

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи освітнього ступеня «магістр»
за спеціальністю 121 «Інженерія програмного забезпечення»
(освітня програма «Інженерія програмного забезпечення»)

на тему:

**«Розробка інформаційної системи для класифікації
резюме кандидатів на основі NLP»**

Виконав студент групи ПЗм-23-2
ЛАВРЕНЧУК Віталій Олександрович

Керівник роботи:
МОРОЗОВ Андрій Васильович

Рецензент
ЧИЖМОТРЯ Олексій Володимирович

Житомир – 2024

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

«ЗАТВЕРДЖУЮ»

Зав. кафедри інженерії

програмного забезпечення

Тетяна ВАКАЛЮК

«01» жовтня 2024 р.

ЗАВДАННЯ
на кваліфікаційну роботу

Здобувач вищої освіти: **ЛАВРЕНЧУК Віталій Олександрович**

Керівник роботи: **МОРОЗОВ Андрій Васильович**

Тема роботи: **«Розробка інформаційної системи для класифікації резюме кандидатів на основі NLP»**, затверджена Наказом закладу вищої освіти від **«01» жовтня 2024 р. № 520/с**

Вихідні дані для роботи: **предметом дослідження є процес проектування та розробка інформаційної системи для класифікації резюме кандидатів на основі NLP.**

Керівники / консультанти з магістерської кваліфікаційної роботи із зазначенням розділів, що їх стосуються:

Розділ	Керівник/ консультант	Завдання видав	Завдання прийняв
1	МОРОЗОВ А.В.	20.09.2024р.	20.09.2024р.
2	МОРОЗОВ А.В.	15.10.2024р.	15.10.2024р.
3	МОРОЗОВ А.В.	25.11.2024р.	25.11.2024р.

РЕФЕРАТ

Кваліфікаційна робота освітнього ступеня «магістр» присвячена розробці інформаційної системи для класифікації резюме кандидатів на основі NLP. Пояснювальна записка складається з вступу, трьох розділів та списку використаної літератури. Пояснювальна записка до кваліфікаційної магістерської роботи містить 96 сторінок, 37 ілюстрацій, 1 таблицю, 25 сторінок додатків та 24 посилань на зовнішні джерела.

Метою роботи є розробка інформаційної системи для класифікації резюме кандидатів на основі NLP.

В роботі проаналізовано та порівняно різні методи класифікації текстових даних, зокрема для автоматичної класифікації резюме кандидатів. У процесі виконання роботи були досліджені різні підходи до обробки та аналізу текстових даних, що дозволяють ефективно вирішити завдання класифікації.

В рамках роботи проведено порівняння результатів різних моделей класифікації, розглянуті їх переваги та недоліки, а також визначено найбільш ефективний метод для конкретної задачі.

Робота є важливим кроком у розвитку систем автоматичного аналізу текстової інформації та може бути корисною для подальших досліджень у цій галузі.

КЛЮЧОВІ СЛОВА: ІНФОРМАЦІЙНА СИСТЕМА, NLP, ШТУЧНИЙ ІНТЕЛЕКТ, РОЗРОБКА, ПРОГРАМУВАННЯ, АНАЛІЗ ДАНИХ, PYTHON.

ABSTRACT

The qualification work for the educational degree "Master" is devoted to the development of an information system for classifying candidates' resumes based on NLP. The explanatory note consists of an introduction, three sections and a list of used literature. The explanatory note to the qualification master's thesis contains 96 pages, 37 illustrations, 1 table, 25 pages of appendices and 24 links to external sources.

The purpose of the work is to develop an information system for classifying candidates' resumes based on NLP.

The work analyzes and compares various methods for classifying text data, in particular for automatic classification of candidates' resumes. In the process of performing the work, various approaches to processing and analyzing text data were investigated, which allow to effectively solve the classification task.

As part of the work, the results of different classification models were compared, their advantages and disadvantages were considered, and the most effective method for a specific task was determined.

The work is an important step in the development of automatic text information analysis systems and may be useful for further research in this area.

KEYWORDS: INFORMATION SYSTEM, NLP, ARTIFICIAL INTELLIGENCE, DEVELOPMENT, PROGRAMMING, DATA ANALYSIS, PYTHON.

ЗМІСТ

ВСТУП.....	7
РОЗДІЛ 1. АНАЛІЗ ЗАДАЧІ ТА ТЕОРЕТИЧНІ ОСНОВИ ЇЇ ВИРІШЕННЯ.....	9
1.1.Аналіз задачі та методів її вирішення	9
1.2. Інтелектуальний аналіз текстових даних.....	9
1.3. Обробка природної мови (NLP).....	13
Висновки до розділу 1.....	18
РОЗДІЛ 2. АНАЛІЗ МЕТОДІВ КЛАСИФІКАЦІЇ РЕЗЮМЕ.....	20
2.1. Методи класифікації текстових даних	20
2.2. Особливості застосування NLP для аналізу резюме	21
2.3. Порівняння існуючих методів класифікації	26
Висновки до розділу 2.....	34
РОЗДІЛ 3. РОЗРОБКА ІНФОРМАЦІЙНОЇ СИСТЕМИ.....	36
3.1. Збір та підготовка даних	36
3.2. Аналіз даних із використанням моделей	47
3.3.Порівняльний аналіз моделей класифікації.....	59
Висновки до розділу 3.....	64
ВИСНОВКИ.....	66
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ	68
ДОДАТКИ.....	71

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

ШІ – (англ. artificial intelligence) – штучний інтелект

NLP – (англ. Natural language processing) – обробка природної мови

TF-IDF – (англ. Term Frequency-Inverse Document Frequency) – зворотна частота документа

BoW – (англ. Bag of Words) – мішок слів

BERT – (англ. Bidirectional Encoder Representations from Transformers) – двоспрямовані кодувальні представлення з трансформерів

GPT – (англ. Generative Pre-trained Transformer) – генеративний попередньо тренований трансформер

IAT – інтелектуальний аналіз тексту

SVC – (англ. Support Vector Classifier) – класифікатор на основі методу опорних векторів

SVM – (англ. Support Vector Machines) – метод опорних векторів, алгоритм машинного навчання для задач класифікації та регресії

MNB – (англ. Multinomial Naive Bayes) – багатомірний наївний байєсівський алгоритм

KNN – (англ. K-Nearest Neighbors) – алгоритм К найближчих сусідів

RNN – (англ. Recurrent Neural Network) – рекурентна нейронна мережа

LSTM – (англ. Long Short-Term Memory) – довготривала короткочасна пам'ять

ВСТУП

Актуальність теми. У сучасному світі інформаційні технології значно впливають на автоматизацію різноманітних бізнес-процесів, зокрема у сфері управління персоналом. Одним із найважливіших викликів для компаній є ефективна обробка великої кількості резюме кандидатів, які надходять під час відбору персоналу. Традиційні методи аналізу резюме вимагають значних людських ресурсів, що може уповільнювати процес рекрутингу та знижувати його ефективність.

Технології обробки природної мови (NLP, Natural Language Processing) дозволяють автоматизувати аналіз текстової інформації, виділяючи ключові аспекти та підвищуючи точність оцінки кандидатів. Інтеграція NLP у процес класифікації резюме дозволяє значно скоротити час на відбір потенційних співробітників, забезпечити об'єктивність оцінки та оптимізувати використання ресурсів компанії.

Розробка інформаційної системи для класифікації резюме кандидатів на основі NLP є актуальною задачею, що має значний практичний і науковий інтерес. Впровадження такої системи може стати важливим етапом у вдосконаленні процесів найму персоналу в компаніях різних масштабів.

Мета роботи – розробка інформаційної системи, здатної автоматизувати процес класифікації резюме кандидатів за допомогою методів NLP.

Для досягнення мети визначено наступні завдання:

- провести аналіз існуючих підходів та методів обробки текстових даних у задачах класифікації резюме;
- зібрати та підготувати набір даних для навчання і тестування;
- розробити інформаційну систему, що використовує сучасні методи NLP для класифікації резюме;
- провести експериментальне тестування розробленої системи та оцінити її ефективність.

Об'єктом дослідження є процес автоматизації аналізу резюме кандидатів.

Предметом дослідження є застосування методів обробки природної мови для класифікації текстових даних у резюме кандидатів.

Наукова новизна роботи полягає у використанні сучасних методів NLP та машинного навчання для автоматизації класифікації резюме.

Практичне значення полягає у її можливому впровадженні у сферу рекрутингу для підвищення ефективності процесів відбору персоналу.

Апробація. За темою кваліфікаційної магістерської роботи було опубліковано статті у збірнику наукових праць [23, 24].

РОЗДІЛ 1. АНАЛІЗ ЗАДАЧІ ТА ТЕОРЕТИЧНІ ОСНОВИ ЇЇ ВИРІШЕННЯ

1.1. Аналіз задачі та методів її вирішення

У сучасних умовах розвитку інформаційних технологій аналіз текстових даних, зокрема в контексті обробки резюме, став важливою задачею для автоматизації процесу відбору кандидатів на різноманітні позиції. Завдяки значному збільшенню обсягу даних, що зберігаються в електронному вигляді, виникає потреба у створенні ефективних систем, здатних автоматично обробляти, аналізувати та класифікувати текстову інформацію. Зокрема, особливу увагу слід приділити таким завданням, як витягнення ключових навичок, досвіду, освіти, а також класифікація кандидатів за відповідними критеріями.

Основною задачею є класифікація резюме на основі їхнього вмісту. Враховуючи великий обсяг даних, цей процес не може бути виконаний вручну без значних часових витрат, тому його автоматизація є важливою для підвищення ефективності рекрутингу. Зокрема, завдання полягає у розпізнаванні ключових характеристик кожного резюме, таких як навички, досвід, кваліфікація, а також у визначенні найбільш підходящих кандидатів для конкретних вакансій.

Крім того, необхідно врахувати різноманітні варіації написання інформації в резюме, що можуть виникати через відмінності у стилях, форматах та мовах. Це ставить додаткові вимоги до створення системи, здатної враховувати такі варіанти та коректно їх обробляти.

Для вирішення поставленої задачі, використовуються сучасні методи обробки природної мови (NLP), які включають різноманітні підходи до аналізу тексту, в тому числі статистичні методи, методи машинного навчання, а також методи глибокого навчання.

1.2. Інтелектуальний аналіз текстових даних

Аналіз текстових даних – це процес, який дозволяє отримувати корисну інформацію з текстового контенту шляхом його обробки, інтерпретації та

структурування. Він об'єднує методи статистики, машинного навчання та обробки природної мови (NLP), завдяки чому неструктуровані дані, такі як документи, повідомлення чи записи в соціальних мережах, перетворюються на структуровану інформацію. Ця інформація може бути використана для прийняття важливих рішень, автоматизації процесів або виявлення ключових тенденцій.

Аналіз тексту має широку сферу застосування. У бізнесі він допомагає зрозуміти клієнтів, аналізуючи їхні відгуки чи настрої. У рекрутингу сприяє автоматизації відбору резюме, забезпечуючи відповідність кандидатів вакансіям. У медицині дозволяє обробляти великі обсяги медичних записів чи наукових публікацій для пошуку ключових даних. В юридичній сфері допомагає класифікувати документи чи прогнозувати судові рішення. В освітньому процесі аналіз тексту може бути використаний для оцінювання відповідей студентів або оптимізації навчальних матеріалів. Також цей підхід активно застосовується в науці, де аналізуються великі масиви літератури, і в безпеці, наприклад, для виявлення кіберзагроз.

Загалом аналіз текстових даних має велике значення в сучасному світі, оскільки дозволяє обробляти величезні обсяги текстової інформації та витягувати з неї знання, які раніше могли бути недоступними через складність і обсяг даних.

Також частиною комп'ютерного аналізу даних є Text mining. Text mining, також відомий як інтелектуальний аналіз текстових даних, це процес перетворення неструктурованого тексту в структурований формат для виявлення значущих шаблонів і нових ідей. Інтелектуальний аналіз тексту може бути використаний для аналізу величезних колекцій текстових матеріалів, щоб виявити ключові поняття, тенденції та приховані зв'язки [1]. Основні процеси text mining описано на рис. 1.1.



Рис. 1.1. Процес Text mining

Застосовуючи передові аналітичні методи, такі як Naïve Bayes, Support Vector Machines (SVM) та інші алгоритми глибокого навчання, може бути досліджена виявлено приховані зв'язки в неструктурованих даних.

Текст є одним із найпоширеніших типів даних у базах даних. Залежно від бази даних ці дані можуть бути організовані як: структуровані дані, неструктуровані дані, напівструктуровані дані.

Структуровані дані – це дані, які стандартизовано в табличний формат із численними рядками та стовпцями, що полегшує зберігання та обробку для аналізу та алгоритмів машинного навчання. Структуровані дані можуть включати такі вхідні дані, як імена, адреси та номери телефонів.

Неструктуровані дані не мають попередньо визначеного формату даних. Він може включати текст із джерел, як-от соціальних мереж чи оглядів продуктів, або мультимедійні формати, як-от відео- та аудіофайли.

Напівструктуровані дані є сумішшю структурованих і неструктурованих форматів даних. Хоча вона має певну структуру, вона не має достатньої структури, щоб відповідати вимогам реляційної бази даних. Приклади напівструктурованих даних включають файли XML, JSON і HTML.

Ключовими завданнями ІАТ є категоризація текстів, пошук інформації, обробка змін у колекціях текстів, розробка засобів представлення інформації для користувача.

Категоризація документів полягає у зіставленні документів з колекції з однією або декількома групами (класами, кластерами) схожих між собою текстів (наприклад, по темі або стилем). Категоризація може відбуватися як за участю людини, так і без неї.

У першому випадку (класифікація документів), система ІАТ повинна віднести тексти до вже визначених (зручних для неї) класів. Для цього необхідно провести кероване навчання, для чого користувач повинен надати системі ІАТ як перелік класів, так і зразки документів, що належать цим класам.

Другий випадок категоризації називається кластеризацією документів. При цьому система ІАТ повинна сама визначити множину кластерів, за якими можуть бути розподілені тексти, — в машинному навчанні відповідне завдання називається некерованим навчанням. У цьому випадку користувач повинен повідомити системі ІАТ кількість кластерів, на яке йому хотілося б розбити оброблювану колекцію (передбачається, що в алгоритм програми вже закладена процедура вибору ознак) [2].

Технологія інтелектуального аналізу тексту в даний час широко застосовується до широкого кола урядових, дослідницьких та бізнес-потреб. Всі три групи можуть використовувати інтелектуальний аналіз тексту для управління документами та пошуку документів, що стосуються їх повсякденної діяльності. Професіонали з правових питань можуть використовувати інтелектуальний аналіз тексту для електронного відкриття. Уряди і військові групи використовують інтелектуальний аналіз тексту для цілей національної безпеки та розвідки. Наукові дослідники об'єднують підходи інтелектуального аналізу тексту в зусиллях для організації великих наборів текстових даних (тобто, розв'язання проблеми неструктурованих даних), для визначення ідей, переданих через текст (наприклад, аналіз тональності тексту у соціальних мережах) і підтримувати наукові відкриття в галузі природничих наук та в

галузі біоінформатики. У бізнесі інтелектуальний аналіз тексту використовується для підтримки конкурентної розвідки та автоматичного розміщення оголошень серед багатьох інших заходів.

1.3. Обробка природної мови (NLP)

Ключовим засобом аналізу та обробки текстових даних є NLP. Обробка природної мови (NLP) – це технологія машинного навчання, яка дає комп'ютерам можливість інтерпретувати, маніпулювати та розуміти людську мову. Сьогодні організації мають великі обсяги голосових та текстових даних із різних каналів зв'язку, таких як електронні листи, текстові повідомлення, стрічки новин соціальних мереж, відео, аудіо та багато іншого. Вони використовують програмне забезпечення NLP для автоматичної обробки цих даних, аналізу намірів чи настроїв у повідомленні та реагування на людське спілкування в режимі реального часу [3]. Структура NLP складається з багатьох частин, котрі зображено на рис. 1.2.

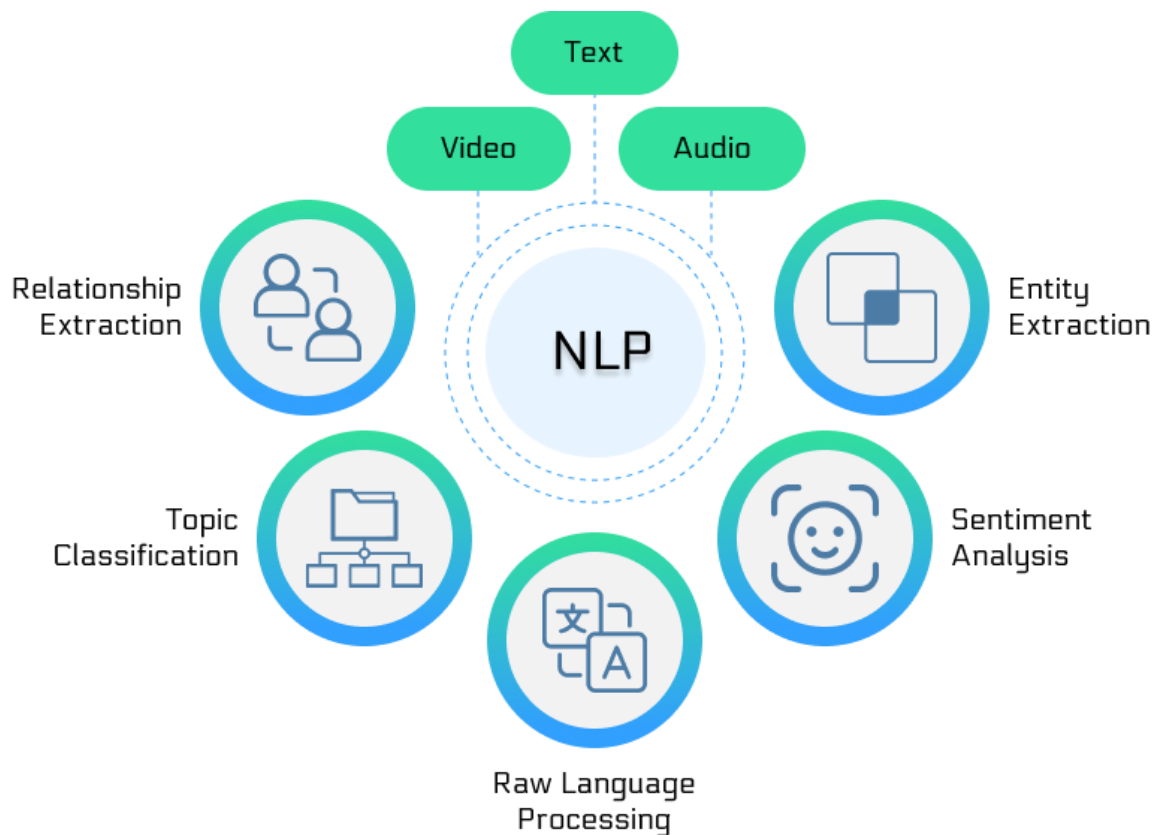


Рис. 1.2. Структура NLP

До основних завдань обробки природної мови (NLP) належать кілька ключових аспектів (рис. 1.3).

Видобування даних передбачає вивчення інформації, пошук зв'язків і закономірностей між ними. Синтез мовлення спрямований на озвучення або прочитання тексту голосом, який максимально наближений до природного.

Розпізнавання мови охоплює визначення тексту з картинок, відсканованих документів або PDF-файлів, а також розпізнавання мовлення, що продукується людським голосом. Генерування природної мови дозволяє перетворювати комп'ютерні дані у природну людську мову. Машинний переклад забезпечує автоматичний переклад текстів з однієї мови на іншу, проте це завдання є складним, оскільки машини не володіють знаннями, притаманними людині, що ускладнює розуміння фраз. Питально-відповідальні системи займаються пошуком відповідей на запитання, сформульовані людською мовою.

До інших завдань належить розпізнавання або визначення тем тексту, що передбачає поділ тексту на частини з визначенням головної теми кожної з них. Інформаційний пошук включає процес пошуку, розпізнавання й видобування інформації.

Добування даних передбачає отримання семантичної інформації з тексту, а також встановлення зв'язків між об'єктами в тексті, наприклад, хто з ким працює. Спрощення тексту полягає у зміні, розширенні або іншій обробці інформації для спрощення структури чи граматики тексту зі збереженням основної ідеї. Розв'язання лексичної багатоманітності спрямоване на визначення можливих значень багатозначних слів і вибір найбільш підходящого значення відповідно до контексту.

Також важливими завданнями є розпізнавання абревіатур, заголовків, детектування окремих лінгвістичних одиниць і морфологічна декомпозиція, яка дозволяє перетворювати терміни (зокрема, технічні або медичні) у зрозумілу форму [4].

КЛАСИФІКАЦІЯ Дозволяє розділяти та групувати запити на конкретні тематичні класи, розуміючи безпосередньо сутність, або контекст, про який йде мова	РЕЛЕВАНТНИЙ ПОШУК Дозволяє здійснити пошук та перенесення потрібних даних відповідно до заданого контексту
МАШИННИЙ ПЕРЕКЛАД Здійснення перекладу з однієї мови на іншу	СЕНТИМЕНТ АНАЛІЗ Здійснює аналіз настроїв користувачів при комунікації, їх психологічно-емоційний стан
ПРОГНОЗУВАННЯ Дозволяє спрогнозувати результат на основі закономірностей виявлених у даних, які були надані для навчання	ПЕРЕВІРКА ГРАМОТНОСТІ ТЕКСТІВ Перевірка орфографії, пунктуації, корегування стилістики введених даних
УПРАВЛІННЯ ГОЛОСОМ Дозволяє перетворити аудіо-дані голосу у текст	ГЕНЕРАЦІЯ ТЕКСТУ Дозволяє згенерувати відповідь або потрібний тематичний текст відповідно до стилістики і типу даних, на яких модель була навчена
САМЕРИЗАЦІЯ ДАНИХ Опрацювання великих масивів даних та надання короткого висновку-заключення відповідно до їх змісту	ВИЛУЧЕННЯ ДАНИХ Вилучення з великого масиву тексту конкретних даних, які стосуються необхідного контексту

Рис. 1.3. Завдання NLP

Розвиток NLP можна розглядати через низку етапів, що охоплюють більше ніж півстоліття наукових досягнень та технологічних інновацій [5].

Початок розробки NLP можна віднести до 1950-х років, коли комп'ютери лише починали використовуватись для автоматизації різних обчислювальних завдань. Одним з перших значущих кроків стало створення алгоритмів для машинного перекладу. У 1950 році Аллан Тьюрінг запропонував свою відому тестову концепцію, звану Тестом Тьюрінга, що стала важливим етапом у розвитку ідеї машинного «розуміння» людської мови.

Перші спроби автоматичного перекладу між російською та англійською мовами проводилися в рамках проекту, фінансованого урядом США в 1950-х роках. Це був один із перших великих досліджень у галузі обробки мови, хоча результати не виправдали сподівань через складність природних мов та обмеження обчислювальних потужностей того часу.

У період лінгвістичних підходів (1970-1990-ті роки) NLP зосереджувався на лінгвістичних моделях, що спираються на граматику. Однією з основних ідей був синтаксичний аналіз — побудова деревовидних структур для представлення

граматичних відносин між словами в реченні. Одним з основних напрямків було використання контекстно-вільних граматики (CFG), які дозволяють аналізувати і створювати правильні речення на основі їх синтаксису.

Протягом цього часу розвивалися різні підходи до аналізу тексту, такі як морфологічний аналіз (визначення коренів, суфіксів та інших лексичних елементів), а також створювались перші словники і лексичні бази даних для підтримки обробки тексту.

У 1990-х роках почався перехід від граматичних моделей до статистичних методів. Це відбулося завдяки появі потужніших комп'ютерів і великих корпусів текстів, які дозволили обробляти великі обсяги даних. Статистичні моделі ґрунтуються на ймовірностних методах, які оцінюють ймовірність появи тих чи інших слів або фраз у контексті іншого слова. Одним із таких методів був наївний байєсівський класифікатор, який дозволяє визначати ймовірність того, що певний текст належить до певної категорії (наприклад, позитивний чи негативний відгук).

Застосування статистичних підходів також призвело до створення нових технік, таких як моделі скалярних просторових подібностей для вимірювання подібності між словами та фразами, а також автоматичних методів тематичного моделювання.

З початком 2010-х років обробка природної мови зазнала революції завдяки розвитку глибоких нейронних мереж. Зокрема, рекурентні нейронні мережі (RNN) та довготривала пам'ять (LSTM) дозволили моделювати залежності в текстах, які мають велику кількість елементів (наприклад, виведення повних речень або передбачення наступного слова).

У 2018 році компанія Google презентувала BERT, нову архітектуру, що революціонізувала NLP. Ця модель заснована на трансформерах – алгоритмах, які дозволяють паралельно обробляти весь контекст тексту, що дає змогу отримати набагато точніші результати. Трансформери працюють із увагою, яка дозволяє моделі фокусуватися на найважливіших частинах тексту для виконання завдання, що значно підвищує якість обробки природної мови. Такі моделі, як

GPT-3, BERT та T5, дозволяють виконувати складні завдання в NLP, включаючи класифікацію тексту, машинний переклад, створення текстів та багато інших.

Розуміння природної мови є однією з найбільш складних задач у сфері штучного інтелекту, оскільки для ефективного розпізнавання людської мови потрібно, щоб система мала глибоке розуміння контексту навколишнього середовища та могла взаємодіяти з ним. Визначення того, що саме означає «розуміти», є основним викликом для штучного інтелекту.

Проте, незважаючи на досягнення, є одна велика проблема, яка залишається не вирішеною, і полягає вона у самій природі людської мови. Людське мовлення часто є неоднозначним, і це створює великі труднощі в його автоматичному розумінні. Існують різні види неоднозначностей.

Структура речення яка може мати декілька інтерпретацій називається синтаксична неоднозначність. Наприклад, в українському прислів'ї «Час – не кінь, не підженеш і не зупиниш» не зовсім зрозуміло, чи йдеться про час, чи про коня, і як саме їх порівнюють.

Смисловою неоднозначністю називається неоднозначність, при якій слово має кілька значень, і контекст може допомогти визначити, яке з них правильне. Наприклад, в реченні «Де знайти ключ до того замку?» слово «замок» може означати або фортецю, або механізм для замикання.

Відмінкова неоднозначність – одна і та сама форма слова може мати різні значення залежно від контексту. У фразях «Усі були схвильовані перед концертом» і «Не треба давати перед!» слово «перед» може позначати час або місце, і це змінює сенс фрази.

Референційна неоднозначність визначається в ситуаціях, коли займенник вказує на декілька можливих об'єктів, але система не може визначити, до чого саме він відноситься. Наприклад, у фразі «Відкрий полицку та дістань мокру парасольку, я хочу її висушити» займенник «її» може означати як полицку, так і парасольку, і без контексту система може зробити неправильне припущення.

Окрім цього, ще однією складною проблемою є синонімія, коли різні слова можуть виражати одне й те саме поняття. Це ускладнює пошукові системи,

оскільки одне і те саме поняття може бути виражено різними словами, і система не завжди може коректно визначити релевантні документи, де ці синоніми використовуються.

Ці проблеми особливо сильно впливають на системи машинного перекладу, оскільки складно передати точну семантичну та синтаксичну структуру речення в іншу мову, зберігаючи його значення. Визначити правильне відображення структури тексту у внутрішньому представленні, яке зрозуміє комп'ютер, залишається складним завданням.

Для вирішення цих проблем іноді потрібно додавати додаткові знання про певні галузі, щоб система могла правильно інтерпретувати неоднозначності. Хоча сьогодні не існує програм, здатних «розуміти» всі види неоднозначностей у різних сферах, є спеціалізовані програми, які можуть ефективно працювати в обмежених контекстах і враховувати неоднозначності в вузьких сферах.

Висновки до розділу 1

У першому розділі роботи було проведено аналіз задачі та розглянуто теоретичні основи аналізу текстових даних та обробки природної мови (NLP), а також огляд існуючих інформаційних систем для аналізу резюме.

Було охарактеризовано основні підходи до обробки текстових даних, а також основні етапи та методи, які використовуються в Text Mining. Розглянуто процеси витягнення корисної інформації з текстів, такі як класифікація, кластеризація та аналіз емоційного забарвлення тексту. Особливу увагу було приділено важливості попередньої обробки даних, що є необхідним етапом для досягнення високої точності аналізу.

Описано етапи розвитку NLP, основні задачі та методи, які використовуються для обробки текстів на природній мові, такі як токенізація, лематизація, частиномовний аналіз, синтаксичний та семантичний аналіз. Визначено ключові завдання NLP, такі як розпізнавання мовленнєвих одиниць, розуміння контексту і генерування тексту. Обговорені основні обмеження та

виклики, що стоять перед цією галуззю, включаючи обробку неструктурованих даних та потребу в розумінні контексту, що змінюється в залежності від ситуації.

У результаті дослідження було виявлено, що аналіз текстових даних і обробка природної мови є потужними інструментами для автоматизації процесів відбору кандидатів на основі їх резюме. Вони дають змогу покращити ефективність роботи HR-фахівців і забезпечити глибше розуміння текстових даних, що має важливе значення для успішного розвитку автоматизованих систем на основі AI. Однак на даний момент існують певні обмеження в точності і масштабованості цих технологій, що потребує подальшого удосконалення алгоритмів та моделей.

РОЗДІЛ 2. АНАЛІЗ МЕТОДІВ КЛАСИФІКАЦІЇ РЕЗЮМЕ

2.1. Методи класифікації текстових даних

Класифікація текстових даних – це процес, який полягає у віднесенні тексту до певної категорії або класу на основі його змісту. У контексті аналізу резюме класифікація дозволяє автоматично визначати ключові характеристики кандидатів, такі як професійні навички, рівень освіти чи досвід роботи. Цей процес є важливим етапом при створенні інформаційних систем для аналізу резюме, оскільки забезпечує структуроване представлення неструктурованих даних [6].

Методи класифікації текстових даних можна умовно поділити на традиційні та сучасні підходи.

Традиційні методи включають моделі на основі правил, які використовують заздалегідь визначені правила для аналізу тексту. Наприклад, такі системи можуть шукати ключові слова або фрази для віднесення тексту до певної категорії. Ці методи вирізняються простотою реалізації, зрозумілістю та точністю у вузькоспеціалізованих завданнях, проте мають обмеження у масштабуванні та низьку гнучкість при роботі з великими обсягами даних або неоднозначними текстами [7].

До традиційних підходів належать також статистичні методи, які базуються на ймовірнісних підходах, таких як Naive Bayes. Ці алгоритми аналізують частоту появи слів у тексті для визначення його приналежності до певного класу. Наприклад, під час аналізу резюме можна оцінювати, наскільки часто зустрічаються слова, пов'язані з технічними спеціальностями.

Ще одним традиційним методом є використання машинного навчання, зокрема логістичної регресії та дерев рішень. Ці алгоритми працюють із навчальними даними, попередньо перетвореними в числовий формат, наприклад, за допомогою технік TF-IDF або Bag of Words. Логістична регресія підходить для бінарної класифікації, наприклад, оцінки відповідності резюме заданій вакансії,

а дерева рішень надають зрозумілу структуру для прийняття класифікаційних рішень [8].

Сучасні підходи до класифікації тексту базуються на глибокому навчанні. Рекурентні нейронні мережі (RNN) використовуються для аналізу послідовностей, таких як текст, і враховують контекст і порядок слів. Наприклад, вони допомагають визначати професійні навички чи досвід у резюме. Трансформери, такі як BERT, стали проривом у сфері NLP завдяки здатності враховувати контекст усіх слів у реченні одночасно. Це дозволяє їм досягати високої точності в класифікації тексту, розпізнаючи складні залежності між словами [9].

На практиці для класифікації резюме часто застосовують комбінації методів. Наприклад, первинний аналіз може виконуватись на основі правил для виділення загальних категорій, а потім уточнюватися за допомогою глибокого навчання. Інтеграція традиційних і сучасних методів дозволяє досягти більшої точності та адаптивності систем аналізу резюме.

Таким чином, класифікація текстових даних є ключовим елементом у розробці систем для аналізу резюме. Вибір методу залежить від специфіки завдання, обсягу даних і бажаного рівня точності.

2.2. Особливості застосування NLP для аналізу резюме

Обробка тексту в задачах аналізу резюме є багатоступеневим процесом, який забезпечує перетворення неструктурованого тексту у структуровані дані, придатні для класифікації або вилучення інформації.

Першим кроком є перевірка наявності непотрібних слів у тексті. Цей етап передбачає очищення тексту від слів або символів, які не несуть значущої інформації для аналізу. Прикладом цього процесу може бути видалення чисел, спеціальних символів, HTML-тегів, посилань тощо. Метою цього етапу є зменшення обсягу даних, зберігаючи лише корисну інформацію, що в свою чергу підвищує точність моделей NLP. Для відкидання непотрібної інформації

використовуються регулярні вирази (RegEx), бібліотеки очищення тексту (наприклад, Python's re).

Виконавши початкове очищення тексту потрібно виконати токенизацію. Токенизація – це розбивка тексту на окремі слова або токени. Це може включати в себе видалення пунктуації та розділення тексту на окремі слова. Метою токенизації є спрощення роботи з текстом, дозволяючи аналізувати кожне слово окремо. Для токенизації використовуються наступні інструменти: `nltk.word_tokenize` (Python), `tokenizer` для багатofункціональних NLP завдань [10].

Приклад: Речення «This is a sample» токенизується як ['This', 'is', 'a', 'is', 'sample']. Процес токенизації зображено на рис. 2.1.

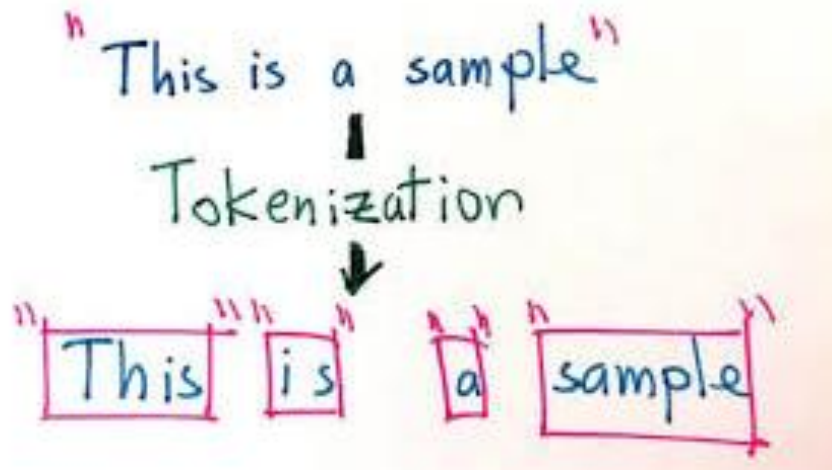


Рис. 2.1. Процес токенизації

Наступним важливим етапом в обробці текстових даних при аналізі тексту є видалення стоп слів (stop words), зокрема у задачах обробки природної мови (NLP). Стоп слова – це слова, які часто зустрічаються в тексті, але не несуть значущого смислового навантаження для конкретної задачі. Прикладом таких слів є артиклі, прийменники, сполучники, займенники, тощо, наприклад: «і», «в», «на», «це», «для».

Текст часто містить велику кількість стоп слів, що можуть збільшувати розмір текстових даних, але не додають корисної інформації для аналізу. Видалення таких слів допомагає зменшити обсяг даних, що аналізуються, і підвищити ефективність алгоритмів.

Стоп слова можуть заважати виділенню важливих термінів і концептів у тексті. Їх видалення дозволяє зосередитись на більш значущих словах, що може підвищити точність класифікації або інших моделей аналізу.

Багато алгоритмів NLP, таких як побудова моделей для тематичного моделювання, класифікації тексту чи пошукових систем, можуть працювати значно швидше і з більшою ефективністю, якщо не потрібно обробляти зайві слова.

Стоп слова можуть мати негативний вплив на алгоритми пошуку схожих документів або пошукових систем, оскільки вони можуть домінувати в частотних статистиках і знижувати релевантність результатів.

Після видалення стоп слів слід виконати процес зведення слів до їх кореня або основи, що дозволяє зменшити варіативність слів та зберегти основну смислову одиницю для подальшого аналізу. Цей процес має назву стемінг (stemming) Стемінг не завжди зберігає граматичні форми слова, а скоріше фокусується на видаленні префіксів і суфіксів для того, щоб знайти їх «стем» (основу) [11]. Процес стемінгу зображено на рис. 2.2.

Стемінг дозволяє звести слова, які мають різні граматичні форми, до одного кореня. Наприклад, слова «gunning», «gunner», «gap» можуть бути зведені до одного кореня «gun». Це дозволяє системі працювати з меншим набором унікальних слів, що важливо для аналізу текстів, таких як пошукові системи або класифікація документів.

В обробці тексту часто є необхідність враховувати синонімічні форми слів (наприклад, «запуск» і «запустити»). Стемінг допомагає зменшити цю складність, оскільки всі варіації зводяться до одного базового кореня.

В процесах, як-от пошукові системи або класифікація текстів, зведення слів до кореня зменшує кількість варіацій слів, що може прискорити обробку текстів і зробити аналіз більш ефективним. Наприклад, можна швидше порівнювати слова, знаючи, що всі вони походять від одного кореня.

Стемінг може допомогти знизити «шум» у тексті, виключаючи варіативність граматичних форм. Це може бути корисним при аналізі великих

масивів тексту або в задачах тематичного моделювання, де важливо працювати з основними концептами, а не з конкретними граматичними формами.

Stemming

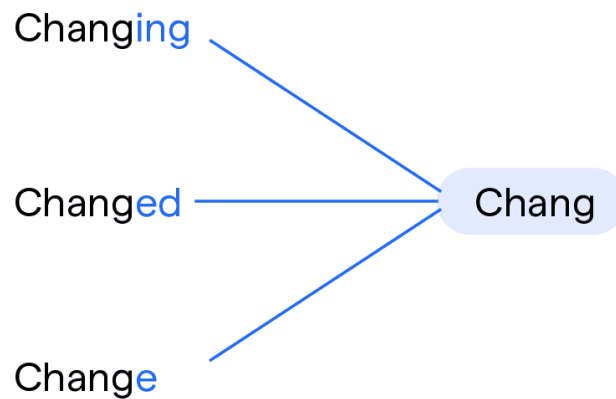


Рис. 2.2. Процес стемінгу

Наступним важливим етапом підготовки тексту о аналізу є виконання лематизації. Лематизація (Lemmatization) – це процес перетворення слова в його базову або словникову форму (лему), з урахуванням його частини мови та контексту. Відмінність лематизації від стемінгу полягає в тому, що лематизація враховує граматичні особливості слова (частина мови, відмінок, час тощо), що дозволяє отримати більш точну і значущу базову форму [12]. Процес лематизації зображено на рис. 2.3.

Лематизація зберігає більше контекстної інформації, ніж стемінг, оскільки вона враховує частину мови, з якою пов'язане слово. Наприклад, слово «better» буде перетворене в «good», якщо це прикметник, а «running» – в «run», якщо це дієслово.

Лема – це словникова форма слова, тобто та форма, яку можна знайти в словнику. Наприклад:

«running» → «run»

«was» → «be»

«mice» → «mouse»

Лематизація зберігає смислову значущість слова, на відміну від стемінгу, який може призвести до втрати сенсу через зведення слів до надто загальних форм.

Lemmatization

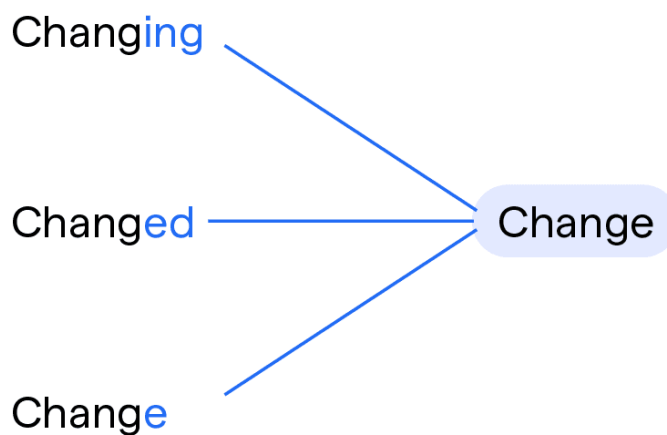


Рис. 2.3. Процес лематизації

Лематизація та Стемінг є дуже схожими поняттями тому слід визначити їх відмінність.

Лематизація більш точна, оскільки зберігає граматичні і контекстуальні нюанси, а стемінг просто обрізає суфікси і префікси.

Лематизація зазвичай вимагає більше обчислювальних ресурсів і часу, ніж стемінг, оскільки вона потребує аналізу частини мови та контексту.

Лематизація підходить для задач, де важлива точність і смислове значення, наприклад, у системах автоматичного перекладу, аналізі настроїв, класифікації текстів.

2.3. Порівняння існуючих методів класифікації

У сучасних задачах машинного навчання, зокрема в області обробки текстових даних, важливим кроком є вибір оптимальних методів класифікації для аналізу та прогнозування. Класифікація – це процес, при якому модель машинного навчання використовується для присвоєння об'єктам певних категорій на основі їхніх ознак. У контексті обробки резюме, метою є класифікація документів на основі різноманітних параметрів.

Моделі класифікації використовуються для автоматичного визначення категорії (класу) об'єкта на основі аналізу його характеристик. Вони можуть бути як простими, так і складними, і застосовуються в багатьох сферах: від медичної діагностики до фільтрації спаму в електронних листах. Вибір моделі класифікації залежить від специфіки завдання, якості та обсягу даних, а також вимог до точності та швидкості роботи системи.

Для класифікації текстових даних застосовуються різноманітні моделі, від традиційних статистичних методів до сучасних глибоких нейронних мереж. Кожен з цих методів має свої особливості, переваги та обмеження, і для кожного завдання потрібно обирати оптимальний підхід.

При виконанні даного проекту буде задіяно K-Nearest Neighbors (KNN), Multinomial Naive Bayes, Support Vector Classifier (SVC), Decision Tree та Random Forest моделі.

Алгоритм k-найближчих сусідів (KNN) – це непараметричний класифікатор із контрольованим навчанням, який використовує близькість для класифікацій або прогнозів щодо групування окремої точки даних. Це один із популярних і найпростіших класифікаторів і класифікаторів регресії, які використовуються сьогодні в машинному навчанні.

Алгоритм KNN (рис. 2.4.) може бути використаний як для задач регресії, так і для задач класифікації, але зазвичай застосовується для класифікації, ґрунтуючись на припущенні, що схожі точки знаходяться поруч одна з одною.

Для задач класифікації мітка класу призначається на основі більшості голосів – тобто, мітка, яка найчастіше зустрічається серед сусідніх даних,

використовується. Хоча це технічно називається «голосуванням за принципом більшості», в літературі зазвичай використовується термін «голосування більшістю». Різниця між цими термінами полягає в тому, що «голосування більшістю» технічно вимагає більшості більше ніж 50%, що в основному працює, коли є лише два класи. Коли ж є кілька класів, наприклад, чотири категорії, то не обов'язково мати 50% голосів для визначення класу; можна призначити мітку класу при більш ніж 25% голосів [13].

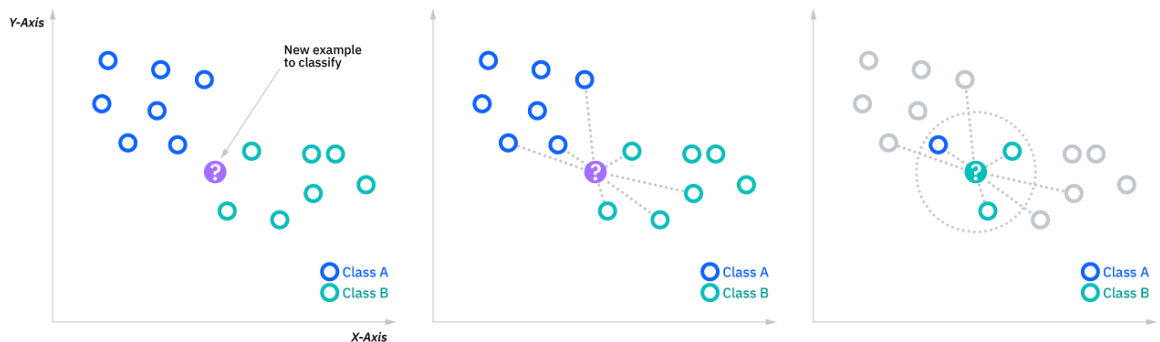


Рис. 2.4. Алгоритм KNN

Задачі регресії використовують схожий підхід до задач класифікації, але в цьому випадку для передбачення використовуються середні значення найближчих k сусідів. Основна відмінність полягає в тому, що класифікація застосовується для дискретних значень, а регресія – для неперервних. Однак перед тим, як зробити класифікацію, необхідно визначити відстань. Найчастіше використовують евклідову відстань.

Також варто зазначити, що алгоритм KNN є частиною сімейства моделей «повільного навчання» (lazy learning), що означає, що він лише зберігає навчальний набір даних, не проходячи етап навчання. Це також означає, що всі обчислення відбуваються під час класифікації або прогнозування. Оскільки цей алгоритм сильно залежить від пам'яті для збереження всіх навчальних даних, його також називають методом навчання на основі екземплярів або методами, заснованими на пам'яті.

Ідеї, пов'язані з моделлю KNN, належать Евелін Фікс і Джозефу Ходжесу, які представили їх у своїй роботі 1951 року, а Томас Кавер розвинув їх концепцію

у своїх дослідженнях в роботі «Nearest Neighbor Pattern Classification». Хоча цей алгоритм не такий популярний, як раніше, він досі є одним із перших алгоритмів, які вивчають у науці про дані завдяки своїй простоті та точності. Однак, зі збільшенням розміру набору даних, KNN стає все менш ефективним, що знижує загальну продуктивність моделі. Алгоритм KNN зазвичай використовують для простих систем рекомендацій, розпізнавання шаблонів, добування даних, прогнозування фінансових ринків, виявлення вторгнень та в інших сферах [14].

Multinomial Naive Bayes (MNB) – це дуже популярний і ефективний алгоритм машинного навчання, який базується на теоремі Байєса. Він широко використовується для завдань класифікації текстів, де потрібно працювати з дискретними даними, такими як підрахунок слів у документах. У цій статті ми розглянемо MNB і реалізуємо його.

MNB належить до ймовірнісних методів, що базуються на теоремі Байєса. Він називається «наївним», оскільки передбачає незалежність ознак, тобто присутність однієї ознаки не впливає на присутність іншої (що на практиці може бути неправдою) [15].

MNB є ймовірнісним класифікатором для обчислення розподілу ймовірностей текстових даних, що робить його добре підходящим для даних з ознаками, які представляють дискретні частоти або підрахунки подій у різних задачах обробки природної мови (NLP).

Термін «мультиноміальний» відноситься до типу розподілу даних, передбаченого моделлю. Особливостями в класифікації тексту є, як правило, кількість слів або частота термінів. Мультиноміальний розподіл використовується для оцінки ймовірності побачити певний набір кількості слів у документі [16]. Алгоритм MNB зображено на рис. 2.5.

Bayes' Theorem

Conditional probability: Probability of a label y occurring as a result of the given features x .

Joint probability: Probability of features x occurring together with label y .

$$P(y|x) = \frac{P(x,y)}{P(x)}$$

Marginal probability: Probability of features x occurring across all possible labels y .

Рис. 2.5. Алгоритм MNB

SVC (Support Vector Classifier): SVC це специфічна імплементація алгоритму SVM (Support Vector Machine).

SVM – це алгоритм навчання з учителем, який класифікує дані, знаходячи оптимальну лінію або гіперплощину, що максимізує відстань між класами в N -вимірному просторі [17].

SVM зазвичай використовуються для задач класифікації. Вони розрізняють два класи, знаходячи оптимальну гіперплощину, яка максимізує відстань між найближчими точками даних з різних класів. Кількість ознак у вхідних даних визначає, чи є гіперплощина лінією в двовимірному просторі або площиною в n -вимірному просторі. Оскільки можна знайти кілька гіперплощин для розрізнення класів, максимізація відстані між точками дозволяє алгоритму знайти найкращу межу для відокремлення класів. Це, у свою чергу, дозволяє йому добре узагальнювати нові дані та робити точні прогнози класифікації. Лінії, що прилягають до оптимальної гіперплощини, відомі як підтримувальні вектори, оскільки ці вектори проходять через точки даних, що визначають максимальний відступ.

Алгоритм SVM (рис. 2.6.) широко використовується в машинному навчанні, оскільки він може обробляти як лінійні, так і нелінійні задачі класифікації. Однак, коли дані не є лінійно роздільними, для перетворення даних

у простір вищої розмірності використовуються ядрові функції, що дозволяє забезпечити лінійне розділення [18].

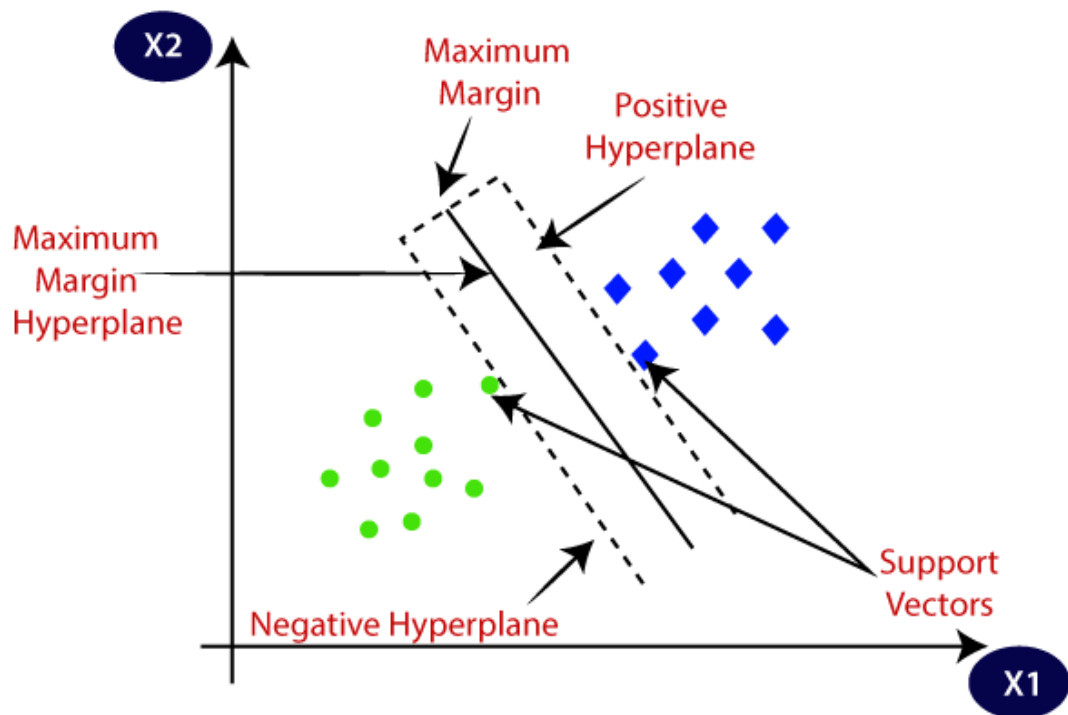


Рис. 2.6. Алгоритм SVM

Дерева рішень у машинному навчанні використовуються як передбачувальні моделі, що відображають знання про об'єкт (представлені гілками) у множину рішень. Це один з підходів до передбачувального моделювання у статистиці, добуванні даних та машинному навчанні. Навчання на дереві рішень – це метод, який зазвичай використовується в аналізі даних. Мета – створити модель, яка передбачає значення цільової змінної на основі кількох вхідних змінних [19].

Дерево рішень – це просте представлення для класифікації прикладів. Для цього розділу припустимо, що всі вхідні дані Особливість мають скінченні дискретні області визначення, і існує одна цільова ознака, яка називається «класифікація». Кожен елемент домену класифікації називається «класом». Дерево рішень або дерево класифікації - це дерево, в якому кожен внутрішній (нелистовий) вузол позначений вхідною ознакою. Дуги, що надходять від вузла, позначеного вхідною ознакою, позначаються кожним із можливих значень цільової ознаки, або дуга веде до підлеглого вузла прийняття рішень на іншій

вхідній ознаці. Кожен лист дерева позначається класом або розподілом ймовірностей по класах, що означає, що набір даних класифікований деревом або до певного класу, або до певного розподілу ймовірностей (яке, якщо дерево рішень коректно побудоване, зміщений до певних підмножин класів) [20]. Алгоритм дерева рішень зображено на рис. 2.7.

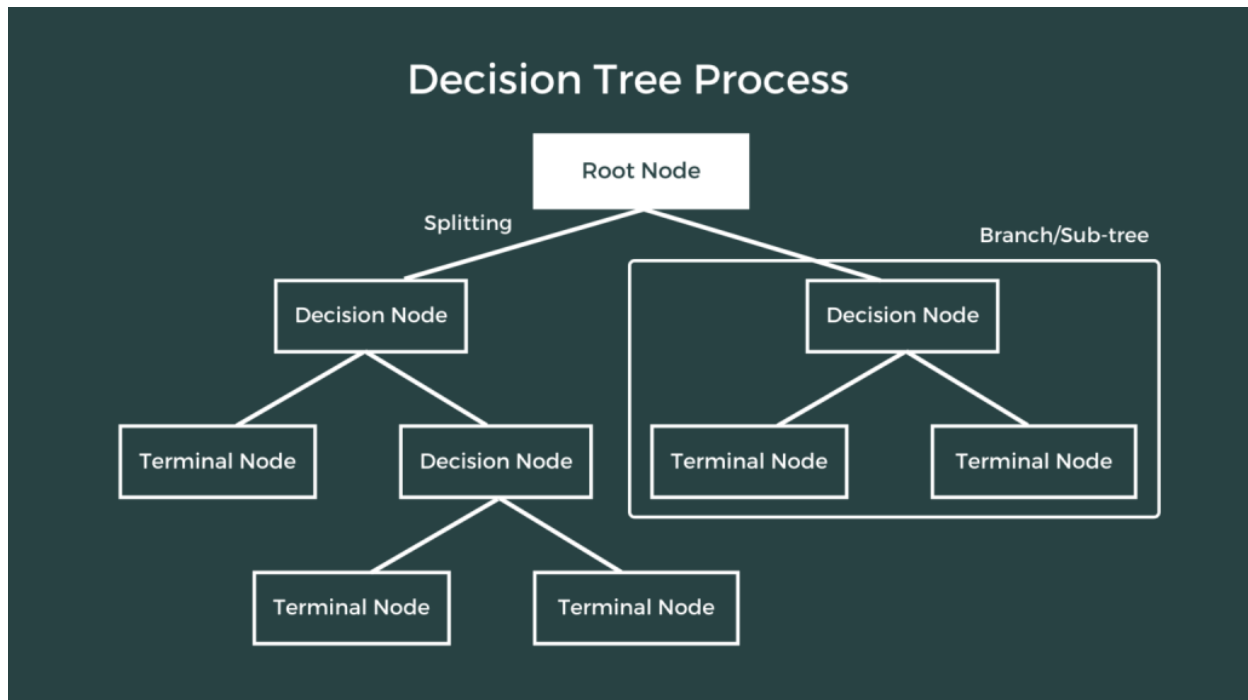


Рис. 2.7. Алгоритм дерева рішень

Random forest – ансамблевий метод машинного навчання для класифікації, регресії та інших завдань, який працює за допомогою побудови численних дерев прийняття рішень під час тренування моделі й продукує моду для класів (класифікацій) або усереднений прогноз (регресія) побудованих дерев. Недоліком є схильність до перенавчання [21].

Алгоритм Random Forest (рис. 2.8.) є потужною технікою навчання на деревах у машинному навчанні. Він працює шляхом створення декількох дерев рішень під час етапу навчання. Кожне дерево будується, використовуючи випадкову підмножину набору даних для оцінки випадкової підмножини ознак у кожному розподілі. Ця випадковість вводить варіативність між окремими деревами, що знижує ризик перенавчання та покращує загальну точність прогнозування.

Під час прогнозування алгоритм об'єднує результати всіх дерев, використовуючи голосування (для задач класифікації) або усереднення (для задач регресії). Цей процес колективного прийняття рішень, підтримуваний численними деревами та їхнім аналізом, забезпечує стабільні та точні результати. Випадкові ліси широко використовуються для функцій класифікації та регресії завдяки їх здатності обробляти складні дані, знижувати перенавчання та надавати надійні прогнози в різних умовах [22].

Random Forest використовує потужність ансамблевого навчання, створюючи сукупність дерев рішень. Ці дерева подібні до окремих експертів, кожен із яких спеціалізується на певному аспекті даних. Важливо, що вони працюють незалежно один від одного, мінімізуючи ризик того, що модель буде надмірно залежати від особливостей одного дерева.

Щоб кожне дерево рішень в ансамблі забезпечувало унікальний погляд, Random Forest застосовує випадковий вибір ознак. Під час навчання кожного дерева обирається випадкова підмножина ознак. Ця випадковість забезпечує, що кожне дерево зосереджує увагу на різних аспектах даних, створюючи різноманітний набір предикторів у межах ансамблю.

Техніка баггінгу є ключовим елементом стратегії навчання Random Forest. Вона передбачає створення кількох бутстреп-вибірок із початкового набору даних, дозволяючи зразкам обиратися з поверненням. Це створює різні підмножини даних для кожного дерева рішень, вводячи варіативність у процес навчання та роблячи модель більш стійкою.

Під час прогнозування кожне дерево рішень у Random Forest подає свій голос. Для задач класифікації фінальний прогноз визначається за модою (найчастішим прогнозом) серед усіх дерев. У задачах регресії береться середнє значення індивідуальних прогнозів дерев. Цей механізм внутрішнього голосування забезпечує збалансований і колективний процес прийняття рішень.

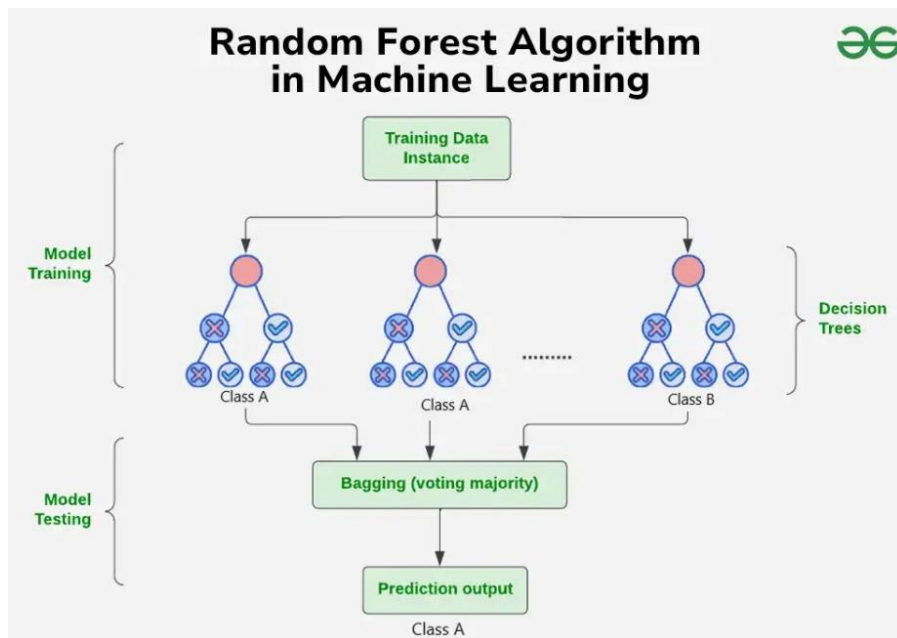


Рис. 2.8. Алгоритм Random Forest

Вибір найкращої моделі для класифікації ґрунтується на низці критеріїв, які дозволяють оцінити продуктивність, точність та відповідність моделі до специфіки задачі.

Точність є ключовим показником для класифікації. Вона визначається як відсоток правильно передбачених класів щодо загальної кількості випадків. Моделі порівнюються за точністю на тестовому наборі даних, щоб визначити, яка з них краще ідентифікує класи.

Окрім точності, враховуються додаткові метрики. Precision (Точність) – частка правильних передбачень серед усіх передбачених позитивних класів. Recall (Повнота) – частка правильних передбачень серед усіх фактичних позитивних класів. F1-score – середнє гармонічне між Precision та Recall, що дозволяє збалансувати оцінку для задач із нерівномірними класами.

Для великих обсягів даних або обмежень обчислювальних ресурсів обираються моделі, які демонструють високу швидкість навчання та передбачень, наприклад, Multinomial Naive Bayes чи Random Forest. Моделі, які вимагають значних ресурсів (наприклад, SVC з нелінійним ядром), використовуються лише у разі, якщо вони значно перевершують інші за точністю.

Також враховується здатність моделі адаптуватися до реальних даних, які можуть містити шум (зайві або некоректні дані). Наприклад, Random Forest має вбудовану стійкість до шуму завдяки використанню ансамблю дерев, тоді як Decision Tree може бути більш чутливим до перенавчання.

Всі моделі, розглянуті у дослідженні, оцінюються на однакових наборах даних. Порівняння проводиться за допомогою вищезазначених метрик, а також графіків, що відображають ефективність кожної моделі.

Висновки до розділу 2

Було проведено комплексний аналіз методів класифікації, які можуть бути використані для обробки та класифікації текстових даних у резюме. Детально розглянуто основні підходи до класифікації текстових даних, що дозволяє сформулювати розуміння базових алгоритмів та їх застосування у задачах обробки тексту.

Проаналізовано особливості застосування методів обробки природної мови (NLP) для аналізу резюме. Було описано ключові етапи підготовки текстових даних: перевірка наявності зайвих слів, токенизація, видалення стоп-слів, стемінг та лематизація. Ці кроки є необхідними для підвищення ефективності роботи моделей класифікації та зменшення впливу шуму в даних.

Виконано порівняння існуючих методів класифікації, які використовуватимуться у роботі. Було розглянуто переваги, недоліки та особливості моделей.

K-Nearest Neighbors (KNN) — це простий і інтуїтивно зрозумілий алгоритм, який ефективний для роботи з невеликими наборами даних. Multinomial Naive Bayes добре підходить для роботи з текстовими даними, особливо в задачах класифікації, які базуються на частотності слів. Support Vector Classifier (SVC) забезпечує високу точність у задачах класифікації завдяки побудові оптимальної гіперплощини. Decision Tree є інтерпретованим підходом, який легко застосовується до різних задач. Random Forest — це потужний ансамблевий метод, який поєднує рішення кількох дерев для підвищення точності

класифікації. Таким чином, проведений аналіз дозволив визначити основні методи, які будуть застосовані у подальшій роботі, та підготувати основу для їх практичного використання. Вибрані моделі забезпечують баланс між точністю, швидкістю обробки та можливістю роботи з текстовими даними, що робить їх доцільними для задачі класифікації резюме.

РОЗДІЛ 3. РОЗРОБКА ІНФОРМАЦІЙНОЇ СИСТЕМИ

3.1. Збір та підготовка даних

Підготовка текстових даних є важливим етапом у розробці інформаційної системи для класифікації резюме. На цьому етапі здійснюється аналіз, очищення та трансформація даних, що забезпечує їх готовність для навчання моделей машинного навчання.

Для виконання задачі класифікації використовувався набір даних, що складається з двох колонок. Перша колонка, `Resume`, містить текстові дані резюме, які включають опис досвіду роботи, навичок, освіти, досягнень та іншої релевантної інформації. Друга колонка, `Category`, містить мітки, що визначають професійний напрямок або спеціалізацію відповідного резюме, наприклад, «React JS Developer» чи «SQL Developer».

Першим етапом розробки інформаційної системи для класифікації резюме кандидатів із застосуванням методів обробки природної мови (NLP) є аналіз та попередня обробка вихідних даних. Для цього спочатку було імпортовано необхідні бібліотеки, які використовуватимуться в даному проєкті, а також здійснено завантаження та попередній огляд набору даних.

Бібліотеки, що було використано, включають `Numpy`, яка призначена для роботи з багатовимірними масивами, матрицями, лінійною алгеброю та статистичними обчисленнями. `Pandas` використовується для завантаження, обробки та аналізу табличних даних у форматі `DataFrame`. Бібліотека `String` допомагає виконувати операції над текстовими даними, такі як видалення пунктуації та спеціальних символів. `Spacy` застосовується для обробки природної мови (NLP), зокрема для токенізації, лематизації та видалення стоп-слів. Для створення графіків, візуалізації даних і представлення результатів використовується `matplotlib.pyplot`. Бібліотека `Warnings` служить для приглушення попереджень, що забезпечує чистий і зрозумілий вивід.

Лістинг 3.1. Імпортування бібліотек та вивід даних

```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g.
pd.read_csv)
import string # special operations on strings
import spacy # language models
from matplotlib.pyplot import imread-   from matplotlib import
pyplot as plt
from wordcloud import WordCloud
%matplotlib inline
import warnings

data = pd.read_csv(«Resume.csv»)
```

	Resumes	Category
0	Naveen Sadhu\n\n\n\n\nTitle: software develo...	React JS Developer Resume
1	Ui-Developer/ React JS Developer \n\nNAME: KRI...	React JS Developer Resume
2	SUSOVAN BAG \n\nSeeking a challenging po...	React JS Developer Resume
3	SHAIK ABDUL SHARUK \n\n2 years' Experience i...	React JS Developer Resume
4	MD KHIZARUDDIN RAUF \n\n It EXPERIENCE \n\n ...	React JS Developer Resume
...	...	...
74	\n [pic]\n [pic]\n\n • 3.3 years of IT ex...	Workday resume
75	JYOTI VERMA\n\n\n\n\nPROFESSIONAL SUMM...	Workday resume
76	G Himaja\n\n ...	Workday resume
77	\nName : Naresh Babu Cherukuri\n\n\nObjective...	Workday resume
78	\n Hari Krishna ...	Workday resume

Рис. 3.1. Виведення початкових даних

Після імпортування даних було виконано перевірку набору даних на наявність пропущених значень та надано загальну інформацію про його структуру:

За допомогою команди `data.isnull().sum()` було проаналізовано кожен стовпець набору даних на наявність пропущених значень. Це дозволило виявити стовпці з можливими пропусками для подальшої обробки (рис. 3.2.).

```
Resumes      0  
Category      0  
dtype: int64
```

Рис. 3.2. Перевірка наявності порожніх значень

Використовуючи команду `data.info()`, було отримано загальну інформацію про набір даних (рис. 3.3.), зокрема:

- Загальну кількість рядків і стовпців.
- Імена стовпців.
- Типи даних у кожному стовпці.
- Кількість ненульових значень у кожному стовпці.

Ця інформація дає змогу зрозуміти типи даних та перевірити їхню цілісність перед подальшою обробкою.

```
<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 79 entries, 0 to 78  
Data columns (total 2 columns):  
#   Column      Non-Null Count  Dtype  
---  ---  
0   Resumes     79 non-null    object  
1   Category    79 non-null    object  
dtypes: object(2)  
memory usage: 1.4+ KB
```

Рис. 3.3. Загальна інформація про набір даних

Наступним кроком підготовки текстових даних було виконання перевірки наявності непотрібних слів у наборі даних.

Лістинг 3.2. Перевірка наявності непотрібних слів

```
data.Resumes.str.contains('https://').value_counts() / len(data)  
data.Resumes.str.contains('@').value_counts() / len(data)
```

Код `data.Resumes.str.contains('https://').value_counts() / len(data)` підраховує частку записів у стовпці `Resumes`, що містять посилання, які починаються з `https://`. Цей аналіз допомагає оцінити, скільки резюме містять посилання на сторонні ресурси.

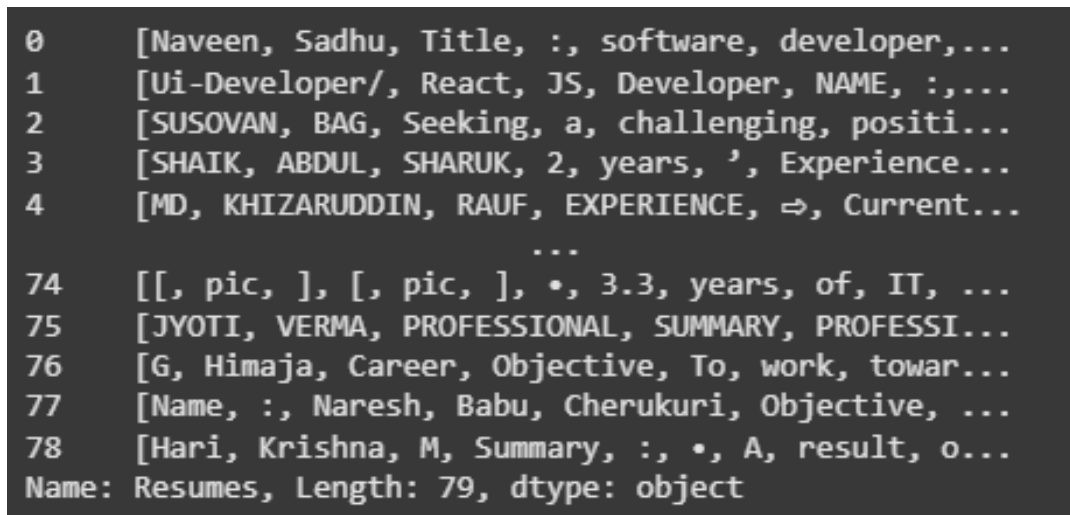
Код `data.Resumes.str.contains('@').value_counts() / len(data)` визначає частку записів, які містять символ `@`, що може свідчити про присутність електронних адрес у тексті. Це дозволяє оцінити, скільки записів можуть містити персональні або контактні дані.

Обидві перевірки допомагають визначити обсяг тексту, який потребує очищення від зайвих деталей, перш ніж переходити до подальшого аналізу.

Одним з найважливіших кроків є виконання токенизації. Спочатку імпортуються необхідні функції для токенизації з бібліотеки `nltk`. Використовуються функції `word_tokenize` та `TweetTokenizer` для обробки тексту. За допомогою `data.Resumes.map(word_tokenize)` кожне резюме в стовпці `Resumes` розбивається на окремі токени (слова) за допомогою функції `word_tokenize`. Результат початкової токенизації зображено на рис. 3.4.

Лістинг 3.3. Імпортування бібліотек для токенизації та виконання початкової токенизації

```
from nltk import word_tokenize
from nltk.tokenize import word_tokenize
tokens = data.Resumes.map(word_tokenize)
```



```
0      [Naveen, Sadhu, Title, :, software, developer,...
1      [Ui-Developer/, React, JS, Developer, NAME, :,...
2      [SUSOVAN, BAG, Seeking, a, challenging, positi...
3      [SHAIK, ABDUL, SHARUK, 2, years, ', Experience...
4      [MD, KHIZARUDDIN, RAUF, EXPERIENCE, ⇨, Current...
...
74     [[, pic, ], [, pic, ], •, 3.3, years, of, IT, ...
75     [JYOTI, VERMA, PROFESSIONAL, SUMMARY, PROFESSI...
76     [G, Himaja, Career, Objective, To, work, towar...
77     [Name, :, Naresh, Babu, Cherukuri, Objective, ...
78     [Hari, Krishna, M, Summary, :, •, A, result, o...
Name: Resumes, Length: 79, dtype: object
```

Рис. 3.4. Результат початкової токенизації

Текст очищається від зайвих пробілів на початку та в кінці кожного резюме за допомогою методу `strip()`. Після цього видаляються всі пусті рядки за допомогою фільтрації. Це важливо для того, щоб не обробляти порожні дані.

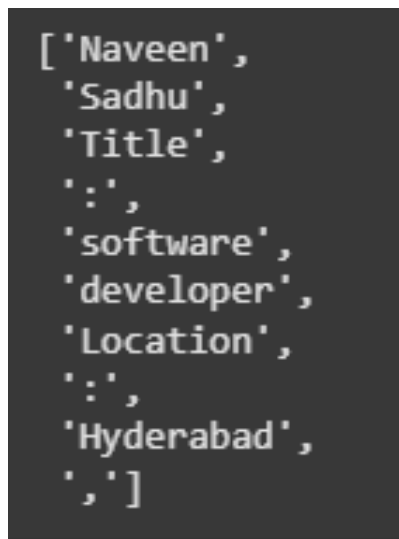
Лістинг 3.4 Очищення тексту від зайвих пробілів та пустих рядків

```
data = [Text.strip() for Text in data.Resumes]
data = [Text for Text in data if Text]
```

Далі було виконано токенизацію за допомогою TweetTokenizer. Функція TweetTokenizer використовується для токенизації тексту, що містить хештеги або інші символи, які часто зустрічаються в текстах (наприклад, в соціальних мережах). Зокрема, параметр strip_handles=True дозволяє видалити хендли (псевдоніми користувачів). Після токенизації перші 10 елементів виводяться за допомогою review_tokens[0:10] (рис 3.5.).

Лістинг 3.5. Токенизація за допомогою TweetTokenizer

```
from nltk.tokenize import TweetTokenizer
tknzs = TweetTokenizer(strip_handles=True)
review_tokens = tknzs.tokenize(review)
review_tokens[0:10]
```



```
['Naveen',
 'Sadhu',
 'Title',
 ':',
 'software',
 'developer',
 'Location',
 ':',
 'Hyderabad',
 ',']
```

Рис. 3.5. – Результат токенизації за допомогою TweetTokenizer

Для подальшої роботи текст очищається від знаків пунктуації. Це робиться для того, щоб зберегти лише слова.

Лістинг 3.6. Видалення знаків пунктуації

```
no_punc_text =
review_tokens_text.translate(str.maketrans('', '', string.punctuatio
n))
no_punc_text[0:200]
```

Рис. 3.6. Результат видалення знаків пунктуації

Далі застосовується регулярний вираз для видалення всіх посилань в тексті. Це необхідно для того, щоб не зберігати зайву інформацію, таку як веб-адреси.

Лістинг 3.6. Видалення посилань

```
import re
no_url_text = re.sub(r'http\S+', '', no_punc_text)
no_url_text[0:500]
```

Наступним етапом після успішного виконання прибирання лишніх слів є видалення стоп-слів з токенизованого тексту. Для виконання цього етапу імпортуються бібліотеки з пакету `nlTK`, зокрема функція `stopwords` для завантаження списку стоп-слів. Для роботи з токенами та стоп-словами завантажуються два ресурси з бібліотеки `nlTK`:

- `nlTK.download('punkt')` – завантажує ресурс для токенизації.
- `nlTK.download('stopwords')` – завантажує список англійських стоп-слів.

За допомогою `stopwords.words('english')` отримується список англійських стоп-слів. Це стандартний набір слів, які зазвичай не несуть інформації для аналізу, тому їх слід виключити з тексту. Далі створюється новий список токенів `no_stop_tokens`, в якому з оригінальних токенів `text_tokens` залишаються тільки ті, які не входять до списку стоп-слів (рис. 3.7.).

Лістинг 3.6. Видалення стоп-слів

```
import nlTK
from nlTK.corpus import stopwords

nlTK.download('punkt')

nlTK.download('stopwords')

my_stop_words = stopwords.words('english')
no_stop_tokens = [word for word in text_tokens if not word in
my_stop_words]
```

```
['naveen',  
 'sadhu',  
 'title',  
 'software',  
 'developer',  
 'location',  
 'hyderabad',  
 'india',  
 'professional',  
 'summary',  
 '1',  
 'year',  
 'overall',  
 'it',  
 'experience',  
 'html',  
 'sql',  
 'reactjs',  
 'nodejs',  
 'experience',  
 'working',  
 'application',  
 'development',  
 'testing',  
 'tool',  
 'hard',  
 'working',  
 'authentic']
```

Рис. 3.7. Результат видалення стоп-слів

Після завершення етапу стемінгу було застосовано лематизацію до текстового набору даних. Вона схожа на стемінг, але на відміну від стемінгу, лематизація враховує граматичні контексти слів і забезпечує точнішу основу для подальшого аналізу. У даному випадку застосовується лематизація за допомогою бібліотеки spaCy, яка надає потужний інструмент для обробки природних мов. Спочатку створюється об'єкт `doc`, який містить текст, що потребує лематизації. Потім для кожного токена (слова) в об'єкті `doc` застосовується метод `lemma_`. Це дозволяє отримати лемму для кожного слова. Результат лематизації зображено на рис. 3.8.

Лістинг 3.7. Виконання лематизації

```
lemmas = [token.lemma_ for token in doc]  
clean_review = ' '.join(lemmas)
```

```
'naveen sadhu title software developer location hyderabad india professional summary 1 year overall it experience html sql reactjs nodejs experience work application development testing tool hard work enthusiastic excellent communication interpersonal analytical skill strong ability perform part team exceptional ability learn new concept education bachelor technology computer science engineering marri laxman reddy institute technology management affiliate jntuh skill technology mern stack framework framework 7 ide eclipse visual studio code database sql mongo database operating system window unix professional experience current project client inline 4 usa organization edvenswa tech pvt ltd technology mern stack html css bootstrap description the project goal provide whenever user want servicing bike go directly application book slot bike servicing why whenever customer problem directly book service service provider take bike servicing contribution understand implement feature negotiate scope resolve conflict priority translate requirement meaningful story team deliver application deployment develop optimize code per requirement personal detail name naveen sadhu dob 2203 1997 address hyderabad language know english hindi telugu marital status single yours sincerely naveen sadhu uideveloper react js developer name krishna professional summary over 32 year extensive experience react js web developer 1 year experience react js software developer extensive experience develop web page use html xml css javascript react js redux json experience phase sdlc like requirement analysis implementation maintenance extensive experience agile scrum extensive knowledge develop single page application spa good expertise analyze document object model dom layout dom function java script function cascade style across crossbrowser
```

Рис. 3.8. Результат лематизації

Після виконання всіх етапів очищення тексту, таких як видалення непотрібних слів, токенизація, лематизація та видалення стоп-слів, дані готові до наступного етапу аналізу. Очищені тексти тепер представляють собою структуровану інформацію, яку можна використовувати для подальшої обробки. Наступним важливим кроком є перетворення тексту на числові ознаки, що дозволяє здійснити подальший аналіз даних, використовуючи алгоритми машинного навчання.

Векторизація тексту – це процес перетворення текстових даних у числові вектори, що дає змогу системам аналізу працювати з ними як з кількісними значеннями. Один із таких методів векторизації – CountVectorizer, який перетворює текст у матрицю, де кожен елемент відповідає кількості входжень конкретного слова в текстах. Цей етап є важливим, оскільки дає змогу машинним алгоритмам працювати з текстами в числовій формі для подальшої класифікації, кластеризації чи іншого аналізу текстових даних.

У цьому випадку використовується метод CountVectorizer з параметром `ngram_range=(1,3)`. Це означає, що векторизація здійснюється для біграм (2 слова) і триграм (3 слова) поряд з унікальними словами (уніграми), що дозволяє враховувати не лише окремі слова, але й їх комбінації, що можуть бути важливими для розуміння контексту тексту.

- `analyzer='word'` вказує, що векторизація буде здійснюватися на рівні слів.
- `ngram_range=(1,3)` дозволяє створювати з тексту одно-, дво- та триграмні фрази.

– `max_features=4000` обмежує кількість унікальних фраз (слова та їх комбінації) до 4000 найбільш часто зустрічаються елементів.

За допомогою цього методу `bow_matrix_ngram` (Bag-of-Words матриця) буде містити векторизовані представлення текстів, де кожен рядок представляє один документ (резюме), а кожен стовпчик – частоту появи певної n-грами в цьому документі.

Лістинг 3.8. Виконання векторизації

```
cv_ngram_range =  
CountVectorizer(analyzer='word', ngram_range=(1, 3), max_features=400  
0)  
bow_matrix_ngram = cv_ngram_range.fit_transform(lemmas)
```

Для аналізу частоти слів у очищених резюме, зосередженого на визначенні найбільш поширених слів із виключенням стоп-слів та розділових знаків, був створений наступний код.

Лістинг 3.9. Аналіз частоти слів

```
oneSetofStopWords = set(stopwords.words('english')+['`', '>', '<'])  
totalWords = []  
Sentences = data['Absolute_Clean_Resumes'].values  
cleanedSentences = ««  
for records in Sentences:  
    cleanedSentences += records  
    required_words = nltk.word_tokenize(records)  
    for word in required_words:  
        if word not in oneSetofStopWords and word not in  
string.punctuation:  
            totalWords.append(word)  
wordfreqdist = nltk.FreqDist(totalWords)  
mostcommon = wordfreqdist.most_common(100)
```

Спершу формується набір стоп-слів, до якого входять стандартні англійські слова, що не несуть смислового навантаження. Цей набір використовується для фільтрації слів, які не будуть враховані під час аналізу.

Далі очищені резюме, що зберігаються в колонці 'Absolute_Clean_Resumes' датафрейму, зчитуються в список речень. Для кожного речення виконуються наступні кроки:

- Токенізація тексту.
- Перевірка кожного слова на наявність у списку стоп-слів та серед розділових знаків. Якщо слово не є стоп-словом і не є розділовим знаком, воно додається до списку 'totalWords'.

Після збору всіх слів з резюме створюється частотний розподіл за допомогою функції 'nltk.FreqDist()'. Це дозволяє обчислити кількість появ кожного слова в тексті. Метод 'most_common(100)' отримує 100 найчастіше вживаних слів (рис. 3.9).

```
[('experience', 602),  
 ('application', 531),  
 ('server', 501),  
 ('report', 472),  
 ('peoplesoft', 443),  
 ('workday', 427),  
 ('using', 395),  
 ('integration', 368),  
 ('project', 365),  
 ('data', 332),  
 ('database', 300),  
 ('process', 290),  
 ('system', 280),  
 ('business', 271),  
 ('web', 241),  
 ('tool', 239),  
 ('sql', 237),  
 ('client', 222),  
 ('worked', 219),  
 ('role', 214),  
 ('knowledge', 210),  
 ('service', 206),  
 ('requirement', 202),  
 ('environment', 201),  
 ('involved', 189),  
 ('team', 187),  
 ('skill', 185),  
 ('like', 180),  
 ('support', 178),  
 ('technology', 176),  
 ('hcm', 173),  
 ('good', 169),  
 ('...', 163)]
```

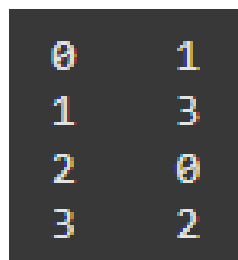
Рис. 3.9. Результат аналізу частоти слів

Наступний етап передбачає перетворення текстових даних у числовий формат для подальшого використання в алгоритмах машинного навчання, які працюють лише з числовими значеннями. Для цього застосовується метод кодування категорій за допомогою класу `LabelEncoder` з бібліотеки `sklearn`. У процесі роботи визначається список колонок, які потрібно закодувати, зокрема колонка `Category`, що містить категорії резюме. Далі для кожної колонки у списку виконується кодування, під час якого кожна унікальна текстова категорія отримує числове представлення (рис. 3.10). Ці числові значення замінюють текстові дані у колонці, створюючи підготовлений набір даних для аналізу або навчання моделей.

Лістинг 3.10. Перетворення текстових даних у числовий формат

```
from sklearn.preprocessing import LabelEncoder

var_mod = ['Category']
le = LabelEncoder()
for i in var_mod:
    data[i] = le.fit_transform(data[i])
targetCounts =
data['Category'].value_counts().reset_index()['Category']
targetLabels =
data['Category'].value_counts().reset_index()['index']
```



0	1
1	3
2	0
3	2

Рис. 3.10. Результат перетворення текстових даних у числовий формат

Після даного етапу здійснюється векторизація текстових даних за допомогою TF-IDF (Term Frequency-Inverse Document Frequency). Це важливий крок у підготовці даних для аналізу, який забезпечує представлення тексту у числовому вигляді, необхідному для роботи алгоритмів машинного навчання.

Спершу з текстових даних (`Absolute_Clean_Resumes`) і цільових категорій (`Category`) формуються змінні `requiredText` і `requiredTarget`. Потім створюється

об'єкт `TfidfVectorizer`, який налаштовується на врахування підтиску частоти (`sublinear_tf=True`) і видалення стоп-слів англійської мови (`stop_words='english'`). Векторизатор навчається на текстових даних методом `fit`, а текстові дані трансформуються у векторне представлення за допомогою методу `transform`, результат якого зберігається у змінній `WordFeatures`. Результат векторизації зображено на рис. 3.11.

Лістинг 3.11 Виконання векторизації текстових даних

```
requiredText = data['Absolute_Clean_Resumes'].values
requiredTarget = data['Category'].values
word_vectorizer = TfidfVectorizer(sublinear_tf=True,
stop_words='english')
word_vectorizer.fit(requiredText)
WordFeatures = word_vectorizer.transform(requiredText)
```

```
array([[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,  
       1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,  
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 3, 3, 3, 3, 3, 3, 3,  
       3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3]])
```

Рис. 3.11. Результат векторизації текстових даних

3.2 Аналіз даних із використанням моделей

Після завершення етапу збору та підготовки даних, на якому було здійснено очищення тексту, видалення непотрібної інформації та трансформацію даних у зручний для аналізу формат, слід перейти безпосередньо до аналізу даних. На даному етапі буде побудовано різноманітні моделі для подальшого дослідження та інтерпретації отриманих даних. Даний крок слід почати з візуалізації для кращого розуміння розподілу категорій та трендів, що присутні у наборі даних. Цей крок дозволить виявити можливі аномалії, а також допоможе в подальшому побудувати та налаштувати відповідні моделі для класифікації даних, що є наступним кроком на шляху до отримання корисних інсайтів.

Для відображення к-сті резюме кандидатів по кожній з категорій побудовано наступні діаграми зображені на рис. 3.12 та рис. 3.13.:

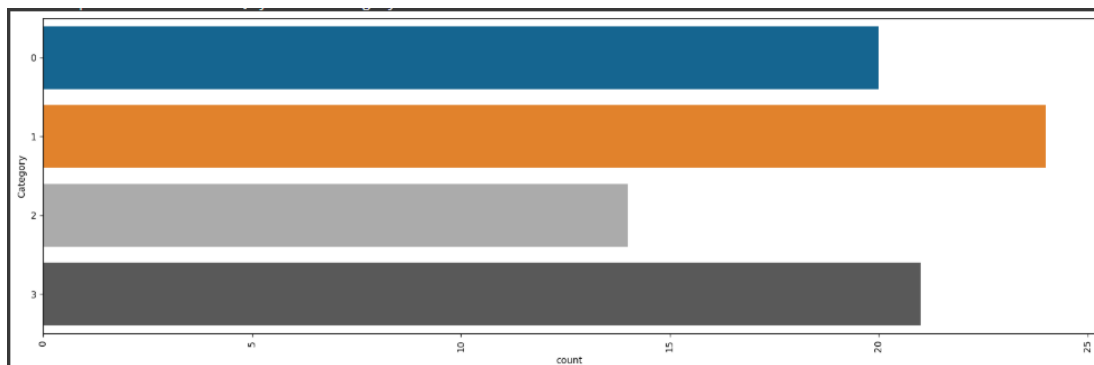


Рис. 3.12. Столпчикова діаграма к-сті резюме по категоріям

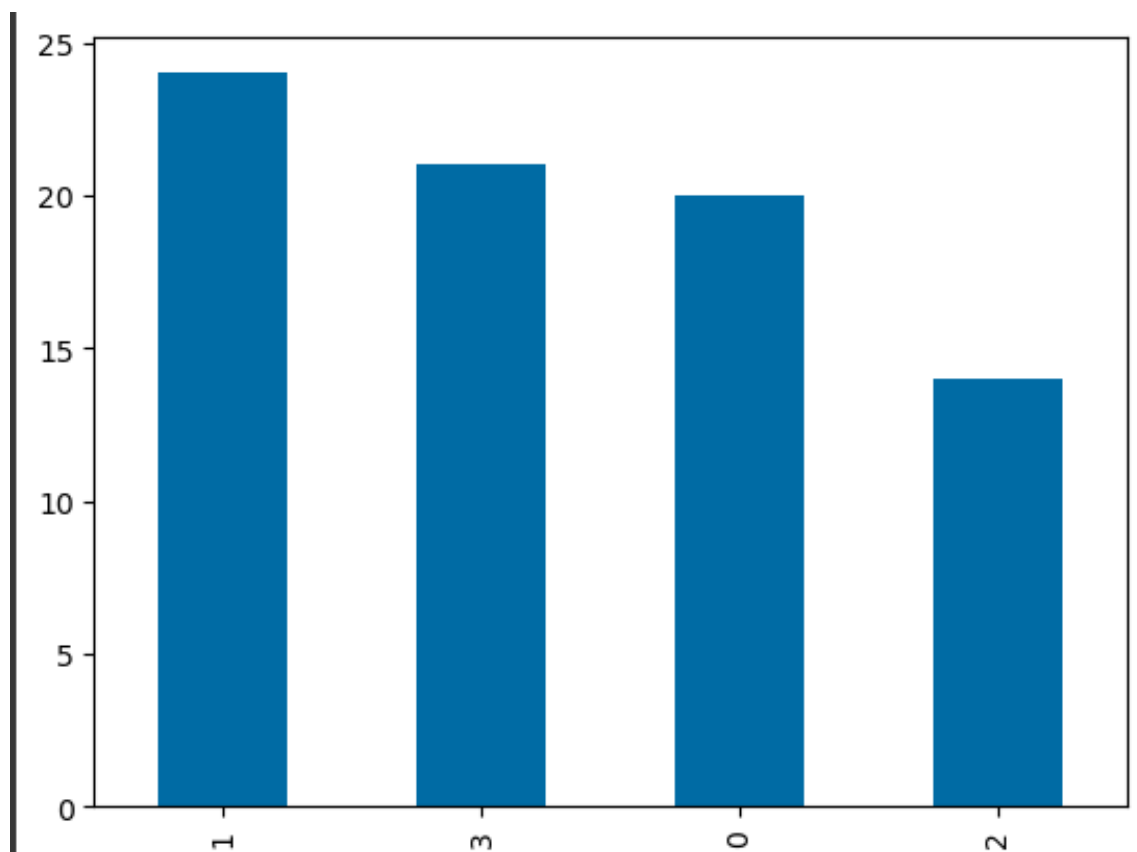


Рис. 3.13. Столпчикова діаграма к-сті резюме по категоріям

Після того як ми візуалізували розподіл категорій у наших даних, стає очевидним, які групи мають найбільшу або найменшу кількість записів. Така візуалізація допомагає краще зрозуміти структуру нашого набору даних та визначити, чи є необхідність у подальшій балансуванні класів. Тепер, маючи чітке уявлення про дані, можна перейти до побудови класифікаційних моделей, які дозволять передбачати категорії на основі текстових ознак.

Для цього буде використано різні підходи до класифікації, зокрема методи на основі машинного навчання, такі як Support Vector Classifier (SVC), K-Nearest Neighbors (KNN), Multinomial Naive Bayes, Decision Tree та Random Forest.

Після підготовки даних і їх векторизації наступним етапом є поділ на тренувальну та тестову вибірки для побудови моделей. Для цього використовується метод `train_test_split` з бібліотеки `sklearn.model_selection`, який дозволяє розділити набір даних на дві частини – тренувальну та тестову. В даному випадку 30% даних виділено для тестування, а 70% для навчання моделі. Використання параметра `stratify=requiredTarget` гарантує, що пропорції класів будуть збережені в обох вибірках.

Лістинг 3.12. Поділ даних на тренувальну та тестову частини

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(WordFeatures,
requiredTarget, random_state=42, test_size=0.3,
stratify=requiredTarget)
```

В даній роботі розглядаються два підходи до побудови моделі класифікації за допомогою Support Vector Classifier (SVC), а саме лінійна модель з ядром `linear`, та нелінійна модель з поліноміальним ядром `poly`.

Модель з лінійним ядром добре працює, коли дані можуть бути лінійно розділені. Вибір параметра `C` визначає баланс між максимізацією граничного розділення та мінімізацією помилок класифікації.

Лістинг 3.13. Побудова моделі SVC(linear)

```
from sklearn.svm import SVC
from sklearn.metrics import classification_report,
plot_confusion_matrix, accuracy_score

# Лінійна модель
svcl = SVC(kernel='linear', C=1.0)
svcl.fit(X_train, y_train)

# Прогнози
svcl_pred_train = svcl.predict(X_train)
svcl_pred_test = svcl.predict(X_test)
```

```
# Оцінка точності
svcl_train_accuracy = accuracy_score(y_train, svcl_pred_train)
svcl_test_accuracy = accuracy_score(y_test, svcl_pred_test)
```

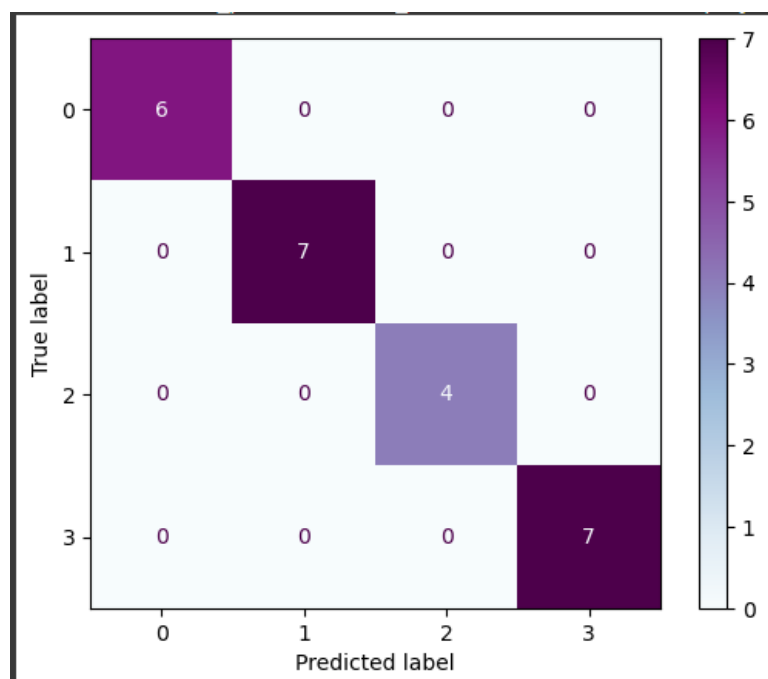


Рис. 3.14. Матриці неточностей для передбачень лінійної моделі SVC

Отже, модель показала ідеальні результати для всіх класів.

Для кожного класу (0, 1, 2, 3) точність становить 1.00 (або 100%). Це означає, що серед усіх передбачень для кожного класу не було жодного помилкового передбачення. Модель завжди правильно класифікувала об'єкти, віднесені до кожного класу.

Для кожного класу повнота також 1.00 (або 100%). Це свідчить про те, що модель знайшла всі реальні приклади кожного класу без жодних пропущених.

F1-score для кожного класу також дорівнює 1.00. Це баланс між precision і recall, і оскільки обидва показники ідеальні, F1-score також є максимальним.

Модель з нелінійним ядром дозволяє враховувати нелінійні взаємозв'язки між ознаками. Параметр degree визначає степінь полінома.

Лістинг 3.14. Побудова моделі SVC(poly)

```
svcn = SVC(kernel='poly', degree=2)
svcn.fit(X_train, y_train)
svcn_pred_train = svcn.predict(X_train)
svcn_pred_test = svcn.predict(X_test)
```

```

svcn_train_accuracy = accuracy_score(y_train, svcn_pred_train)
svcn_test_accuracy = accuracy_score(y_test, svcn_pred_test)
from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt
matrix_train = confusion_matrix(y_train, svcn_pred_train)
matrix_test = confusion_matrix(y_test, svcn_pred_test)
sns.heatmap(matrix_train, annot=True, fmt='d', cmap='coolwarm')
plt.title(«Confusion Matrix - Train Data»)
plt.xlabel(«Predicted»)
plt.ylabel(«Truth»)
plt.show()

sns.heatmap(matrix_test, annot=True, fmt='d', cmap='coolwarm')
plt.title(«Confusion Matrix - Test Data»)
plt.xlabel(«Predicted»)
plt.ylabel(«Truth»)
plt.show()

```

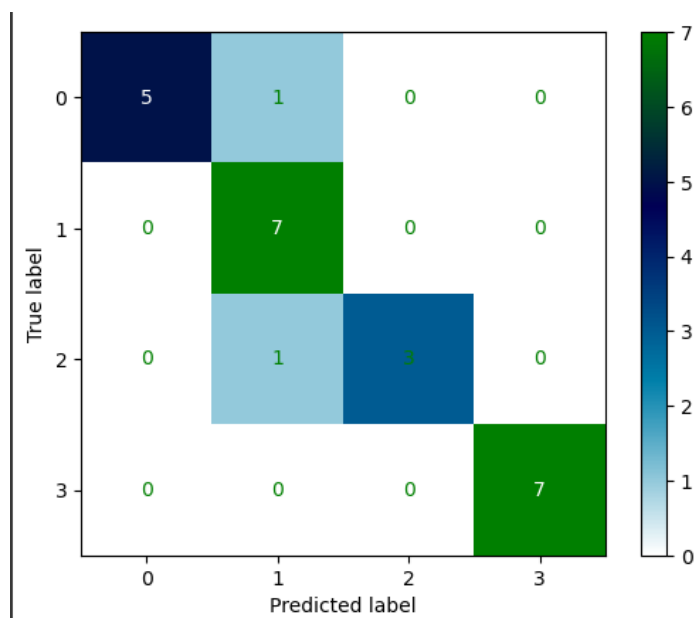


Рис. 3.15. Матриці неточностей для передбачень нелінійної моделі SVC

Для класу 0 точність становить 1.00 (100%), що означає, що модель завжди правильно класифікувала об'єкти цього класу. Повнота класу 0 становить 0.83

(83%), тобто модель знайшла 83% реальних прикладів цього класу. F1-score класу 0 дорівнює 0.91, що є гармонійним балансом між точністю і повнотою.

Для класу 1 точність становить 0.78 (78%), що означає, що 22% передбачень для цього класу були помилковими. Повнота класу 1 становить 1.00 (100%), і модель знайшла всі реальні приклади цього класу. F1-score класу 1 дорівнює 0.88, демонструючи сильну, але не ідеальну узгодженість між точністю і повнотою.

Для класу 2 точність становить 1.00 (100%), і всі передбачення цього класу були правильними. Повнота класу 2 становить 0.75 (75%), що означає, що модель не знайшла 25% реальних прикладів цього класу. F1-score класу 2 дорівнює 0.86, що свідчить про гарний, але не ідеальний результат.

Для класу 3 точність та повнота становлять 1.00 (100%), що означає, що модель не допустила жодних помилок у передбаченнях та знайшла всі реальні приклади цього класу. F1-score класу 3 дорівнює 1.00, що відображає ідеальний баланс між точністю і повнотою. Модель K-Nearest Neighbors (KNN) класифікує об'єкти на основі їх близькості до сусідніх об'єктів у багатовимірному просторі ознак.

Лістинг 3.15. Побудова моделі KNN

```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=10, p=2)
knn.fit(X_train, y_train)

knn_pred_train = knn.predict(X_train)
knn_pred_test = knn.predict(X_test)

knn_train_accuracy = accuracy_score(y_train, knn_pred_train).round(3)
knn_test_accuracy = accuracy_score(y_test, knn_pred_test).round(3)

train_cm2 = confusion_matrix(y_train, knn_pred_train)
```

```
test_cm2 = confusion_matrix(y_test, knn_pred_test)
```

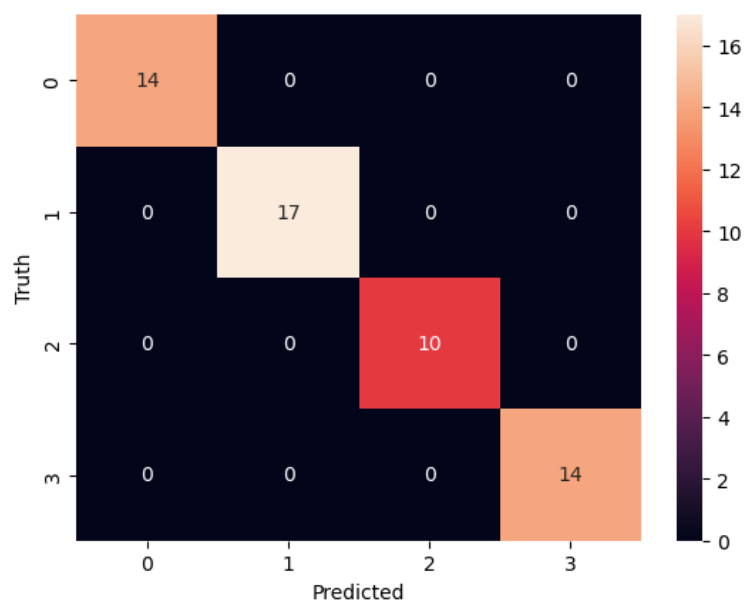


Рис. 3.16. Матриці неточностей для передбачень моделі KNN (тренувальні дані)

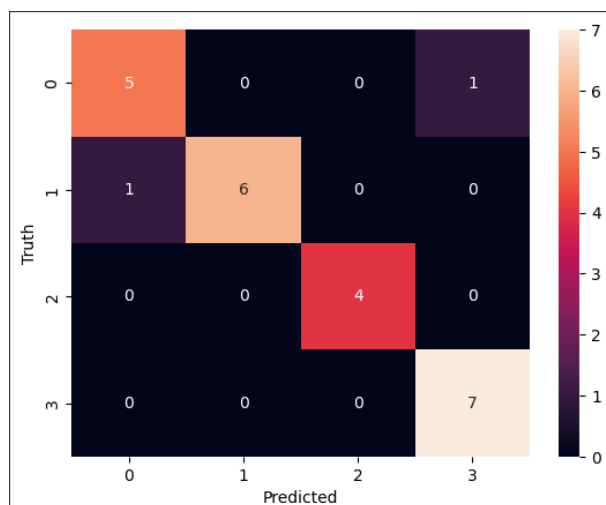


Рис. 3.17. Матриці неточностей для передбачень моделі KNN(тестувальні дані)

Модель правильно класифікувала 83% прикладів класу 0. Всі передбачення для класу 1 були правильними. Модель не зробила жодного помилкового передбачення для класу 2. 88% передбачень для класу 3 виявилися правильними.

Модель знайшла 83% усіх реальних прикладів класу 0. 86% реальних прикладів класу 1 були правильно виявлені. Всі реальні приклади класу 2 були знайдені. Модель виявила всі реальні приклади класу 3.

Баланс між точністю і повнотою класу 0 є стабільним. Високе значення свідчить про ефективну класифікацію класу 1. Клас 2 має ідеальний баланс між точністю і повнотою. Клас 3 має високу ефективність.

Naive Bayes Classifier — це простий і ефективний алгоритм класифікації, який базується на теоремі Байєса.

Лістинг 3.16. Побудова моделі Multinomial Naive Bayes

```
nb = MultinomialNB()
nb.fit(X_train, y_train)
nb_pred_train = nb.predict(X_train)
nb_pred_test = nb.predict(X_test)
nb_train_accuracy = accuracy_score(y_train, nb_pred_train)
nb_test_accuracy = accuracy_score(y_test, nb_pred_test)
train_cm3 = confusion_matrix(y_train, nb_pred_train)
test_cm3 = confusion_matrix(y_test, nb_pred_test)
```

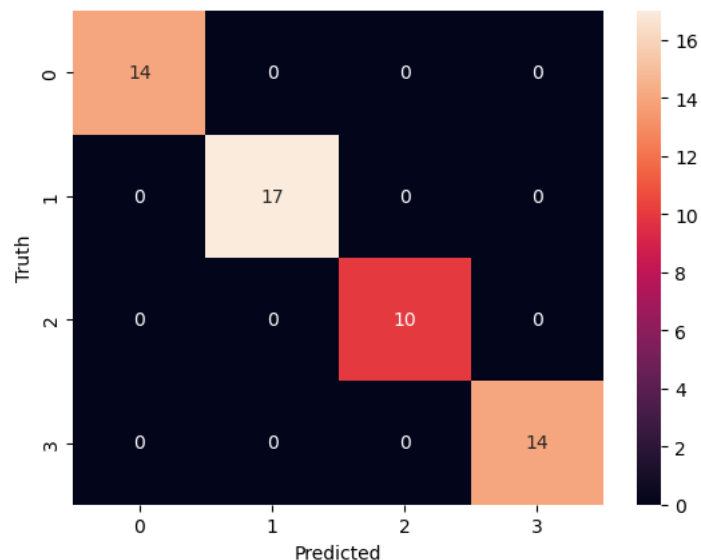


Рис. 3.18. Матриці неточностей для передбачень моделі Multinomial Naive Bayes (тренувальні дані)

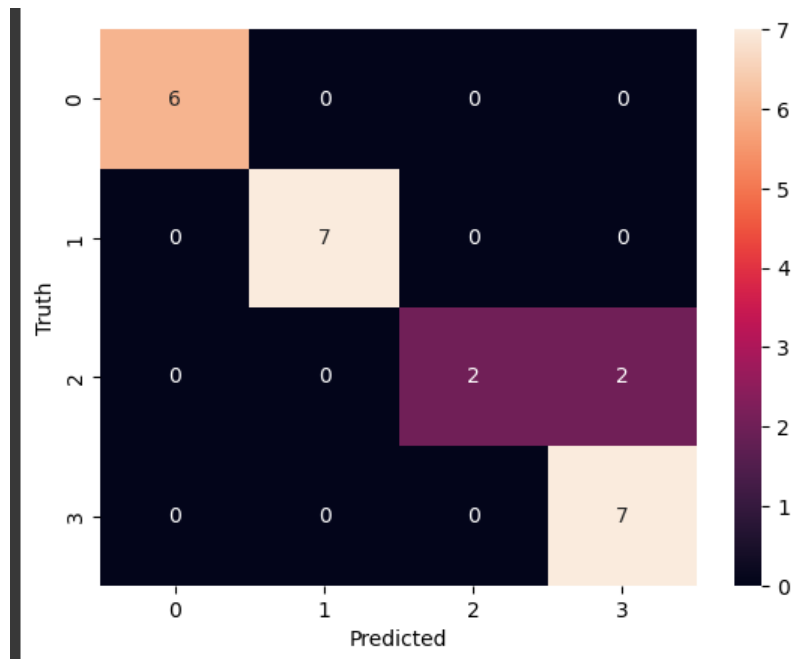


Рис. 3.19. Матриці неточностей для передбачень моделі Multinomial Naive Bayes (тестувальні дані)

Для класів 0 і 1 точність становить 1.00, що вказує на повну відсутність помилкових позитивних передбачень для цих класів. Для класу 2 точність також становить 1.00, проте це не компенсує проблеми з іншими метриками. Для класу 3 точність трохи нижча — 0.78, що означає наявність певної кількості помилкових позитивних передбачень.

Класи 0, 1 і 3 мають повноту 1.00, що свідчить про здатність моделі знаходити всі справжні приклади для цих класів. Проте для класу 2 повнота значно нижча — 0.50, що означає, що модель змогла правильно класифікувати лише половину реальних прикладів цього класу.

F1-оцінка для класів 0 і 1 становить 1.00, що свідчить про ідеальний баланс між точністю і повнотою. Для класу 3 F1-оцінка становить 0.88, що вказує на прийнятний рівень продуктивності, хоча й не ідеальний. Для класу 2 F1-оцінка знизилася до 0.67 через низьку повноту, що свідчить про проблеми з класифікацією цього класу.

Дерево рішень (Decision Tree) є алгоритмом машинного навчання, який використовує ієрархічну структуру вузлів для прийняття рішень. Воно ітеративно ділить дані на підгрупи, орієнтуючись на значення одного або

декількох ознак, щоб досягти максимальної точності. Нижче наведено аналіз отриманих метрик:

Лістинг 3.17. Побудова моделі Decision Tree

```
dt = DecisionTreeClassifier(max_depth=2)
dt.fit(X_train,y_train)
dt_pred_train = dt.predict(X_train)
dt_pred_train
dt_pred_test = dt.predict(X_test)
dt_pred_test
dt_train_accuracy = accuracy_score(y_train, dt_pred_train)
dt_test_accuracy = accuracy_score(y_test, dt_pred_test)
train_cm4 = confusion_matrix(y_train, dt_pred_train)
test_cm4 = confusion_matrix(y_test, dt_pred_test)
```

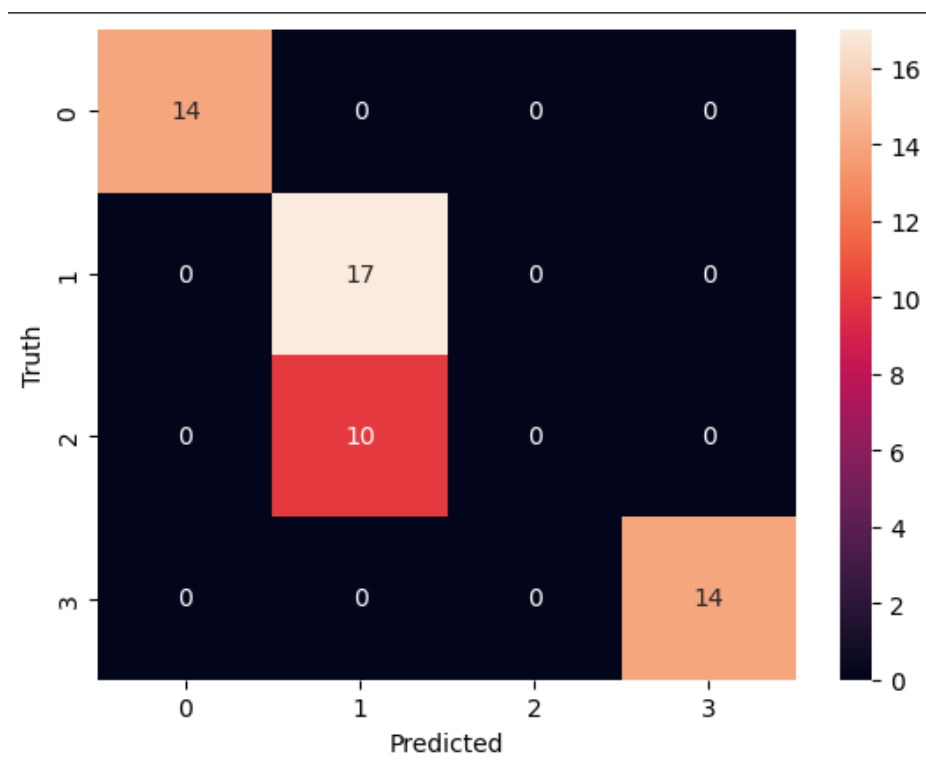


Рис. 3.20. Матриці неточностей для передбачень моделі Decision Tree (тренувальні дані)

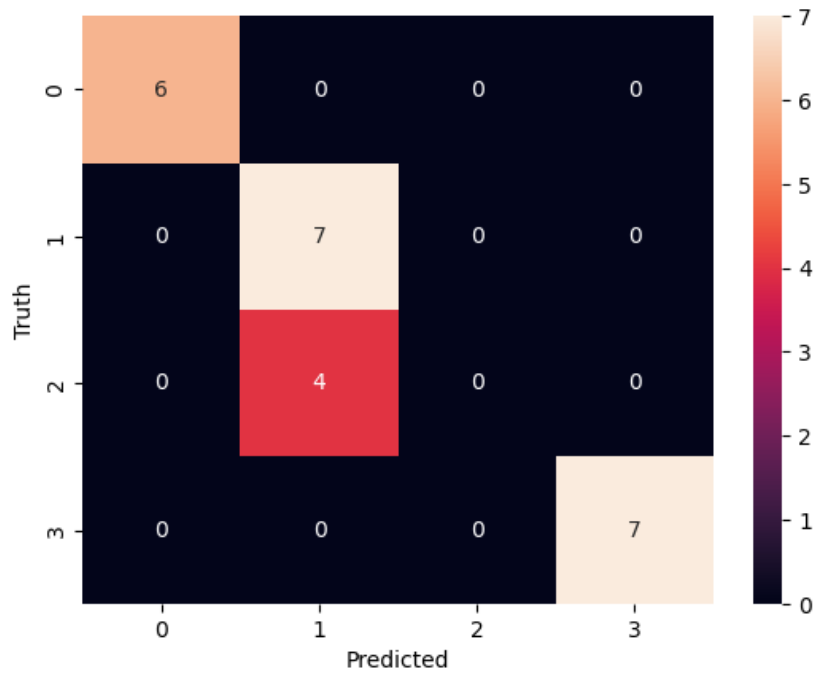


Рис. 3.21. Матриці неточностей для передбачень моделі Decision Tree (тестувальні дані)

Для класів 0 і 3 точність становить 1.00, що означає, що всі передбачення для цих класів є абсолютно правильними. Для класу 1 точність помірна — 0.64, що вказує на значну кількість помилкових позитивних передбачень. Для класу 2 точність дорівнює 0.00, що свідчить про повну відсутність коректних передбачень цього класу.

Повнота для класів 0, 1 і 3 становить 1.00, що означає, що модель успішно класифікує всі реальні приклади цих класів. Для класу 2 повнота дорівнює 0.00, що вказує на те, що модель не знайшла жодного справжнього прикладу цього класу.

F1-оцінка для класів 0 і 3 становить 1.00, що відображає ідеальний баланс між точністю і повнотою. Для класу 1 F1-оцінка складає 0.78, що вказує на прийнятну, але не ідеальну продуктивність. Для класу 2 F1-оцінка становить 0.00 через нульові значення точності та повноти.

Алгоритм Random Forest є ансамблевим методом машинного навчання, який складається з великої кількості дерев рішень (Decision Trees). Кожне дерево створюється на основі випадкового підбору підмножини даних і ознак. Рішення

приймається голосуванням усіх дерев, що сприяє підвищенню точності та зменшенню ризику перенавчання.

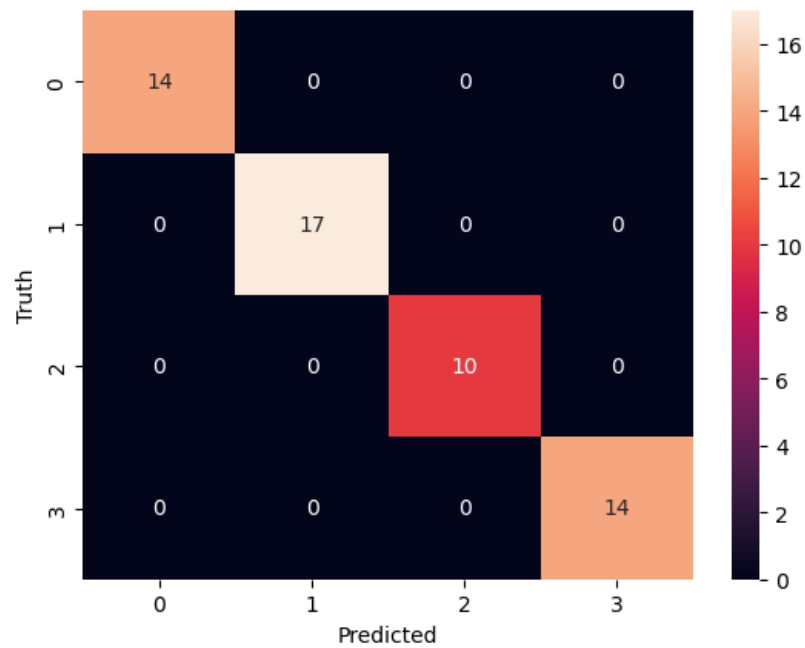


Рис. 3.22. Матриці неточностей для передбачень моделі Decision Tree (тренувальні дані)

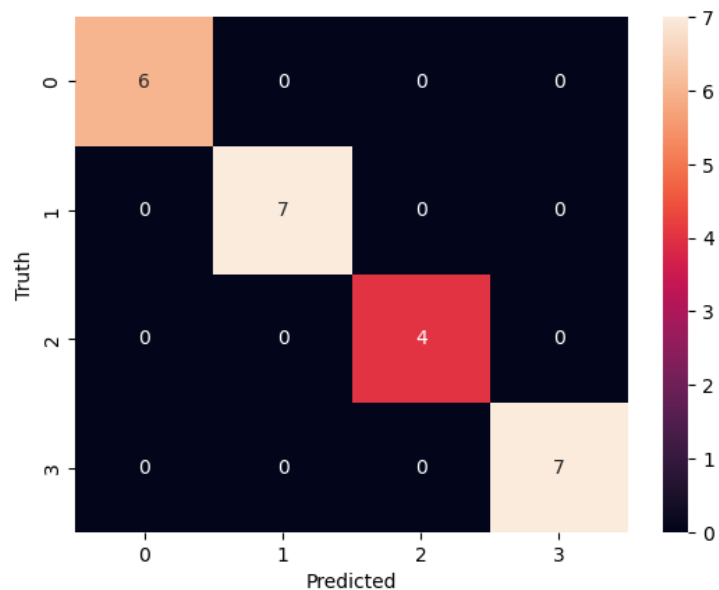


Рис. 3.23. Матриці неточностей для передбачень моделі Decision Tree (тестувальні дані)

Точність для всіх класів (0, 1, 2, 3) становить 1.00, що означає, що модель завжди правильно класифікує всі передбачення без помилкових позитивних результатів.

Повнота для всіх класів також дорівнює 1.00, що свідчить про те, що модель успішно розпізнає всі реальні приклади кожного класу без будь-яких пропусків.

F1-оцінка для всіх класів становить 1.00, що вказує на ідеальний баланс між точністю та повнотою.

3.3. Порівняльний аналіз моделей класифікації

Після розробки та тренування моделей класифікації потрібно виконати їх детальний аналіз. Слід зосередитись на оцінюванні якості кожної моделі за допомогою ключових метрик та візуалізації їх результатів. Це дозволить зрозуміти сильні та слабкі сторони кожного підходу, а також визначити, яка модель демонструє найкращу продуктивність на тестових даних.

Основна мета полягає у виявленні моделі, яка забезпечує високу точність, повноту та F1-метрику, враховуючи баланс між продуктивністю на тренувальній і тестовій вибірках.

У ході дослідження було застосовано такі моделі:

- Linear SVC (лінійний метод опорних векторів)
- Non-linear SVC (нелінійний метод опорних векторів)
- K-Nearest Neighbors (KNN)
- Naive Bayes Classifier
- Decision Tree Classifier
- Random Forest Classifier

1. Linear SVC (лінійний метод опорних векторів)

Таблиця 3.1

Метрики оцінки якості класифікації моделі SVC(linear)

Клас	Precision	Recall	F1-Score
0	1.00	1.00	1.00
1	1.00	1.00	1.00
2	1.00	1.00	1.00
3	1.00	1.00	1.00

Таблиця 3.2

Метрики оцінки якості класифікації моделі SVC(poly)

Клас	Precision	Recall	F1-Score
0	1.00	0.83	0.91
1	0.78	1.00	0.88
2	1.00	0.75	0.86
3	1.00	1.00	1.00

Нелінійна SVC показала високі результати, але не для всіх класів однаково. Хоча класи 0, 2 і 3 досягли високої точності, повноти й F1-score, клас 1 мав трохи нижчий рівень точності (78%).

Лінійна SVC показала ідеальні результати, досягнувши 100% точності, повноти та F1-метрики. Це свідчить про те, що модель повністю правильно класифікувала всі об'єкти тестової вибірки. Нелінійна SVC, у свою чергу, продемонструвала більш реалістичні показники: точність 91.67%, повнота 89.58%, F1-метрика 91.03%, що свідчить про здатність моделі адаптуватися до складніших даних. Вона краще підходить для ситуацій, де класи є нелінійно розділними, і може забезпечити більш узагальнені результати.

2. K-Nearest Neighbors (KNN)

Таблиця 3.3

Метрики оцінки якості класифікації моделі KNN

Клас	Precision	Recall	F1-Score
0	0.83	0.83	0.83
1	1.00	0.86	0.92
2	1.00	1.00	1.00
3	0.88	1.00	0.93

Модель K-Nearest Neighbors (KNN) продемонструвала високу ефективність класифікації, особливо для класів 1, 2 і 3, де значення Precision, Recall і F1-score наближаються до 1.0, що свідчить про точність та повноту передбачень. Клас 2 досяг ідеальних результатів із повним балансом між точністю і повнотою. Для класу 0 продуктивність дещо нижча, зі значеннями точності та повноти на рівні 0.83, що вказує на деякі помилки в класифікації.

Незважаючи на це, F1-score для цього класу демонструє стабільний баланс між точністю і повнотою. Загалом, модель є ефективною, особливо для класів із меншою кількістю.

3. Naive Bayes Classifier

Таблиця 3.4

Метрики оцінки якості класифікації моделі Multinomial Naive Bayes

Клас	Precision	Recall	F1-Score
0	1.00	1.00	1.00
1	1.00	1.00	1.00
2	1.00	0.550	0.67
3	0.78	1.00	0.88

Загальний результат для Naive Bayes Classifier показує хорошу продуктивність на тестових даних. Для класів 1, 2, і 3 метрики precision, recall і f1-score наближаються або дорівнюють 1.00, що вказує на високу точність і здатність моделі знаходити всі правильні передбачення для цих класів.

Для класу 0 точність, повнота і F1-оцінка становлять 0.83, що свідчить про трохи нижчу продуктивність у порівнянні з іншими класами.

Загалом модель демонструє високу точність класифікації, забезпечуючи точні передбачення для більшості класів, що робить її ефективною для задач класифікації з цими даними.

4. Decision Tree Classifier

Таблиця 3.5

Метрики оцінки якості класифікації моделі Decision Tree

Клас	Precision	Recall	F1-Score
0	1.00	1.00	1.00
1	0.64	1.00	0.78
2	0.00	0.00	0.00
3	1.00	1.00	1.00

Модель Decision Tree добре класифікує класи 0 і 3. Однак для класу 1 вона демонструє посередню точність, а для класу 2 — повну неспроможність розпізнавати об'єкти цього класу. Це може бути пов'язано з недостатньою

глибиною дерева (`max_depth=2`), яка обмежує здатність моделі навчитися складнішим залежностям у даних.

5. Random Forest Classifier

Таблиця 3.6

Метрики оцінки якості класифікації моделі Random Forest

Клас	Precision	Recall	F1-Score
0	1.00	1.00	1.00
1	1.00	1.00	1.00
2	1.00	1.00	1.00
3	1.00	1.00	1.00

Модель Random Forest із заданими параметрами (`n_estimators=250`, `max_features=0.8`, `max_depth=2`) демонструє ідеальну продуктивність на тестових даних. Висока точність, повнота та F1-оцінка свідчать про те, що модель здатна ефективно класифікувати всі класи.

Таблиця 3.7

Загальна характеристика метрик класифікаторів

Класифікатор	Точність (тренувальні дані)	Точність (тестувальні)	Precision	Recall	F1-Score
SVM (Linear)	1.00	1.00	1.00	1.00	1.00
SVM (Non_Linear)	1.00	0.916	0.94	0.85	0.91
KNN	1.00	0.917	0.92	0.92	0.92
Naive-Bayes	1.00	0.916	0.94	0.87	0.88
Decision Tree	0.81	0.83	0.65	0.75	0.69
Random Forest	1.00	1.00	1.00	1.00	1.00

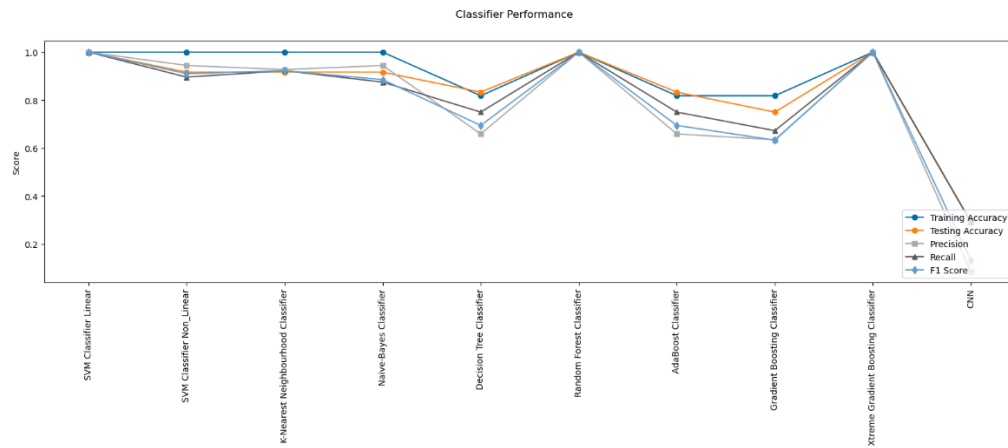


Рис. 3.24. Лінійний графік продуктивності різних класифікаторів за метриками.

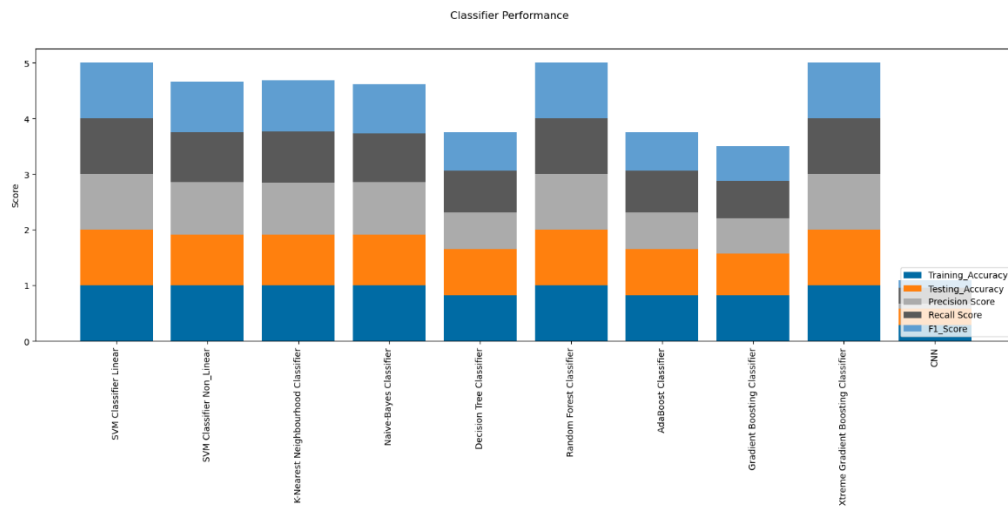


Рис. 3.25. Накопичена стопчаста діаграма продуктивності різних класифікаторів за метриками.

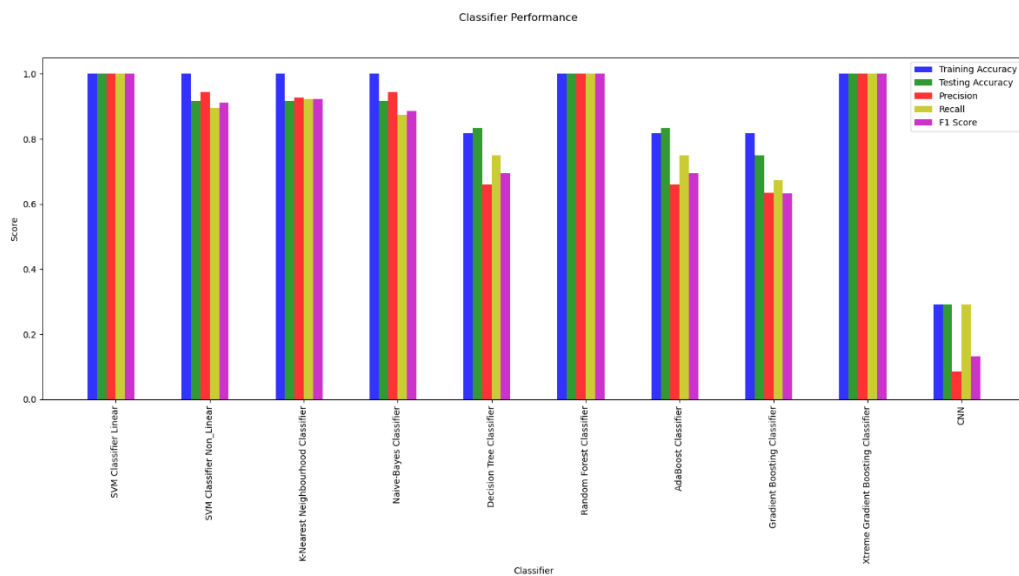


Рис. 3.26. Групована стопчаста діаграма продуктивності різних класифікаторів за метриками.

Отже, Random Forest є найкращим вибором для цієї задачі. Random Forest показав ідеальні результати по всіх класах для всіх метрик, що свідчить про його високу здатність правильно класифікувати різноманітні типи даних. Модель добре справляється з різними класами, навіть якщо класи мають різну кількість представників у навчальному наборі даних. Завдяки використанню декількох дерев рішень, Random Forest зазвичай не страждає від переобучення, що робить його більш універсальним для різних задач.

Решта класифікаторів мають ряд недолків. Модель KNN досягла високої точності для деяких класів, проте її стабільність може бути проблемною, особливо для класів з меншою кількістю даних або складними характеристиками. Це може призвести до менш ефективної класифікації в ситуаціях, коли класи мають схожі ознаки або є складними для розрізнення.

Хоча дерева рішень показали хороші результати в деяких випадках, вони виявили значні проблеми з точністю для класу 1 і не змогли класифікувати деякі приклади з класу 2. Це свідчить про те, що Decision Tree може не мати достатньої гнучкості для правильної класифікації всіх класів, а також може бути схильна до переобучення на даних.

Незважаючи на свою ефективність в загальному випадку, Naïve Bayes може мати обмеження при роботі з даними, де існують складні залежності між ознаками. Це припущення про незалежність ознак часто не відповідає реальним даним, що може знизити точність моделі.

Висновки до розділу 3

У цьому розділі було виконано комплексний аналіз текстових даних, починаючи з їх збору та підготовки і завершуючи порівняльним аналізом ефективності різних класифікаційних моделей. Спочатку було здійснено обробку даних, включаючи очищення від непотрібних та стоп-слів, а також застосування таких технік, як лематизація, стемінг та векторизація. Цей етап дозволив створити чистий і структурований набір даних, готовий до подальшої роботи з моделями.

Наступним кроком була побудова та навчання кількох класифікаційних моделей, таких як Support Vector Classifier (SVC), K-Nearest Neighbors (KNN), Multinomial Naive Bayes, Decision Tree та Random Forest. Кожна з цих моделей була протестована на тестовому наборі даних, і для кожної були обчислені відповідні метрики якості, включаючи точність, повноту, F1-оцінку та інші.

У результаті порівняльного аналізу моделей було побудовано графіки, які наочно показують ефективність кожної з них за різними критеріями. Найбільш універсальною та стабільною моделлю для даної задачі виявився Random Forest. Ця модель продемонструвала високу точність, хорошу здатність до узагальнення та стійкість до різних характеристик даних, що робить її найкращим вибором для вирішення поставленої задачі.

Таким чином, підсумовуючи результати аналізу, можна стверджувати, що Random Forest є оптимальним варіантом для подальшого використання в задачах класифікації, забезпечуючи баланс між точністю, стабільністю та універсальністю.

ВИСНОВКИ

У ході виконання цієї роботи було детально розглянуто проблему класифікації текстових даних на прикладі аналізу резюме кандидатів, а також застосування методів обробки природної мови (NLP) для вирішення цієї задачі. Робота складається з трьох основних етапів: аналізу теоретичних аспектів задачі, застосування класифікаційних моделей та порівняння їх ефективності, а також розробки інформаційної системи для класифікації.

У першому розділі було проведено детальний огляд теоретичних основ обробки текстових даних та природної мови, що дозволило зрозуміти ключові етапи та методи, які використовуються в Text Mining і NLP. Розглянуто важливість попередньої обробки текстів, яка включає видалення непотрібних слів, лематизацію, стемінг і векторизацію, що є необхідними етапами для досягнення високої точності аналізу. Зроблено висновок, що ефективна обробка текстових даних є складним і багатогранним процесом, що вимагає ретельного підходу до вибору методів та інструментів.

Другий розділ був присвячений порівнянню основних методів класифікації текстових даних, що використовуються для класифікації резюме кандидатів. Охарактеризовано п'ять основних моделей: Support Vector Classifier (SVC), K-Nearest Neighbors (KNN), Multinomial Naive Bayes, Decision Tree і Random Forest. Для кожної моделі було розглянуто її принцип роботи, переваги та недоліки. Було визначено, що кожна з моделей має свої сильні та слабкі сторони в контексті аналізу текстових даних, і вибір оптимальної моделі залежить від специфіки завдання, обсягу даних та необхідної точності.

У третьому розділі було реалізовано практичний етап роботи, який включав збір і підготовку текстових даних, їх обробку та побудову кількох класифікаційних моделей. Підготовка даних включала очищення від непотрібних та стоп-слів, лематизацію, стемінг і векторизацію, що дозволило створити структурований набір даних для подальшої роботи. Для кожної моделі було обчислено ряд метрик, таких як точність, повнота, F1-оцінка та інші, що

дозволило оцінити їх ефективність. Порівняння результатів моделей показало, що найбільш стабільною і ефективною для класифікації резюме є модель Random Forest, яка продемонструвала високі показники точності, здатність до узагальнення та стійкість до різноманітних варіацій даних.

Виконана робота показала, що процес обробки текстових даних і класифікація резюме є складною задачею, яка потребує не лише правильного вибору методів NLP, але й ретельного налаштування класифікаційних моделей для досягнення оптимальних результатів. Важливою складовою успішного вирішення цієї задачі є попередня обробка даних, яка допомагає значно покращити якість моделей. Порівняльний аналіз моделей показав, що, хоча різні підходи можуть мати свої переваги в певних випадках, для класифікації резюме виявилася найбільш ефективною модель Random Forest, яка забезпечує високу точність і стійкість до різноманітних типів даних.

Робота продемонструвала, що використання сучасних методів обробки природної мови та класифікації текстових даних є потужним інструментом для автоматизації процесів, таких як аналіз резюме. Подальший розвиток таких систем може дозволити значно підвищити ефективність рекрутингових процесів та знизити часові витрати на обробку кандидатів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Ah-Hwee T. Text Mining: The state of the art and the challenges [Електронний ресурс] / Tan Ah-Hwee. – 2000. – Режим доступу до ресурсу: https://www.researchgate.net/profile/Ah-Hwee-Tan/publication/2471634_Text_Mining_The_state_of_the_art_and_the_challenges/links/54b924610cf269d8cbf73381/Text-Mining-The-state-of-the-art-and-the-challenges.pdf.
2. Shamshiri A. Text mining and natural language processing in construction [Електронний ресурс] / A. Shamshiri, K. Rok Ryu, J. Young Park. – 2024. – Режим доступу до ресурсу: <https://www.sciencedirect.com/science/article/abs/pii/S0926580523004600>.
3. Stryker C. What is NLP (natural language processing)? [Електронний ресурс] / C. Stryker, J. Holdsworth. – 2024. – Режим доступу до ресурсу: <https://www.ibm.com/topics/natural-language-processing>.
4. Daniel C. Natural Language Processing (NLP): Definition and Principles [Електронний ресурс] / Daniel Cole. – 2023. – Режим доступу до ресурсу: <https://datascientest.com/en/natural-language-processing-definition-and-principles>
5. Jones K. Natural Language Processing: A Historical Review* [Електронний ресурс] / Karen Sparck Jones // University of Cambridge. – 1994. – Режим доступу до ресурсу: <https://aclanthology.org/www.mt-archive.info/Zampolli-1994-Sparck-Jones.pdf>.
6. Saeed N. Data mining classification algorithms: An overview [Електронний ресурс] / N. Saeed, M. Tarig, A. Tarig // IASE. – 2021. – Режим доступу до ресурсу: <https://www.sciencegate.com/IJAAS/2021/V8I2/1021833ijaas202102001.html>.
7. Setnes M. Rule-based modeling: precision and transparency [Електронний ресурс] / M. Setnes, R. Babuska, H. Verbruggen // IEEE. – 1998. – Режим

доступу

до

ресурсу:

<https://ieeexplore.ieee.org/abstract/document/661100/authors>.

8. Athey S. Machine Learning Methods That Economists Should Know About [Электронный ресурс] / S. Athey, W. Guido. – 2019. – Режим доступа до ресурсу: <https://www.annualreviews.org/content/journals/10.1146/annurev-economics-080217-053433>.
9. Holdsworth J. What is deep learning? [Электронный ресурс] / J. Holdsworth, M. Scapicchio. – 2024. – Режим доступа до ресурсу: <https://www.ibm.com/topics/deep-learning>.
10. Webster J. TOKENIZATION AS THE INITIAL PHASE IN NLP [Электронный ресурс] / J. Webster, C. Kit // City Polytechnic of Hong Kong. – 1992. – Режим доступа до ресурсу: <https://aclanthology.org/C92-4173.pdf>.
11. Singh S. A Survey on various Stemming Algorithms [Электронный ресурс] / S. Singh, R. Pateriya // INTERNATIONAL JOURNAL OF COMPUTER ENGINEERING IN RESEARCH TRENDS. – 2015. – Режим доступа до ресурсу: https://ijcert.org/ems/ijcert_papers/V2I57.pdf.
12. Plisson J. A Rule based Approach to Word Lemmatization [Электронный ресурс] / J. Plisson, N. Lavrac, D. Mladenic // Department of Knowledge Technologies Jožef Stefan Institute – Режим доступа до ресурсу: <https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=5319539616e81b02637b1bf90fb667ca2066cf14>.
13. Zhang Z. Introduction to machine learning: k-nearest neighbors [Электронный ресурс] / Zhongheng Zhang. – 2016. – Режим доступа до ресурсу: <https://pmc.ncbi.nlm.nih.gov/articles/PMC4916348/>.
14. Peterson L. K-nearest neighbor [Электронный ресурс] / Leif E. Peterson. – 2009. – Режим доступа до ресурсу: http://scholarpedia.org/article/K-nearest_neighbor.
15. Structure extended multinomial naive Bayes [Электронный ресурс] / L.Jiang, S. Wang, C. Li, L. Zhang. – 2016. – Режим доступа до ресурсу: <https://www.sciencedirect.com/science/article/abs/pii/S0020025515006921>.

16. Jindal R. Classification and Analysis of Textual data using Naive Bayes with TF-IDF [Електронний ресурс] / R. Jindal, Chingmuankim // IEEE. – 2022. – Режим доступу до ресурсу: <https://ieeexplore.ieee.org/document/10000309>.
17. Lau K. Online training of support vector classifier [Електронний ресурс] / K. Lau, Q. Wu. – 2003. – Режим доступу до ресурсу: <https://www.sciencedirect.com/science/article/abs/pii/S0031320303000384>.
18. Chang C. Training v-Support Vector Classifiers: Theory and Algorithms [Електронний ресурс] / C. Chang, C. Lin. – 2001. – Режим доступу до ресурсу: <https://direct.mit.edu/neco/article-abstract/13/9/2119/6558/Training-v-Support-Vector-Classifiers-Theory-and?redirectedFrom=fulltext>.
19. QUINLAN J. Learning Decision Tree Classifiers [Електронний ресурс] / J. R. QUINLAN // University of Sydney, Sydney, Australia. – 1996. – Режим доступу до ресурсу: <https://dl.acm.org/doi/pdf/10.1145/234313.234346>.
20. What is a decision tree? [Електронний ресурс] – Режим доступу до ресурсу: <https://www.ibm.com/topics/decision-trees>.
21. Parmar A. https://sci-hub.se/10.1007/978-3-030-03146-6_86 [Електронний ресурс] / A. Parmar, R. Katariya, V. Patel // Agrawal Institute of Engineering and Technology, Navsari, Gujarat, India. – 2019. – Режим доступу до ресурсу: https://sci-hub.se/10.1007/978-3-030-03146-6_86.
22. Ahmed Salman H. Random Forest Algorithm Overview [Електронний ресурс] / H. Ahmed Salman, A. Kalakech, A. Steiti // Babylonian Journal of Machine Learning. – 2024. – Режим доступу до ресурсу: <https://mesopotamian.press/journals/index.php/BJML/article/view/417>.
23. CUTTING-EDGE TECHNOLOGIES FOR ENHANCED EMPLOYMENT INTERACTIONS. // Технічна творчість: Збірник наукових праць / – Хмельницький: ХНУ, 2023. – С. 116–117.
24. IMPLEMENTATION OF A RECOMMENDATION SYSTEM FOR PROFESSIONAL DEVELOPMENT. // Технічна творчість: Збірник наукових праць / – Хмельницький: ХНУ, 2023. – С. 116–117.

ДОДАТКИ

Код програми

```
# -*- coding: utf-8 -*-
«««NLP final.ipynb

Automatically generated by Colab.

Original file is located at
    https://colab.research.google.com/drive/1zPF5k4j5JO6dXMHA-
    YPcrl5kr_b16yck
«««

#!pip install textract
#!apt-get install -y antiword
#!pip install spacy
#!python -m spacy download en_core_web_sm
#!pip install WordCloud

# Commented out IPython magic to ensure Python compatibility.
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g.
pd.read_csv)
import string # special operations on strings
import spacy # language models
from matplotlib.pyplot import imread
from matplotlib import pyplot as plt
from wordcloud import WordCloud
# %matplotlib inline
import warnings
warnings.filterwarnings('ignore')
```

```

#RESUME CSV FILE IS BEING EXTRACTED BY CONNECTING GOOGLE DRIVE TO
THE COLLAB

data = pd.read_csv(«Resume.csv»)
data

data.isnull().sum()

data.info()

«««# Checking unnecessary words in the data»»»

data.Resumes.str.contains('https://').value_counts() / len(data)

data.Resumes.str.contains('@').value_counts() / len(data)

«««# TOKENIZATION»»»

from nltk import word_tokenize
from nltk.tokenize import word_tokenize
tokens = data.Resumes.map(word_tokenize)
tokens

data = [Text.strip() for Text in data.Resumes] #Removes both
leading and trailing white space
data = [Text for Text in data if Text] #filters out empty string

data

review = ' '.join(data)
review

from nltk.tokenize import TweetTokenizer

```

```

tknzh = TweetTokenizer(strip_handles=True)
review_tokens = tknzh.tokenize(review)
review_tokens[0:10]

review_tokens_text = ' '.join(review_tokens)
review_tokens_text[0:500]

no_punc_text =
review_tokens_text.translate(str.maketrans('', '', string.punctuatio
n))
no_punc_text[0:200]

import re
no_url_text = re.sub(r'http\S+', '', no_punc_text)
no_url_text[0:500]

text_tokens = word_tokenize(no_url_text)
text_tokens[0:10]

«««# Removing stopwords»»»

import nltk
from nltk.corpus import stopwords
nltk.download('punkt')
nltk.download('stopwords')
my_stop_words = stopwords.words('english')
no_stop_tokens = [word for word in text_tokens if not word in
my_stop_words]
no_stop_tokens

print(no_stop_tokens[0:100])

lower_words = [Text.lower() for Text in no_stop_tokens]

```

```
lower_words
```

```
«««# Applying Stemming»»»
```

```
from nltk.stem import PorterStemmer, WordNetLemmatizer
ps = PorterStemmer()
stemmed_tokens = [ps.stem(word) for word in lower_words]
stemmed_tokens
```

```
nlp = spacy.load('en_core_web_sm')
doc = nlp(' '.join(lower_words))
doc
```

```
«««# Applying Lemmatization»»»
```

```
lemmas = [token.lemma_ for token in doc]
lemmas
```

```
clean_review = ' '.join(lemmas)
clean_review
```

```
«««# Applying Text Processing On Dataframe»»»
```

```
data = pd.read_csv('Resume.csv')
data
```

```
def remove_pattern(text, pattern_regex):
    r = re.findall(pattern_regex, text)
    for i in r:
        text = re.sub(i, '', text)

    return text
```

```
«««# **Converting into clean resumes**»»»
```

```
data['Clean_Resumes'] =  
np.vectorize(remove_pattern)(data['Resumes'], '@[\w]*')  
data.head()
```

```
«««# **Removing URLs from Clean_Resumes**»»»
```

```
clean_Resumes = []  
for index, row in data.iterrows():  
    words_without_links = [word for word in  
row.Clean_Resumes.split() if 'http' not in word]  
    clean_Resumes.append(' '.join(words_without_links))  
data['Clean_Resumes'] = clean_Resumes  
data.head()
```

```
«««# **removing spaces**»»»
```

```
data = data[data['Clean_Resumes']!='']  
data.drop_duplicates(subset=['Clean_Resumes'], keep= False)  
data = data.reset_index(drop=True)  
data.head(10)
```

```
«««# **Making text lowercase, removing text in square brackets,  
links, punctuation and remove words containing numbers**»»»
```

```
def clean_text(text):  
    text = text.lower()  
    text = re.sub('!', '', text)  
    text = re.sub('\[.*?\]', '', text)  
    text = re.sub('⇒', '', text)  
    text = re.sub(':', '', text)
```

```

text = re.sub('•', '', text)
text = re.sub('https?://\S+|www.\.\S+', '', text)
text = re.sub('<.*?>+', '', text)
text = re.sub('[%s]' % re.escape(string.punctuation), '', text)
text = re.sub('\n', '', text)
text = re.sub('\w*\d\w*', '', text)

return text

data['Clean_Resumes'] = data['Clean_Resumes'].apply(lambda x:
clean_text(x))

data.head()

«««# **Removing emojis from Clean Resumes**»»»

def remove_emoji(text):
    emoji_pattern = re.compile(«[«
                                u»\U0001F600-\U0001F64F» # emoticons
                                u»\U0001F300-\U0001F5FF» # symbols &
pictographs
                                u»\U0001F680-\U0001F6FF» # transport &
map symbols
                                u»\U0001F1E0-\U0001F1FF» # flags (iOS)
                                u»\U00002702-\U000027B0»
                                u»\U000024C2-\U0001F251»
                                »]+», flags=re.UNICODE)

    return emoji_pattern.sub(r'', text)

data['Clean_Resumes'] = data['Clean_Resumes'].apply(lambda x:
remove_emoji(x))

data

«««# **Removing stop words from Clean_Resumes and Making it
Absolute_Clean_Resumes**»»»

```

```

my_stop_words = stopwords.words('english')

cleaned_resumes = []

for index, row in data.iterrows():

    words_without_stopwords = [word for word in
row.Clean_Resumes.split() if word not in my_stop_words]

    cleaned_resumes.append(' '.join(words_without_stopwords))

data['Absolute_Clean_Resumes'] = cleaned_resumes
data.head(10)

Tokenized_Resume = data['Absolute_Clean_Resumes'].apply(lambda x:
x.split())
Tokenized_Resume.head(10)

«««# **Applying Lemmatization on Abosulte_Clean_Resumes**»»««

word_lemmatizer = WordNetLemmatizer()
Tokenized_Resume = Tokenized_Resume.apply(lambda x:
[word_lemmatizer.lemmatize(i) for i in x])
Tokenized_Resume.head(10)

for i, tokens in enumerate(Tokenized_Resume):
    Tokenized_Resume[i] = ' '.join(tokens)

data['Absolute_Clean_Resumes'] = Tokenized_Resume
data.head(10)

«««# **Applying Count Vectorizer**»»««

```

```

from sklearn.feature_extraction.text import CountVectorizer,
TfidfVectorizer

cv = CountVectorizer()

review_cv = cv.fit_transform(lemmas)

review_cv

cv.get_feature_names()

«««# **Applying Count vectorizer on NGrams**»»»

cv_ngram_range =
CountVectorizer(analyzer='word', ngram_range=(1, 3), max_features=400
0)

bow_matrix_ngram = cv_ngram_range.fit_transform(lemmas)

print(cv_ngram_range)

print(bow_matrix_ngram)

data.drop(['Resumes', 'Clean_Resumes'], axis=1, inplace=True)
data

oneSetofStopWords = set(stopwords.words('english')+['`', '»', '«'])
totalWords = []
Sentences = data['Absolute_Clean_Resumes'].values
cleanedSentences = ««
for records in Sentences:
    cleanedSentences += records
    required_words = nltk.word_tokenize(records)
    for word in required_words:
        if word not in oneSetofStopWords and word not in
string.punctuation:

```

```

totalWords.append(word)

wordfreqdist = nltk.FreqDist(totalWords)
mostcommon = wordfreqdist.most_common(100)

wordfreqdist

mostcommon

from sklearn.preprocessing import LabelEncoder

var_mod = ['Category']
le = LabelEncoder()
for i in var_mod:
    data[i] = le.fit_transform(data[i])

data.Category.value_counts()

data.info()

targetCounts =
data['Category'].value_counts().reset_index()['Category']
targetLabels =
data['Category'].value_counts().reset_index()['index']
targetCounts
targetLabels

requiredText = data['Absolute_Clean_Resumes'].values
requiredTarget = data['Category'].values

word_vectorizer = TfidfVectorizer(sublinear_tf=True,
stop_words='english')
word_vectorizer.fit(requiredText)
WordFeatures = word_vectorizer.transform(requiredText)

```

```

requiredTarget

WordFeatures

word_vectorizer

«««# Sentiment analysis»»»

#!pip install textblob

from textblob import TextBlob

# compute the sentiment polarity and subjectivity for each resume
text

data['polarity'] = data['Absolute_Clean_Resumes'].apply(lambda x:
TextBlob(x).sentiment.polarity)

data['subjectivity'] = data['Absolute_Clean_Resumes'].apply(lambda
x: TextBlob(x).sentiment.subjectivity)

# compute the average polarity and subjectivity for each category
categories = data['Category'].unique()
for category in categories:
    mask = data['Category'] == category
    avg_polarity = data[mask]['polarity'].mean()
    avg_subjectivity = data[mask]['subjectivity'].mean()
    print('Average sentiment polarity for category {}:
{: .2f}'.format(category, avg_polarity))
    print('Average sentiment subjectivity for category')

«««# Topic modeling»»»

#!pip install pyLDAvis

```

```

import pandas as pd
import gensim
from gensim.corpora import Dictionary
from gensim.models.ldamodel import LdaModel
from gensim.models.coherencemodel import CoherenceModel
import pyLDAvis.gensim_models

# Tokenize the text
documents = data['Absolute_Clean_Resumes'].apply(lambda x:
x.split())

# Create dictionary
dictionary = Dictionary(documents)

# Filter out words that occur too frequently or too rarely
dictionary.filter_extremes(no_below=5, no_above=0.5)

# Create corpus
corpus = [dictionary.doc2bow(doc) for doc in documents]

# Train LDA model
num_topics = 10
lda_model = LdaModel(corpus=corpus,
                      id2word=dictionary,
                      num_topics=num_topics,
                      random_state=42,
                      chunksize=100,
                      passes=10,
                      alpha='auto',
                      per_word_topics=True)

# Compute coherence score
coherence_model_lda = CoherenceModel(model=lda_model,
texts=documents, dictionary=dictionary, coherence='c_v')
coherence_lda = coherence_model_lda.get_coherence()
print('Coherence Score: ', coherence_lda)

```

```

# Visualize topics

vis = pyLDAvis.gensim_models.prepare(lda_model, corpus,
dictionary)

pyLDAvis.display(vis)

# Print top topics and their keywords
for idx, topic in lda_model.print_topics(-1):
    print('Topic: {} \nWords: {}'.format(idx, topic))

««««# VIZ»»»»

import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore')
import seaborn as sns
plt.figure(figsize=(20,6))
plt.xticks(rotation=90)
sns.countplot(y=«Category», data=data)

from matplotlib.gridspec import GridSpec
import numpy as np
targetCounts = data['Category'].value_counts()
targetLabels = data['Category'].unique()
# Make square figures and axes
plt.figure(1, figsize=(8,8))
the_grid = GridSpec(2, 2)

cmap = plt.get_cmap('coolwarm')
colors = [cmap(i) for i in np.linspace(0, 1, 3)]
plt.subplot(the_grid[0, 1], aspect=1, title='CATEGORY
DISTRIBUTION')

source_pie = plt.pie(targetCounts, labels=targetLabels,
autopct='%1.1f%%', shadow=True, colors=colors)

```

```

plt.show()

data['Category'].value_counts(sort=True).plot.bar()

«««# MODEL BUILDING»»»

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(WordFeatures,

requiredTarget, random_state=42, test_size=0.3,
stratify=requiredTarget)

X_train

X_test

X_train.shape

X_test.shape

y_train.shape

y_test.shape

y_train

y_test

from sklearn.metrics import f1_score, precision_score,
confusion_matrix, recall_score, accuracy_score,
classification_report

from sklearn.svm import SVC

```

```

from sklearn.ensemble import RandomForestClassifier,
AdaBoostClassifier, GradientBoostingClassifier

from sklearn.naive_bayes import MultinomialNB

from sklearn.tree import DecisionTreeClassifier

from xgboost import XGBClassifier


«««# SUPPORT VECTOR MACHINE CLASSIFICATION (SVC)


## Linear functions
«««

from sklearn.metrics import f1_score, precision_score,
confusion_matrix, recall_score, accuracy_score,
classification_report


from sklearn.svm import SVC

svcl = SVC(kernel='linear', C = 1.0)
svcl.fit(X_train,y_train)


svcl_pred

svcl_pred_train = svcl.predict(X_train)
svcl_pred_train

svcl_pred_test = svcl.predict(X_test)
svcl_pred_test


#train data
svcl_train_accuracy = accuracy_score(y_train,svcl_pred_train)
svcl_train_accuracy


#test data
svcl_test_accuracy = accuracy_score(y_test,svcl_pred_test)

```

```

svcl_test_accuracy

print(classification_report(y_test,svcl_pred_test))

from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(svcl, X_test, y_test, cmap='BuPu')

«««## NON-LINEAR»»»

svcn = SVC(kernel='poly', degree=2)
svcn.fit(X_train,y_train)

svcn_pred_train = svcn.predict(X_train)
svcn_pred_train

svcn_pred_test = svcn.predict(X_test)
svcn_pred_test

#On train data
svcn_train_accuracy = accuracy_score(y_train,svcn_pred_train)
svcn_train_accuracy

#on test data
svcn_test_accuracy = accuracy_score(y_test,svcn_pred_test)
svcn_test_accuracy

print(classification_report(y_test,svcn_pred_test))

matrix_train = confusion_matrix(y_train, svcn_pred_train)
matrix_train

matrix_test = confusion_matrix(y_test, svcn_pred_test)

```

```

matrix_test

from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(svcn, X_test, y_test, cmap='ocean_r')

sns.heatmap(matrix_train,annot=True)
plt.xlabel(«Predicted»)
plt.ylabel(«Truth»)

sns.heatmap(matrix_test,annot=True)
plt.xlabel(«Predicted»)
plt.ylabel(«Truth»)

accuracy_1 = round(accuracy_score(y_test,svcl_pred_test),4)
precision_1 =
round(precision_score(y_test,svcl_pred_test,average='macro'),4)
recall_1 =
round(recall_score(y_test,svcl_pred_test,average='macro'),4)
f1_1 = round(f1_score(y_test,svcl_pred_test,average='macro'),4)
accuracy_2 = round(accuracy_score(y_test,svcn_pred_test),4)
precision_2 =
round(precision_score(y_test,svcn_pred_test,average='macro'),4)
recall_2 =
round(recall_score(y_test,svcn_pred_test,average='macro'),4)
f1_2 = round(f1_score(y_test,svcn_pred_test,average='macro'),4)

#Linear
print('Accuracy Score  : ', accuracy_1)
print('Precision Score : ', precision_1)
print('Recall Score    : ', recall_1)
print('f1-Score        : ', f1_1)

#Non-linear

```

```

print('Accuracy Score  : ', accuracy_2)
print('Precision Score : ', precision_2)
print('Recall Score    : ', recall_2)
print('f1-Score        : ', f1_2)

«««# KNN»»»

from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=10, p=2)
knn.fit(X_train,y_train)

knn_pred_train = knn.predict(X_train)
knn_pred_train

knn_pred_test = knn.predict(X_test)
knn_pred_test

knn_train_accuracy =
accuracy_score(y_train,knn_pred_train).round(3)
knn_train_accuracy

knn_test_accuracy = accuracy_score(y_test,knn_pred_test).round(3)
knn_test_accuracy

print(classification_report(y_test,knn_pred_test))

train_cm2 = confusion_matrix(y_train,knn_pred_train)
train_cm2

test_cm2 = confusion_matrix(y_test,knn_pred_test)
test_cm2

```

```

sns.heatmap(train_cm2,annot=True)
plt.xlabel('Predicted')
plt.ylabel('Truth')

sns.heatmap(test_cm2,annot=True)
plt.xlabel('Predicted')
plt.ylabel('Truth')

from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(knn, X_test, y_test, cmap='CMRmap')

accuracy_3 = round(accuracy_score(y_test,knn_pred_test),4)
precision_3 = round(precision_score(y_test,knn_pred_test,average =
'macro'),4)
recall_3 = round(recall_score(y_test,knn_pred_test, average =
'macro'),4)
f1_3 = round(f1_score(y_test,knn_pred_test, average = 'macro'),4)

print('Accuracy Score  : ', accuracy_2)
print('Precision Score : ', precision_2)
print('Recall Score    : ', recall_2)
print('f1-Score       : ', f1_2)

«««# Naive-Bayes Classifier»»»

nb = MultinomialNB()
nb.fit(X_train,y_train)

nb_pred_train = nb.predict(X_train)
nb_pred_train

nb_pred_test = nb.predict(X_test)

```

```

nb_pred_test

nb_train_accuracy = accuracy_score(y_train, nb_pred_train)
nb_train_accuracy

nb_test_accuracy = accuracy_score(y_test, nb_pred_test)
nb_test_accuracy

train_cm3 = confusion_matrix(y_train, nb_pred_train)
train_cm3

test_cm3 = confusion_matrix(y_test, nb_pred_test)
test_cm3

sns.heatmap(train_cm3,annot=True)
plt.xlabel('Predicted')
plt.ylabel('Truth')

sns.heatmap(test_cm3,annot=True)
plt.xlabel('Predicted')
plt.ylabel('Truth')

plot_confusion_matrix(nb, X_test, y_test, cmap='tab20b')

print(classification_report(y_test,nb_pred_test))

accuracy_4 = round(accuracy_score(y_test,nb_pred_test),4)
precision_4 = round(precision_score(y_test,nb_pred_test,average =
'macro'),4)
recall_4 = round(recall_score(y_test,nb_pred_test, average =
'macro'),4)
f1_4 = round(f1_score(y_test,nb_pred_test, average = 'macro'),4)

```

```

print('Accuracy Score  : ', accuracy_4)
print('Precision Score  : ', precision_4)
print('Recall Score     : ', recall_4)
print('f1-Score         : ', f1_4)

«««# Decision Tree Classifier»««

dt = DecisionTreeClassifier(max_depth=2)
dt.fit(X_train,y_train)

dt_pred_train = dt.predict(X_train)
dt_pred_train

dt_pred_test = dt.predict(X_test)
dt_pred_test

dt.tree_.max_depth

dt.tree_.node_count

dt_train_accuracy = accuracy_score(y_train, dt_pred_train)
dt_train_accuracy

dt_test_accuracy = accuracy_score(y_test, dt_pred_test)
dt_test_accuracy

train_cm4 = confusion_matrix(y_train, dt_pred_train)
train_cm4

test_cm4 = confusion_matrix(y_test, dt_pred_test)
test_cm4

```

```

sns.heatmap(train_cm4,annot=True)
plt.xlabel('Predicted')
plt.ylabel('Truth')

sns.heatmap(test_cm4,annot=True)
plt.xlabel('Predicted')
plt.ylabel('Truth')

plot_confusion_matrix(dt, X_test, y_test, cmap='flare_r')

print(classification_report(y_test,dt_pred_test))

accuracy_5 = round(accuracy_score(y_test,dt_pred_test),4)
precision_5= round(precision_score(y_test,dt_pred_test,average =
'macro'),4)
recall_5 = round(recall_score(y_test,dt_pred_test, average =
'macro'),4)
f1_5 = round(f1_score(y_test,dt_pred_test, average = 'macro'),4)

print('Accuracy Score  : ', accuracy_5)
print('Precision Score : ', precision_5)
print('Recall Score    : ', recall_5)
print('f1-Score       : ', f1_5)

«««# Random Forest Classifier»»»

RFC =
RandomForestClassifier(n_estimators=250,max_features=0.8,max_depth
=2)
RFC.fit(X_train,y_train)

RFC_pred_train = RFC.predict(X_train)

```

```

RFC_pred_train

RFC_pred_test = RFC.predict(X_test)
RFC_pred_test

RFC_train_accuracy = accuracy_score(y_train,RFC_pred_train)
RFC_train_accuracy

RFC_test_accuracy = accuracy_score(y_train,RFC_pred_train)
RFC_test_accuracy

train_cm5 = confusion_matrix(y_train, RFC_pred_train)
train_cm5

test_cm5 = confusion_matrix(y_test, RFC_pred_test)
test_cm5

sns.heatmap(train_cm5,annot=True)
plt.xlabel('Predicted')
plt.ylabel('Truth')

sns.heatmap(test_cm5,annot=True)
plt.xlabel('Predicted')
plt.ylabel('Truth')

plot_confusion_matrix(RFC, X_test, y_test, cmap='plasma')

print(classification_report(y_test,RFC_pred_test))

accuracy_6 = round(accuracy_score(y_test,RFC_pred_test),4)
precision_6 = round(precision_score(y_test,RFC_pred_test,average =
'macro'),4)

```

```

recall_6 = round(recall_score(y_test,RFC_pred_test, average =
'macro'),4)

f1_6 = round(f1_score(y_test,RFC_pred_test, average = 'macro'),4)


print('Accuracy Score  : ', accuracy_6)
print('Precision Score  : ', precision_6)
print('Recall Score     : ', recall_6)
print('f1-Score         : ', f1_6)

=


import matplotlib.pyplot as plt

fig, ax = plt.subplots(figsize=(20,5))
fig.suptitle('Classifier Performance')

ax.plot(df['Classifier'], df['Training_Accuracy'], marker='o')
ax.plot(df['Classifier'], df['Testing_Accuracy'], marker='o')
ax.plot(df['Classifier'], df['Precision Score'], marker='s')
ax.plot(df['Classifier'], df['Recall Score'], marker='^')
ax.plot(df['Classifier'], df['F1_Score'], marker='d')
ax.set_ylabel('Score')
ax.legend(['Training Accuracy', 'Testing Accuracy', 'Precision',
'Recall', 'F1 Score'], loc='lower right')

plt.xticks(rotation=90)

plt.show()

import matplotlib.pyplot as plt
import numpy as np

```

```

fig, ax = plt.subplots(figsize=(20,6))
fig.suptitle('Classifier Performance')

metrics = ['Training_Accuracy', 'Testing_Accuracy', 'Precision
Score', 'Recall Score', 'F1_Score']

# Plot bars for each classifier and each metric
bottom = np.zeros(len(df))
for metric in metrics:
    ax.bar(df['Classifier'], df[metric], bottom=bottom)
    bottom += df[metric]

# Set axis labels and legend
ax.set_ylabel('Score')
ax.legend(metrics, loc='lower right')
plt.xticks(rotation=90)
plt.show()

```

```

import matplotlib.pyplot as plt
import numpy as np

```

```

fig, ax = plt.subplots(figsize=(20, 7))
fig.suptitle('Classifier Performance')

metrics = ['Training Accuracy', 'Testing Accuracy', 'Precision
Score', 'Recall Score', 'F1_Score']

bar_width = 0.1
opacity = 0.8
index = np.arange(len(df['Classifier']))

# Plot bars for each classifier and each metric
rects1 = ax.bar(index, df['Training_Accuracy'], bar_width,
alpha=opacity, color='b', label='Training Accuracy')

```

```

rects2 = ax.bar(index + bar_width, df['Testing_Accuracy'],
bar_width, alpha=opacity, color='g', label='Testing Accuracy')

rects3 = ax.bar(index + 2*bar_width, df['Precision Score'],
bar_width, alpha=opacity, color='r', label='Precision')

rects4 = ax.bar(index + 3*bar_width, df['Recall Score'],
bar_width, alpha=opacity, color='y', label='Recall')

rects5 = ax.bar(index + 4*bar_width, df['F1_Score'], bar_width,
alpha=opacity, color='m', label='F1 Score')

# Set axis labels and legend
ax.set_xlabel('Classifier')
ax.set_ylabel('Score')
ax.set_xticks(index + 2.5*bar_width)
ax.set_xticklabels(df['Classifier'], rotation=90)
ax.legend()

plt.show()

```