципі персептрону, комп'ютерної моделі сприйняття інформації мозком. Перцептрон став однією з перших моделей нейромереж.

Відштовхуючись від всього вищесказаного, можна сформулювати визначення: машинне навчання – клас методів штучного інтелекту (*artificial intelligence*), орієнтований на рішення прикладних задач. Характерною рисою МН є навчання в процесі рішення безлічі подібних завдань, що подібно до людської поведінки в процесіоволодіння будь-яким ремеслом шляхом спроб і помилок. Для побудови такихметодів використовуються засоби математичної статистики, чисельних методів, теорії ймовірностей, теорії графів та інші різні техніки роботи з даними у цифровій формі.

Ціль МН полягає в передбаченні результатів по вхідним даними на підставі якоїсь математичної моделі. Щоб реалізувати концепцію МН, експерти розробляють алгоритми спільного призначення, які можуть застосовуватися до великого класу завдань навчання. Якщо потрібно вирішити певну проблему, достатньо передати алгоритму більше конкретні дані, тут мова йде про навчання на прикладах . У більшості випадків комп'ютер використовує дані як джерело інформації, порівнює свій результат з бажаним, а потім вноситьвиправлення. Чим більше даних чи «досвіду» накопичує комп'ютер, тим краще він справляється зі своєю роботою – як і людина [26]. Але потрібно враховувати те, що дані мають бути надані не лише у великому кількості, але повинні бути також різноманітні, інакше машині буде важко знайти закономірності, а це може привести до неточності результату.

У МН є три основні складники: дані (*data*), ознаки (*features*) та алгоритм навчання (*learning algorithm*). Як було сказано вище, матеріал, на якому

проходитиме навчання, повинен складатися хоча б з кількох тисяч. Їх можна збирати вручну або за допомогою парсера, який здійснює автоматичний збір інформації з заданого ресурсу. Іноді можна проводити дослідження вже на готових базах даних або, як їх часто називають, датасетами (*data set*), але вони є великою рідкістю.

По-друге, складним завданням є вибір ознак, характеристик,на які буде спиратися машина в процесі навчання. Вони можуть бути зовсім різними залежно від мети та галузі дослідження. Дуже важливо виділити потрібні закономірності, щоб вони були властиві не лише навчальній вибірці, а й наступним прецедентам. Крім того, в деяких методах МН не використовуються заздалегідь відомі класи ознак.

Коли ознаки потрапляють на входи МН, система намагається помітити закономірності між ознаками, а на виході генерується результат цих процесів. Результати роботи прийнято називати міткою (*label*), оскільки система позначає входи певним прогнозом, до якої категорію потрапить вихід після класифікації. А саму систему, за якою елементу приписується відповіді, прийнято називати цільовою функцією [27].

Слід також пам’ятати, що для рішення одного завдання частіше всього знайдеться не один метод. Від обраного алгоритму рішення залежатиме точність, швидкість роботи та розмір готової моделі. Але все одно не можна забувати, що якість все-таки в більшою мірою залежить від наданих програмі даних. Існує велике кількість різних алгоритмів, які застосовуються як окремо, так і в сукупності.

Таким чином, МН можна визначити, як послідовність дій:

– подання класифікованих елементів даних на формальній мові, які машина може інтерпретувати;

– оцінка – функція, яка дозволяє визначити корисність тих чи інших класифікаторів;

– оптимізація – пошук найкращих класифікаторів.

Після закінчення роботи система має видати в ідеалі правильні відповіді, а

людина при цьому витратить менше робочої сили на обробку великої кількості даних. У такому випадку можна буде рахувати, що машиною була підібрана коректна цільова функція.

Але вибір оптимальної моделі – це складний та ітеративний процес, що складається з повторення циклів навчання, конструювання ознак, вибору моделі і налаштування гіперпараметрів (так називають параметри алгоритмів МН, значення яких встановлюються перед запуском процесу навчання). Після кожної ітерації виконується оцінка результатівз метою отримати найкращу комбінацію моделі, параметрів та ознак. Цей процес зображений на рис. 2.3 можна називати трійкою вибору моделі (*Model Selection Triple – MST*). Вибір моделі є ітеративною і дослідницької операцією, оскільки простір *MST* зазвичай нескінченний, і аналітики, як правило, не можуть апріорі знати, який *MST* дасть задовільну точність. Отже, мета цього процесу полягає в тому, щоб уявити ітерацію як основу науки машинного навчання, яка повинна полегшуватися, а не обмежуватись [28].

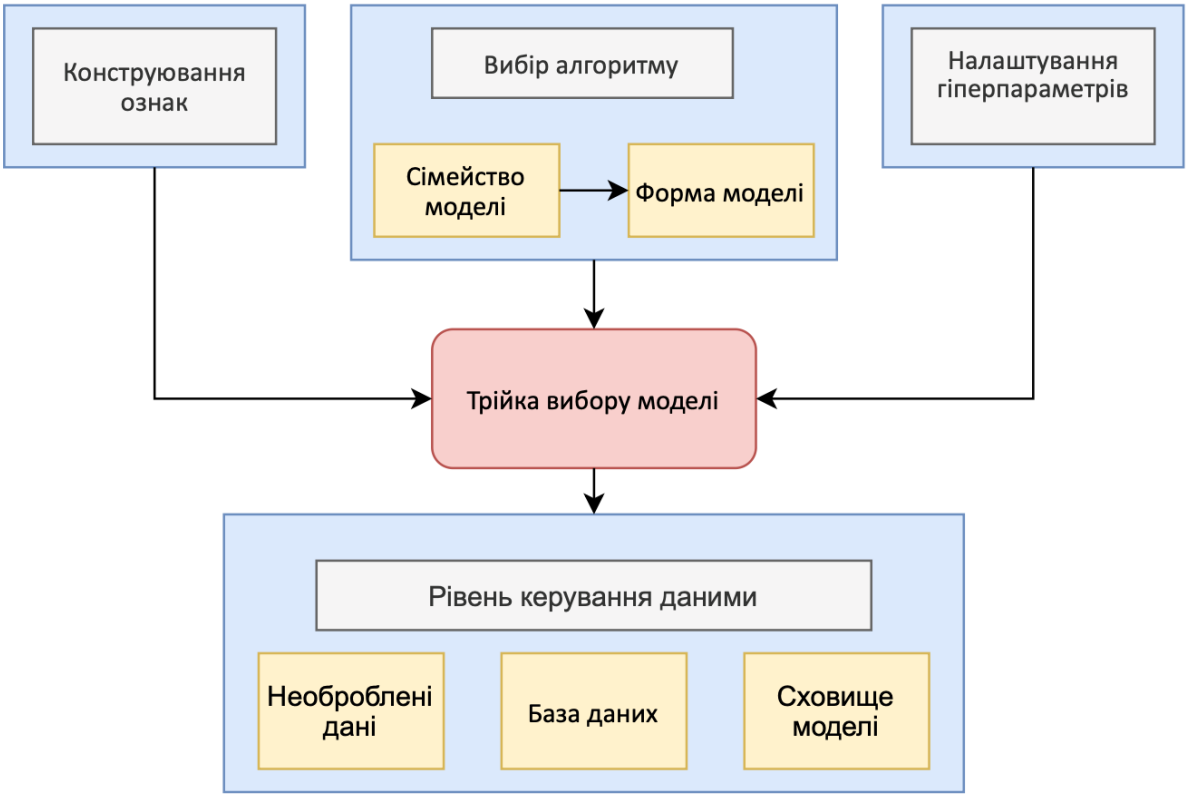


Рисунок 2.3 – Трійка вибору моделі

Також Вікхем (*Wickham*) із співавторами у своїй статті, що вийшла в 2015 році, усунули неоднозначність перевантаженого терміну «модель», описавши три базових випадки його використання в машинному навчанні: сімейство моделі, форма моделі та навчена модель. Широке поняття сімейства моделей описує відносини між змінними та поставленою метою (наприклад, лінійна модель передбачає одну відповідь за формулою лінійного рівняння і припускає, що випадкові помилки мають нормальний розподіл). Форма моделі представляє собою конкретний екземпляр *MST*: набір ознак, обраний алгоритм і конкретні гіперпараметри. Зрештою, навченою моделлю вважається та модель, яка навчилася на певному наборі даних, і здатна робити передбачення на нових, раніше не відомих даних [29].

Якщо повернутися до питання про додатки, обробки природної мови, то основний завданням аналізу тексту є інтерпретація того, що відбувається в кожному з цих перетворень. При цьому з кожним наступним перетворенням текст стає все менш значущим для нас, тому що все менше нагадує людську мову.

2.3 Класифікація в текстовому аналізі

Важливість МН зростає не тільки у дослідженнях, які мають відношення до комп'ютерних наук; його роль у повсякденному житті стає ще більше. МН дозволяє користуватися надійними фільтрами поштового спаму, випробуваними пошуковими механізмами, зручними додатками з розпізнавання тексту та мови та, можливо, що в найближчому майбутньому – безпечними безпілотними автомобілями.

Існує три основні типи МН: навчання з учителем (*supervised learning*), навчання без вчителя (*unsupervised learning*), навчання з підкріпленням (*reinforcement learning*). У даній кваліфікаційній роботі буде докладно описано навчання з вчителем.

Навчання з учителем і без поєднує те, що це два напрями класичного

навчання, перші алгоритми якого мають статистичну природи. Їхнє завдання полягало в пошуку закономірності в цифрах, оцінці близькості точок у просторі та обчислення напрямків векторів. Хоч дані алгоритми зароджувалися в 1950-х роках, їх актуальність не застаріває. Сьогодні на класичних алгоритмах тримається безліч звичних нам речей, наприклад, коли на сайті зустрічається блок «Рекомендовані статті» або коли банківська карта може бути заблоковано через підозрілі операції. Крім того, класичне навчання є більш вигідним, так як не вимагає надзвичайно великих наборів даних, для деяких завдань великі датасети недоступні або їх придбання виявиться дорогим і трудомістким.

Тепер ґрунтовно розглянемо один із типів класичного МН. Головна мета навчання з вчителем – навчити модель на розміченій навчальній вибірці, щоб надалі модель змогла надавати категорії новим екземплярам, спираючись на шаблони, виявлені в процесі навчання. Тут поняття «з вчителем» відноситься до набору зразків, де бажані вхідні мітки вже відомі.

Як приклад розглянемо фільтрацію поштового спаму, можна навчити модель з використанням алгоритму машинного навчання з вчителем на вибірці помічених поштових повідомлень, вони мають бути марковані коректно – спам чи не спам. Завдання навчання з учителем з мітками дискретних класів, такими, як в нашому прикладі фільтрації поштового спаму, також називається завданням класифікації.

Класифікація є основною формою аналізу тексту, її можна зустріти у розділенні вебсторінок та сайтів за тематичними каталогами, в аналізі тональності, в визначенні мови тексту і в показі найбільш релевантної реклами.

Класифікація – це підкатегорія навчання з учителем, де метою є прогнозування категоріальних міток класів, до яких належать нові зразки на основі минулих спостережень. Такі мітки класів представляють собою дискретні невпорядковані значення, які можуть сприйматися як приналежність до груп зразка [30]. Раніше згаданий приклад виявлення поштового спаму демонстрував типове завдання двійкової класифікації, коли алгоритм МН вивчав набір правил,

щоб проводити відмінність між двома можливими класами: спам або не спам. Однак набір міток класів зовсім необов'язково має двійкову природу, класифікація такого типу буде називатися багатокласовою, характерним прикладом такий завдання вважається розпізнавання рукописних символів. У такому разі навчальний набір даних буде отримувати безліч рукописних прикладів для кожної літери алфавіту. Тепер, якщо користувач за допомогою пристрою введення надасть новий рукописний символ, тоді прогнозуюча модель буде в стані з певною точністю передбачити коректну літеру алфавіту для введеного рукописного символу. Але така система МН не буде здатна правильно розпізнати, наприклад, цифри,якщо їх від самого початку не було в навчальній вибірці.

Найскладніша частина прикладного аналізу тексту – керування та збір предметно-орієнтованого корпуса для побудови моделей. Друга поскладності частина – вироблення аналітичного рішення конкретного прикладного завдання [21].

В даний час розроблено велике число моделей і механізмів класифікації, які математично різноманітніші, ніж лінійні моделі, використовувані здебільшого для завдань регресії, які спрямовані на прогнозування безперервних результатів, тобто замість категорій передбачається число. Програми з аналізу текстових даних мають вибір з широкого кола сімейств моделей – від методів на основі екземплярів, які використовують вимір відстані між точками в просторі і байєсовські ймовірності, до лінійних і нелінійних апроксимацій та нейронних мереж. Однак в основі всіх сімейств моделей класифікації лежить той самий базовий процес: можна одночасно тестувати кілька моделей для рішення конкретного завдання і потім порівнювати їх шляхом прийомів перехресної перевірки для вибору найбільш ефективних.

Сам процес класифікації ділиться на два наступних етапи, представлених на рис. 2.4, – навчання та експлуатація. На першому етапі корпус текстових документів перетворюється в вектори ознак. Далі ознаки документів разом з їх мітками, розпізнавання яких хочемо навчити модель, передаються в алгоритм

класифікації, який визначає свій внутрішній стан і виявлені шаблони. Після навчання можна векторизувати новий, раніше невідомий документ і передати результат алгоритму, який поверне передбачену мітку категорії документа.

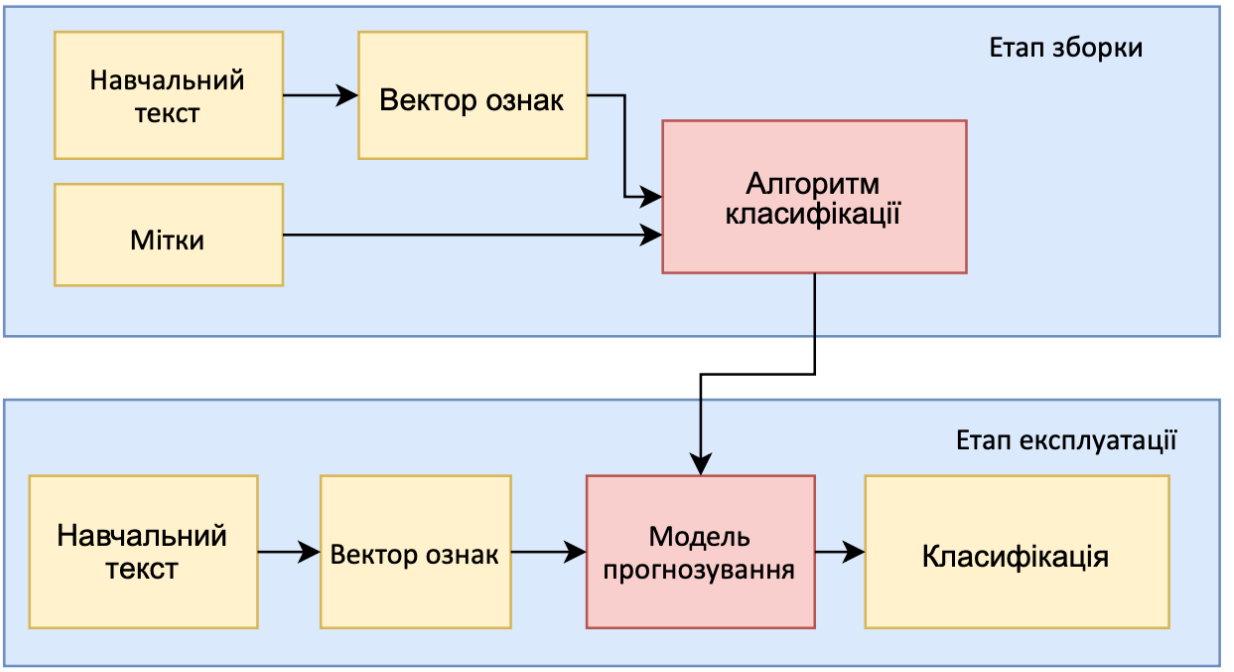


Рисунок 2.4 – Процес класифікації

Наприклад може стояти завдання двійкової класифікації, яка розпізнає тільки два класи, і обидва вважаються протилежними один одному як, наприклад, так і ні або, в нашому випадку,корисний чи не корисний зворотній зв'язок. Якщо подивитися на питання з точки зору ймовірності, то двійковий класифікатор з класами A і B припускає, що $P(B) = 1 - P(A)$.

Однак насправді такий однозначний зв'язок, поділ чітко на два класи, зустрічається досить рідко, зокрема, при аналізі емоційного забарвлення, якщо документ не позитивний, значить чи це, що він негативний? Якісь документи можуть носити і нейтральне забарвлення, як це часто буває насправді, додавання третього класу в класифікатор може в значній мірі покращити розпізнавання позитивних та негативних текстів. В результаті це завдання стає завданням багатокласової класифікації.

Як говорилося в попередньому розділі, часто при обробці текстових даних

створюються додаткові лексичні ресурси. Для покращення бінарної класифікації, виконуваної алгоритмом машинного навчання можна використовувати методи з раніше згаданого аналізу тональності, популярна дисципліна із широкої галузі обробки природної мови. Даний підхід не має на увазі вилучення фактів з інформації, він займається тільки ступенем емоційного забарвлення, аналіз тональності відстежує саме емоції авторів в відношенні до певної теми.

Як і в інших завданнях прикладної лінгвістики, основні підходи до автоматичного визначення тональності тексту можна розділити на дві великі групи. Алгоритми першої групи засновані на правилах (*rule-based*), а алгоритми другої групи використовують методи МН.

Існує кілька способів отримати думку з великої колекції текстів без використання МН, нижче перерахуємо основні з них та коротко обговоримо принцип їх роботи.

Побудова правил. Алгоритми, створювані на основі правил, дозволяють враховувати семантичні, структурні особливості різних слів та самої мови. Заздалегідь створюються шаблони, за цим шаблоном із тексту витягуються n-компонентні ланцюжки (*n*-грами), їх тональність визначається на основі правил, і з урахуванням словників; наприклад, правила можуть будуватися по моделі «якщо …, то…». Але реалізація такого методу стикається знизкою проблем:

– вузька сфера застосування набору правил у зв’язку з тим, що формат написання різних повідомлень в мережі Інтернет достатньо сильно відрізняється від прийнятих норм природної мови літературної форми;

– потрібно формувати певний корпус різних лінгвістичних правил, який зобов’язаний враховувати велику частину різних конструктивних мовних особливостей;

– алгоритм не зможе бути перенесений іншою мовою через зв’язок з унікальною структурою певної мови [39].

Словники оцінювання лексики. У словниках оцінювання лексики зберігаються слова і словосполучення, кожному з яких привласнений рівень

емоційної оцінки. Залежно від словника використовуються різні шкали оцінок: буде приписана лише одна, або дві тональні оцінки. Також існують набори слів, яким приписуються різні емоції (такі як «страх», «здивування», «віра», гнів, «радість» , тощо) і тональні оцінки (позитивна або негативна).

2.4 Огляд класифікаторів на основі машинного навчання

Вибір найбільш відповідного алгоритму класифікації для вирішення певного завдання вимагає практики і досвіду. Кожен класифікатор має специфічні особливості і робить деякі припущення. Насправді слід порівнювати показники ефективності для кількох різних алгоритмів машинного навчання для досягнення найкращого результату. На ефективність моделі, тобто на її здатність робити коректні прогнози, сильно впливають дані, що лежать в її основі, які передаються алгоритму класифікації на навчання: в яких може бути багато шуму, також досить часто класи екземплярів виявляються нелінійно роздільними, що автоматично ускладнює алгоритм.

У даному розділі буде розглянуто чотири відомих класифікатора: логістична регресія (*Logistic Regression*), метод k-найближчих сусідів (*k-nearest neighbors algorithm – KNN*), випадковий ліс (*Random Forest*) та *XGBoost*.

Логістична регресія. Лінійні моделі представляють собою клас моделей, які широко використовуються на практиці, вони роблять прогноз, використовуючи лінійну функцію. Дані моделі застосовуються в завданнях регресії і класифікації.

Основна ідея лінійного класифікатора полягає в тому, що ознаковий простір може бути розділений гіперплощиною на два напівпростори, в кожному з яких прогнозується одне з двох значень цільового класу. Якщо лінійний поділ можливо виконати без помилок, то навчальна вибірка вважається лінійно роздільною.

Логістична регресія (*Logistic regression*) дуже широко використовується в

виробничому середовищі для двійкової класифікації.Незважаючи на свою назву, вона є саме алгоритмом класифікації, а не регресії.

В основі логістичної регресії лежить імовірнісна модель. Задля пояснення цієї ідеї введемо поняття коефіцієнта переваги, цей показник визначає перевагу на користь певної події. Даний коефіцієнт може бути записаний наступним чином:

$$\frac{p}{(1-p)}, \tag{2.1}$$

де p – це ймовірність позитивного події, тобто. подія є реальним.

Наприклад, якщо хочемо спрогнозувати чи є у пацієнта якесь захворювання або, як в випадку цього дослідження, дізнатися відноситься чи коментар до класу позитивний відгук про застосунок, то ми маємо на увазі позитивну подію, якій можна присвоїти мітку класу $y = 1$.

Далі визначається логіт-функція (*logit)*, яка рівна натуральному логарифму коефіцієнта переваги:

$$logit(p) = log\frac{p}{(1-p)}. \tag{2.2}$$

Вхідними значеннями логіт-функції є числа в діапазоні від 0 до 1, які потім трансформуються в значення по всьому діапазону дійсних чисел. Їх можна використовувати для вираження лінійної залежності між значеннями ознак і логарифмом:

$$logit\big(p(y=1|x)\big) = w_0x_0 + w_1x_1 + \ldots + w_nx_n = \sum_{i=0}^{n} w_ix_i = w^Tx, \tag{2.3}$$

де $p(y=1|x)$ – умовна ймовірність того, що конкретний екземпляр

належить класу 1 за певними ознаками x, а w_0 означає елемент зміщення при додатковому вхідному значенні x_0 рівному 1.

Але насправді нас цікавить передбачення ймовірності того, що заданий зразок належить до зазначеного класу – це, у свою чергу, зворотній запис логіт-функції, яка називається логістичною сигмоїдальною функцією або просто сигмоїдальною, оскільки її графік має характерну форму літери *S*, як показано на рис. 2.5. Формула записується таким чином:

$$sig(t) = \frac{1}{1 + e^{-t}}. \tag{2.4}$$

В даному випадку t є лінійною комбінацією ваг та ознак екземплярів:

$$t = w_0 x_0 + w_1 x_1 + \cdots + w_n x_n = \sum_{i=0}^{n} w_i x_i = w^T x. \tag{2.5}$$

На представленому графіку видно, що функція $sig(t)$ прагне до 1, коли t прагне до нескінченності, тому що e^{-t} стає занадто малим для великих значень t. Аналогічним чином $sig(t)$ наближається до 0 в результаті знаменника, що збільшується. Підсумовуючи вищесказане, сигмоїдальній функції на вхід подаються речовічисла і перетворюються в значення в рамках діапазону [0, 1] з точкою перетину $sig(t) = 0.5$.

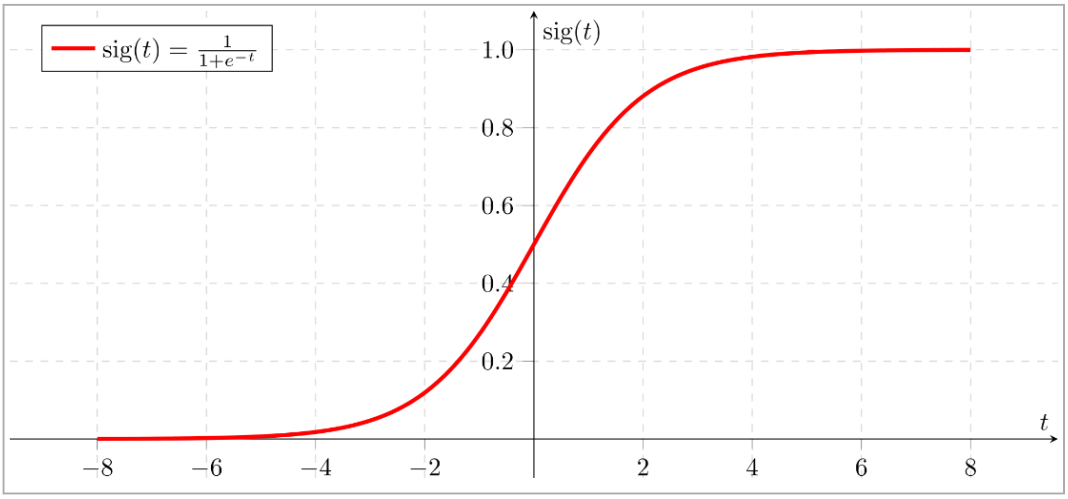


Рисунок 2.5 – Графік сигмоїдальної функції

Вихід даної функції трактується як ймовірність віднесення певного зразка до позитивної події, тобто до класу 1, при заданих ознаках і підібраних вагах w ($sig(t) = P(y = 1|x; w)$).

Значить, якщо для одного конкретного коментаря отримаємо $sig(t) = 0.7$, то можливість того, що цей текст є позитивним відгуком, складає 70%. Відповідно, шанс того, що даний коментар належить класу 0, може бути обчислений так: $sig(t) = P(y = 0|x; w) = 1 - P(y = 1|x: w) = 0.3$ (або 30%). Передбачену ймовірність після можна трансформувати в бінарний результат з використанням порогової функції:

$$\hat{y} = \begin{cases} 1, \text{якщо } sig(t) \geq 0.5 \\ 0 \text{ в іншому випадку} \end{cases}. \quad (2.6)$$

Прогнозування не просто чіткої відповіді чи відноситься екземпляр до класу 1 або 0, а саме *ймовірності* приналежності до певного класу є в багатьох завданнях дуже важливою бізнес-вимогою. Наприклад, в задачі оцінки кредитоспроможності людини (*кредитний скоринг*), де частіше всього застосовується логістична регресія, частонеобхідно спрогнозувати ймовірність неповернення кредиту. Клієнтів, які звернулися за отриманням кредиту, сортують по спаданню взалежно від передбаченої ймовірності, і одержують рейтинг надійності клієнтів. Отже, така властивість алгоритму може бути корисною при роботі зі складними даними, де потрібно підвищити або знизити поріг для прийняття рішення.

Однак при навчанні моделі логістичної регресії фокусуємось не на помилках, а більше на максимізації ймовірності відповідності передбачених значень оцінці максимальної правдоподібності (*Maximum-Likelihood Estimation – MLE*) [31]. Коротко розглянемо, яким чином налаштовуються параметри моделі, ваги w.

Спочатку визначаємо функцію витрат – міру того, наскільки помилкова модель з точки зору її здібності оцінювати взаємозв'язок між x і y. Зазвичай вона

виражається як різниця або відстань між прогнозованим значенням і фактичним значенням [32]. Потім мінімізуємо її з метою знайти оптимальні ваги для класифікаційної моделі. Перш ніж вивести цю функцію для логістичної регресії, визначимо правдоподібність L, яку бажано підняти до максимального рівня при побудові моделі з умовою, що в наборі даних індивідуальні зразки один від одного незалежні. Формула правдоподібності виглядає так:

$$L(w) = P(y|x; w) = \\ = \prod_{i=1}^{n} P\left(y^i \middle| x^i; w\right) = \prod_{i=1}^{n} (sig\left(t^i\right))^{y^i} (1 - sig\left(t^i\right))^{1-y^i}. \quad (2.7)$$

Набагато легше на практиці довести до максимуму натуральний логарифм цього рівняння. Такий варіант називається логарифмічною функцією правдоподібності:

$$logL(w) = \sum_{i=1}^{n} y^i \log\left(sig\left(t^i\right)\right) + \left(1 - y^i\right)\log\left(1 - sig(t^i\right). \quad (2.8)$$

Використати логарифмічну функцію краще, так як вона знижуєчислову втрату значущості, якщо величини правдоподібності надто малі. Також стоїть відзначити, що базова реалізації алгоритму в бібліотеці *Scikit-learn* оптимізує роботу алгоритму за допомогою регуляризації. Вона допомагає знайти компроміс між перенавчанням і недонавчанням (див. рис. 2.6). Модель вважається перенавченою, коли вона дає якісні результати на навчальних даних, але погано узагальнюється на тестових даних, які не траплялися раніше. У протилежній ситуації модель буде занадто слабка, щоб виявити структуру в навчальному наборі даних, та тому буде малоефективною на невідомих зразках даних.

Метод регуляризації корисний тим, що обробляє колінеарність, сильний

взаємозв'язок між ознаками, фільтрує шум в даних і, отже, запобігає перенавчанню, коли модель просто запам'ятовує передані їй навчальні екземпляри. Ідея, що лежить в основі даного методу, полягає в тому, щоб скоротити складність моделі, штрафуючи екстремальні значення ваг.

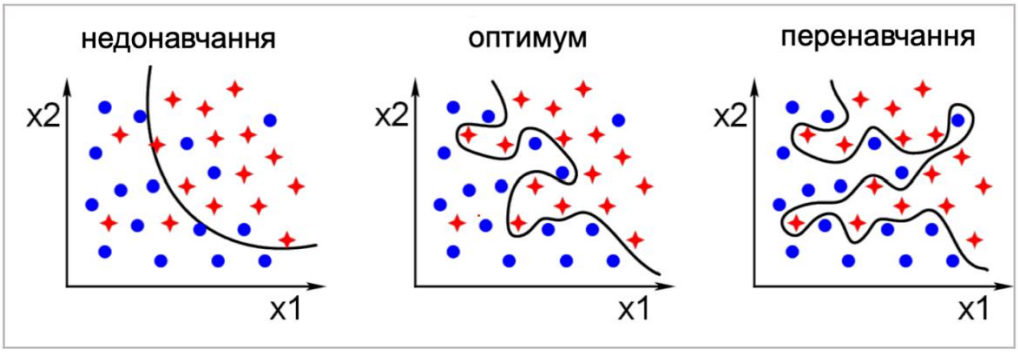


Рисунок 2.6 – Проблема перенавчання і недонавчання

Перша форма регуляризації, яку розглянемо – це *L2*-регуляризація, вона сприяє появі малих, але ненульових, вагових коефіцієнтів моделі, та записується таким чином:

$$\lambda \|w\|_2^2 = \lambda \sum_{j=1}^{m} w_j^2. \tag{2.9}$$

У формулі лямбда позначає параметр регуляризації, за допомогоюцього параметра можемо керувати тим, наскільки якісно відбувається підгонка до навчальних даних, при цьому ваги залишаються низькими. Отже, коли λ збільшується, підвищується і сила регуляризації.

Ще один підхід до скорочення складності моделі – регуляризація*L1*, яку можна представити в наступному вигляді:

$$\lambda \|w\|_1 = \lambda \sum_{j=1}^{m} |w_j|. \tag{2.10}$$

L1-регуляризація відрізняється від *L2* тим, що вона зазвичай повертає розряджені вектори ознак, тобто багато ваг дорівнюють 0 [33]. Таким чином, дану регуляризацію можна сприймати як прийом, що дозволяє відібрати важливі ознаки, оскільки розрідженість на практиці може бути, коли оперуємо даними з великим числом ознак, з яких не всі є релевантними.

Можемо регуляризувати функцію витрат для логістичної регресії шляхом додавання, наприклад, члена регуляризації *L2*, він буде скорочувати ваги моделі в ітеративному процесі навчання:

$$logL(w) = \sum_{i=1}^{n} \left[y^i \log\left(sig\left(t^i\right)\right) + \left(1 - y^i\right) \log\left(1 - sig\left(t^i\right)\right) \right] + \lambda \|w\|^2 \tag{2.11}$$

У результаті витрати наближаються до 0, якщо робимо коректний прогноз, що зразок має мітку того чи іншого класу. Однак у випадку неправильного прогнозу, витрати прагнуть до нескінченності. Таким чином, при некоректних прогнозах класу штрафуємо модель більш високими витратами.

Метод k-найближчих сусідів. Алгоритм МН з учителем k-найближчих сусідів або *KNN* є типовим прикладом лінивого учня [34]. Даний метод носить таку назву не тільки через свою об'єктивну простоту, а й через те, що він зберігає інформацію, передану йому під час навчання, без налаштування цільової функції та використовує її на тестових даних. Фактично, *KNN* не має фази навчання, оскільки він тільки запам'ятовує всю навчальну вибірку, але на етапі класифікації нових даних алгоритм безпосередньо шукає клас точок, які ближче всього до цільового об'єкту.

З цього слідує, що алгоритми МН можуть бути розділені на параметричні і непараметричні моделі [35]. Використовуючи параметричні алгоритми, оцінюємо параметри на навчальних даних для виведення функції, яка буде здатна передбачати мітки нових точок даних, у такому разі навчальний набір даних надалі не буде потрібен. Стандартним прикладом групи параметричних

моделей є логістична регресія, описана раніше.

Непараметричні алгоритми навпаки не можуть бути описані за допомогою конкретного набору параметрів, кількість яких заздалегідь відома. Їх число буде зростати разом з навчальним набором даних. *KNN* відноситься до метричних алгоритмів класифікації, підкатегорії непараметричних моделей. З вище сказаного слід, що непараметричні моделі не роблять явних припущень про глобальні закони, яким підпорядковуються дані. Єдине припущення, яке роблять метричні методи, – це те, що властивості об'єктів можна дізнатися, маючи подання про його сусідів.

Реалізацію алгоритму можна розбити на три етапи:

– вибрати число *k*, тобто кількість екземплярів навчальною вибірки, які розглядатиметься при класифікації нового об'єкта та метрику відстані;

– знайти *k* найближчих сусідів для цільового зразка;

– призначити мітку класу на основі мажоритарного голосування.

На рис. 2.7 показано, як новій точці даних (зеленому колу) присвоюється клас червоного трикутника, так як з трьох найближчих сусідів трикутник набрав більшість голосів, а клас синього квадрата отримав лише один голос.

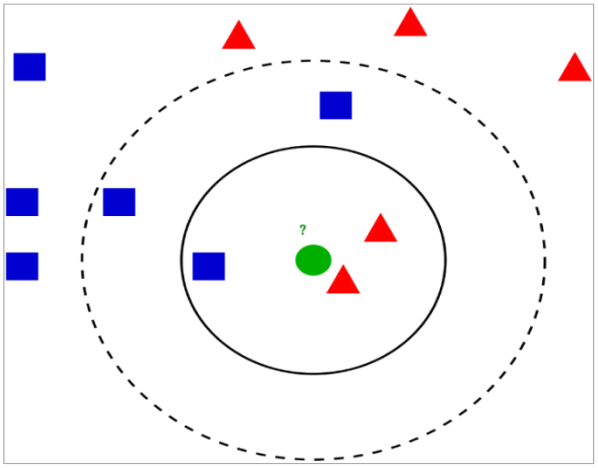


Рисунок 2.7 – Призначення тестового зразка класу трикутника по мажоритарному голосуванню

Таким чином, алгоритм *KNN* дуже легко інтерпретувати, крім того, він швидко адаптується до появи нових зразків у навчальній вибірці. Однак

обчислювальна складність при класифікації нових зразків буде лінійно зростати зі збільшенням кількості навчальних даних, тільки якщо ці дані не мають занадто значного числа ознак вимірів.

У стандартній реалізації *KNN* використовують просте незважене голосування. Це передбачає, що всі *k* екземплярів мають однакове право голосу в незалежності від відстані до класифікованого об'єкта. Тим не менш, чим далі зразок навчального набору даних розташований від об'єкта, що класифікується, в ознаковому просторі, тим нижча його значущість щодо класу. Тому для покращення результатів роботи моделі можна ввести зважування прикладів, при якому вага значимості «сусіда» буде зменшуватися при віддаленості від цільового об'єкт. У такому випадку використовують зважене голосування.

Метод *KNN* знаходить в навчальних даних *k* прикладів, найбільш схожих на точку, що класифікується, спираючись на обрану метрику відстані. Існує ряд способів, як обчислити відстань між двома точками в просторі, в переважній більшості випадків використовується евклідова відстань, обчислювана за формулою:

$$d(x, y) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}. \qquad (2.12)$$

Також достатньо часто застосовується манхеттенська метрика, оскільки у зрозумілому просторі вона більш стійка до викидів. Припустимо, існує два майже ідентичних об'єкта в просторі з тисячею ознак, але при цьому вони сильно відрізняються за однією з ознак. Отже, з великою ймовірністю є викид у цій ознаці, і швидше за все ці об'єкти таки близькі. Якби в такій ситуації скористалися евклідовою метрикою, то відмінність в єдиній ознаці посилилося б і зробила б об'єкти далі один від одного через квадрат у формулі. На манхеттенській відстані навпаки ця відмінність нівелюється, так як використовується саме модуль:

$$d(x,y) = \sum_{i=1}^{n} |x_i - y_i|. \quad (2.13)$$

Якщо не писати алгоритм самостійно з нуля, а використовувати готовий метод з бібліотеки *Scikit-learn* для машинного навчання, то за замовчуванням буде використовуватися метрика Мінковського, яка представляє собою узагальнення евклідової (при цьому параметр p буде дорівнює 2) і манхеттенська відстань ($p = 1$), в стандартній реалізації $p = 2$:

$$d(x,y) = \left(\sum_{i=1}^{n} |x_i - y_i|^p \right)^{1/p}. \quad (2.14)$$

Виходячи з вищесказаного, стає ясно, що у даного алгоритму є ряд недоліків: він не надто ефективний в завданнях, де потрібна велика кількість даних, також чутливий до неінформативних ознак. Крім того, для якісної роботи алгоритму потрібна не тільки метрична близькість об'єктів, але і їх семантична близькість. Однак у методу *KNN* є безліч застосувань в реальному світі, попри його недоліки. Якщо правильно обробити вихідні дані, алгоритм можна використовувати для пошуку схожих по семантиці документів, у разі якщо теми документів будуть схожі за умови, що векторні уявлення близькі один до одного.

Навчання на основі дерев рішень.

Алгоритми класифікації, які базуються на деревах прийняття рішень, як можемо припустити з назви, поділяють дані, задаючи певні питання.

В якості прикладу розглянемо дерево на рис. 2.8, яке показує виживання пасажирів «Титаніка». Цифри під листям показують ймовірність виживання і відсоток спостережень в листі. Подивившись на діаграму можна зробити висновок, що шанси на виживання були високі, якщо пасажир трансатлантичного пароплава був жінкою чи маленьким хлопчик з кількома родичами на борту. Таким чином, у вузлах дерева міститься правило або питання, що задається, в листі, відповідно, знаходиться підмножина об'єктів, які

задовольняють усім правилам гілки, яка закінчується даними листом.

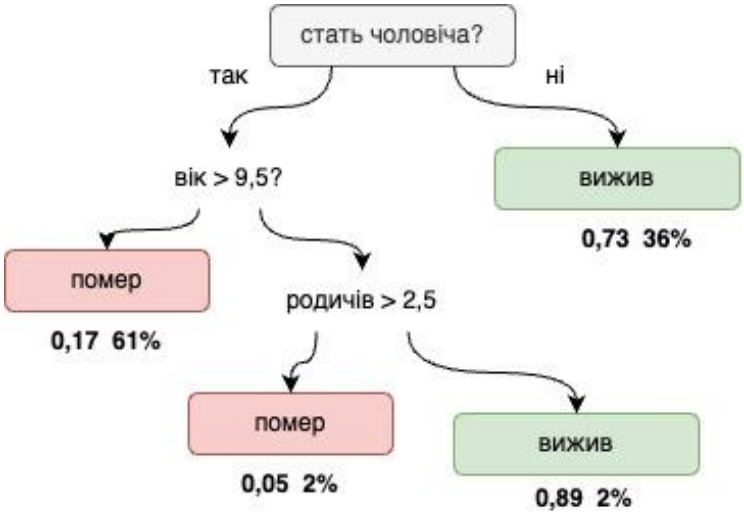


Рисунок 2.8 – Дерево рішень задачі виживання на Титаніку

Використовуючи наступний алгоритм, починаємо шлях від кореня дерева та розбиваємо дані за конкретною ознакою, яка в результаті дає найбільший приріст інформації, далі це поняття буде розкрито детальніше. Потім процедура рекурсивно повторюється: до кожної підмножини знову застосовується правило, поки не буде досягнута деяка умова зупинення роботи алгоритму, тобто заздалегідь визначаємо максимальну глибину дерева, щоб уникнути перенавчання. В останньому вузлі перевірка та розбиття не проводиться, і він оголошується листом, він визначає рішення для кожного, хто потрапив в нього.

При формуванні правила розбиття в вузлах дерева прийняття рішень необхідно вибрати ознаку, по якій це буде зроблено. Загальне правило полягає в наступному: обраний атрибут повинен розбити множину спостережень у вузлі так, щоб підмножини в результаті отримували приклади з однаковими мітками, вказівними на приналежність до класу, або були максимально наближені до нього. Для цього існують два основних критерії – теоретико-інформаційний і статистичний.

Перший критерій заснований на поняттях теорії інформації, як випливає з назви. Запозичене поняття інформаційна ентропія в аналізі даних і МН використовується як міра класової однорідності підмножини спостережень, які

вийшли під час поділу алгоритмом навчального набору даних на класи [36]. Її можна написати в вигляді формули:

$$H = -\sum_{i=1}^{n} \frac{N_i}{N} \log\left(\frac{N_i}{N}\right), \tag{2.15}$$

де n – загальне число елементів вихідної множини;

N – кількість прикладів в підмножині;

N_i – число прикладів, які належать класу i.

Отже, найкращою ознакою для розбиття буде та, яка максимально знизить ентропію фінальної підмножини.

Однак на практиці кажуть не про ентропії, а про інформації, величини зворотної до ентропії. Таким чином, оптимальний атрибут розбиття забезпечує найбільший приріст інформації (*information gain*):

$$IG(A) = info(S) - info(S_A), \tag{2.16}$$

де $info(S)$ вказує на інформацію про підмножину S до розбиття, а $info(S_A)$ вже визначає інформацію про туж підмножину після розбиття її за атрибутом A.

Статистичний підхід спирається на використання індексу Джині, даний статистичний показник показує, наскільки часто приклад, обраний з навчальної множини випадковим чином, буде розпізнаний некоректно, при умови, що цільові значення цієї меожини були отримані із деякого статистичного розподілу.

З цього випливає, що індекс Джині є показником відстані між двома розподілами – цільових значень і передбачень моделі, – тому логічно зробити висновок, що чим менша ця відстань, тим вища якість роботи моделі. Показник можна уявити в вигляді наступної формули:

$$Gini(Q) = 1 - \sum_{i=1}^{n} p_i^2, \tag{2.17}$$

де Q – наявна в результаті множина;

n – кількість класів в множині;

p_i – ймовірність класу i, виражена як відносна частота прикладів відповідного класу.

Цей індекс, очевидно, варіюється від 0 до 1. Він дорівнюватиме 0, якщо всі приклади Q будуть віднесені до одного класу, та дорівнює 1 в випадку, коли всі класи мають рівну ймовірність і представлені в рівних пропорціях.

Було розглянуто, як влаштоване одне вирішальне дерево, та його принцип роботи може створити враження, що одного такого класифікатора було б достатньо для вирішення завдання, якби була підібрана оптимальна глибина дерева. Але на сьогоднішній день замість одного дерева прийняття рішень широко застосовується алгоритм випадковий ліс, оскільки він відрізняється ефективністю класифікації та легкістю масштабованості. Його можна розглядати, як ансамбль одиночних дерев. Ціль ансамблевих методів полягає в тому, щоб об'єднати різні класифікатори в один, який матиме кращу ефективність узагальнення, чим кожен індивідуальний класифікатор окремо.

Головна ідея випадкового лісу – усереднити множину дерев рішень для побудови більш надійної моделі, яка менш схильна до перенавчання і має високу ефективність узагальнення. Для оцінки якості роботи алгоритму існує розкладання на три компонента – зміщення (*bias*), розкид (*variance*) і шум (*noise*) – у літературі воно відоме як *bias-variance decomposition*. Кожна частина розкладеної помилки моделі відповідає за окремий фактор: зміщення, розкид або дисперсія, шум.

Зміщення – здатність моделі наблизити найкращу серед усіх можливих моделей, наприклад, якщо намагатися вирішити абсолютно нелінійне завдання за допомогою лінійної моделі, то навчена модель буде вважатися зміщеною, тобто занадто слабкою для цього завдання.

Розкид або дисперсія – стійкість моделі до змін у навчальній вибірці, так дерева рішень будуть дуже сильно змінюватися в залежності від переданих даних, отже, у них великий розкид.

Шум – характеристика складності та суперечливості даних. Якщо, наприклад, у вибірці є два об’єкти з однаковими ознаками, але різними мітками, то тоді модель точно не зможе мати нульову помилку.

З цього випливає, що при вирішенні будь-якого завдання методами МН маємо на меті знайти таку модель, яка є незміщеною, значить, досить складною, щоб пристосуватися до цих даних, і яка має низький розкид, що дозволяє їй знайти загальні закономірності, а не залежати від невеликих змін у вибірці. З цієї причини поодинокі дерева рішень не будуть найкращим вибором алгоритму, у них занадто висока дисперсія. Випадковий ліс, навпаки, допомагає скоротити розкид без збільшення усунення.

Алгоритм випадковий ліс представляє собою комбінацію бегінга (*bagging, bootstrap aggregation*) і метод випадкових підпросторів при виборі ознак прийняття рішень деревами. Технологія бегінга припускає, що з навчальної вибірки береться n прикладів, вони вибираються рівноймовірно, з повтореннями. Таким чином, отримаємо нову вибірку, в якій деякі елементи вихідної вибірки можуть не буті, при цьому інші можуть увійти кілька разів. Потім, повторивши процедуру k раз, отримаємо k моделей, навчених на k вибірках, передбачення яких усереднимо і отримаємо остаточну відповідь. Отже, класифікатори не виправляють помилки інших моделей, а компенсують їх при голосуванні. Процес генерації підвибірок за допомогою вибору екземплярів вихідної вибірки з поверненням або повтореннями називається бутстрепом (*bootstrap*).

Представимо алгоритм випадкового лісу у вигляді чотирьох етапів.

Перший етап – згенерувати бутстреп-вибірку з елементів.

Другий етап – створити дерево прийняття рішень з отриманої вибірки. Також в кожному вузлі потрібно:

– випадковим чином вибрати піднабір атрибутів без повернення всіх

ознак, метод випадкових підпросторів зменшує кореляцію між деревами, що дозволяє уникнути перенавчання [28];

– розділити вузол дерева, використовуючи конкретну ознаку, яка забезпечує найкращий поділ, тобто доводить приріст інформації до максимуму.

Третій етап – повторити кроки 1 і 2 k разів.

Четвертий етап – об'єднати прогнози всіх дерев у випадковому лісі та обрати відповідь за більшістю голосів.

Необхідно відзначити, розкид моделі виходить знизити, використовуючи процедуру бегінга, але на зміщення він не впливатиме. Тому можна зробити висновок, що зсув має бути низьким саме у дерев, з яких будується ансамбль. Рішення даної ситуації – це використання глибоких дерев, так як вони будуть докладно запам'ятовувати підбірку даних; з цього випливає, що на тестовому наборі передбачення будуть сильно варіюватися залежно від навчальної підвибірки, але за усереднення прогнозів результат з великою ймовірністю буде близький до істині, дисперсія при цьому буде високою, а усунення низьким. Неглибокі дерева навпаки здатні запам'ятати тільки верхньорівневі залежності в навчальній підвибірці, а вони будуть схожі у всіх підгрупах даних, через це цільова залежність буде описана недостатньо докладно. Змінюючи навчальні підвибірки, отримуватимемо на тестовому об'єкті неточні прогнози моделі, але стабільні, отже, розкид низький, але усунення високе. В результаті, даний алгоритм слід використовувати з глибокими деревами, так як ансамблева модель стійка до шуму від індивідуальних дерев.

Єдиний гіперпараметр алгоритму випадкового лісу, який необхідно підібрати – це кількість дерев ухвалення рішень. Як правило, ефективність моделі прямо пропорційна кількості обраних дерев за рахунок збільшення обчислювальних витрат.

Також можна оптимізувати інші гіперпараметри класифікатора, наприклад, розмір бутстреп-вибірки та кількість ознак, обираються випадковим чином для кожного поділу. Вибираючи розмір бутстреп-вибірки, керуємо компромісом між зміщенням і дисперсією випадкового лісу. Якщо зменшуємо

розмір даної вибірки, то це призведе до збільшення відмінностей між одиночними деревами, оскільки шанс того, що конкретний зразок з'явиться у бутстреп-вибірці, стає низьким. Хоча таке рішення може послабити перенавчання за допомогою захоплення довільності випадкового лісу, але найчастіше це призводить до зниженню загальної ефективності моделі. І зворотна процедура, збільшення розміру бутстреп-вибірки, навпаки з великою ймовірністю може збільшити ступінь перенавчання, так як індивідуальні дерева стануть занадто схожі одне на одне.

Крім того, можемо керувати числом ознак, що використовуються при навчанні одного дерева, і тим самим впливати на якість випадкового лісу. Чим більше ознак, тим менше відчувається ефект роботи ансамблю, тому що збільшується кореляція між одиночними дерева. Відповідно, зменшуючи кількість ознак, робимо самі дерева слабшими. На практиці для класифікації рекомендується брати корінь із усіх ознак для визначення максимальної кількості ознак.

XGBoost. Розглянемо самий актуальний алгоритм на сьогоднішній день, який використовує допоміжний алгоритм бустингу (boosting – посилення). Ідея бустингу полягає в тому, щоб при навчанні моделі послідовно мінімізувати помилки. Через те, що провісники навчаються на помилках, здійснених попередниками, знадобиться менше часу для досягнення правильної відповіді. Цей підхід, як і інші ансамблеві методи, перетворює кількох слабких учнів на сильну модель і шукає компроміс між зміщенням і дисперсією.

Насамперед перш ніж перейти до основного алгоритму *XGBoost*, коротко розглянемо кілька інших алгоритмів бустингу і їх особливості. Різновид бустингу AdaBoost (*adaptive boosting – адаптивний бустинг*) концентрує увагу на недонавченості моделі, з цього випливає, що алгоритм при кожному новому пророкуванні буде фокусуватися на складних підвиборчих даних, чия класова приналежність неочевидна. Припустимо, що хочемо побудувати класифікатор *AdaBoost*, де вирішальне дерево є основним алгоритмом, тоді ваги помилково класифікованих елементів, за принципом адаптивного бустингу, будуть

зростати. На наступній ітерації навчання алгоритм навчатиметься на оновлених коефіцієнтах. В такому разі загальна схема роботи адаптивного бустингу виглядає наступним чином: вагові коефіцієнти збільшуються саме у неправильно класифікованих об’єктів навчальної вибірки, щоб наступна модель змогла зробити для них правильні передбачення. *AdaBoost* через свої техніки послідовного навчання нагадує метод градієнтного спуску, що прагне знайти мінімум функції витрат, при цьому він робить дотичну лінію до функції та ітеративно намагається просуватися в негативному напрямку нахилу, бік глобального мінімуму функції [28]. Адаптивний бустінг відрізняється тим, що він змінює параметри або ваги не тільки одного предиктора, а збирає моделі до ансамблю, який поступово покращує. Але великий недолік даного методу – це відсутність можливості паралельного навчання моделей, так як кожен з предикторів починає навчання тільки після того, як отримав відомості від попереднього, який закінчив цикл навчання.

Інший найбільш загальний алгоритм бустингу – градієнтний, всі сучасні методи бустингу, так чи інакше, є його модифікаціями або окремими випадками. Його принцип роботи схожий з описаним вище інструментом *AdaBoost*. Але градієнтний бустінг відрізняється від адаптивного, який змінює параметри моделі на кожній ітерації, тим, що він рухається до мінімуму функції втрат, спираючись на залишкову помилку минулої моделі. Таким чином, він мінімізує помилку алгоритмом градієнтного спуску, тобто підбирає коефіцієнти так, щоб функція втрат (різниця між істинним значенням та прогнозом моделі) стала мінімальною, отже, передбачення було максимальнонаближено до правильного. Градієнтний бустінг реалізується таким чином:

– спочатку модель будується на навчальній вибірці і робитьпередбачення для всього набору даних;

– потім за істинними і передбачуваними відповідям обчислюються помилки;

– для нової моделі ці помилки будуть цільовою змінною, вона буде

робити нові прогнози, які поєднуватимуться з прогнозами попередніх моделей;

– на наступному етапі знову обчислюються помилки з опорою на правильні значення та передбачені;

– навчальний процес закінчиться, якщо функція втрат не зміниться або якщо буде досягнуто максимальне число предикторів [33].

Алгоритм *XGBoost* (*extreme gradient boosting*) або екстремальний градієнтний бустинг, в свою чергу, є просунутою реалізацією градієнтного бустингу, використовуваного в поєднанні з деревами прийняття рішень.

Екстремальний градієнтний бустинг – це ансамблевий метод дерев, що використовує принцип бустинг слабких учнів (у нашому конкретному випадку це бінарні дереварішень, в вузлах яких розгалуження може вестись тільки в двох напрямках, тобто здійснюється вибір з двох альтернатив (корисний коментар або ні)) завдяки градієнтному спуску, також він покращує архітектуру *Gradient Boosting Machines* (*GBM – альтернативна назва градієнтного бустингу*) за допомогою системи оптимізація.

Даний алгоритм був розроблений для покращення ефективності обчислювальних ресурсів та скорочення витрат часу та пам’яті. Крім паралелізації, *XGBoost* забезпечує обчислення для дуже великих наборів даних, які не можуть поміститися в пам’яті комп’ютера. Також він відсікає гілки дерев, використовуючи параметр максимальної глибини (*max_depth*). Екстремальний градієнтний бустінг спочатку будує дерева розміром з *max_depth*, що забезпечує швидке навчання, так як не потрібно оцінювати параметри регуляризації у кожному вузлі. Після того, як кожне дерево буде збільшено до *max_depth*, алгоритм проходить рекурсивно від нижньої частини дерева до самого верху і перевіряє, чи задовольняє поділ умовам, встановленим у гіперпараметрах. Якщо поділ чи вузол відповідає вимогам, він видаляється з дерево [36].

Виходячи з перерахованих переваг сучасного алгоритму *XGBoost*, можемо зробити припущення, що на практиці він даватиме більш гарні результати, чим його попередник випадковий ліс.

2.5 Вибір технології машинного навчання

Вибір мови програмування розробки алгоритму.

Основним завданням цієї роботи є проведення операцій отримання, обробки та аналізу даних. Виходячи з цього, при виборі мови програмування, основним критерієм є широкий спектр можливостей роботи з даними, що надається мовою.

Мова програмування *R*. *R* – мова програмування (МП) для статистичної обробки даних, а також вільне програмне середовище обчислень з відкритим вихідним кодом. Ця мова програмування широко використовується як статистичне програмне забезпечення для аналізу даних та фактично стала стандартом для статистичних програм [39].

Основною перевагою *R* є підтримка широкого спектру статистичних і чисельних методів, а також можливість розширення за допомогою підключення спеціалізованих пакетів. Пакети є бібліотеками, які надають розробнику доступ до додаткових функцій. Наприклад, пакети передбачення, які містять спеціальні функції нейронних мереж, нелінійної регресії. Також у цій мові передбачена можливість візуалізації даних, що є корисною особливістю під час роботи з даними.

Однак, ця мова має відносно невисоку продуктивність, що значно обмежує можливості розробника при роботі з процесами отримання, передачі та аналізу великих обсягів даних.

Мова програмування *Java*. *Java* – типізована об’єктно-орієнтована мова програмування, розроблена компанією «*Sun Microsystems*».

Програми *Java* зазвичай транслюються в спеціальний байт-код, тому вони можуть працювати на будь-якій комп’ютерній архітектурі за допомогою віртуальної *Java*-машини [40].

Основною перевагою даної мови програмування є можливість інтеграції методів науки про дані безпосередньо до існуючої кодової бази. Оскільки *Java* є високопродуктивною, скомпільованою мовою загального призначення, це

робить її придатною для написання ефективного виробничого коду «*ETL*», а також алгоритмів машинного навчання з використанням обчислювальних засобів.

Недоліком мови програмування *Java* є малий обсяг спеціалізованих бібліотек для передових статистичних методів порівняно з деякими предметно-орієнтованими мовами, розглянутими в даній роботі.

Мова програмування *Python*. *Python* – високорівнева мова програмування загального призначення, орієнтована на підвищення продуктивності розробника та читання коду. Синтаксис ядра *Python* мінімалістичний.

У той же час, стандартна бібліотека включає великий обсяг корисних функцій. Мова програмування *Python* розроблена в 1991 році. З того часу ця мова стала надзвичайно популярною мовою програмування загального призначення і широко використовується в співтоваристві фахівців за даними [41].

Мова програмування *Python* має відносно низький поріг входження для програмістів-початківців. Також є спеціалізовані бібліотеки, які дозволяють розширити наявний функціонал мови для роботи в галузі машинного навчання. Основним недоліком *Python* є порівняно невисока швидкість виконання *Python*-програми, що обумовлено її інтерпретованістю.

Ця мова програмування вважається кращим варіантом для цілей науки про дані, більшість якої зосереджена навколо процесу «*ETL*». *Python* стане в нагоді у випадках, коли завдання, пов’язані з аналізом даних, поєднуються з розробкою вебдодатків, або якщо статистичний код потрібно інкорпорувати в базу даних.

Python, будучи повнофункціональною мовою програмування, підходить для реалізації алгоритмів з їх подальшим практичним використанням. Ця особливість робить *Python* найбільш підходящим МП для розробки алгоритму прогнозування та машинного навчання.

Pandas – це бібліотека з відкритим вихідним кодом, що надає високопродуктивні, прості у використанні структури даних та інструменти аналізу мови програмування *Python*.

Python відмінно підходить для збору даних, але меншою мірою для аналізу

та моделювання даних. *Pandas* допомагає заповнити цю прогалину, дозволяючи виконувати весь робочий процес аналізу даних на *Python* без необхідності переходу на більш відповідну для аналізу даних мову, наприклад *R*.

Особливості:

– швидкий та ефективний об’єкт *DataFrame* для обробки даних з інтегрованою індексацією;

– інструменти для читання та запису даних між структурами даних у пам’яті та різними форматами: *CSV, Microsoft Excel*, бази даних *SQL* та швидкий формат *HDF5*;

– інтелектуальне вирівнювання даних та інтегральна обробка відсутніх даних: автоматичне вирівнювання на основі міток в обчисленнях та легке маніпулювання неструктурованими даними у впорядкованій формі;

– інтелектуальна нарізка на основі ярликів, зручна індексація та підмножина великих наборів даних;

– перетворення даних за допомогою механізму, що дозволяє виконувати операції *split-apply-comb* на наборах даних;

– високопродуктивне об’єднання наборів даних;

– *Pandas* використовується в найрізноманітніших академічних та комерційних галузях, включаючи фінанси, неврологію, економіку, статистику, рекламу, вебаналітику та багато іншого.

Scikit-learn. *Scikit-learn* – це бібліотека для машинного навчання для мови програмування *Python*. Вона має різні алгоритми класифікації, регресії та кластеризації, випадкові ліси, підвищення градієнта, *k*-середнє та *DBSCAN*, призначені для взаємодії з чисельними бібліотеками *NumPy* та *SciPy*.

Natural Language Toolkit (*NLTK*) – це набір бібліотек і програм для символьної та статистичної обробки природних мов, написаних мовою програмування *Python*.

NLTK визначає інфраструктуру, яку можна використати для побудови програм *NLP* у *Python*. Вона надає базові класи для представлення даних, що мають відношення до обробки природної мови; стандартні інтерфейси для

виконання таких завдань: анотування частин мови, синтаксичний розбір і класифікація тексту; стандартні реалізації для кожного завдання, які можуть бути об'єднані для вирішення складних завдань.

Вибір середовища розробки.

Середовище розробки *Visual Studio Code* – повнофункціональна *IDE* від *Microsoft*, яка доступна на *Windows* і *MacOS*. Дане середовище розробки представлене у безкоштовному варіанті. *Visual Studio Code* дозволяє розробляти програми для різних платформ і надає свій власний набір розширень.

Python Tools for Visual Studio дозволяє писати на *Python* в *Visual Studio Code* і включає в себе *Intellisense* для *Python*, налагодження та інші інструменти [42].

Середовище розробки *PyCharm*.

Однією з найбільш популярних повнофункціональних *IDE*, призначених саме для розробки програмного забезпечення на мові програмування *Python* є *PyCharm*. Для середовища розробки існує як безкоштовний «*open-source*», і платний варіанти *IDE*. *PyCharm* доступний на операційних системах *Windows*, *MacOS* та *Linux*.

Запуск та налагодження *Python* коду здійснюється безпосередньо в середовищі *PyCharm*. Крім того, у самій *IDE* передбачена підтримка проєктів та системи управління версіями.

Середовище розробки *Spyder*.

Spyder – «*open-source*» *IDE* для *Python*, спеціально оптимізована для розробки *data science* додатків. *Spyder* поставляється в комплекті з менеджером пакетів *Anaconda*, який дозволяє спростити роботу підключення спеціалізованих пакетів, що розширюють можливості МП *Python* [42].

IDE Spyder користується високою популярністю саме у *data science* розробників. Це пов'язано з тим що саме середовище розробки взаємодіє зі спеціалізованими бібліотеками, як *SciPy*, *NumPy* і *Matplotlib*.

Spyder має ту ж функціональність, що і більшість стандартних *IDE*, наприклад, редактор коду з виділенням синтаксису, автодоповнення коду та вбудованого оглядача документації. Однак, дане середовище розробки має таку

відмінну особливість як наявність провідника змінних. Подібна функція дозволяє переглянути значення змінних у формі таблиці прямо всередині самої *IDE*.

Середовище розробки *Google Colab.*

Google Colab є потужним інструментом для розробки та експериментів з машинним навчанням, особливо для початківців та тих, хто шукає ефективний інструмент без значних витрат. Перерахуємо переваги використання *Google Colab* для машинного навчання.

Безкоштовність та доступність: *Google Colab* надає можливість працювати з потужними ресурсами безкоштовно, що особливо корисно для студентів, дослідників та девелоперів з обмеженим бюджетом.

Швидкий початок роботи: легкий доступ та можливість працювати з *Jupyter Notebooks* дозволяють швидко почати роботу з проєктами машинного навчання без значних затрат часу на налаштування середовища.

Зручна спільна робота: вбудована можливість спільної роботи та обміну кодом робить *Google Colab* ефективним інструментом для колективної розробки та вивчення машинного навчання.

Підтримка *GPU/TPU*: наявність безкоштовного доступу до графічних та тензорних процесорів дозволяє прискорити процеси тренування моделей та обчислень.

Оберемо *Google Colab,* як основне середовище розробки.

2.6 Попередня обробка стеку даних

NLTK – провідна платформа для створення програм *NLP* на *Python*. Вона має легкі у використанні інтерфейси для багатьох мовних корпусів, а також бібліотеки для обробки текстів для класифікації, токенізації, стеммінгу, розмітки, фільтрації та семантичних міркувань. Розглянемо основні етапи попередньої обробки даних на основі бібліотеки *NLTK.*

Токенізація за реченями. Токенізація (іноді – сегментація) за реченями – це

процес поділу письмової мови на пропозиції-компоненти. Ідея виглядає досить простою. В англійській та деяких інших мовах можемо виокремлювати речення кожного разу, коли знаходимо певний знак пунктуації – точку.

Візьмемо невеликий текст опису мобільної версії настільної гри нарди. Щоб зробити токенізацію речень за допомогою *NLTK*, можна скористатися методом `nltk.sent_tokenize` (див. рис. 2.9). На виході отримаємо 3 окремі речення (див. рис. 2.10).

```
import nltk
nltk.download('punkt')
text = "Backgammon is one of the oldest known board games. Its history
      can be traced back nearly 5,000 years to archaeological
      discoveries in the Middle East. It is a two-player game where
      each player has fifteen checkers which move between twenty-four
      points according to the roll of two dice."
sentences = nltk.sent_tokenize(text)
for sentence in sentences:
    print(sentence)
    print()
```

Рисунок 2.9 – Токенізація методом `nltk.sent_tokenize`

```
[Output]:
Backgammon is one of the oldest known board games.
Its history can be traced back nearly 5,000 years to archaeological
discoveries in the Middle East.
It is a two-player game where each player has fifteen checkers which
move between twenty-four points according to the roll of two dice.
```

Рисунок 2.10 – Результат виконання методу `nltk.sent_tokenize`

Токенізація за словами. Токенізація (іноді – сегментація) за словами – це процес поділу речень на слова-компоненти. В англійській та багатьох інших мовах, які використовують ту чи іншу версію латинського алфавіту, пробіл – це непоганий роздільник слів.

Тим не менш, можуть виникнути проблеми, якщо будемо використовувати тільки пробіл – в англійській складові іменники пишуться по-різному і іноді через пропуск. І тут знову допомагають бібліотеки.

Візьмемо речення з попереднього прикладу та застосуємо до нього метод

`nltk.word_tokenize` (див. рис. 2.11).

```
for sentence in sentences:
    words = nltk.word_tokenize(sentence)
    print(words)
    print()

[Output]:
['Backgammon', 'is', 'one', 'of', 'the', 'oldest', 'known', 'board',
'games', '.']

['Its', 'history', 'can', 'be', 'traced', 'back', 'nearly', '5,000',
'years', 'to', 'archaeological', 'discoveries', 'in', 'the', 'Middle',
'East', '.']

['It', 'is', 'a', 'two-player', 'game', 'where', 'each', 'player',
'has', 'fifteen', 'checkers', 'which', 'move', 'between', 'twenty-
four', 'points', 'according', 'to', 'the', 'roll', 'of', 'two',
'dice', '.']
```

Рисунок 2.11 – Токенізація методом `nltk.sent_tokenize`

Лематизація та стемінг тексту. Зазвичай тексти містять різні граматичні форми одного й того ж слова, і навіть можуть зустрічатися однокореневі слова. Лематизація і стемінг мають на меті привести всі словоформи, які зустрічаються до однієї, нормальної словникової форми.

Приведення різних словоформ до однієї:

dog, dogs, dog's, dogs' => dog

Те ж саме, але вже стосовно цілого речення:

the boy's dogs are different sizes => the boy dog be differ size

Лематизація та стемінг – це окремі випадки нормалізації і вони відрізняються.

Стемінг – це грубий евристичний процес, який відрізає «зайве» від кореня слів, часто це призводить до втрати словотворних суфіксів.

Лематизація – це тонший процес, який використовує словник і морфологічний аналіз, щоб у результаті призвести слово до його канонічної формі – лемі.

Відмінність у тому, що стемер (конкретна реалізація алгоритму стемінгу) діє без знання контексту і, відповідно, не розуміє різницю між словами, які мають

різний зміст залежно від частини мови. Однак у стемерів є свої переваги: їх простіше впровадити і вони працюють швидше. Плюс, нижча «акуратність» може мати значення у деяких випадках.

Наприклад, слово *good* – це лема для слова *better*. Стемер не побачить цей зв'язок, тому що тут потрібно звірятися зі словником. Слово *play* – базова форма слова *playing*. Тут упораються і стемінг, і лематизація.

Слово *meeting* може бути як нормальною формою іменника, і формою дієслова *to meet*, залежно від контексту. На відміну від стемінгу, лематизація спробує вибрати правильну лему, спираючись на контекст.

На рисунку 2.12 представлено реалізацію методів лематизації та стемінгу на основі бібліотеки *NLTK*.

```
from nltk.stem import PorterStemmer, WordNetLemmatizer
from nltk.corpus import wordnet

def compare_stemmer_and_lemmatizer(stemmer, lemmatizer, word, pos):
    """
    Print the results of stemming and lemmatization using the passed
stemmer, lemmatizer, word, and pos (part of speech)
    """
    print("Stemmer:", stemmer.stem(word))
    print("Lemmatizer:", lemmatizer.lemmatize(word, pos))
    print()

lemmatizer = WordNetLemmatizer()
stemmer = PorterStemmer()

compare_stemmer_and_lemmatizer(stemmer, lemmatizer, word="seen",
pos=wordnet.VERB)
compare_stemmer_and_lemmatizer(stemmer, lemmatizer, word="drove",
pos=wordnet.VERB)

[Output]:
Stemmer: seen
Lemmatizer: see

Stemmer: drove
Lemmatizer: drive
```

Рисунок 2.12 – Реалізація методів лематизації та стемінгу на основі бібліотеки *NLTK*

Стоп-слова. Стоп-слова – це слова, які викидаються з тексту перед обробкою тексту. Коли застосовуємо машинне навчання до текстів, такі слова можуть додати багато шуму, тому необхідно позбавлятися нерелевантних слів. Під стоп-словами зазвичай розуміють артиклі, вигуки, та інші, які не несуть смислового навантаження. При цьому треба розуміти, що немає універсального списку стоп-слів, все залежить від конкретного випадку.

NLTK має попередньо встановлений список стоп-слів. Розглянемо, як можна прибрати стоп-слова з речення (див. рис. 2.13).

```
import nltk
from nltk.corpus import stopwords
nltk.download('punkt')
nltk.download('stopwords')
# View the list of English stopwords
# print(stopwords.words("english"))
stop_words = set(stopwords.words("english"))
sentence = "Backgammon is one of the oldest known board games."

words = nltk.word_tokenize(sentence)
without_stop_words = [word for word in words if word.lower() not in
stop_words
print(without_stop_words)

[Output]:
['Backgammon', 'one', 'oldest', 'known', 'board', 'games', '.']
```

Рисунок 2.13 – Видалення стоп-слів на основі бібліотеки *NLTK*

Регулярні вирази. Регулярний вираз (*regexp*, *regex*) – це послідовність символів, що визначає шаблон пошуку. Регулярні вирази можемо використовувати для додаткового фільтрування тексту. Наприклад, можна усунути всі символи, які не є словами.

У багатьох випадках пунктуація не потрібна і її легко забрати за допомогою регулярних виразів.

Модуль *re* в *Python* представляє операції з регулярними виразами. Можемо використовувати функцію `re.sub`, щоб замінити все, що підходить під шаблон пошуку, на вказаний рядок (див. рис. 2.14).

```
import re
sentence = "The development of snowboarding was inspired by
skateboarding, sledding, surfing and skiing."
pattern = r"[^\w]"
print(re.sub(pattern, " ", sentence))

[Output]:
The development of snowboarding was inspired by skateboarding
sledding  surfing and skiing
```

Рисунок 2.14 – Заміна всіх «неслів» на пробіл

Мішок слів. Алгоритми машинного навчання не можуть працювати безпосередньо з сирим текстом, тому необхідно конвертувати текст у набори цифр (вектори). Це називається вилученням ознак.

Мішок слів – це популярна та проста техніка вилучення ознак, що використовується під час роботи з текстом. Вона визначає входження кожного слова до тексту.

Щоб використати модель, потрібно:

– визначити словник відомих слів (токенів);

– вибрати рівень присутності відомих слів.

Будь-яка інформація про порядок чи структуру слів ігнорується. Ось чому це називається мішком слів. Ця модель намагається зрозуміти, чи зустрічається знайоме слово у документі, але не знає, де саме воно зустрічається.

TF-IDF. У частотного скорингу є проблема: слова з найбільшою частотністю мають відповідно найбільшу оцінку. У цих словах може бути не так багато інформаційного виграшу для моделі, як у менш частих словах. Один із способів виправити ситуацію – знижувати оцінку слова, яке найчастіше зустрічається у всіх подібних документах. Це називається *TF-IDF*.

TF-IDF (скорочення від *term frequency – inverse document frequency*) – це статистичний міра для оцінки важливості слова в документі, який є частиною колекції або корпусу.

Скоринг по *TF-IDF* зростає пропорційно до частоти появи слова в документі, але це компенсується кількістю документів, що містять це слово.

Формула скорингу для слова *X* у документі *Y*:

$$w_{x,y} = tf_{x,y} \times \log\left(\frac{N}{df_x}\right), \quad (2.18)$$

де $tf_{x,y}$ – частота x в y,

df – кількість документів, що містять x,

N – загальна кількість документів,

TF (*term frequency* – частота слова) – відношення числа входжень слова до загального числа слів документа.

$$TF(term) = \frac{Numbers\ of\ times\ term\ appears\ in\ a\ document}{Total\ number\ of\ items\ in\ the\ document}. \quad (2.19)$$

IDF (*inverse document frequency* – зворотна частота документа) – інверсія частоти, з якою деяке слово зустрічається в документах колекції.

$$IDF(term) = \log\left(\frac{Total\ number\ of\ documents}{Number\ of\ documents\ with\ term\ in\ it}\right). \quad (2.20)$$

У результаті обчислити *TF-IDF* для слова *term* можна так:

$$TFIDF(term) = TF(term) \times IDF(term). \quad (2.21)$$

Можна використовувати клас `TfidfVectorizer` із бібліотеки *sklearn*, щоб обчислити *TF-IDF*.

2.7 Метрики оцінки якості моделі машинного навчання

Для оцінки якості роботи алгоритмів будемо використовувати площу під графіком *ROC* кривої. Дана крива допомагає оцінити якість бінарної

класифікації. *ROC* крива показує залежність частки вірних позитивних класифікацій від частки хибних позитивних класифікацій при варіювання порога вирішального правила (*TPR(FPR)*).

Частка хибних позитивних класифікацій (*False Positive Rate*, *FPR*):

$$TPR(a, X^m) = \frac{\sum_{i=1}^{m}[a(x_i) = +1][y_i = +1]}{\sum_{i=1}^{m}[y_1 = +1]}. \quad (2.22)$$

Частка вірних позитивних класифікацій (*True Positive Rate*, *TPR*):

$$PR(a, X^m) = \frac{\sum_{i=1}^{m}[a(x_i) = +1][y_i = -1]}{\sum_{i=1}^{m}[y_1 = -1]}, \quad (2.23)$$

де $a(x_i)$ – класифікатор, $X^m = (x_1, x_2, \dots, x_m)$ – вибірка об'єктів; $(y_1, y_2, \dots, y_m)$ – відповіді які відповідають об'єктам X^m.

Будується дана *ROC* крива наступним чином. Нехай потрібно розділити множину X на два класи: позитивний $(+1)$ та негативний (-1). Нехай за допомогою класифікатора $a(x)$ якимось чином можемо отримати ймовірність $f(x)$ того, що об'єкт x віднесено до позитивного класу. Тоді алгоритм обчислення *ROC* кривої наступний:

– обчислюємо представників класів $+1$ та -1 у вибірці: m_- і m_+ відповідно.

– упорядкуємо об'єкти $(x_1, x_2, \dots, x_m)$ по спаданню значень $f(x)$.

– початкова точка *ROC* кривої: $(FPR_0, TPR_0) := (0,0)$.

– для кожного i від 1 до m.

Якщо $y_i = -1$, то

$$FPR_i = FPR_{i-1} + \frac{1}{m_-};\ TPR_i := TPR_{i-1}. \quad (2.24)$$

Інакше:

$$FPR_i = FPR_{i-1};\ TPR_i \coloneqq TPR_{i-1} + \frac{1}{m_+}. \quad (2.25)$$

Результат роботи алгоритму представлений на рисунку 2.15.

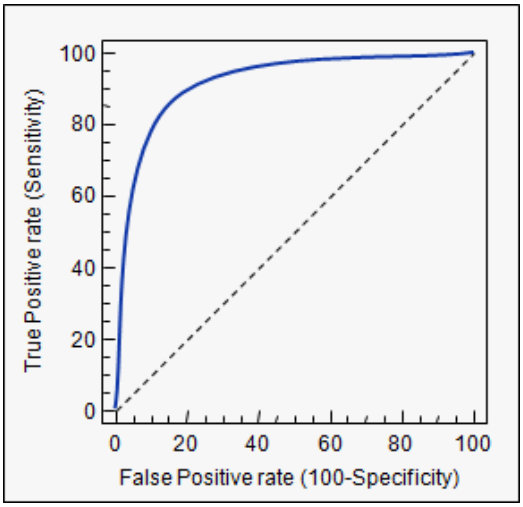


Рисунок 2.15 – *AUC* крива (залежність *TPR* від *FPR*) для 3 різних алгоритмів

Метод оцінки якості класифікатора – площа під ROC кривою. Змінюється від 0.5 (випадковий класифікатор) до 1.0 (ідеальний класифікатор) (Якщо значення менше 0.5, то це значить, що позначили -1 як позитивний клас, а +1 як негативний).

Переваги ROC-кривої в тому, що вона інваріантна щодо відносини помилок першого і другого роду.

Також, часто застосовується *F*-міра, визначається, як:

$$F = 2 * \frac{Precision * Recall}{Precision + Recall}, \quad (2.26)$$

де *Precision* – відношення числа правильно класифікованих об’єктів позитивного класу до загального числа позитивних об’єктів від класифікатора, а *Recall* – відношення числа правильно класифікованих об’єктів позитивного класу до спільного числа позитивних об’єктів.

Для класифікації відгуків за типами: звіт про помилку, запит функціональності, однак, *F*-мера не зовсім показова. Пов’язано це з тим, що в даному завданні найважливішим виявляється відстежити як можна більше звітів про помилки та запити функціональності. Тобто, в даній задачі *Recall* має більше значення, чим *Precision*. Для обліку цього, в *F* метриці можемо ввести додатковий коефіцієнт, який показує «важливість» *Recall* щодо *Precision*. Але для об’єктивних результатів, цього не робимо, і будемо явно обчислювати *Recall*.

Площа під графіком *ROC* кривої обчислюється, як правило, для бінарного класифікатор. При наявності кількох класів (звіти про помилки, запити функціональностей, неінформативні відгуки) навчаємо алгоритми для кожного зі згаданих класів окремо: відгук належить класу А – відгук не належить класу А. Таким чином, для одного набору даних маємо 3 значення метрики для кожного з класів. Аналогічно обчислюємо *Recall* метрику. Це може допомогти побачити деякі особливості класифікації по кожному класу.

3 РЕАЛІЗАЦІЯ СИСТЕМИ

3.1 Збір даних для машинного навчання

Парсери відгуків на вебзастосунки в машинному навчанні грають важливу роль у зборі та підготовці даних для подальшого аналізу та класифікації. Основна ідея полягає в тому, щоб автоматично отримувати та обробляти великий обсяг відгуків з вебсайтів, а потім використовувати ці дані для навчання моделей машинного навчання для автоматизованої аналітики чи класифікації відгуків на позитивні та негативні.

Нижче наведено кроки, які часто включаються в процес розробки парсера відгуків.

Вибір вебсайтів: визначення цільових вебсайтів, з яких ви хочете зібрати відгуки. Кожен вебсайт може мати власну структуру та вигляд даних, тому парсер повинен бути адаптований до конкретного джерела.

Аналіз *HTML*-коду: розглядання *HTML*-коду вебсайту для визначення місць, де розташовані відгуки. Використання інструментів, таких як бібліотека *BeautifulSoup* в *Python*, для ефективного парсингу *HTML*.

Видобуток даних: видобуток текстового вмісту відгуків, а також можливої додаткової інформації, такої як рейтинг, дата написання тощо.

Обробка даних: перетворення зібраних даних у структурований формат. Це може включати в себе очищення тексту від *HTML*-тегів, фільтрацію непотрібних символів, токенізацію і лематизацію слів.

Збереження даних: збереження даних у відповідному форматі для подальшого використання. Це може бути збереження в базі даних, у текстовому файлі або в іншому форматі, який зручний для обробки.

Машинне навчання: використання зібраних даних для тренування моделей машинного навчання для класифікації відгуків. Такі моделі можуть використовувати алгоритми, наприклад, логістичну регресію, метод k-

найближчих сусідів, випадковий ліс чи інші, для передбачення тональності відгуків.

Оцінка результатів: оцінка точності та ефективності розробленої моделі на тестових даних. Це допомагає визначити, наскільки добре ваш парсер та класифікатор працюють на реальних даних.

Цей процес може бути складним і вимагати уважного аналізу та оптимізації кожного етапу. Застосування машинного навчання дозволяє автоматизувати процес класифікації відгуків та отримати значну користь в області аналізу сприйняття користувачами вебзастосунків.

Зважаючи на те, що конкретний парсер буде залежати від структури та джерела відгуків, надамо загальний приклад використання бібліотек *Python* для парсингу відгуків з вебсайту. Для прикладу використаємо бібліотеку *BeautifulSoup* для парсингу *HTML* та *requests* для взаємодії з вебсайтом (див. рис. 3.1). Також можна використовувати інші бібліотеки, які спрощують цей процес.

У цьому прикладі представлено базовий скрипт для парсингу відгуків із вебсайту, що має *HTML*-структуру із блоками відгуків. Важливо вказати правильну структуру вашого вебсайту для адаптації коду.

Для вирішення завдання кваліфікаційної роботи було взято відгуки про вебзастосунок. Вибір ґрунтувався на популярності програмного продукту, і відповідно, величезному числі відгуків для навчання. Але, очевидно, рішення задачі може бути застосовано і до відгуків на інші продукти.

Було отримано відгуки про вебзастосунок англійською мовою. Кожен відгук складається з заголовка, тіла відгуку, включає дату створення, ім’я користувача, оцінку по п’ятибальній шкалі.

Ключовою проблемою на даному етапі є відсутність розмічених даних, тобто неминучість ручної розмітки на певні категорії. Було проведено розмітку вибірки відгуків на звіти про помилки (негативні відгуки), запити функціональностей (позитивні відгуки) і неінформативні відгуки.

```
import requests
from bs4 import BeautifulSoup
import pandas as pd

def scrape_reviews_and_save_to_excel(url,
excel_filename='reviews_data.xlsx'):
    # Завантаження HTML-сторінки з відгуками
    response = requests.get(url)
    if response.status_code == 200:
        # Використання BeautifulSoup для парсингу HTML
        soup = BeautifulSoup(response.text, 'html.parser')

        # Знаходження елементів з відгуками
        reviews = soup.find_all('div', class_='review')

        # Створення списків для даних
        user_names = []
        dates = []
        review_texts = []

        # Обробка кожного відгуку
        for review in reviews:
            user_name = review.find('span', class_='user-name').text
            date = review.find('span', class_='review-date').text
            review_text = review.find('p', class_='review-text').text

            # Зберігання даних у списки
            user_names.append(user_name)
            dates.append(date)
            review_texts.append(review_text)
        # Створення DataFrame з даних
        df = pd.DataFrame({'User': user_names, 'Date': dates,
'Review': review_texts})
        # Збереження даних у файл Excel
        df.to_excel(excel_filename, index=False)
        print(f"Data saved to {excel_filename}")
    else:
        print(f"Failed to retrieve the page. Status code:
{response.status_code}")
# Приклад виклику функції з URL вебсайту з відгуками
url_with_reviews = 'https://example.com/reviews'
scrape_reviews_and_save_to_excel(url_with_reviews)
```

Рисунок 3.1 – Базовий скрипт для парсингу відгуків із вебсайту

Дані відгуки були обрані випадково на всій множині отриманих відгуків і є репрезентативними для всієї вибірки. Кожен із відгуків міг бути віднесений до однієї чи декількох категорій.

Нижче наведемо приклади відгуків.

“After iPad replacement using restore it can’t login or failure time out. Just an ever spinning wheel of uselessness. Multiple reboots, deletes and reinstalls are no help”(звіт про помилки).

“Everything works pretty good не реальні проблеми just wish I was able to see my eBay bucks made per item. And total” (запит функціональності).

“Awesome App! On here all the time doing lots of shopping! Never had one issue, well besides a couple of items not showing up” (неінформативний відгук).

Особливістю відгуків на програмний продукт є середня довжина відгуку (як правило, 3-5 речень), часті помилки в словах та пунктуації, зашумленість тексту знаками пунктуації, що становлять смайли, велика частка відгуків з яскраво вираженим позитивним/негативним емоційним забарвленням.

Після отримання відгуків проводимо початкову передобробку даних.

Залишаємо відгуки, написані тільки на англійській мові. В вибірку потрапили відгуки, написані також і українською мовою, попри те, що країна в відгуку вказана як США.

Трансформуємо усі літери до нижнього регістру. Як правило, користувачі, що пишуть відгуки з мобільних пристроїв не дбають про правильний регістр і дана інформація може бути врахована, але лише після ретельного аналізу.

3.2 Застосування розробленого модуля

Опис лістинга алгоритмів машинного навчання для класифікації відгуків на програмний продукт.

У даному розділі розглядається машинний скрипт, де показано етапи написання класифікаторів. Основні етапи включають:

– завантаження всіх необхідних бібліотек;
– збір відгуків;
– попередня обробка;

– виведення на екран таблиці відгуків за допомогою бібліотеки *pandas* датафрейм та підрахунок кількості відгуків;

– навчання класифікатора на навчальній вибірці;

– застосувати класифікатори (метод опорних векторів (*svm*), логістична регресія, випадковий ліс);

– оцінити точність (*accuracy*) та повноту (*recall*) класифікаторів, провести порівняння.

Маємо вибірку з відгуків. Нам відоме емоційне забарвлення кожного відгуку з вибірки: позитивне чи негативне. Завдання полягає у побудові моделі, яка за текстом відгуку передбачає його емоційне забарвлення.

Імпортуємо бібліотеки (див. рис. 3.2).

```
import pandas as pd # бібліотека для зручної роботи з датафреймами
import numpy as np # бібліотека для зручної роботи зі списками та
матрицями

# бібліотека, де реалізовано основні алгоритми машинного навчання
from sklearn.metrics import *
from sklearn.model_selection import train_test_split
from sklearn.pipeline import Pipeline
```

Рисунок 3.2 – Імпорт бібліотек *Pandas*, *NumPy*, *Sklearn-learn*

Відкриємо файли та створимо масив із текстів та правильних міток для відгуків.

Спочатку йдуть позитивні відгуки, далі негативні (див. рис. 3.3).

Розбиваємо дані на навчальну та тестову вибірки за допомогою функції `train_test_split()` з *sklearn* (див. рис. 3.4).

Векторизатори. Векторизатор перетворює слово чи набір слів у числовий вектор, зрозумілий алгоритму машинного навчання, який звик працювати з числовими табличними даними. Нижче – наведено приклад перетворення слів у двомірний вектор, кожному слову відповідає точка на площині (див. рис. 3.5).

```
from google.colab import drive
drive.mount('/content/gdrive')

# завантажуємо позитивні відгуки
positive = pd.read_csv('/content/gdrive/MyDrive/Colab
Notebooks/data/positive.csv', sep=';', usecols=[3], names=['text'])
positive['label'] = ['positive'] * len(positive) # встановлюємо мітки

# завантажуємо негативні відгуки
negative = pd.read_csv('/content/gdrive/MyDrive/Colab
Notebooks/data/negative.csv', sep=';', usecols=[3], names=['text'])
negative['label'] = ['negative'] * len(negative) # встановлюємо мітки
# з'єднуємо разом
df = positive.append(negative)
# подивимось отримані дані
df.sample(5, random_state=40)
```

Рисунок 3.3 – Завантаження відгуків

```
x_train, x_test, y_train, y_test = train_test_split(df.text,
df.label)
```

Рисунок 3.4 – Розбиття вибірки на навчальну та тестову

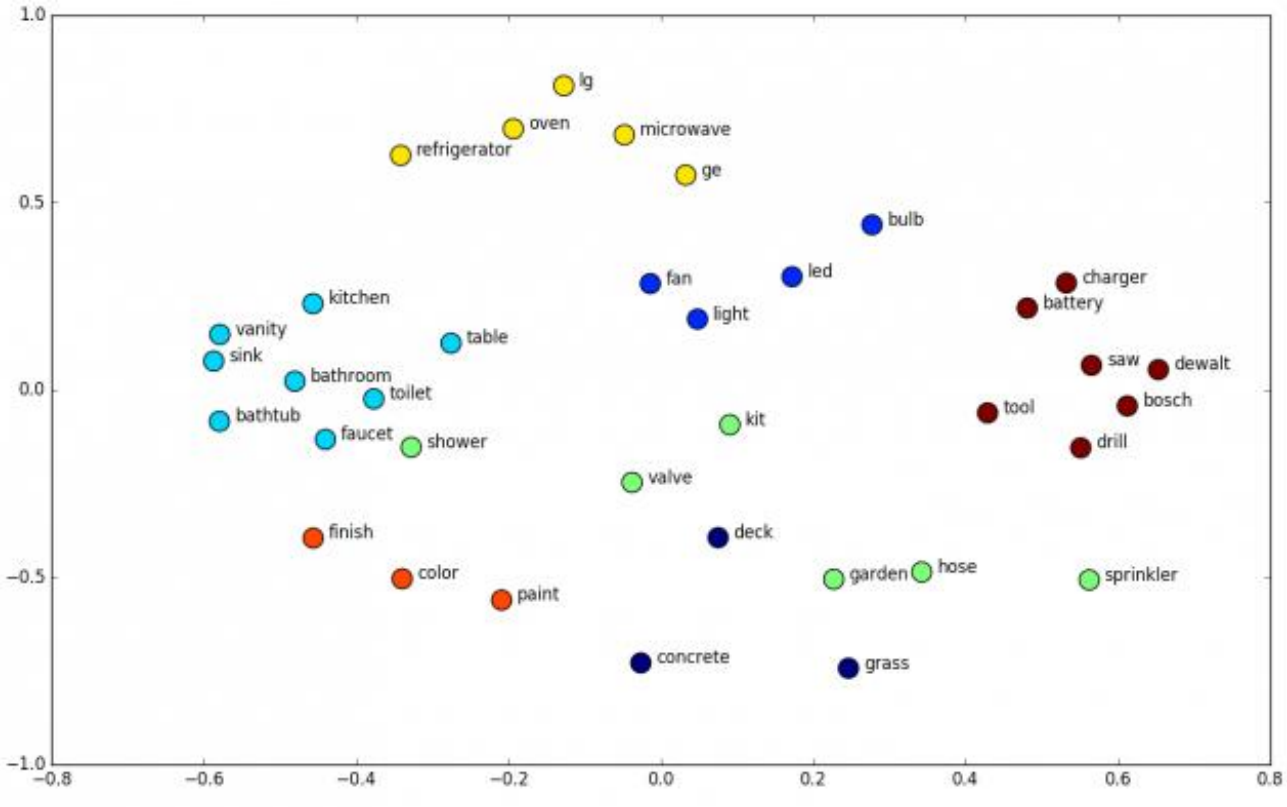


Рисунок 3.5 – Приклад перетворення слів у двомірний вектор

На початковому етапі буде достатньо тих інструментів, які вже є в бібліотеці *sklearn-lear*n (див. рис. 3.6).

```
from sklearn.linear_model import LogisticRegression # можна замінити
на [..]-> провести порівняння точності моделей
from sklearn.feature_extraction.text import CountVectorizer # модель
"мішка слів"
```

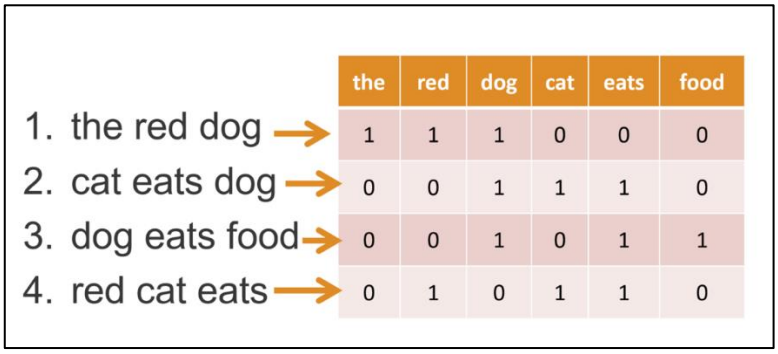
Рисунок 3.6 – Завантаження бібліотеки *scikit-lear*n для використання логістичної регресії та моделі «мішка слів»

Найпростіший спосіб отримати ознаки з текстових даних – векторизатори: `CountVectorizer` і `TfidfVectorizer`.

Об’єкт `CountVectorizer` робить наступне (див. рис. 3.7):

– будує для кожного документа (кожного рядка, що прийшов йому) вектор розмірності *n*, де *n* – кількість слів або *n*-грам у всьому корпусі;

– заповнює кожен і-тий елемент кількістю входження слова в даний документ.



	the	red	dog	cat	eats	food
1. the red dog →	1	1	1	0	0	0
2. cat eats dog →	0	0	1	1	1	0
3. dog eats food →	0	0	1	0	1	1
4. red cat eats →	0	1	0	1	1	0

Рисунок 3.7 – Приклад векторизації

Ініціалізуємо `CountVectorizer()`, вказавши як ознаки уніграми: `vectorizer = CountVectorizer()`.

Після ініціалізації *vectorizer* можна навчити нашими даними.

Для навчання використовуємо навчальну вибірку `x_train`, але на відміну від класифікатора використовуємо метод `fit_transform()`: спочатку навчаємо векторизатор, а потім відразу застосовуємо його до набору даних:

`vectorized_x_train = vectorizer.fit_transform(x_train).`

В `vectorizer.vocabulary_` лежить словник, перетворення слів у їх індекси (див. рис. 3.8).

```
list(vectorizer.vocabulary_.items())[:10]
```

Рисунок 3.8 – Отримання списку перших 10 елементів словника

У нашій вибірці 170125 текстів (звітів), в них зустрічається 243591 різних слова.

Оскільки тепер маємо чисельне уявлення і набір вхідних ознак, то можемо навчити модель логістичної регресії (або будь-яку іншу з наведених у другому розділі моделей) (див. рис. 3.9).

```
clf = LogisticRegression(random_state=42, max_iter=1000, C=1) #
фіксуємо random_state для відтворюваності результатів
clf.fit(vectorized_x_train, y_train)
```

Рисунок 3.9 – Використання моделі логістичної регресії з бібліотеки *scikit-learn* для навчання на вхідних даних

Бібліотека *scikit-learn* має реалізації класифікаторів для методу *k*-найближчих сусідів, випадкового лісу та *XGBoost*. Імпортувати ці класифікатори можна:

- `from sklearn.neighbors import KNeighborsClassifier;`
- `from sklearn.ensemble import RandomForestClassifier;`
- `from xgboost import XGBClassifier.`

З тестовими даними потрібно зробити те саме, що і з даними для навчання: зробити з текстів вектора, які можна передавати в класифікатор для прогнозу класу об’єкта.

У нас вже є навчений векторизатор *vectorizer*, тому використовуємо метод «`transform()`» (просто застосуємо його), а не `fit_transform` (навчити і застосувати) (див. рис. 3.10).

```
vectorized_x_test = vectorizer.transform(x_test)
clf.score(vectorized_x_test,y_test)
```

Рисунок 3.10 – Використання попередньо навченої модель логістичної регресії для оцінки (визначення точності) на тестових даних

Як раніше, для отримання прогнозу у навченого класифікатора використовуємо метод `predict()`.

За допомогою функції `classification_report()`, яка вважає відразу кілька метрик якості класифікації, подивимося на те, наскільки добре пророкуємо позитивну або негативну тональність відгуку (див. рис. 3.11).

```
pred = clf.predict(vectorized_x_test)
print(classification_report(y_test, pred))
```

Рисунок 3.11 – Використання навченої моделі логістичної регресії для здійснення передбачень

Біграми. Спробуємо зробити те ж саме, додавши як ознак біграми (див. рис. 3.12).

```
# ініціалізуємо векторайзер
trigram_vectorizer = CountVectorizer(ngram_range=(1, 2))
# навчаємо його і відразу застосовуємо до x_train
trigram_vectorized_x_train =
trigram_vectorizer.fit_transform(x_train)
# ініціалізуємо та навчаємо класифікатор
clf = LogisticRegression(random_state=42)
clf.fit(trigram_vectorized_x_train, y_train)
trigram_vectorized_x_test = trigram_vectorizer.transform(x_test)
pred = clf.predict(trigram_vectorized_x_test)
print(classification_report(y_test, pred))
```

Рисунок 3.12 – Використання метода «мішка слів» (n-грами) для векторизації текстових даних та навчає модель логістичної регресії на отриманих векторах

TF-IDF векторизація.

`TfidfVectorizer` робить те саме, що і `CountVectorizer`, але як значення видає *tf-idf* кожного слова. Розрахунок *TF-IDF* наведено у другому розділі.

Якщо слово часто зустрічається в одному документі, але загалом по корпусу зустрічається в невеликій кількість документів, у нього високий TF-IDF.

Діємо аналогічно, як з CountVectorizer() (див. рис. 3.13).

```
# ініціалізуємо векторизатор, як змінні використовуємо уніграми та
біграми
tfidf_vectorizer = TfidfVectorizer(ngram_range=(1, 2))
# навчаємо його і відразу застосовуємо до x_train
tfidf_vectorized_x_train = tfidf_vectorizer.fit_transform(x_train)
# ініціалізуємо та навчаємо класифікатор
clf = LogisticRegression(random_state=42)
clf.fit(tfidf_vectorized_x_train, y_train)
# застосовуємо навчений векторизатор до тестових даних
tfidf_vectorized_x_test = tfidf_vectorizer.transform(x_test)
# отримуємо передбачення та виводимо інформацію про якість
pred = clf.predict(tfidf_vectorized_x_test)
print(classification_report(y_test, pred))
```

Рисунок 3.13 – Використання методу TF-IDF для векторизації текстових даних та навчання моделі логістичної регресії на отриманих векторах

Стоп-слова та пунктуація. Стоп-слова – це слова, які часто зустрічаються практично в будь-якому тексті і нічого цікавого не говорять про конретний документ. Для моделі це просто шум. А шум треба прибирати. З тієї ж причини прибирають і пунктуацію (див. рис. 3.14).

```
# Імпортуємо бібліотеку nltk та завантажуємо необхідні дані
import nltk
nltk.download('punkt')
from nltk.corpus import stopwords
nltk.download('stopwords')
# Отримуємо англійські стоп-слова
english_stopwords = stopwords.words("english")
# Виводимо англійські стоп-слова
print(english_stopwords)
```

Рисунок 3.14 – Імпорт стоп-слів для англійської мови з бібліотеки *NLTK*

Знаки пунктуації краще імпортувати з модуля *String*. У ньому зберігаються різні набори констант для роботи з рядками (пунктуація, алфавіт та інші).

Об’єднаємо стоп-слова та знаки пунктуації разом та запишемо в змінну

noise: *noise = stopwords.words('english') + list(punctuation).*

Тепер потрібно навчати модель з урахуванням нових знань про токенізацію та стоп-слова.

Для цього можемо зібрати новий векторизатор, передавши йому на вхід:

– які *n*-грами потрібні, параметр `ngram_range`;

– який токенізатор використовуємо, параметр tokenizer;

– які стоп-слова, параметр `stop_words`.

Використовуємо готовий токенізатор `word_tokenize`, а стоп-слова зберігаються в змінній `noise` (див. рис. 3.15).

```
# ініціалізуємо розумний векторайзер
smart_vectorizer = CountVectorizer(ngram_range=(1, 2),
                                   stop_words=noise)
# навчаємо його і відразу застосовуємо до x_train
smart_vectorized_x_train = smart_vectorizer.fit_transform(x_train)
# ініціалізуємо та навчаємо класифікатор
clf = LogisticRegression(random_state=42)
clf.fit(smart_vectorized_x_train, y_train)
# застосовуємо навчений векторайзер до тестових даних
smart_vectorized_x_test = smart_vectorizer.transform(x_test)
# отримуємо передбачення та виводимо інформацію про якість
pred = clf.predict(smart_vectorized_x_test)
print(classification_report(y_test, pred))
```

Рисунок 3.15 – Використання розумного векторайзера

Іноді пунктуація буває не шумом – головне відштовхуватися від завдання. Перевіримо, що буде, якщо взагалі не прибирати пунктуацію (див. рис. 3.16)?

Варто залишити пунктуацію – і всі метрики дорівнюють 1. Як це вийшло? Серед неї були дуже значні токени.

Подивимося, як один із супер-значних токенів впорається з класифікацією без жодного машинного навчання (див. рис. 3.17).

Символьні n-грами. Тепер як фічі використовуємо, наприклад, уніграми символів. Для цього необхідно встановити `CountVectorizer()` параметр `analyzer=char`, тобто аналізувати символи (див. рис. 3.18).

```
#підключаємо nltk и word_tokenize
import nltk
nltk.download('punkt')
#df['message'] = df['message'].apply(nltk.word_tokenize)
# ініціалізуємо розумний векторайзер stop-words НЕ ВИКОРИСТОВУЄМО!
alternative_tfidf_vectorizer =
TfidfVectorizer(tokenizer=nltk.word_tokenize)
# навчаємо його і відразу застосовуємо до x_train
alternative_tfidf_vectorized_x_train =
alternative_tfidf_vectorizer.fit_transform(x_train)
# ініціалізуємо та навчаємо класифікатор
clf = LogisticRegression(random_state=42)
clf.fit(alternative_tfidf_vectorized_x_train, y_train)
# застосовуємо навчений векторайзер до тестових даних
alternative_tfidf_vectorized_x_test =
alternative_tfidf_vectorizer.transform(x_test)
# отримуємо передбачення та виводимо інформацію про якість
pred = clf.predict(alternative_tfidf_vectorized_x_test)
print(classification_report(y_test, pred))
```

Рисунок 3.16 – Використання розумного векторайзер бібліотеки *NLTK* для токенізації слів та обчислення *TF-IDF*

```
cool_token = ')'
pred = ['positive' if cool_token in tweet else 'negative' for tweet
in x_test]
print(classification_report(pred, y_test))
```

Рисунок 3.17 –Класифікація за «супер-значним» токеном

```
# ініціалізуємо векторайзер для символів
char_vectorizer = CountVectorizer(analyzer='char')

# навчаємо його і відразу застосовуємо до x_train
char_vectorized_x_train = char_vectorizer.fit_transform(x_train)

# ініціалізуємо та навчаємо класифікатор
clf = LogisticRegression(random_state=42)
clf.fit(char_vectorized_x_train, y_train)

# застосовуємо навчений векторайзер до тестових даних
char_vectorized_x_test = char_vectorizer.transform(x_test)

# отримуємо передбачення та виводимо інформацію про якість
pred = clf.predict(char_vectorized_x_test)
print(classification_report(y_test, pred))
```

Рисунок 3.18 – Використання уніграм символів

3.3 Визначення якості класифікатора

Для визначення якості класифікатора використовувалися згадані вище метрики. Для знаходження більше об’єктивний оцінки якості класифікатора використовувалась перехресна перевірка *(cross-validation*) на 3 частинах. Для цього дані розбивалися на *k=3* частин (причому так, щоб співвідношення різних класів у кожній частині було приблизно таким самим, що і на всієї вибірки). Потім, на *k-1* частинах даних проводилося навчання моделі, а решта частина даних використовувалася для тестування і визначення значення метрики. Дана процедура повторювалась *k* раз, і значення метрики для всієї вибірки знаходилося як середнє арифметичне значень на різних її частинах. В практичних експериментах були використані такі класифікатори: логістична регресія, метод *k*-найближчих сусідів, випадковий ліс та *XGBoost*. Отримані результати відображені в таблиці 3.1.

Таблиця 3.1 – Порівняння класифікаторів

	Precision	Recall	f1-score	Support
LogisticRegression				
negative	0.76	0.76	0.76	27975
positive	0.77	0.76	0.77	28734
accuracy			0.76	56709
RandomForest				
negative	0.66	0.82	0.73	27896
positive	0.77	0.59	0.67	28813
accuracy			0.70	56709
KNeighbors				
negative	0.52	0.98	0.68	27892
positive	0.85	0.11	0.20	28817
accuracy			0.54	56709
XGBClassifier				
negative	0.66	0.82	0.73	27896
positive	0.77	0.59	0.67	28813
accuracy			0.70	56709

Більш детальна інформація про поведінку класифікаторів даних представлена в додатку А.

Як видно з таблиці 3.1, загалом, якість класифікації запитів функціональності продукту (позитивних відгуків) нижче, ніж звітів про помилки (негативних звітів). Це відбувається в здебільшого через малу кількість запитів функціональностей у вихідних даних. Інша причина такої поведінки може бути пов'язана із специфічністю запитів функціональностей продукту, які часто представляють собою щось середня між звітами про помилки та неінформативними відгуками, що відображають у загалом, рейтинг товару. Наприклад:

"Everything works pretty good no real problems just wish I was able to see my eBay bucks made per item. And total." Даний відгук слід віднести до запиту функціональності, тим не менше, лише «*just wish*» вказує нам на це.

"PRICES IN THE ADVERTISING is so annoying and wrong. Every time I see, for example, C$2.98 and US $2.25, I know that I'm supposed to pay theu.s. Price but I keep getting charged the C price. I know I need to probably take this up with PayPal, however, I just want to see the U.S. Prices first and just get charged that so I don't have to deal with Paypal on that end. PLEASE CHANGE THIS!" Тільки частина цього відгуку відноситься до запиту функціональності продукту.

Також, цікаво, що при всім наборі настроюваних параметрів, методи *k*-найближчих сусідів, випадковий ліс та *XGBoost* не змогли виявитися кращим, ніж логістична регресія.

Використання *n*-грам замість слів у відгуках покращує класифікацію. Пов'язана ця обставина з великою кількістю помилок у відгуках, а використання *n*-грам дозволяє врахувати помилки у відгуках, рахуючи тільки частини слів.

ВИСНОВКИ

Магістерська кваліфікаційна робота представляє собою глибокий аналіз використання технологій машинного навчання у веброзробці, а саме розробку системи для обробки та класифікації великого обсягу відгуків на вебзастосунки за їхнім характером.

Було детально вивчено сучасні дослідження в галузі машинного навчання та аналізу відгуків. Визначено ключові тенденції та проблеми, що виникають при роботі з великим обсягом текстових даних. Наведені підходи до обробки природної мови (*NLP*) та класифікації текстів, вказуючи на важливість використання методів машинного навчання для покращення результатів.

Встановлено, що застосування машинного навчання для класифікації відгуків:

– дозволяє автоматизувати аналіз та розподіл користувацьких відгуків на позитивні та негативні;

– дозволяє веброзробникам швидше реагувати на критичні питання та вдосконалювати продукт;

– допомогає розробникам швидше визначати проблеми та потреби користувачів, дозволяючи оперативно впроваджувати виправлення та нові функції.

Визначено, що машинне навчання є ключовим методом для обробки та аналізу великого обсягу даних, таких як відгуки на на вебзастосунки, мобільні застосунки. Підкреслено важливість вибору оптимального алгоритму для досягнення бажаних результатів.

Проведений аналіз вказує на значущість обробки природної мови для успішної класифікації відгуків. Висвітлено важливість використання методів токенізації та лематизації для зменшення розміру словника та покращення якості моделі.

Описано основні принципи та завдання класифікації в текстовому аналізі.

Виділено важливість правильного визначення категорій та розробки ефективних моделей для класифікації відгуків.

Проведений огляд основних класифікаторів на основі машинного навчання, таких як логістична регресія, метод *k*-найближчих сусідів, випадковий ліс та *XGBoost*. Зазначено їхні переваги та області застосування в контексті класифікації відгуків на вебзастосунки.

Обрано мову програмування *Python* та середовище розробки *Colab* для реалізації системи. Зазначено важливість цих виборів для забезпечення зручності та ефективності процесу розробки та тестування моделі.

Наведено використання метрик оцінки якості моделі. Підкреслено необхідність вибору відповідних метрик, залежно від конкретної задачі класифікації відгуків.

Описано процес збору великого обсягу відгуків на вебзастосунки для подальшого використання в навчанні моделі. Зазначено важливість створення репрезентативного та збалансованого датасету для досягнення високої точності класифікації.

Проведено розгортання розробленого модуля для класифікації відгуків на позитивні та негативні. Зазначено успішні результати та переваги використання розробленої системи для автоматизованого аналізу великого обсягу текстових даних.

Виконано оцінку якості розробленого класифікатора за допомогою визначених метрик. Зазначено досягнуті результати та здатність моделі ефективно відрізняти позитивні та негативні відгуки.

Магістерська робота є важливим кроком у напрямку розробки та впровадження системи класифікації відгуків з використанням методів машинного навчання та обробки природної мови. Вибір оптимальних технологій, алгоритмів та ефективних метрик визначає успіх розробленої системи. Результати роботи дозволяють ефективно класифікувати великий обсяг відгуків, що є актуальним завданням у сучасному світі веброзробки.

ПЕРЕЛІК ПОСИЛАНЬ

1. Maalej W., Kuztanovic Z., Nabil H. On the automatic classification of app reviews. *Requirement Engineering*. Vol. 21. P. 311–331. URL: https://doi.org/10.1007/s00766-016-0251-9 (дата звернення: 10.09.2023).

2. Li H., Zhang L., Zhang L., Shen J. A user satisfaction analysis approach for software evolution. *Informatics and Computing (PIC)* : IEEE International Conference, 2010. Vol. 2. P. 1093–1097. URL: https://doi.org/10.1109/PIC.2010.5687999 (дата звернення: 10.09.2023).

3. Carreño L. V. G., Winbladh K. Analysis of user comments: an approach for software requirements evolution. *35th International Conference on Software Engineering*. IEEE Press, 2013. P. 582–591. URL: https://doi.org/10.1109/ICSE.2013.6606604 (дата звернення: 10.09.2023).

4. Pagano D., Maalej W. User feedback in the appstore: an empirical study. *International Conference on Requirements Engineering*. RE'13, 2013. P. 125–134. URL: https://doi.org/10.1109/RE.2013.6636712 (дата звернення: 10.09.2023).

5. Hoon L., Vasa R., Schneider J.-G., Grundy J. An analysis of the mobile app review landscape: trends and implications. Technical report, Swinburne University of Technology, 2013. 27 p. URL: https://bit.ly/47kYtL4 (дата звернення: 10.09.2023).

6. Maalej W., Nabil H. Bug report, feature request, or simply praise? On automatically classifying app reviews. *23rd International Requirements Engineering Conference (RE)*. IEEE, 2015. P. 116–125. URL: https://doi.org/10.1109/RE.2015.7320414 (дата звернення: 10.09.2023).

7. Jalbert N., Weimer W. Automated duplicate detection for bug tracking systems. *International Conference on Dependable Systems and Networks With FTCS and DCC (DSN), IEEE*. P. 52–61. URL: https://doi.org/10.1109/DSN.2008.4630070 (дата звернення: 10.09.2023).

8. Alipour A., Hindle A., Stroulia E. A contextual approach towards more accurate duplicate bug report detection. *10th Working Conference on Mining Software*

Repositories (MSR), 2013. P. 183–192. URL: https://doi.org/10.1109/MSR.2013.6624026 (дата звернення: 10.09.2023).

9. Joachims T. Text categorization with Support Vector Machines: Learning with many relevant features. *European Conference on Machine Learning*. London : Springer, 1998. P. 137–142. URL: https://doi.org/10.1007/BFb0026683 (дата звернення: 10.09.2023).

10. Cavnar W. B., Trenkle J. M. *N-Gram-Based Text Categorization*. 3rd Annual Symposium on Document Analysis and Information Retrieval. Las Vegas : NV, 1994. P. 161–175. URL: https://bit.ly/3QSUekv (дата звернення: 10.09.2023).

11. Rocha H, Oliveira G, Maques-Neto H. NextBug: A Tool for Recommending Similar Bugs in Open-Source Systems. *Brazilian Conference on Software: Theory and Practice – Tools Track*. Maceio : CBSoft Tools, 2014. P. 53–60. URL: https://bit.ly/47bllge (дата звернення: 10.09.2023).

12. Wang X., Zhang L., Xie T., Anvik J. An approach to detecting duplicate bug reports using natural language and execution information. *30th International Conference on Software Engineering*, Leipzig: ACM/IEEE, 2008. P. 461–470. URL: https://doi.org/10.1145/1368088.1368151 (дата звернення: 10.09.2023).

13. Anvik J., Hiew L., Murphy G. Who should fix this bug? *28th International Conference on Software Engineering*. (20-28 May 2006). Shanghai, 2006. P. 361–370. URL: https://doi.org/10.1145/1134285.1134336 (дата звернення: 10.09.2023).

14. Mishra N., Schreiber R., Stanton I. Clustering Social Networks. Workshop on *Algorithms and Models for the Web-Graph*. Springer. 2007. P. 56–67. URL: https://doi.org/10.1007/978-3-540-77004-6_5 (дата звернення: 10.09.2023).

15. Sun C., Lo D., Khoo S. Towards more accurate retrieval of duplicate bug reports. *26th International Conference on Automated Software Engineering*. 2011. P. 253–262. URL: https://doi.org/10.1109/ASE.2011.6100061 (дата звернення: 10.09.2023).

16. Tian Y., Sun C., Lo D. Improved Duplicate Bug Report Identification. *16th European Conference on Software Maintenance and Reengineering*, 2012. P. 385–390. URL: https://doi.org/10.1109/CSMR.2012.48 (дата звернення: 10.09.2023).

17. Rijsbergen C. J., Robertson S. E. New models in probabilistic information retrieval. London : British Library. 1980. 123 p.

18. Mikolov T., Sutskever I., Chen K. Distributed representations of words and phrases and their compositionality. *27th Annual Conference on Neural Information Processing Systems*. (5-8 December). Nevada. 2013. P. 3111–3119. URL: https://doi.org/10.48550/arXiv.1310.4546 (дата звернення: 10.09.2023).

19. Mikolov T., Chen K., Corrado G., Dean J. *Efficient Estimation of Word Representations in Vector Space* 2013. URL: https://doi.org/10.48550/arXiv.1301.3781 (дата звернення: 10.09.2023).

20. Morin F., Bengio Y. Hierarchical probabilistic neural network language model. *The Tenth International Workshop on Artificial Intelligence and Statistics*. 2005. Vol. 5. P. 246–252. URL: https://bit.ly/3FUwiH1 (дата звернення: 10.09.2023).

21. Bengfort B., Bilbro R., Ojeda T. Applied Text Analysis with Python: Enabling Language Aware Data Products with Machine Learning. O'Reilly, 2018. 330 p.

22. Grus J. Data Science from Scratch: First Principles with Python. O'Reilly, 2019. 409 p.

23. Mason H. The Next Generation of Data Products URL: https://u.to/Lk8qIA (дата звернення: 10.09.2023).

24. Loukides M. What is data science? URL: https://www.oreilly.com/radar/what-is-data-science/ (дата звернення: 10.09.2023).

25. Bengfort B. The Age of the Data Product. URL: https://www.districtdatalabs.com/the-age-of-the-data-product (дата звернення: 10.09.2023).

26. Cielen D., Meysman A.; Ali M. Introducing Data Science: Big Data, Machine Learning, and more, using Python tools. Manning, 2016. 320 p.

27. Букія Г. Т. Аналіз тональності. *Прикладна та комп'ютерна лінгвістика*. 2017. №3. С. 123–139.

28. Kumar A., McCann R., Naughton J., Patel J. Model Selection Management Systems: The Next Frontier of Advanced Analytics. *ACM SIGMOD Record*. 2015. Vol.

44., №4. P. 17–22. URL: https://doi.org/10.1145/2935694.2935698 (дата звернення: 10.09.2023).

29. Wickham H., Cook D., Hofmann H. Visualizing statistical models: Removing the blindfold. *Statistical Analysis and Data Mining: The ASA Data Science Journal*. 2015. №8(4). P. 203–225. URL: https://doi.org/10.1002/sam.11271 (дата звернення: 10.09.2023).

30. Raschka S. Python Machine Learning: Machine and Deep Learning with Python, Scikit-learn and TensorFlow. Packt Publishing, 2019. 772 p.

31. Sarkar D. Text Analytics with Python. A Practitioner's Guide to Natural Language Processing. NY: Apress Media, 2019. 674 p.

32. Conor Mc. Machine learning fundamentals (I): Cost functions and gradient descent. URL: https://bit.ly/3SBqhqe (дата звернення: 10.09.2023).

33. Géron A. Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow. O'Reilly Media. 856 p.

34. Mesevage T. Machine Learning Classifiers. URL: https://bit.ly/47bB5Qj (дата звернення: 10.09.2023).

35. Brownlee J. Parametric and Nonparametric Machine Learning Algorithms. URL: https://bit.ly/49BL03h (дата звернення: 10.09.2023).

36. Sam T. Entropy: How Decision Trees Make Decisions. URL: https://bit.ly/3uaus2e (дата звернення: 10.09.2023).

37. Kashnitsky Y. Bagging and Random Forest. URL: https://bit.ly/46b8dGK (дата звернення: 10.09.2023).

38. Aliyev V. Machine Learning Algorithms from Start to Finish in Python: Logistic Regression. URL: https://bit.ly/3MH1QnP (дата звернення: 10.09.2023).

39. R Tutorial. URL: https://www.w3schools.com/r/ (дата звернення: 10.09.2023).

40. Java Introduction https://www.w3schools.com/java/java_intro.asp (дата звернення: 10.09.2023).

41. Python Developer's Guide URL: https://devguide.python.org/ (дата звернення: 10.09.2023).

42. The Most Popular IDEs for Developers in 2023. URL: https://bit.ly/3QEnHgM (дата звернення: 10.09.2023).

ДОДАТОК А

Оцінки якості моделей машинного навчання

Таблиця А.1 – Оцінки якості методу логістичної регресі

	Precision	Recall	f1-score	Support
LogisticRegression				
negative	0.76	0.76	0.76	27975
positive	0.77	0.76	0.77	28734
accuracy			0.76	56709
macro avg	0.76	0.76	0.76	56709
weighted avg	0.76	0.76	0.76	56709
negative	0.76	0.76	0.76	27975
LogisticRegression+bigrams				
negative	0.78	0.77	0.78	27975
positive	0.78	0.78	0.78	27975
accuracy			0.78	56709
macro avg	0.78	0.78	0.78	56709
weighted avg	0.78	0.78	0.78	56709
LogisticRegression+TF-IDF				
negative	0.77	0.75	0.76	27975
positive	0.76	0.78	0.77	28734
accuracy			0.76	56709
macro avg	0.76	0.76	0.76	56709
weighted avg	0.76	0.76	0.76	56709
LogisticRegression+stop-words+punctuation				
negative	0.76	0.76	0.76	27975
positive	0.77	0.77	0.77	28734
accuracy			0.76	56709
macro avg	0.76	0.76	0.76	56709
weighted avg	0.76	0.76	0.76	56709
LogisticRegression+n-грами				
negative	1.00	0.99	0.99	27975
positive	0.99	1.00	0.99	28734
accuracy			0.99	56709
macro avg	0.99	0.99	0.99	56709
weighted avg	0.99	0.99	0.99	56709

Таблиця А.2 – Оцінки якості методу *k*-найближчих сусідів

	Precision	Recall	f1-score	Support
LogisticRegression				
negative	0.66	0.82	0.73	27896
positive	0.77	0.59	0.67	28813
accuracy			0.70	56709
macro avg	0.71	0.70	0.70	56709
weighted avg	0.71	0.70	0.70	56709
negative	0.66	0.82	0.73	27896
LogisticRegression+bigrams				
negative	0.67	0.80	0.73	27896
positive	0.76	0.61	0.68	28813
accuracy			0.70	56709
macro avg	0.71	0.78	0.70	56709
weighted avg	0.71	0.78	0.70	56709
LogisticRegression+TF-IDF				
negative	0.59	0.91	0.71	27953
positive	0.81	0.38	0.52	28756
accuracy			0.64	56709
macro avg	0.70	0.76	0.62	56709
weighted avg	0.70	0.76	0.62	56709
LogisticRegression+stop-words+punctuation				
negative	0.64	0.83	0.72	27758
positive	0.77	0.56	0.65	28951
accuracy			0.69	56709
macro avg	0.71	0.76	0.69	56709
weighted avg	0.71	0.76	0.69	56709
LogisticRegression+n-грами				
negative	1.00	0.99	0.99	27758
positive	0.99	1.00	0.99	28951
accuracy			0.99	56709
macro avg	0.99	0.99	0.99	56709
weighted avg	0.99	0.99	0.99	56709

Таблиця А.3 – Оцінки якості методу випадкового лісу

	Precision	Recall	f1-score	Support
LogisticRegression				
negative	0.52	0.98	0.68	27892
positive	0.85	0.11	0.20	28817
accuracy			0.54	56709
macro avg	0.69	0.55	0.44	56709

Продовження табл. А.3

	Precision	Recall	f1-score	Support
LogisticRegression				
weighted avg	0.69	0.54	0.43	56709
negative	0.52	0.98	0.68	27892
LogisticRegression+bigrams				
negative	0.50	0.99	0.67	27975
positive	0.85	0.05	0.10	27975
accuracy			0.51	56709
macro avg	0.68	0.78	0.39	56709
weighted avg	0.68	0.78	0.38	56709
LogisticRegression+TF-IDF				
negative	0.77	0.75	0.76	27975
positive	0.76	0.78	0.77	28734
accuracy			0.76	56709
macro avg	0.76	0.76	0.76	56709
weighted avg	0.76	0.76	0.76	56709
LogisticRegression+stop-words+punctuation				
negative	0.76	0.76	0.76	27975
positive	0.77	0.77	0.77	28734
accuracy			0.76	56709
macro avg	0.76	0.76	0.76	56709
weighted avg	0.76	0.76	0.76	56709
LogisticRegression+n-грами				
negative	1.00	0.99	0.99	27975
positive	0.99	1.00	0.99	28734
accuracy			0.99	56709
macro avg	0.99	0.99	0.99	56709
weighted avg	0.99	0.99	0.99	56709

Таблиця А.4 – Оцінки якості методу *XGBoost*

	Precision	Recall	f1-score	Support
LogisticRegression				
negative	0.66	0.82	0.73	27896
positive	0.77	0.59	0.67	28813
accuracy			0.70	56709
macro avg	0.71	0.70	0.70	56709
weighted avg	0.71	0.70	0.70	56709
negative	0.66	0.82	0.73	27896
LogisticRegression+bigrams				
negative	0.73	0.63	0.67	27959

Продовження табл. А.4

	Precision	Recall	f1-score	Support
LogisticRegression+bigrams				
positive	0.68	0.77	0.72	28750
accuracy			0.70	56709
macro avg	0.70	0.70	0.70	56709
weighted avg	0.70	0.70	0.70	56709
LogisticRegression+TF-IDF				
negative	0.77	0.75	0.76	27975
positive	0.76	0.78	0.77	28734
accuracy			0.76	56709
macro avg	0.76	0.76	0.76	56709
weighted avg	0.76	0.76	0.76	56709
LogisticRegression+stop-words+punctuation				
negative	1.00	1.00	1.00	27959
positive	1.00	1.00	1.00	28750
accuracy			1.00	56709
macro avg	1.00	1.00	1.00	56709
weighted avg	1.00	1.00	1.00	56709
LogisticRegression+n-грами				
negative	1.00	1.00	1.00	27975
positive	1.00	1.00	1.00	28734
accuracy			1.00	56709
macro avg	1.00	1.00	1.00	56709
weighted avg	1.00	1.00	1.00	56709