НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
Навчально-науковий інститут прикладного системного аналізу
Кафедра штучного інтелекту

До захисту допущено:
Завідувач кафедри
_____ Чумаченко О. Іл.
«__» _____20__ р.

Дипломна робота

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Системи та методи штучного інтелекту»
спеціальності 122 «Комп'ютерні науки»
на тему: «Автоматизована система навчання напівкерованої машини опорних векторів»

Виконав: студент IV курсу, групи КІ-91
Дідок Тарас Андрійович _____

Керівник:
професор, д.т.н. Синєглазов Віктор Михайлович _____

Консультант з економічного розділу:
доцент, к.е.н. Рощина Надія Василівна _____

Консультант з нормоконтролю:
фахівець першої категорії кафедри ШІ
Гончарук Максим Миколайович _____

Рецензент:
професор кафедри АКІК НАУ
к.т.н. Філяшкін Микола Кирилович _____

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

Київ – 2023 року

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
Навчально-науковий інститут прикладного системного аналізу
Кафедра штучного інтелекту

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 122 «Комп'ютерні науки»

Освітньо-професійна програма «Системи та методи штучного інтелекту»

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____ О. І. Чумаченко

«__» _____20__ р.

ЗАВДАННЯ

на дипломну роботу студенту

Дідку Тарасу Андрійовичу

1. Тема роботи: «Автоматизована система навчання напівкерованої машини опорних векторів», керівник роботи Синєглазов Віктор Михайлович, затверджений наказом по університету від «30» травня 2023 року №2065-с.
2. Термін подання студентом роботи 9.06.2023
3. Вихідні дані до роботи: двовимірні точкові вибірки: "місяці", що не перетинаються, "місяці", які пересікаються, вкладені кола; текстові набори даних: повідомлення "спам" та "не спам", скарги та відгуки студентів університету.
4. Зміст роботи: Аналіз предметної області, обробка літературних джерел, дослідження алгоритму методу напівкерованої машини опорних векторів та його реалізацій, розробка програмного продукту для розв'язку задачі класифікації, оцінювання та порівняльний аналіз отриманих результатів.
5. Перелік ілюстративного матеріалу: схеми нейронних мереж, опис класифікацій машинного навчання, опис методу опорних векторів та його

напівкерованої модифікації, опис наборів даних, результати роботи побудованої моделі на вибірках.

6. Консультанти розділів роботи:

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв
Економічний	Рощина Надія Василівна, канд. економ. наук, доцент		

7. Дата видачі завдання: 23 лютого 2023 року

Календарний план

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітка
1.	Огляд літератури за темою	12.03.2023	
2.	Підготовка першого розділу	16.04.2023	
3.	Підготовка другого розділу	7.05.2023	
4.	Розробка програмного продукту	18.05.2023	
5.	Підготовка третього розділу	21.05.2023	
6.	Підготовка економічної частини	24.05.2023	
7.	Оформлення розділів відповідно до нормоконтролю	26.05.2023	
8.	Оформлення дипломної роботи	28.05.2023	
9.	Підготовка презентації доповіді	28.05.2023	

Студент Тарас ДІДОК

Керівник Віктор СИНЄГЛАЗОВ

РЕФЕРАТ

Тема: “Автоматизована система навчання напівкерованої машини опорних векторів”

Дипломна робота містить: 101 с., 9 табл., 46 рис., 2 додатки, 37 джерел

КЛАСИФІКАЦІЯ, МАШИННЕ НАВЧАННЯ, НАПІВКЕРОВАНЕ НАВЧАННЯ, МЕТОД ОПОРНИХ ВЕКТОРІВ, НАПІВКЕРОВАНА МАШИНА ОПОРНИХ ВЕКТОРІВ

У роботі розглянуто та проаналізовано методи напівкерованого навчання, а саме напівкеровану машину опорних векторів та різні підходи до її реалізації. Робота обраного підходу була представлена та досліджена на практичній задачі, а саме класифікації двовимірних точкових вибірок різної форми, а також задачі бінарної та багатокласової класифікації текстів.

Об’єкт дослідження: методи напівкерованого навчання як спосіб подолання проблеми маркування даних.

Предмет дослідження: метод опорних векторів та його модифікація для задачі напівкерованого навчання.

ABSTRACT

The topic: “Automated training system for semi-supervised support vector machine”

Thesis: 101 p., 9 tabl., 46 fig., 2 appendices, 37 sources

CLASSIFICATION, MACHINE LEARNING, SEMI-SUPERVISED LEARNING, SUPPORT VECTOR MACHINE, SEMI-SUPERVISED SUPPORT VECTOR MACHINE

The work examines and analyzes semi-supervised learning methods, in particular the semi-supervised support vector machine and various approaches to its implementation. Results of applying the chosen approach were presented and examined on a practical task, namely the classification of two-dimensional datasets points of various shapes, as well as the task of binary and multi-class text classification.

Research object: semi-supervised learning methods as a way to overcome the problem of data labeling.

Research subject: the support vector machine and its modification for semi-supervised learning tasks.

ЗМІСТ

ПЕРЕЛІК ПРИЙНЯТИХ ПОЗНАЧЕНЬ ТА СКОРОЧЕНЬ

ML — Machine Learning

MSE — Mean Squared Error

RBF — Radial Basis Function

RL — Reinforcement Learning

S3VM — Semi-Supervised Support Vector Machine

S4VM — Safe Semi-Supervised Support Vector Machine

SL — Supervised Learning

SSL — Semi-Supervised Learning

SVM — Support Vector Machine

UL — Unsupervised Learning

ВСТУП

На сьогоднішній день машинне навчання є надзвичайно актуальною сферою досліджень. Методи машинного навчання дедалі частіше застосовуються в різних галузях науки, техніки, медицини, бізнесу та інших.

Однією з важливих перепон у застосуванні цих методів є проблема маркування даних, тобто необхідність співставляти наявні дані з певними мітками, що дають змогу навчити модель розпізнавати окремі класи об’єктів у вибірках чи виконувати певні операції над цими даними. Складність процесу маркування полягає у тому, що він вимагає значних витрат часу та ресурсів, а також залучення спеціалістів у відповідній області, які можуть правильно і якісно розмаркувати наявні дані.

Одним з підходів до вирішення цієї проблеми є застосування методів напівкерованого навчання. За рахунок використання як маркованих, так і немаркованих даних в процесі навчання вони дають змогу досягти досить високої точності моделі при мінімальному залученні експертів для маркування даних.

Метод опорних векторів є одним з найвідоміших та найбільш часто використовуваних методів машинного навчання. Його модифікація для використання у напівкерованому навчанні дозволяє ефективно вирішувати ряд різноманітних задач, при цьому частково вирішуючи проблему маркування даних.

Об’єктом дослідження дипломної роботи є аналіз результатів використання напівкерованих методів машинного навчання для задачі класифікації на різних варіантах вибірок даних у порівнянні з класичними методами навчання з учителем.

Предметом дослідження визначено метод опорних векторів та його модифікацію для задачі напівкерованого навчання.

Робота за своєю структурою складається з чотирьох розділів. У першому розділі буде проведено загальний огляд предметної області, а саме: поняття

машинного навчання та його застосувань; формулювання задачі напівкерованого навчання та основних його припущень; а також характерних особливостей методу опорних векторів. У другому розділі проводиться уточнена постановка задачі напівкерованої машини опорних векторів, досліджується ряд підходів до її реалізації та обирається конкретний метод для програмної реалізації. Третій розділ описує вибір засобів для програмування, а також представлення та порівняльний аналіз результатів роботи алгоритму на різних наборах вхідних даних. У четвертому розділі було проведено функціонально-вартісний аналіз застосунку.

РОЗДІЛ 1 МАШИННЕ НАВЧАННЯ ТА ЙОГО ОЗНАКИ

1.1 Машинне навчання та його використання у різних сферах життєдіяльності людини

Машинне навчання — це окрема галузь штучного інтелекту, в межах якої розробляються алгоритми та моделі, що можуть “навчатися” на основі наявної інформації і застосовувати отримані знання для вирішення задач, для яких вони не були явно запрограмованими. Системи машинного навчання досліджують та виявляють різноманітні закономірності у великих масивах даних, і використовують ці закономірності для прогнозування чи прийняття рішень при надходженні нових, раніше невідомих даних.

Алгоритми машинного навчання в тій чи іншій формі використовуються у багатьох сферах життєдіяльності людини. Серед основних напрямів їх застосування можна виділити наступні:

- Розпізнавання зображень та відео;
- Обробка природньої мови — інтерпретація мови людини та генерація зрозумілих відповідей при взаємодії з користувачем;
- Рекомендаційні системи — для створення персоналізованих рекомендацій на основі вподобань та поведінки клієнтів;
- Методи передбачень — наприклад для прогнозування погодних умов чи курсів валют;
- Алгоритми автопілоту — застосовуються у автономних транспортних засобах для прийняття рішень в залежності від показників навколишнього середовища;
- Діагностичні моделі — використовуються у медицині для аналізу даних про пацієнтів, формування діагнозу та прогнозування шляхів лікування захворювань;

- Алгоритми для аналізу релевантності повідомлень — використовуються у пошукових системах для оцінки відповідності різних результатів запиту користувача;
- Системи страхування та виявлення шахрайства;
- Системи автоматизації виробництва та контролю якості продукції

1.2 Штучні нейронні мережі, їх класифікація та структура

Нейронні мережі, також відомі як Artificial Neural Networks (ANNs) або Simulated Neural Networks (SNNs), є підрозділом машинного навчання і основою Deep Learning алгоритмів. Їх назва та структура базуються на дослідженнях людського мозку, і тому способі, яким «біологічні нейрони» надсилають один одному сигнали [1].

Штучні нейронні мережі складаються з кількох шарів вузлів (нод), які поділяються на вхідний шар, вихідний шар та велику кількість прихованих шарів. Кожен вузол, або інакше штучний нейрон, пов’язаний з іншими і має певну присвоєну вагу (weight) та граничне значення (threshold). Якщо вихідні дані на будь-якому окремому вузлі перевищать зазначене порогове значення, цей вузол «активується», надсилаючи вихідні дані до наступного шару мережі. У іншому випадку ці дані на наступні шари не передаються.

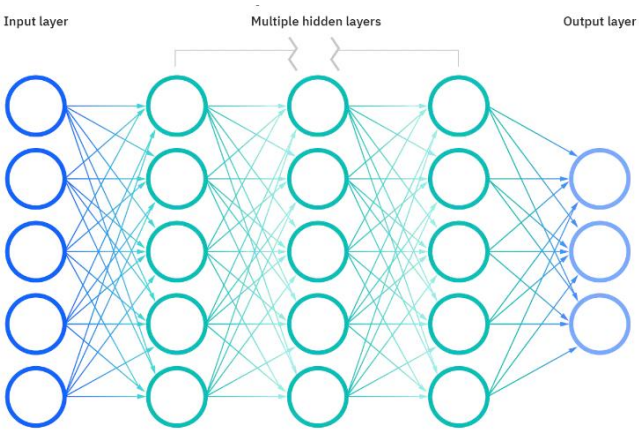


Рисунок 1.1 – Внутрішня будова нейронної мережі [1]

На початкових етапах нейронні мережі досить сильно покладаються на набір тренувальних даних для навчання та покращення їх точності. Проте, коли ці алгоритми вже гарно налаштовані і показують високу точність, їх можна ефективно застосовувати для вирішення великої кількості різноманітних задач, наприклад створення пошукових алгоритмів.

1.2.1 Структура нейронних мереж

Структура нейронної мережі визначається в першу чергу організацією її нейронів. Оскільки кожен вузол мережі має вхідні та вихідні сигнали, завдяки яким він взаємодіє з іншими вузлами, то їх взаємне розташування може суттєво впливати на загальний вигляд мережі.

Крім того, на структуру мережі можуть впливати функції активації, що використовуються для обробки сигналів, які передаються між нейронами, а також методи навчання, що використовуються для підбору та оптимізації вагових коефіцієнтів вузлів мережі.

Структура нейронної мережі може бути одношаровою, двошаровою чи багатошаровою, а також мати різну кількість вузлів на кожному шарі.

Формування мережі починається з вхідного шару. Кількість нейронів у ньому зазвичай буде дорівнювати кількості досліджуваних змінних чи властивостей вхідних даних. Після визначення вхідних вузлів кожному з них присвоюється ваговий коефіцієнт, який допомагає визначити відносну важливість тієї чи іншої змінної. Відповідно змінні з більшими “вагами” дають більш вагомий внесок у формування вихідного результату. На наступному кроці знаходиться сума добутків всіх вхідних значень на відповідні коефіцієнти, і отриманий результат передається у так звану функцію активації. Якщо отримане значення перевищує заданий поріг, то відповідний вузол моделі “активується”, передаючи результат на наступний шар мережі. В такому випадку вихідні дані

вузлів на попередньому шарі моделі стають вхідними для вузлів на наступному шарі. Цей процес передачі даних від одного шару до іншого визначає таку нейронну мережу як мережу прямого поширення (feedforward model). Замість жорсткого порогового значення, яке дає на виході лише 0 або 1, можна також застосовувати “згладжені” функції активації, наприклад сигмоїду. В такому випадку результат буде лежати у діапазоні від 0 до 1, і може розглядатися як імовірність [1].

Останній шар нейронної мережі називається вихідним, і результати, отримані на вузлах цього шару, відповідають розв’язкам досліджуваної задачі. Наприклад, у випадку класифікації результуючим значенням може бути присвоєння до того чи іншого класу, а у випадку регресії — передбачуване наступне значення.

Проміжні шари, що знаходяться між вхідним та вихідним, називаються прихованими. Вони використовуються для формування більш складної моделі і дослідження зв’язків та патернів у даних, за рахунок використання різних лінійних та нелінійних функцій активації на різних шарах мережі. У найпростішій реалізації нейронної мережі приховані шари можуть бути відсутні, і інколи навіть такої моделі достатньо для вирішення певних нескладних задач.

Для оцінки якості моделі можуть застосовуватися різні метрики, наприклад MSE. У такому випадку маємо на меті мінімізувати значення MSE, поступово налаштовуючи вагові коефіцієнти вузлів та, можливо, їх порогові значення і функції активації.

1.2.2 Класифікація нейронних мереж

Найчастіше нейронні мережі класифікують за їх типом. Серед найбільш відомих та часто застосовуваних типів можна виділити наступні [1]:

- Мережі прямого поширення (Feedforward models).

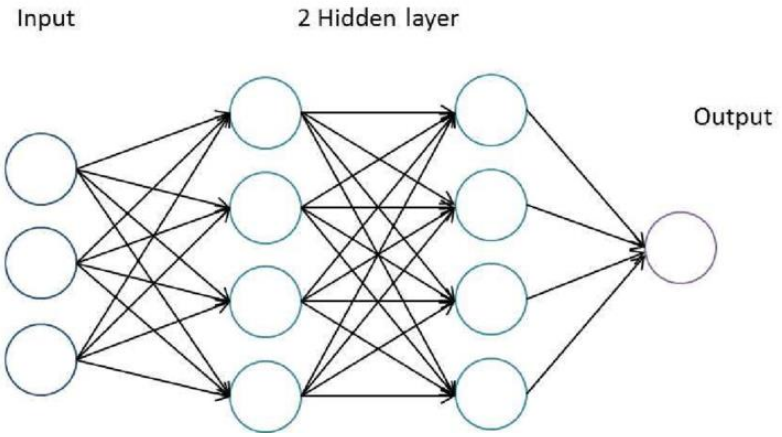


Рисунок 1.2 – Схема мережі прямого поширення [2]

У таких моделях інформація може передаватися між шарами тільки в одному напрямку, без циклів чи повторень. Мережі цього типу часто застосовуються наприклад у задачах розпізнавання образів;

- Рекурентні нейронні мережі (Recurrent neural networks).

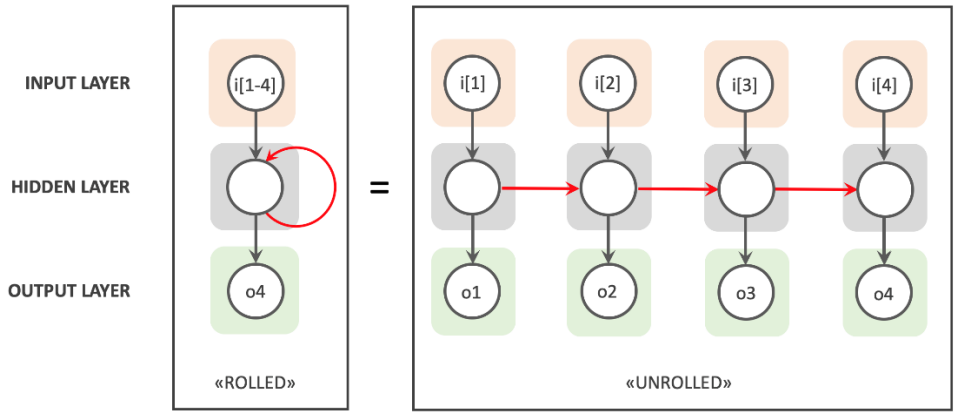


Рисунок 1.3 – Схема рекурентної нейронної мережі [3]

У таких моделях дані можуть передаватися як у прямому напрямку, так і у зворотному, тобто утворювати цикли. За рахунок зворотних зв’язків такі мережі можуть проявляти динамічну поведінку, використовуючи свої внутрішні стани (так звану “пам’ять”) для обробки послідовностей входів довільної довжини. Вони часто застосовуються у задачах розпізнавання мовлення та неперервного рукописного тексту;

- Згорткові нейронні мережі (Convolutional neural networks).

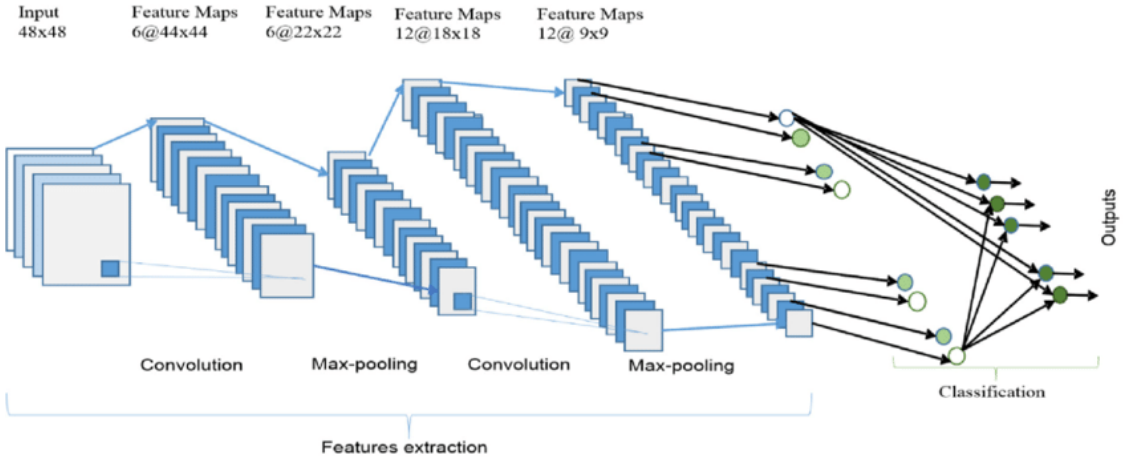


Рисунок 1.4 – Діаграма згорткових нейронних мереж [4]

Вони застосовуються у задачах аналізу зображень, розпізнавання образів, комп'ютерного зору. Ці мережі використовують принципи лінійної алгебри, зокрема множення матриць, для виявлення закономірностей у даних. Такі мережі зазвичай складаються зі згорткових, об'єднувальних (pooling) та повністю з'єднаних (fully connected) шарів. Задачею цих шарів є відповідно проведення операції згортки через застосування фільтрів, зменшення розмірностей карт ознак, та прогнозування на основі цих вилучених ознак;

- Генеративні змагальні мережі (Generative adversarial networks)

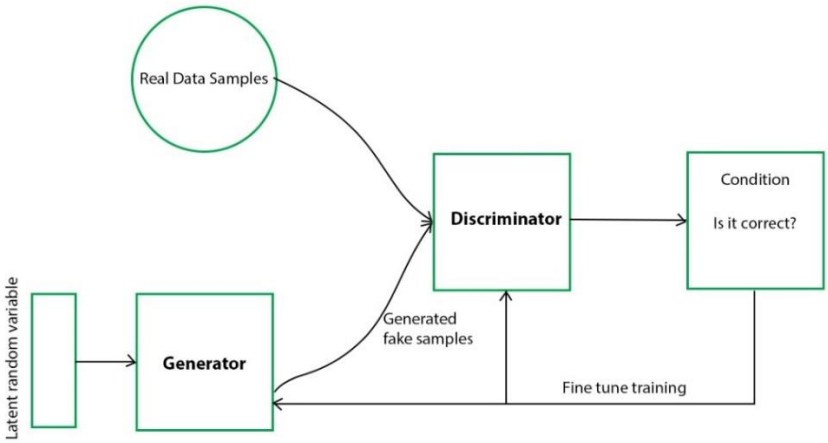


Рисунок 1.5 – Схема генеративної змагальної мережі [5]

Ці моделі використовуються для доповнення даних, а також створення відео та зображень. В них фактично застосовуються дві мережі, одна з яких (генератор) створює нові зразки на основі навчальної вибірки, а інша (дискримінатор) намагається визначити, чи є певний елемент вибірки реальним, чи фальшивим, тобто згенерованим першою мережею;

- Модульні нейронні мережі (Modular neural networks).

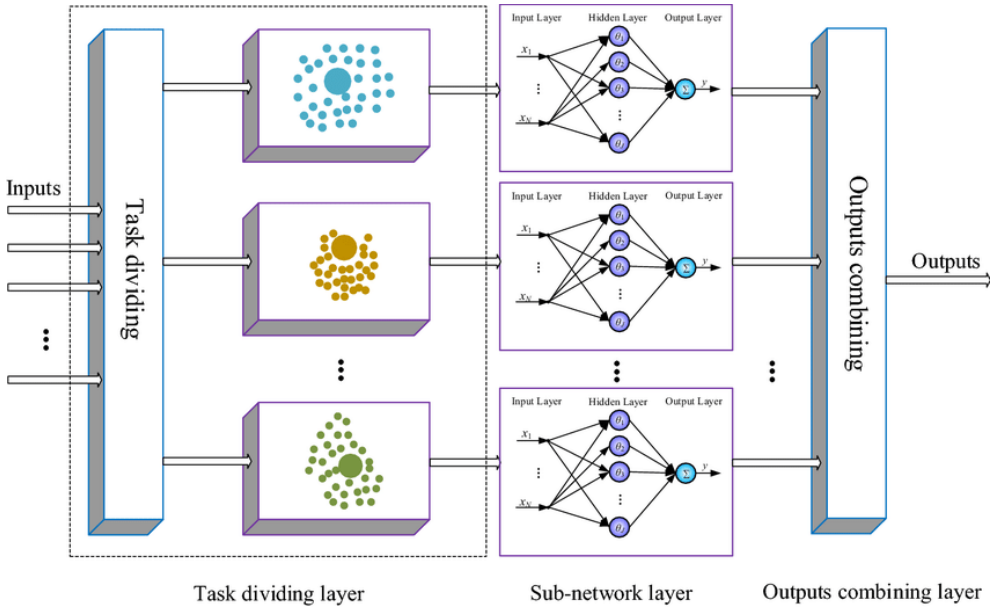


Рисунок 1.6 – Загальна структура модульної нейронної мережі [6]

Вони складаються з кількох менших мереж, або модулів, що відповідають за розв’язання окремих чітко заданих підзадач. Такі моделі широко використовуються у системах управління та робототехніці;

- Мережі радіальних базисних функцій (Radial Basis Function Networks).

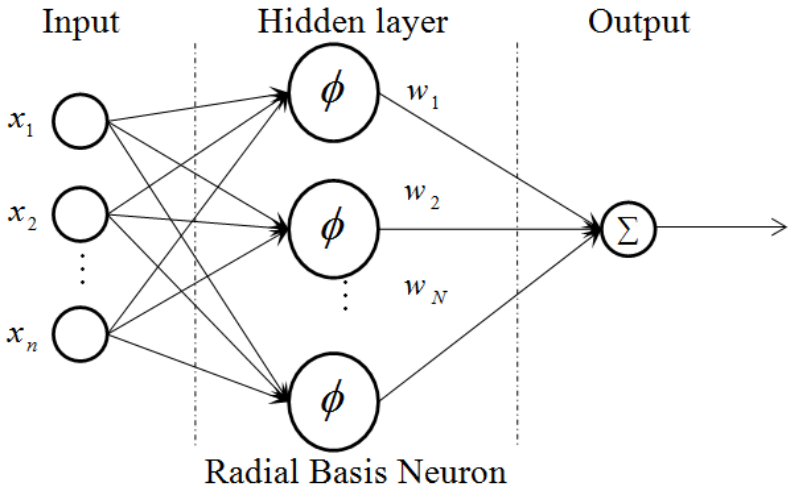


Рисунок 1.7 – Схема мережі радіальних базисних функцій [7]

Такі мережі використовують радіальні базисні функції (математичні функції, що залежать від відстані до центральної точки) для перетворення вхідних даних у простір з вищою розмірністю. Вони зазвичай використовуються у задачах регресії та апроксимації функцій.

1.3 Класифікація методів машинного навчання (SL, SSL, UL, RL)

Існує кілька основних класифікацій методів машинного навчання. Однією з них є поділ в залежності від того, які алгоритми та набори даних використовуються при навчанні моделі. Згідно з цією класифікацією, методи ML можуть належати до однієї з наступних категорій:

1) Навчання з учителем (supervised learning);
2) Навчання без вчителя (unsupervised learning);
3) Навчання з частковим залученням вчителя, або напівкероване навчання (semi-supervised learning);
4) Навчання з підкріпленням (reinforcement learning).

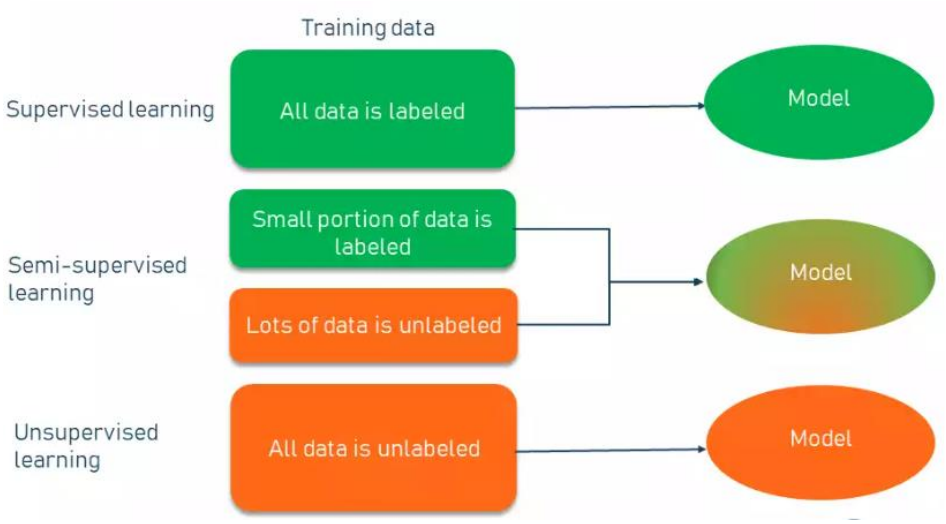


Рисунок 1.8 – Види методів машинного навчання залежно від набору тренувальних даних [11]

1.3.1 Навчання з учителем (SL)

Основною характеристикою SL алгоритмів є використання повністю розмічених наборів даних при тренуванні моделі [12]. Відповідно ми заздалегідь знаємо, до якого “класу” належить будь-який елемент з наявної вибірки.

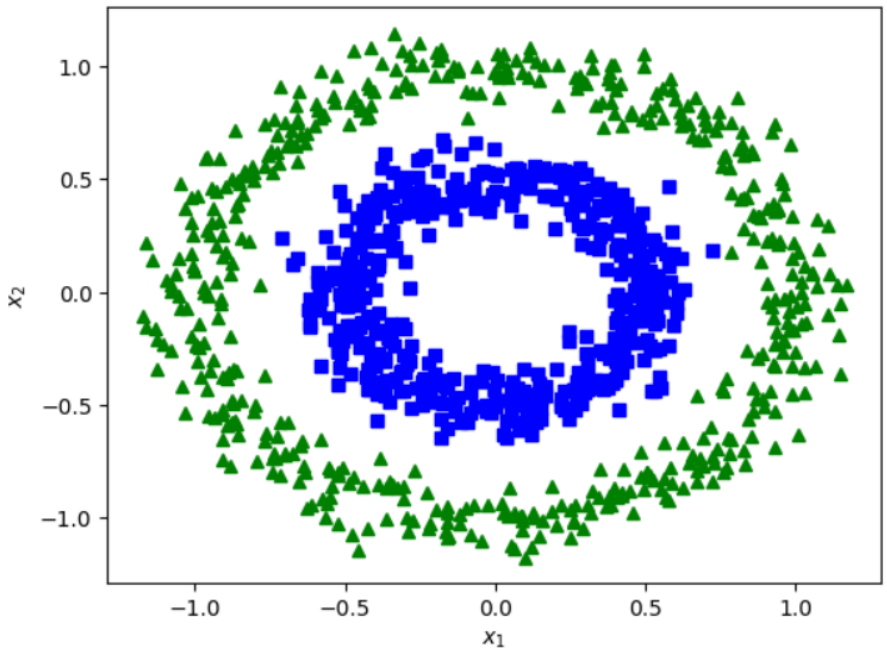


Рисунок 1.9 – Повністю розмічений набір тренувальних даних для навчання з учителем

Навчання з учителем зазвичай використовується для вирішення двох типів задач: класифікації та регресії.

- В задачах класифікації алгоритми намагаються якнайточніше розподілити наявні дані до різних категорій, тобто зробити висновок про належність кожного елементу вибірки до тієї чи іншої категорії залежно від його характеристик. Найчастіше для класифікації використовуються такі алгоритми як лінійні класифікатори, метод опорних векторів, дерева рішень, метод k-найближчих сусідів тощо;
- В задачах регресії досліджується зв’язок між залежними та незалежними змінними у даних. Вони часто застосовуються для прогнозування, наприклад доходу від продажів чи коливань температури. Популярними алгоритмами регресії є лінійна, логістична та поліноміальна регресія.

Серед переваг SL підходу можна виділити наступні:

1) Відносна простота реалізації алгоритмів, адже нам заздалегідь відомі реальні мітки для тестових даних;
2) Точність прогнозування при наявності великої та репрезентативної вибірки;
3) Зрозумілі вихідні результати — SL методи можуть надавати результати, які легко інтерпретувати, що дозволяє зрозуміти, як саме модель робить прогнози. Це може бути дуже корисним наприклад у галузі медицини;

Серед основних недоліків навчання з учителем можна відзначити:

1) Залежність від даних — якщо наявні дані є неповними, зашумленими чи упередженими (biased), то отримана в результаті навчання SL модель може погано узагальнювати нові, невідомі дані. Крім того, існує ще і проблема маркування, адже цей процес є дуже затратним і вимагає багато ресурсів;
2) Перенавчання — SL моделі можуть іноді надмірно підлаштовуватися під дані з навчальної вибірки, що призводить до погіршення ефективності

моделі при роботі з новими, невідомими раніше даними. Зазвичай це відбувається у тих випадках, коли структура моделі є занадто складною, або коли навчальна вибірка є недостатньо репрезентативною відносно загального розподілу даних

1.3.2 Навчання без вчителя (UL)

Методи UL використовують для аналізу та кластеризації немаркованих наборів даних. Ці алгоритми намагаються знайти приховані закономірності та зв’язки між елементами вибірок без необхідності залучення людини. Здатність виявляти подібності та відмінності у даних дають змогу застосовувати UL підхід наприклад в задачах розпізнавання зображень чи сегментації ринку.

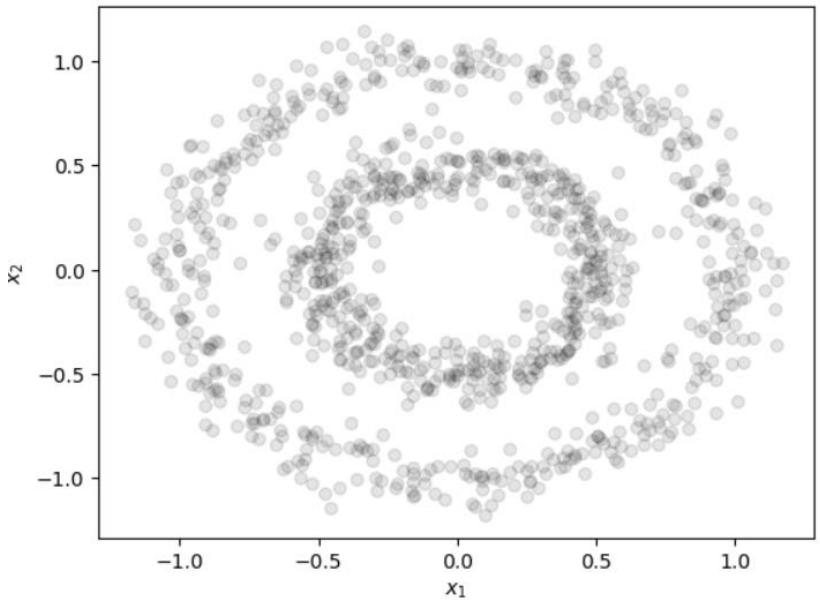


Рисунок 1.10 – Повністю немаркований набір тренувальних даних для навчання без учителя

Навчання без вчителя зазвичай застосовується для вирішення таких задач, як кластеризація, зниження розмірності та навчання асоціативним правилам.

- В задачах кластеризації алгоритми намагаються поділити наявні дані з вибірки на окремі групи (кластери) залежно від подібності окремих

елементів. Варто зазначити, що реальна кількість кластерів може бути заздалегідь невідома;

- Асоціативні правила представляють собою набір “правил”, за якими знаходяться зв’язки між змінними у вибірці. Такі методи часто застосовуються у задачах “ринкового кошику”, для дослідження взаємозв’язків між товарами компанії і формування системи рекомендацій для клієнтів;
- Зменшення розмірності — це підхід машинного навчання, який використовується, коли кількість ознак у певному наборі даних є надто великою. Він дозволяє зменшити кількість вхідних даних, зберігаючи при цьому їхню цілісність. Відповідні алгоритми часто використовуються на етапі попередньої обробки даних, наприклад при видаленні шумів з зображень.

До основних переваг UL методів можна віднести:

1) Відсутність необхідності у розмічених даних, які у багатьох реальних задачах досить складно отримати;
2) Знаходження прихованих зв’язків між змінними, які можуть не бути одразу помітними;
3) Адаптивність — UL моделі можуть адаптуватися до нових даних без необхідності повторного маркування, що у певних випадках робить їх більш гнучкими та масштабованими;

Серед недоліків навчання без вчителя можна виділити наступні:

1) Висока обчислювальна складність, пов’язана з великою розмірністю тренувальних даних;
2) Вищий ризик отримання неточних результатів, у порівнянні з SL методами;

3) Відсутність об’єктивних метрик оцінювання, за якими можна було б порівнювати результати різних моделей чи визначати, в який саме момент потрібно завершити процес навчання;
4) Складнощі при інтерпретації результатів — оскільки UL алгоритми досліджують приховані зв’язки між даними, часто буває складно сказати, чому модель отримала певний результат, і чому вона вважає певні змінні більш важливими, ніж інші

1.3.3 Напівкероване навчання (SSL)

Напівкероване навчання об’єднує в собі особливості SL та UL методів, застосовуючи для навчання як марковані, так і немарковані дані. Такий підхід є корисним, коли у наявних даних важко визначити важливі ознаки, а також коли розмір наявної вибірки є досить великим.

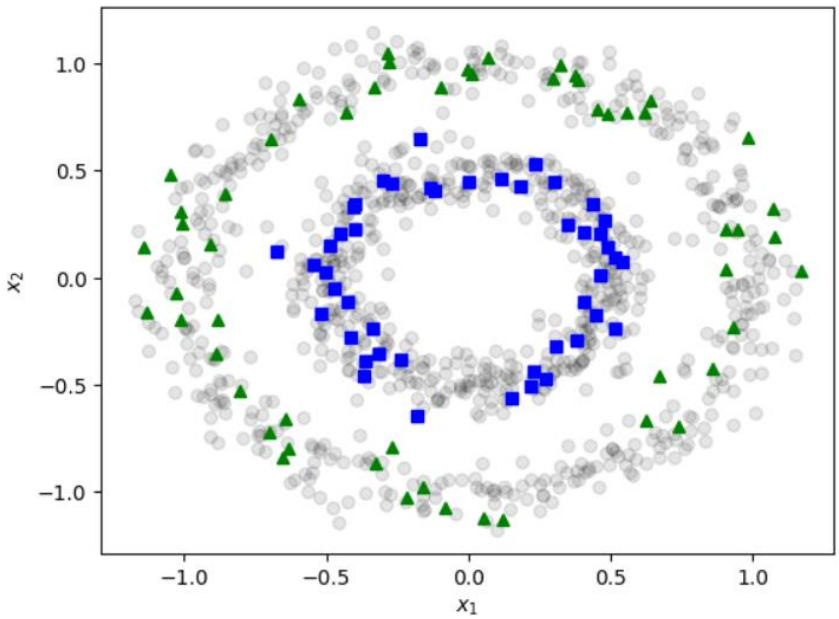


Рисунок 1.11 – Частково маркований набір тренувальних даних для задач напівкерованого навчання

SSL підхід ідеально підходить для застосування у медичній сфері, де навіть невелика кількість розмаркованих даних може призвести до значного

підвищення точності моделі порівняно з навчанням на повністю нерозміченій вибірці (без вчителя). З іншого боку, використання навчання з учителем у цих задачах також не є ефективним, адже зазвичай наявна тільки невелика кількість маркованих даних, яких недостатньо для якісного дослідження відносно загальної задачі.

1.3.4 Навчання з підкріпленням (RL)

У задачах навчання з підкріпленням розглядається так званий інформаційний агент, і певне середовище, в якому він оперує. Агент може отримувати необхідні для навчання дані заздалегідь, як у попередньо розглянутих підходах, а також безпосередньо під час виконання операцій у середовищі.

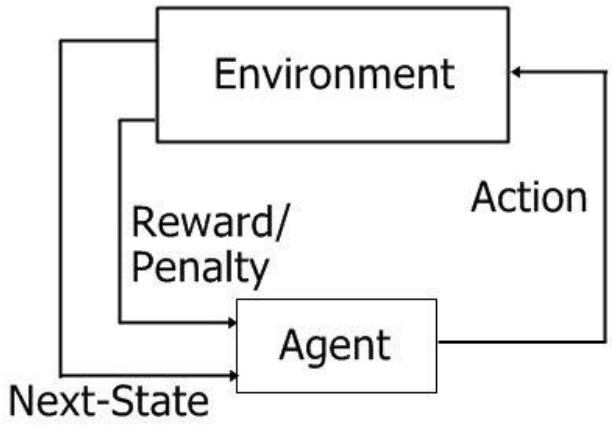


Рисунок 1.12 – Схема процесу навчання з підкріпленням

Для визначення оптимальних дій у RL підході використовується механізм нагород та штрафів. Тим операціям, які вважаються корисними, присвоюються додатні значення оцінки – щоб заохотити агента в подальшому частіше обирати саме ці дії. Небажаним рішенням відповідно призначаються від’ємні штрафи. Ідея такого підходу полягає в тому, що при ітеративному виконанні певного набору операцій і аналізу отриманих результатів, після деякої кількості повторів

агент навчиться досягати мети таким шляхом, при якому значення сумарної оцінки буде максимальним.

1.4 Постановка загальної задачі напівкерованого навчання

Як зазначалося у попередньому підрозділі, навчання з частковим залученням вчителя (SSL) є варіацією машинного навчання, у якій для тренування моделі застосовуються і розмічені, і нерозмічені дані з вибірки. При використанні такого підходу можна сформулювати наступну постановку задачі навчання:

Нехай маємо вибірку даних $X = \{x_1, x_2, \dots, x_n\}$ та набір міток $Y = \{y_1, \dots, y_l\}$, причому $l < n$. Серед цього набору маємо групу розмічених даних виду $(X_l, Y_l) = \{(x_{1:l}, y_{1:l})\}$ і групу нерозмічених даних $X_u = \{x_{l+1:m}\}$, які використовуються при навчанні моделі. Зазвичай значення l значно менше за m ($l \ll m$), тобто маємо більше нерозмічених тренувальних зразків, ніж розмічених. Крім того, маємо також множину нерозмічених елементів $X_t = \{x_{m+1:n}\}$, які не використовуються при тренуванні і формують так звану тестову вибірку.

За таких вхідних умов необхідно знайти функцію розподілу $F(a): X \to Y$, при знаходженні якої мають застосовуватися як (X_l, Y_l), так і X_u.

1.4.1 Основні припущення у задачах SSL

При розв’язанні задач SSL використовуються деякі припущення, без яких неможливо було б сформулювати ефективні узагальнення алгоритмів розв’язання для різних наборів даних. Серед основних припущень можна виділити наступні:

- Припущення плавності (Smoothness Assumption). Його ідея полягає у тому, що два елементи вибірки x_1 та x_2, що лежать близько один від одного, з більшою імовірністю матимуть однакові значення міток y_1, y_2;
- Припущення кластерності (Cluster Assumption). Воно є дещо подібним до попереднього, і працює у випадках, коли об'єкти різних класів утворюють кластери. Ідея цього припущення полягає в тому, що два об'єкти, що належать до одного кластеру, з більшою імовірністю матимуть однакові значення міток;
- Припущення про поділ в областях низької щільності (Low-density Separation Assumption). Його ідея полягає в тому, що межі поділу класів з більшою імовірністю будуть проходити в областях низької щільності, тобто там, де наявна менша кількість елементів вибірки;
- Припущення надмірності ознак (Manifold Assumption). Суть цього припущення у тому, що для опису значної кількості реальних наборів багатовимірних даних насправді достатньо лише деяких з найважливіших їх ознак. Таким чином, при попередній обробці даних можна застосовувати методи зниження розмірності, що допоможуть спростити розв'язування задачі, зберігаючи при цьому достатньо високу точність моделі.

1.5 Метод опорних векторів, його особливості та характерні риси

Метод опорних векторів (Support Vector Machine, SVM) є одним з найбільш відомих методів машинного навчання і належить до категорії навчання з вчителем. Він може використовуватися для вирішення як задач класифікації, так і регресії.

1.5.1 Поняття гіперплощини

Основна ідея методу полягає у знаходженні найкращої межі поділу, яка могла б розділити багатовимірний простір на окремі області таким чином, щоб при надходженні нових даних їх можна було чітко віднести до тієї чи іншої області (класу).

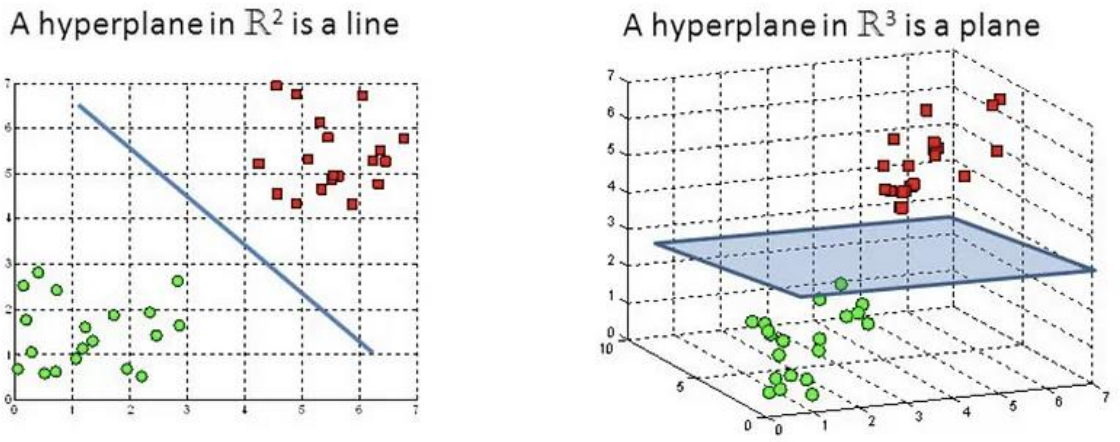


Рисунок 1.13 – Гіперплощини у просторах різної розмірності [13]

Зазвичай існує багато можливих гіперплощин, які розділяють дані на два класи. SVM метод має на меті знаходження такої гіперплощини, що матиме максимальний відступ (margin), тобто максимальну відстань від представників обох класів. Збільшення цієї відстані дозволяє з більшою впевненістю класифікувати нові елементи вибірки у майбутньому.

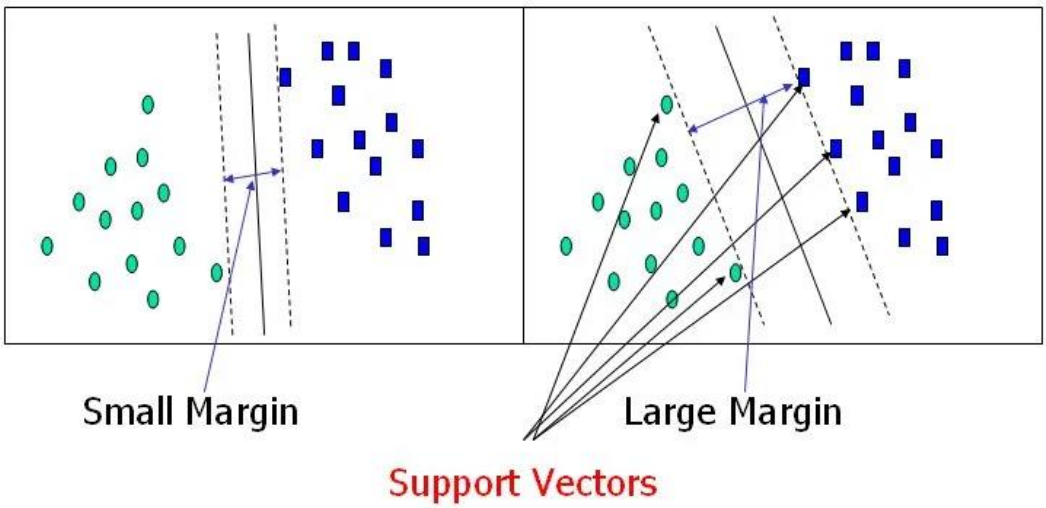


Рисунок 1.14 – Гіперплощини з різною шириною відступу [13]

1.5.2 Опорні вектори

Елементи, що знаходяться найближче до межі розподілу класів, називаються опорними векторами. SVM визначений таким чином, що лише опорні вектори визначають розташування роздільної поверхні і ширину її межі [14], тоді як усі інші спостереження не вносять ніякого впливу. Такий підхід також дозволяє покращити продуктивність моделі, оскільки навіть більш великих вибірок даних нам потрібно враховувати при розрахунках лише відносно невелику кількість зразків.

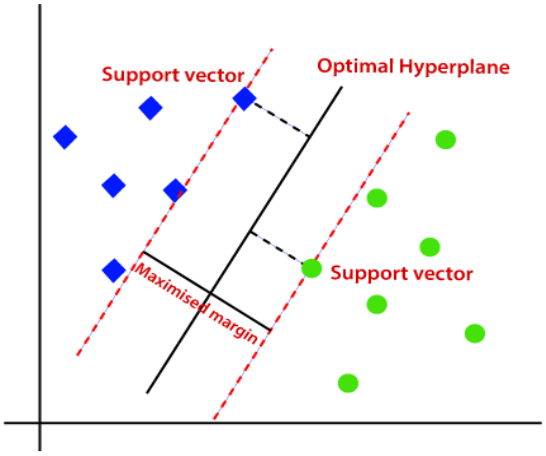


Рисунок 1.15 – Опорні вектори у методі SVM [14]

1.5.3 Формулювання задачі для методу опорних векторів

Розглянемо постановку задачі бінарної класифікації з використанням методу опорних векторів.

Нехай маємо множину об’єктів $X = \{\vec{x}_1, \vec{x}_2, \ldots, \vec{x}_n\}, \vec{x}_i \in \mathbb{R}^n$. Кожен елемент цієї множини може належати до одного з двох класів $Y = \{-1; 1\}$. Необхідно побудувати такий алгоритм класифікації $F(\vec{a}): X \to Y$, який міг би правильно розподіляти об’єкти до того чи іншого класу [16].

В просторі $\mathbb{R}^n$ гіперплощина, яка розділятиме два класи, визначається рівнянням

$$\langle \vec{w}, \vec{x} \rangle + b = 0, \tag{1.1}$$

де параметр $\vec{w}$ задає вектор нормалі до гіперплощини, а відношення $\frac{b}{\|\vec{w}\|}$ — відстань від неї до початку координат.

Побудована гіперплощина поділяє простір $\mathbb{R}^n$ на два підпростори: $\langle \vec{w}, \vec{x} \rangle + b > 0$ та $\langle \vec{w}, \vec{x} \rangle + b < 0$. Відповідно, для того, щоб ця площина правильно розділяла елементи двох класів C_1 та C_2, має виконуватися умова

$$\begin{cases} \langle \vec{w}, \vec{x} \rangle + b > 0, \forall x \in C_1 \\ \langle \vec{w}, \vec{x} \rangle + b < 0, \forall x \in C_2 \end{cases} \tag{1.2}$$

1.5.3.1 Випадок лінійно роздільної вибірки

Якщо задана вибірка є лінійно роздільною, тобто існує хоча б одна гіперплощина, яка повністю розділяє класи C_1 та C_2, то для алгоритму класифікації $F(\vec{a})$ можна використати signum-функцію:

$$F(\vec{a}) = sign(\langle \vec{w}, \vec{x} \rangle + b) = sign\left(\sum_{i=1}^{k} w_i x_i + b\right) \tag{1.3}$$

Відступ від цієї гіперплощини до кожного елементу вибірки буде додатною (умова поділу класів). Оскільки SVM має на меті знаходження найкращої роздільної площини, її намагаються побудувати таким чином, щоб ширина отриманого відступу була якнайбільшою.

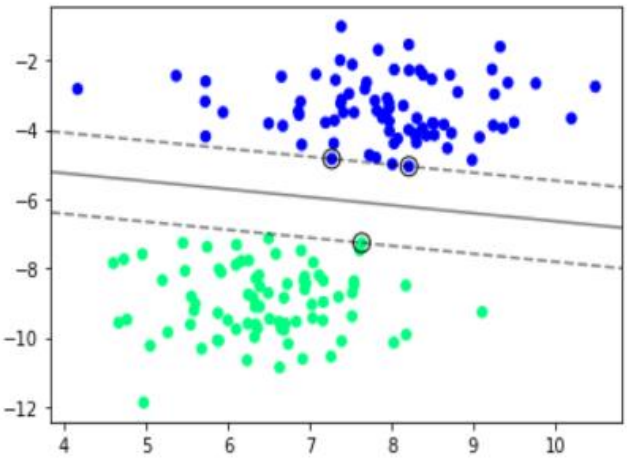


Рисунок 1.16 – Випадок лінійно роздільних класів [17]

Оптимальне рішення при такій постановці досягається за рахунок мінімізації вектору параметрів $\vec{w}$. Враховуючи також обмеження на мінімальну ширину відступу, отримуємо наступну задачу умовної оптимізації в умовах квадратичного програмування:

$$\begin{cases} \min\limits_{\vec{w},b} \dfrac{1}{2}\|\vec{w}\| \\ y_i(\langle \vec{w}, \vec{x}_i \rangle + b) \geq 1, \;\; i \in 1,2,\dots,n \end{cases} \tag{1.4}$$

Оскільки класи в такій постановці є лінійно роздільними, відповідна реалізація має назву SVM з “жорстким відступом” (hard margin), тобто випадок, коли жоден елемент не може опинитися всередині роздільної смуги чи з неправильного боку від неї.

В практичних реалізаціях для розв’язування такої задачі її зазвичай представляють у двоїстому вигляді. Двоїста задача записується із застосуванням методу множників Лагранжа, і для випадку лінійно роздільної вибірки визначається наступним чином:

$$\begin{cases} \max\limits_{\vec{\lambda}} \sum\limits_{i=1}^{n} \lambda_i - \dfrac{1}{2}\sum\limits_{i=1}^{n}\sum\limits_{j=1}^{n} \lambda_i \lambda_j y_i y_j \langle \vec{x}_i, \vec{x}_j \rangle \\ \lambda_i \geq 0, i = 1,2,\dots,n \\ \sum\limits_{i=1}^{n} \lambda_i y_i = 0 \end{cases}, \tag{1.5}$$

де λ_i – змінні двоїстої задачі.

Опорними векторами будуть ті елементи вибірки, для яких відповідні значення $\lambda_r \neq 0$.

1.5.3.2 Випадок лінійно нероздільної вибірки

В реальних задачах лінійно роздільні вибірки практично ніколи не зустрічаються. Майже завжди досліджувані класи не мають чіткої межі (частково перекриваються), або у даних наявні викиди, що порушують умову лінійної роздільності класів. У такому випадку розглянута у попередньому підрозділі задача не матиме рішень.

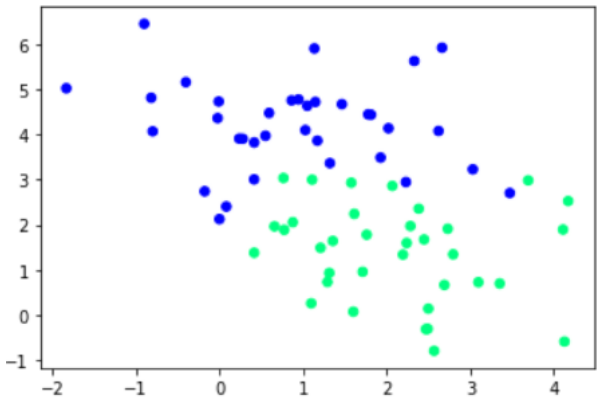


Рисунок 1.17 – Лінійно нероздільні класи [17]

Тоді задані обмеження необхідно дещо послабити, дозволивши деяким елементам вибірки потрапляти всередину роздільної смуги, або навіть по інший бік від неї — якщо це призводить до кращої узагальнюючої здатності алгоритму. Для кожного такого об’єкту з’являється величина ξ_i (помилка моделі на відповідному векторі $\vec{x}_i$), значення якої повинно бути якнайменшим. В такому випадку маємо реалізацію SVM з “нежорстким відступом” (soft margin), умови якої записуються наступним чином:

$$\begin{cases} \min\limits_{\vec{w},b,\xi} \frac{1}{2}\|\vec{w}\|^2 + C\sum\limits_{i=1}^{n}\xi_i \\ y_i(\langle \vec{w}, \vec{x}_i\rangle + b) \geq 1 - \xi_i, \quad i = 1, \dots, n \\ \xi_i \geq 0, \quad i = 1, \dots, n \end{cases}, \tag{1.6}$$

де параметр $C > 0$ — додатна константа, за допомогою якої знаходять баланс між максимізацією ширини смуги та мінімізацією суми помилок ξ_i. Можна також сказати, що вона відповідає за "розмиття" відступу — чим меншим є значення C, тим більш широким та розмитим стає відступ, включаючи у себе більшу кількість елементів з обох класів [17]. І навпаки, при збільшенні C ширина роздільної смуги буде зменшуватися.

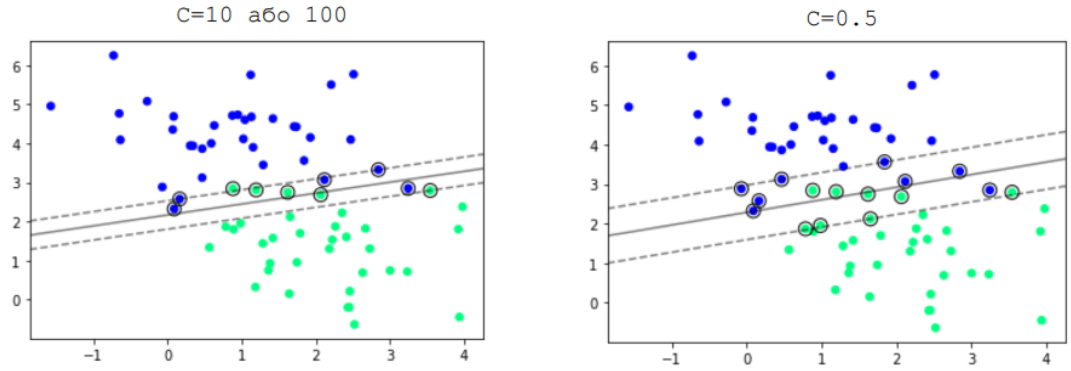


Рисунок 1.18 – Зміна ширини відступу в залежності від параметру C [17]

Оптимальне значення параметру C зазвичай залежить від конкретної вибірки даних. Його налаштовують за допомогою перехресної перевірки.

У двоїстій формі формулювання задачі досить схоже до лінійно роздільного випадку:

$$\begin{cases} \max\limits_{\vec{\lambda}} \sum\limits_{i=1}^{n}\lambda_i - \frac{1}{2}\sum\limits_{i=1}^{n}\sum\limits_{j=1}^{n}\lambda_i\lambda_j y_i y_j \langle \vec{x}_i, \vec{x}_j\rangle \\ 0 \leq \lambda_i \leq C, i = 1,2, \dots, n \\ \sum\limits_{i=1}^{n}\lambda_i y_i = 0 \end{cases} \tag{1.7}$$

Як можна бачити, єдина зміна у порівнянні з лінійно роздільним випадком — це більш строге обмеження на λ_i.

1.5.3.3 Узагальнення нелінійного випадку, процедура kernel trick

Існує також ще один підхід до вирішення проблеми лінійної роздільності вибірки — так званий kernel trick. Його ідея полягає у припущенні, що набір даних, який не є лінійно роздільним в деякому просторі $\mathbb{R}^n$, може стати лінійно роздільним при переході у деякий інший простір H (в загальному випадку, це простір більшої розмірності).

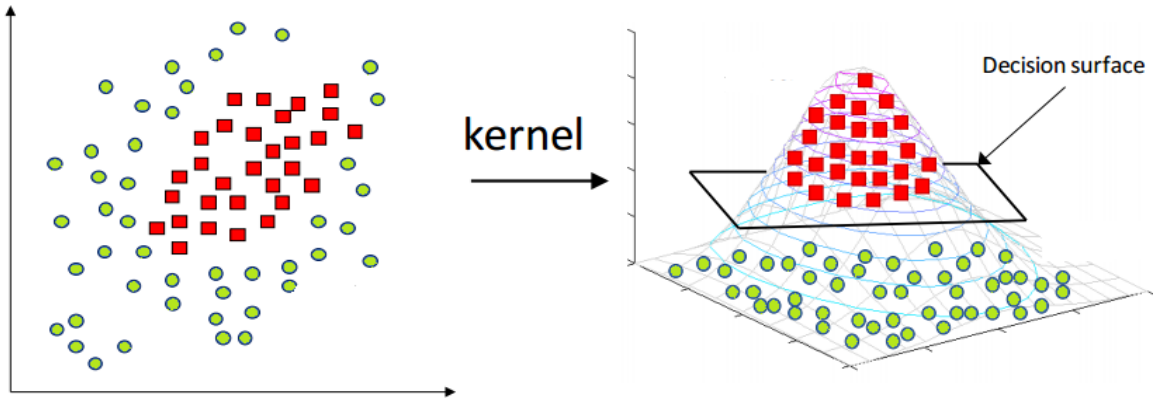


Рисунок 1.19 – Процедура kernel trick [14]

Побудова SVM у новому просторі відбуватиметься так само, як і в $\mathbb{R}^n$, лише замість векторів ознак $\vec{x}$ будуть використовуватися функції переходу (образи) $\psi(\vec{x})$: $\mathbb{R}^n \to H$. Відповідно скалярний добуток $\langle \vec{x}_i, \vec{x}_j \rangle$ також зміниться на $\langle \psi(\vec{x}_i), \psi(\vec{x}_j) \rangle$.

Оскільки при побудові задачі класифікації використовуються лише скалярні добутки від $\vec{x}$, а не самі вектори ознак у явному вигляді, то ми маємо можливість замінити скалярний добуток в $\mathbb{R}^n$ на ядро — функцію, що представляє собою скалярний добуток у просторі H. Сам новий простір, а також образи ψ при цьому можна взагалі не задавати в явному вигляді, обмежившись лише вибором власне ядра.

Використовуючи такий підхід, отримуємо формулювання двоїстої задачі у вигляді

$$\begin{cases} \max\limits_{\vec{\lambda}} \sum\limits_{i=1}^{n} \lambda_i - \frac{1}{2} \sum\limits_{i=1}^{n} \sum\limits_{j=1}^{n} \lambda_i \lambda_j y_i y_j K(\vec{x}_i, \vec{x}_j) \\ 0 \le \lambda_i \le C, i = 1,2, \dots, n \\ \sum\limits_{i=1}^{n} \lambda_i y_i = 0 \end{cases} \quad (1.8)$$

де $K(\vec{x}_i, \vec{x}_j) = \langle \psi(\vec{x}_i), \psi(\vec{x}_j) \rangle$ — ядро.

1.5.4 Проблема вибору ядра

На жаль, не існує одної ідеальної функції ядра, яка б підходила для будь-яких наборів даних. Тому при розв'язуванні задач за допомогою SVM методу зазвичай досліджуються результати при використанні різних ядер, і обирається те з них, яке найкраще працює для конкретної вибірки.

Серед функцій ядра, які застосовуються найчастіше, можна відзначити такі:

- Лінійне ядро (Linear kernel, або також Non-kernel) – представляє собою звичайний скалярний добуток векторів. Використання такого ядра потребує порівняно менших обчислювальних затрат, і воно показує досить гарні результати для задач категоризації текстів [18];
- Поліноміальне ядро (Polynomial kernel) – визначається власне поліномом певного степеню. Його часто використовують в обробці зображень, і воно гарно працює для випадку, коли усі тренувальні дані є нормалізованими;
- RBF ядро – мабуть найбільш популярна функція ядра серед усіх. Зазвичай дослідження починаються саме з нього, оскільки це ядро гарно працює для випадку лінійно нероздільної вибірки і дозволяє побудувати достатньо якісну межу поділу, не маючи попередніх знань про структуру даних [19]. Відповідна функція представляє собою експоненту;

- Сигмоїдне ядро – визначається функцією гіперболічного тангенсу. Його часто обирають при роботі з нейронними мережами;

Існують також і інші ядра, як наприклад ANOVA чи функція Бесселя, проте вони використовуються не так часто.

1.5.5 Переваги та недоліки SVM

Серед основних переваг методу опорних векторів можна виділити наступні [17]:

- Процес навчання моделі зводиться до задачі квадратичного програмування;
- Етап прогнозування після завершення навчання моделі займає дуже мало часу;
- На результат впливають тільки точки, що знаходяться найближче до меді відступу, тому метод SVM може застосовуватися для багатовимірних даних, зокрема коли кількість ознак є більшою за розмірність навчальної множини;
- Можливість використовувати kernel trick як ефективне узагальнення на випадок нелінійної вибірки

До основних недоліків SVM можна віднести:

- Великі обчислювальні витрати;
- Необхідність підбору коректного значення C за допомогою перехресної перевірки;
- Відсутність імовірнісної інтерпретації результатів

1.6 Висновки до першого розділу

У даному розділі було розглянуто поняття машинного навчання та сфери його застосування, штучні нейронні мережі та їх види, а також класифікації методів машинного навчання. Крім того, було сформульовано постановку задачі напівкерованого навчання в загальному випадку, а також проаналізовано метод опорних векторів та наведено його характерні особливості.

РОЗДІЛ 2 НАПІВКЕРОВАНА МАШИНА ОПОРНИХ ВЕКТОРІВ

2.1 Загальна постановка задачі S3VM методу

Напівкерована машина опорних векторів (Semi-Supervised Support Vector Machine, S3VM) є модифікацією методу опорних векторів, яка застосовується для вирішення задач навчання з частковим залученням вчителя.

Основна ідея S3VM полягає у знаходженні такої гіперплощини між розміченими даними, яка мала б найбільшу ширину відступу відносно наявних нерозмічених даних [22]. Для цього використовуються припущення Cluster Assumption та Low-density Separation Assumption, розглянуті раніше.

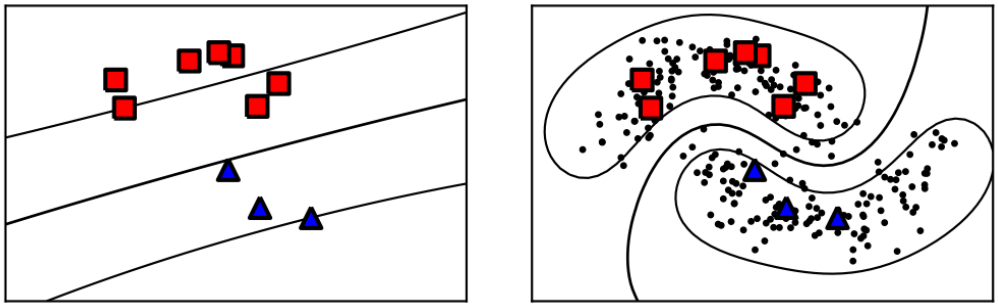


Рисунок 2.1 – Порівняння гіперплощин, знайдених SVM (зліва) та S3VM (справа) [23-24]

Нагадаємо, що формулювання задачі SVM в класичній формі мало наступний вигляд:

$$\begin{cases} \min\limits_{\vec{w},b,\xi} \dfrac{1}{2}\|\vec{w}\|^2 + C\sum\limits_{i=1}^{n}\xi_i \\ y_i(\langle \vec{w}, \vec{x}_i\rangle + b) \geq 1 - \xi_i, \qquad i = 1, \dots, n \\ \xi_i \geq 0, \;\; i = 1, \dots, n \end{cases} \tag{2.1}$$

У випадку S3VM (для бінарної класифікації) в умови додаються два нових обмеження на нерозмічені дані. Одне з них відповідає за обчислення помилки класифікації за припущення, що певний елемент вибірки належить до першого класу, а друге — за помилку при умові належності елементу до другого класу.

При цьому у цільовій функції знаходиться мінімум з двох можливих помилок класифікації. Остаточне значення міток для нерозмічених елементів відповідатиме результатам з меншою помилкою [25]. В такому формулюванні задачу S3VM у загальному вигляді можна записати як

$$\begin{cases} \min\limits_{w,b,\xi,z^+,z^-} \dfrac{1}{2}\|\vec{w}\|^2 + C_1\sum\limits_{i=1}^{l}\xi_i + C_2\sum\limits_{j=l+1}^{l+k}\min\left(z_j^+, z_j^-\right) \\ y_i(\langle\vec{w},\vec{x}_i\rangle - b) + \xi_i \geq 1, \quad \xi_i \geq 0, i = 1,\dots,l \\ \vec{w}\cdot\vec{x}_j - b + z_j^+ \geq 1, \quad z_j^+ \geq 0, j = l+1,\dots,l+k \\ -\left(\vec{w}\cdot\vec{x}_j - b\right) + z_j^- \geq 1, \;\; z_j \geq 0 j = l+1,\dots,l+k \end{cases}, \tag{2.2}$$

де z_j^+, z_j^- — помилки класифікації j-го нерозміченого елементу вибірки відносно першого та другого класів.

Важливо відмітити, що при такому формулюванні буде розв’язуватися задача неопуклої оптимізації, яку вже не можна вирішити методами квадратичного програмування. Це є однією з основних проблем при реалізації алгоритму S3VM, оскільки розрахунки для вирішення такої задачі матимуть велику (в загальному випадку – експоненційну) обчислювальну складність, і на даний момент не існує єдиного універсального підходу чи методу для подолання цієї проблеми.

Також при використанні напівкерованої машини опорних векторів потрібно враховувати так зване обмеження балансу (Balance Constraint). Це обмеження відповідає за кількісний розподіл немаркованих даних між класами, і допомагає уникнути випадку, коли абсолютну більшість таких зразків буде виділено лише в один клас.

В класичній постановці для розв’язання задач методом S3VM використовується так званий трансдуктивний підхід. На відміну від класичного (індуктивного) підходу, де ми маємо тренувальну вибірку для навчання моделі і тестову вибірку для перевірки її результатів, у трансдуктивному підході для навчання застосовуються усі наявні дані, як розмічені, так і не розмічені. Відповідно оцінка якості моделі відбувається на тих же немаркованих даних, на яких вона навчалася.

2.2 Огляд підходів до реалізації S3VM

Напівкерована машина опорних векторів у базовій постановці має два істотні недоліки, які ускладнюють ефективне застосування цього методу. Сам алгоритм методу є незмінним, проте різні його реалізації застосовують різні підходи до вирішення цих проблем.

Перша проблема, як було згадано в попередньому підрозділі, полягає у високій обчислювальній складності алгоритму. Без її вирішення застосування S3VM для великих вибірок стає неефективним, а саме при наявності значної кількості нерозмічених даних S3VM показує помітні покращення ефективності порівняно зі звичайним методом опорних векторів.

Для вирішення цієї проблеми було проведено чимало досліджень, які можна умовно розділити на чотири категорії [26]:

1) Підхід на основі глобальної комбінаторної оптимізації. До цієї категорії належить наприклад метод гілок і меж, який розв'язує S3VM задачу глобально і показує хорошу продуктивність на малих вибірках даних;
2) Підхід на основі глобальних евристичних алгоритмів. Сюди можна віднести наприклад transductive SVM реалізацію, у якій поступово збільшується вплив нерозмічених даних (тобто значення коефіцієнту C_2); або Continuation Method, який спочатку вводить допоміжну гладку функцію, а потім поступово зменшує її "гладкість" (smoothness), наближаючись до реальної цільової функції S3VM;
3) Підхід з використанням опуклої релаксації (convex relaxation). До представників цієї групи відносять методи релаксації на основі напіввизначеної оптимізації (Semi-definite programming, SDP) та алгоритмів мінімаксу;
4) Підхід на основі ефективних методів неопуклої оптимізації. Як приклади можна навести реалізацію UniverSVM, яка застосовує процедуру увігнуто-

опуклої оптимізації (Concave-Convex Procedure, CCCP), а також meanS3VM, де використовується змінна оптимізація (Alternating optimization).

Крім цих підходів, які зосередженні на зменшенні обчислювальної складності алгоритму S3VM, проводяться також дослідження щодо вирішення другої важливої проблеми методу — “безпеки” його використання, тобто перевірки, у яких випадках наявність нерозмічених даних допомагає при навчанні моделі, і в яких – погіршує її точність. Результатами цих досліджень є наприклад алгоритм S4VM (Safe Semi-Supervised Support Vector Machine).

Розглянемо більш детально окремі методи реалізації напівкерованої машини опорних векторів.

2.2.1 Реалізація з використанням SDP

Один з підходів до реалізації S3VM полягає у застосуванні методів SDP (Semidefinite Programming – оптимізації з використанням напіввизначених матриць) для спрощення складності обчислень цільової функції.

Для розв’язання обирається двоїста задача. У випадку напівкерованого навчання її формулювання у векторно-матричному вигляді має вигляд

$$\begin{cases} \min\limits_{\vec{Y}} \max\limits_{\vec{\lambda}} 2\vec{\lambda}^T\vec{e} - \vec{\lambda}^T\left(K \circ \vec{Y}\vec{Y}^T\right)\vec{\lambda} \\ 0 \le \lambda_i \le C \\ \vec{Y} = \begin{pmatrix} \vec{Y}_l \\ \vec{Y}_u \end{pmatrix} \\ \vec{Y}_u \in \{-1; 1\}^u \end{cases}, \tag{2.3}$$

де $\vec{\lambda} = (\lambda_1, \dots, \lambda_l, \lambda_{l+1}, \dots, \lambda_n)$ це вектор двоїстих змінних, K – повна матриця ядер, і $\vec{Y}$ – повний вектор міток [27].

SDP підхід має на меті використання деяких спостережень та припущень для спрощення обмеження на матрицю $\Gamma = \vec{Y}\vec{Y}^T$, таким чином, щоб звести її до невід'ємно визначеної матриці.

Отримана після проведення таких перетворень задача представлятиме собою SDP задачу, для розв'язання яких вже існують добре досліджені методи. Ці методи можуть знаходити оптимальне рішення за поліноміальний час [28], що є помітним покращенням порівняно зі стандартним формулюванням S3VM, яке в загальному випадку має експоненційну обчислювальну складність.

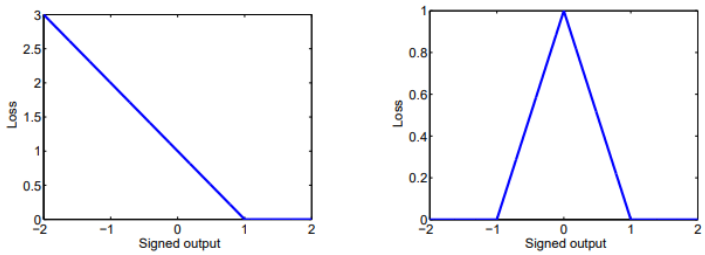
Недоліком цього підходу є те, що такий метод доцільно застосовувати лише на невеликих вибірках. Для можливості його використання на більших наборах даних необхідно приймати додаткові припущення та спрощення, при наявності яких вже не можна гарантувати отримання якісного результату.

2.2.2 Continuation method S3VM

Інший підхід полягає у застосуванні неперервних методів для розв'язання S3VM задачі в її прямому формулюванні [29]. Цільова функція для SSL випадку має вигляд

$$L(\vec{w}) = \min \frac{1}{2}\vec{w}^T\vec{w} + C_1 \sum_{i=1}^{n} l\big(y_i(\vec{w}^T\vec{x}_i + b)\big) + C_2 \sum_{j=n+1}^{m} l\big(|\vec{w}^T\vec{x}_j + b|\big) \qquad (2.4)$$

де $l(k) = \max(0; 1-k)^p$ — функція завісних втрат (hinge loss), яка додає штраф за помилки класифікації. Тут третій доданок відповідає за помилки класифікації на нерозмічених даних, і представляє собою неопуклу функцію, яка породжує локальні мінімуми.



(a) (b)

Рисунок 2.2 – Графіки функцій втрат для розмічених (a) та нерозмічених (b) даних [30]

Окрім локального мінімуму, така цільова функція має ще одну проблему — вона має схильність давати незбалансовані результати, в яких усі нерозмічені елементи вибірки, або більшість з них, класифікуються лише до одного класу. Для вирішення цієї проблеми використовується додаткове обмеження балансу, записане у формі

$$\frac{1}{m}\sum_{i=n-1}^{m}\vec{w}^T\vec{x}_i + b = \frac{1}{n}\sum_{i=1}^{n} y_i\,, \tag{2.5}$$

де n – кількість розмічених тренувальних даних, а $(m-n)$ це кількість нерозмічених об’єктів.

Основна ідея спрощення задачі за допомогою неперервного методу полягає в тому, щоб замість початкової цільової функції мінімізувати спочатку її згладжений варіант. При достатньому рівні згладжування задача оптимізації стане опуклою, і для неї можна буде знайти глобальний мінімум. Після цього рівень згладжування буде поступово зменшуватися, і на кожному кроці знаходитиметься нове значення мінімуму на основі попереднього. Алгоритм продовжуватиме працювати до тих пір, доки коефіцієнт згладжування не досягне нуля [30].

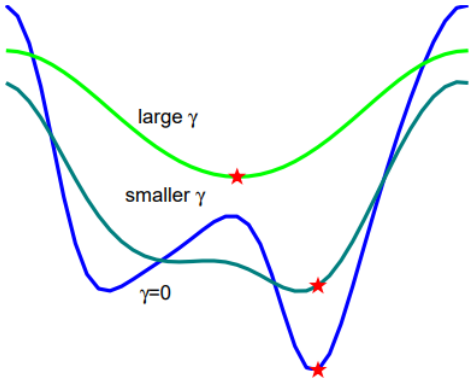


Рисунок 2.3 – Процес згладжування цільової функції [30]

Алгоритм такої реалізації S3VM можна сформулювати наступним чином:

0) Початкові умови: маємо функцію розподілу f, початкову точку x_0 і ряд коефіцієнтів згладжування $\gamma_0 > \gamma_1 > \cdots > \gamma_{p-1} > \gamma_p = 0$;
1) Обираємо функцію згладжування $f_\gamma(x)$, наприклад гауссіан;
2) Ітеративно знаходимо локальні мінімуми $\vec{x}_{i+1}$ для $f_{\gamma i}$ з використанням відповідного коефіцієнту γi, беручи розв'язок $\vec{x}_i$ з попереднього кроку за початкову точку;
3) Дійшовши до $i = p, \gamma_i = \gamma_p = 0$, отримуємо остаточний результат

Другий крок, тобто оптимізація $L_\gamma(\vec{w})$, може виконуватися наприклад за допомогою градієнтного спуску.

Як і SDP метод, підхід з використанням неперевного згладжування зосереджений на пришвидшенні обчислень алгоритму. При вдалому виборі функції згладжування, кількості ітерацій та гіперпараметрів моделі така реалізація S3VM матиме кубічну обчислювальну складність [30].

Певним недоліком continuation методу є складнощі при роботі з багатокласовими задачами. Залежно від наявного набору даних, модель з такою реалізацією може показувати дещо нижчу точність, ніж її аналоги, які базуються на інших підходах.

2.2.3 “Безпечний” алгоритм S3VM

Як показують емпіричні експерименти, у багатьох випадках використання немаркованих даних разом з розміченими даними при навчанні моделі дає можливість помітно покращити її точність. Тим не менш, також трапляються випадки, коли наявність нерозмічених зразків у навчальній вибірці не тільки не допомагає при вирішенні задач, але й призводить до погіршення результатів [31].

Для того, щоб ця проблема не давала значного впливу на результати методу S3VM, було запропоновано його модифікацію, яка зосереджена на покращенні “безпеки” алгоритму. Під безпекою тут мається на увазі, що точність та ефективність застосування S3VM не буде статистично значно гіршою, ніж при використанні SL підходу, незалежно від досліджуваної вибірки.

2.2.3.1 S3VM-us

Один з варіантів підвищення безпеки полягає в тому, щоб враховувати лише ті нерозмічені зразки, які з більшою імовірністю несуть в собі корисну інформацію щодо загального розподілу даних, і відсіювати ті з них, які є більш “ризикованими”. Для цього активно використовується припущення кластерності, і UL алгоритм ієрархічної кластеризації як додатковий крок при аналізі вибірки. Відповідний модифікований метод має назву S3VM-us.

Для оцінки надійності того чи іншого нерозміченого елементу x_j з набору даних (для випадку бінарної класифікації між класами $Y_i = \{-1; 1\}$), використовуються найкоротші відстані p_{j-l} та n_{j-l} від нього до найближчих представників першого та другого класів — серед розмічених даних. Різниця між p_{j-l} і n_{j-l} вважається оцінкою надійності. Інтуїтивно, якщо ця різниця буде більшою, тобто x_j знаходитиметься значно ближче до об’єкту з одного класу, ніж до об’єкту з іншого, то x_j матиме більшу надійність. І навпаки, якщо $p_{j-l} \approx n_{j-l}$,

то складно чітко визначити, до якого саме класу належить x_j, відповідно використовувати цей елемент вибірки для навчання моделі може бути ризиковано.

Алгоритм S3VM-us модифікації можна записати наступним чином [26]:

0) Задаються початкові дані: тренувальний набір $D = \{\{x_i, y_i\}_{i=1}^{l}, \{x_j\}_{l+1}^{l+u}\}$, що включає як розмічені, так і нерозмічені зразки; y^{svm} — мітки для нерозмічених елементів x_j, визначені за допомогою стандартної SVM реалізації для випадку навчання з учителем; y^{s3vm} — мітки для x_j, отримані з використанням S3VM; коефіцієнт ϵ;
1) Знаходиться набір немаркованих елементів, для яких прогнози SVM та S3VM не співпадають $S = \{x_j | y^{svm}(x_j) \neq y^{s3vm}(x_j), j = l+1, \dots, l+u\}$;
2) Виконується ієрархічна кластеризація по D і визначаються індекси екземплярів отриманих кластерів $\{Z_i\}_{i=1}^{l+u-1}$
3) Для кожного $x_j \in S$, позначаємо як $Z_{p_{j-l}}$ (і відповідно $Z_{n_{j-l}}$) перший набір, що містить x_j і хоча б один розмічений елемент з “додатного” (і відповідно “від’ємного”) класу. Також знаходяться значення $t_{j-l} = p_{j-l} - n_{j-l}$.
4) Формується набір нерозмічених зразків з високою надійністю $B = \{x_j \in S \,|\, |t_{j-l}| \geq \epsilon |l+u|, j = l+1, \dots, l+u\}$;
5) Якщо $\sum_{x_j \in B} \left(y^{s3vm}(x_j) - y^{svm}(x_j)\right) t_{j-l} \geq 0$, то для всіх елементів $x \in B$ використовується прогноз, зроблений за допомогою S3VM. У іншому випадку — прогноз SVM;
6) Для всіх $x \notin B$ також використовуються результати SVM.

Як показують результати практичного застосування цієї модифікації, вона помітно покращує надійність S3VM алгоритмів. Тим не менш, такий результат досягається за рахунок помітного погіршення продуктивності, що не завжди є

допустимим. Для забезпечення як безпеки, так і достатньої продуктивності алгоритму пропонується підхід Safe Semi-Supervised SVM.

2.2.3.2 S4VM

Як вже згадувалося в одному з попередніх підрозділів, S3VM методи використовують припущення Low-density Separation, згідно з яким межа поділу класів матиме велику ширину межі і пролягатиме в областях низької щільності. Тим не менш, у загальному випадку таких роздільних гіперплощин може бути декілька, і некоректний вибір однієї з них може призвести до значного погіршення продуктивності алгоритму.

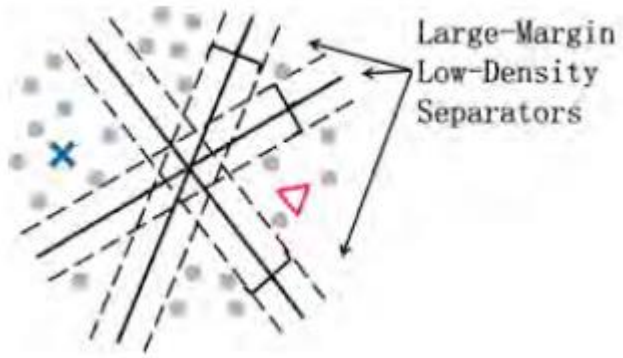


Рисунок 2.4 – Зображення меж розподілу, що мають широкий відступ і знаходяться в областях низької щільності [26]

Саме це спостереження допомогло у формуванні основної ідеї S4VM методу. Фактично, загальний підхід цього методу можна визначити наступним чином:

0) Нехай дано набір тренувальних даних $D = \{\{x_i, y_i\}_{i=1}^{l}, \{x_j\}_{l+1}^{l+u}\}$;
1) На основі цих даних будується набір різноманітних гіперплощин (сепараторів), що пролягають в областях низької щільності і мають широкий відступ — $\{\hat{y}_t\}_{t=1}^{T}$;

2) Елементам $\{x_j\}_{l+1}^{l+u}$ призначаються мітки $\vec{y} = \{y_{l+1}, \dots, y_{l+u}\}$ таким чином, щоб максимізувати покращення продуктивності для будь-якої (навіть найгіршої) межі поділу $\hat{y}_t, t = 1, \dots T$.

В даному випадку “покращення продуктивності” визначається як різниця між приростом кількості елементів, яким було присвоєно правильні мітки, та приростом кількості тих елементів, на яких модель помилилася. У формальному записі ця умова має вигляд

$$\max_{\vec{y} \in \{-1;1\}^u} gain(\vec{y}, \hat{y}, \vec{y}^{svm}) - \lambda\, loss(\vec{y}, \hat{y}, \vec{y}^{svm}), \qquad (2.6)$$

де параметр λ відповідає за те, який ризик вважається допустимим.

Алгоритм S4VM практично завжди показуватиме як мінімум не гіршу точність, ніж стандартний SVM, що використовує для навчання лише розмічені дані, а це дозволяє використовувати його у задачах, де надійність моделі є критичним показником (наприклад у медичній сфері). Проте в якості поступки для досягнення такої надійності даний метод потребує більше часу для виконання розрахунків, оскільки має додатковий етап, який відсутній у інших реалізаціях S3VM – генерацію набору різноманітних роздільних гіперплощин (сепараторів).

2.3 Вибір алгоритму для програмної реалізації

Проаналізувавши існуючі на даний момент алгоритми напівкерованої машини опорних векторів, для практичної реалізації було вирішено обрати підхід з використанням квазі-ньютонівських методів оптимізації. Він належить до тієї ж категорії методів, що і алгоритм неперевного згладжування, розглянутий раніше, і зосереджений на розв’язанні S3VM задачі в прямій постановці за допомогою ефективних алгоритмів оптимізації другого порядку [24].

На відміну від continuation методу, у якому оптимізація відбувається за допомогою градієнтного спуску (тобто використовується інформація про перші похідні цільової функції), квазі-ньютонівський алгоритм застосовує матрицю похідних другого порядку – гессіан, що дозволяє досягати збіжності за значно меншу кількість кроків.

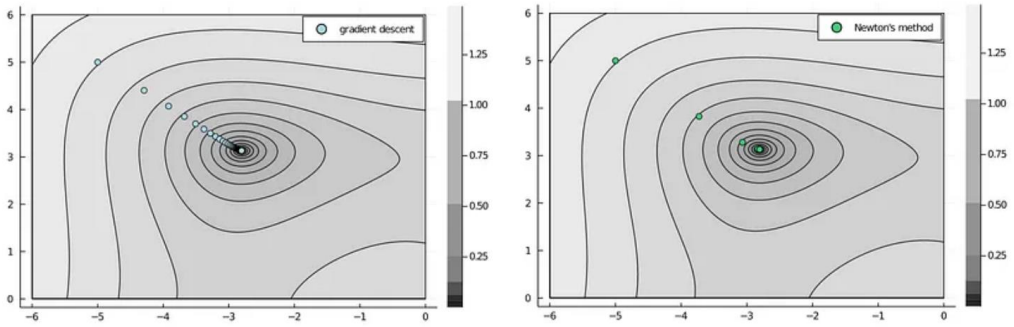


Рисунок 2.5 – Порівняння швидкості збіжності градієнтного спуску (229 кроків) та Ньютонівскього методу (6 кроків) [32]

Обчислення гессіану в загальному випадку є затратною операцією, тому квазі-ньютонівські методи натомість використовують певну його апроксимацію, що дозволяє помітно пришвидшити обчислення.

2.3.1 Визначення критерію оптимізації

Загальна постановка задачі S3VM може записуватися як:

$$\min \|\vec{w}\|^2 + C_1 \sum_{i=1}^{l} \xi_i + C_2 \sum_{j=l+1}^{l+u} \xi_j \,, \tag{2.7}$$

де $\|\vec{w}\|^2$ відповідає за ширину відступу гіперплощини поділу у певному просторі, визначеному функцією ядра, а ξ_i та ξ_j – помилки класифікації на маркованих та немаркованих даних відповідно. C_1 та C_2 це параметри регуляризації, які задаються користувачем.

В практичній реалізації з використанням квазі-ньютонівських методів оптимізації цей критерій оптимізації було переформульовано наступним чином:

- Норму $\|\vec{w}\|^2$ визначимо у вигляді $\sum_{i=1}^{n}\sum_{j=1}^{n} c_i c_j k(\vec{x}_i, \vec{x}_j)$, де $\vec{c}$ – вектор коефіцієнтів (прихованих параметрів) як на марковані, так і немарковані дані;
- Помилки на розмічених даних ξ_i визначатимуться диференційованою функцією логістичних втрат $L(y, f(\vec{x})) = \frac{1}{\gamma}\log(1 + exp(\gamma(1 - yf(\vec{x})))$ (рисунок 2.6 (а));
- Помилки на нерозмічених даних ξ_j визначатимуться диференційованою функцією $L(f(\vec{x})) = \exp(-sf(\vec{x})^2)$, де $s = 3$ (рисунок 2.6 (б))

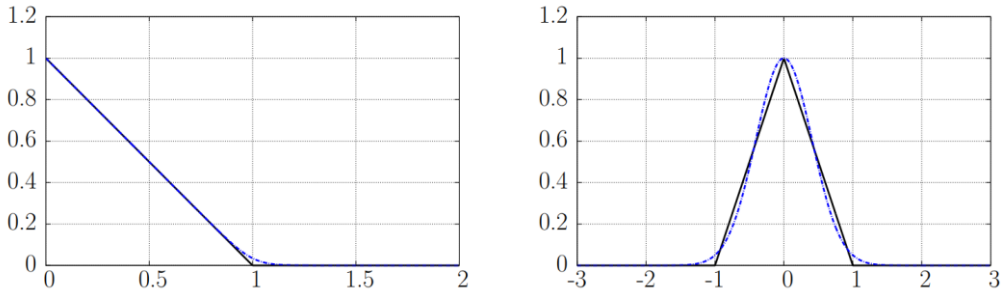


Рисунок 2.6 – Графіки диференційованих функцій втрат (пунктирні лінії) для розмічених (а) та нерозмічених (b) даних [24]

Тобто остаточний критерій оптимізації має вигляд

$$\min \lambda \sum_{i=1}^{n}\sum_{j=1}^{n} c_i c_j k(\vec{x}_i, \vec{x}_j) + \frac{1}{l}\sum_{i=1}^{l}\frac{1}{\gamma}\log(1 + exp(\gamma(1 - y_i f(\vec{x}_i))) + \\ + \frac{\lambda'}{u}\sum_{j=l+1}^{l+u} \exp\left(-3f(\vec{x}_j)^2\right) \quad (2.8)$$

2.3.2 Опис структури алгоритму оптимізації

Основою оптимізації заданої цільової критерії буде процедура ітеративного знаходження її апроксимованого гессіану. Для цього буде використано алгоритм L-BFGS, який належить до класу квазі-ньютонівських методів. Загальний алгоритм S3VM в обраному підході буде працювати за наступними кроками:

0) Визначення початкових умов: задаються набір розмічених даних, набір нерозмічених даних, параметри моделі λ, λ', початкова апроксимація гессіану $H_0 = \eta I, \eta > 0$ та ряд коефіцієнтів згладжування $0 < \alpha_1 < \ldots < \alpha_\tau$;
1) Знаходяться значення вектору параметрів $\vec{c}_0$, з використанням лише розмічених даних;
2) Циклічно проходячи від $i = 1$ до τ проводиться певна кількість ітерацій L-BFGS методу для знаходження гессіану та оптимального значення $\vec{c}_k$ кожному кроці;
3) Отриманий на останньому кроці вектор параметрів $\vec{c}_\tau$ є оптимумом цільової функції, і може використовуватися для визначення міток нерозмічених даних

Значення мітки знаходиметься як

$$y = f(\vec{x}) = \sum_{i=1}^{n} c_i k(\vec{x}_i, \vec{x}) \tag{2.9}$$

2.3.3 Вибір функції ядра

Для практичної реалізації було вирішено вибрати дві функції ядра, на випадок якщо одна з них буде погано підходити для досліджуваного набору даних.

Першою функцією обрано RBF ядро, оскільки воно застосовується найчастіше і досить гарно працює на лінійно нероздільних вибірках.

Другим вибрано лінійне ядро, оскільки воно дозволяє помітно зменшити обчислювальну складність алгоритму для випадку багатовимірних даних.

2.3.4 Проблематика мультикласової задачі

Сама суть методу опорних векторів полягає у знаходженні однієї найкращої площини розподілу даних, тому його класична реалізація не підходить для розв'язання задач, в яких елементи можуть належати більш ніж до двох класів.

У випадку SVM при навчанні з учителем для роботи з такими вибірками зазвичай застосовується модифікований підхід під назвою "один проти одного" (one-vs-one). Його ідея полягає у побудові окремої SVM для кожної пари класів з тренувальної вибірки, і вибору найбільш "надійних" результатів прогнозування з цих моделей для присвоєння відповідних міток тестовій вибірці.

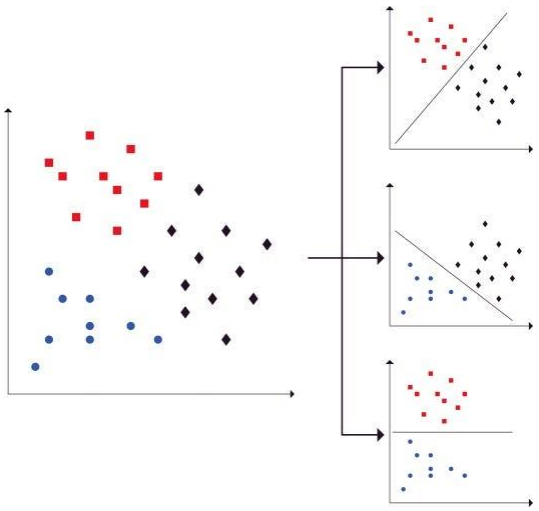


Рисунок 2.7 – Демонстрація підходу One-vs-One [33]

На жаль, при використанні S3VM для напівкерованого навчання застосувати такий підхід не вдасться. Причина цього полягає в тому, що нам в загальному випадку невідомі мітки немаркованих даних, які застосовуються для навчання моделі, і відповідно ми не можемо виділити окремі пари класів.

Внаслідок цього буде застосовано інший метод розв’язання мультикласової задачі – підхід “один проти всіх” (one-vs-rest). Його реалізація передбачає побудову окремої моделі для кожного класу з вибірки, а усі інші класи при цьому будуть вважатися за один. Таким чином, для набору з N класами матимемо N моделей, кожна з яких відповідатиме на запитання, належить певний елемент до відповідного класу, чи до одного з інших класів.

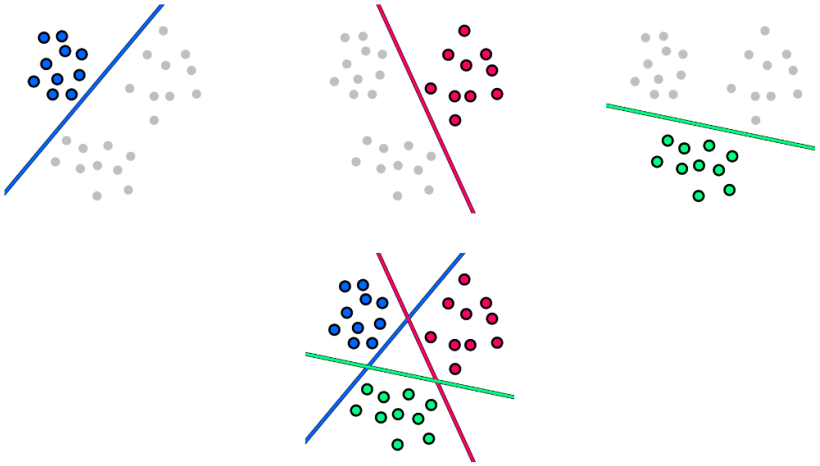


Рисунок 2.8 – Демонстрація підходу One-vs-Rest [33]

Варто зауважити, що для можливості реалізації такого підходу необхідно, щоб S3VM модель повертала результати у вигляді імовірностей — саме вірогідність належності кожного елементу до того чи іншого класу використовується для вибору остаточної мітки. Для цього визначену раніше функцію розподілу $y = f(\vec{x}) = \sum_{i=1}^{n} c_i k(\vec{x}_i, \vec{x})$ у випадку багатокласової вибірки буде модифіковано таким чином, щоб отримані результати лежали у межах від нуля до одиниці.

2.4 Класифікація можливих вибірок для навчання моделі

Для перевірки якості методів напівкерованого навчання важливо дослідити їх результати на різноманітних наборах даних. Це дозволить оцінити надійність застосовуваних методів, їх здатність до узагальнення і можливість

ефективно використовувати нерозмічені дані для покращення роботи моделі, а також знайти їхні недоліки та обмеження.

Розглянемо різні можливі варіанти вибірок даних залежно від того, яким є розподіл класів всередині них:

1) Вибірки з класами низької щільності – у них класи мають відносно небагато елементів, які при цьому знаходяться не дуже близько один до одного;
2) Вибірки з класами високої щільності – протилежний до попереднього випадок, коли у класах є багато об’єктів, відстані між якими є досить малими;
3) Вибірки з класами, що перетинаються – ускладнення для задач класифікації, при якому класи не є лінійно роздільними і не мають чіткої межі поділу;
4) Вибірки із зашумленими даними – імітують реальні ситуації, у яких наявні марковані дані можуть містити помилки чи невідповідності;
5) Вибірки з ієрархічною структурою розподілу даних – у них одні класи знаходяться всередині інших, що ускладнює знаходження межі їх розподілу;
6) Незбалансовані вибірки – випадок, коли кількість елементів у різних класів помітно відрізняється

2.5 Висновки до другого розділу

У даному розділі було сформульовано загальну постановку задачі для S3VM методу та розглянуто декілька можливих його реалізацій. Крім того, було вибрано алгоритм для програмної реалізації, визначено його цільову функцію і спосіб оптимізації, та визначено підхід до розв’язання мультикласової задачі. Також було перелічено кілька варіантів вибірок даних, які було б доцільно дослідити для визначення ефективності побудованої моделі.

РОЗДІЛ 3 РЕАЛІЗАЦІЯ ПРОГРАМНОГО ПРОДУКТУ ТА АНАЛІЗ РЕЗУЛЬТАТІВ

3.1 Вибір засобів програмування

3.1.1 Вибір мови програмування та середовища розробки

В попередньому розділі для реалізації було вибрано модель напівкерованої машини опорних векторів з використанням квазі-ньютонівських методів оптимізації.

Для розробки програмного продукту, обробки вибірок даних та імплементації обраної моделі було вирішено обрати мову програмування Python та інтерактивне середовище програмування Google Colaboratory.

Мова Python є однією з найпопулярніших мов програмування для машинного навчання та аналізу даних. Її перевагами є легкість використання, досить висока продуктивність і наявність великої кількості open-source бібліотек, які спрощують виконання багатьох різноманітних операцій.

Google Colaboratory – це безкоштовний хмарний сервіс на основі Jupyter Notebook, який дозволяє використовувати хмарні ресурси (зокрема RAM та графічні процесори) для прискорення виконання обчислень, а також є легким у налаштуванні, дає можливість відстежувати історію внесених змін і продовжувати розробку з різних пристроїв без необхідності використання сторонніх систем контролю версій.

3.1.2 Вибір додаткових бібліотек

Основною частиною роботи є саме модель напівкерованої машини опорних векторів. Для імплементації S3VM мовою програмування Python існує бібліотека semisupervised [35], в якій містяться реалізації для методів TSVM

(Transductive SVM) та QNS3VM (Quasi-Newton S3VM). Проте відповідні реалізації використовують деякі застарілі методи та функції з інших бібліотек, які на даний момент вже не підтримуються, тому було вирішено відмовитися від використання цієї бібліотеки.

Замість неї було взято за основу інше open-source рішення, яке не було сконвертовано у бібліотеку Python, і проведено його модифікацію та доповнення для роботи з більшими обсягами даних та з мультикласовими задачами.

Крім того, при розробці програмного продукту було використано декілька інших бібліотек для роботи з даними, оцінки продуктивності моделей та графічного відображення результатів.

3.2 Представлення результатів роботи програми

Для дослідження ефективності побудованої моделі було вибрано декілька різних наборів даних і проведено аналіз отриманих результатів в порівнянні з результатами класичного SVM.

3.2.1 Випадок класів, що не перетинаються

Спочатку було розглянуто найпростіший випадок – коли у вибірці з двовимірними даними наявні два класи, що не перетинаються.

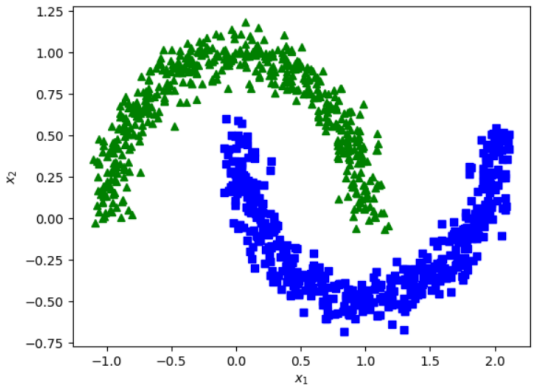


Рисунок 3.1 – Класи, що не перетинаються

Для першої перевірки 5% даних було взято як розмічені, а решту як нерозмічені.

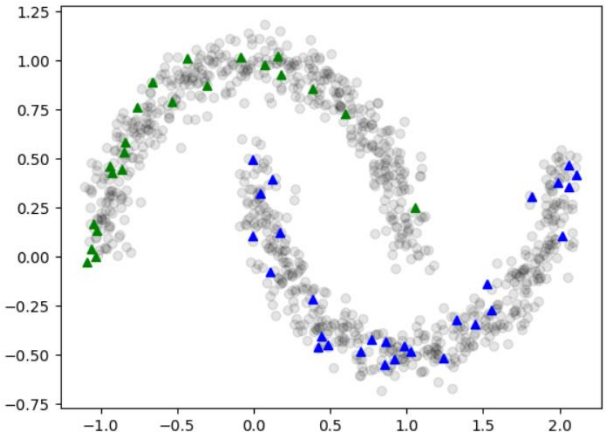


Рисунок 3.2 – Використання лише 5% даних в якості розмічених

В якості метрики оцінювання було обрано Accuracy score, тобто відсоток правильно розмічених елементів. Отримали наступні результати:

Таблиця 3.1 – Метрики якості для першого набору даних

Модель	Accuracy score
SVM	0.923
qnS3VM	1

На графіках показано, як SVM та S3VM класифікували вибірку. Тут і далі жовтим кольором позначено немароквані елементи, які було класифіковано правильно, а червоним – відповідно неправильні результати класифікації.

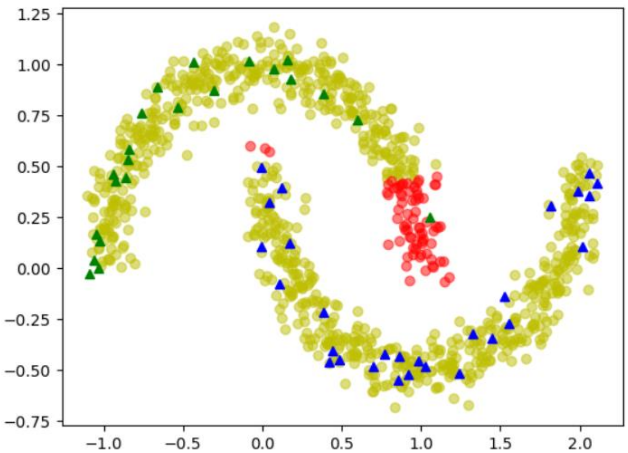


Рисунок 3.3 – Результати класифікації першої вибірки з допомогою SVM

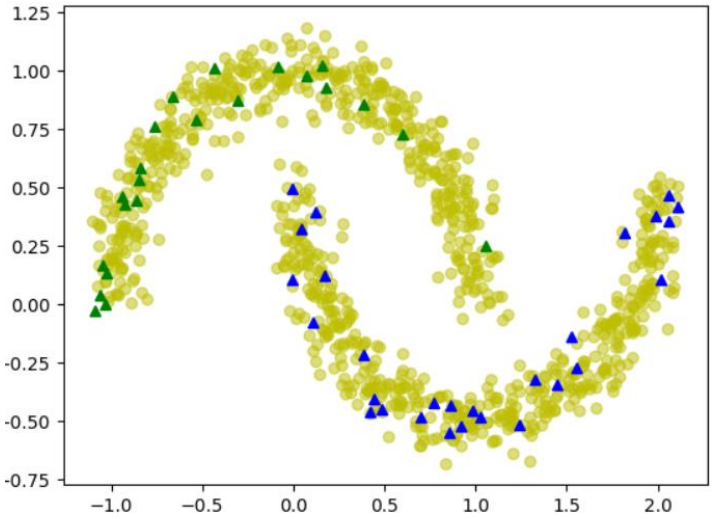


Рисунок 3.4 – Результати класифікації першої вибірки з допомогою S3VM

3.2.2 Випадок класів, що пересікаються

Далі задачу було ускладнено, взявши до розгляду класи, що частково перетинаються.

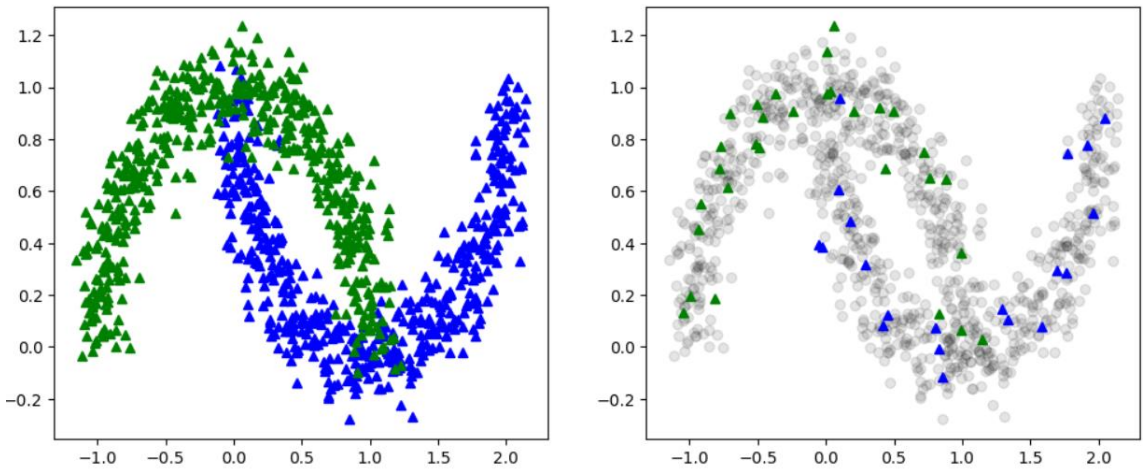


Рисунок 3.5 – Випадок класів, що перетинаються (100% та 5% розмічених даних)

Таблиця 3.2 – Метрики якості для другого набору даних

Модель	Accuracy score
SVM	0.895
qnS3VM	0.942

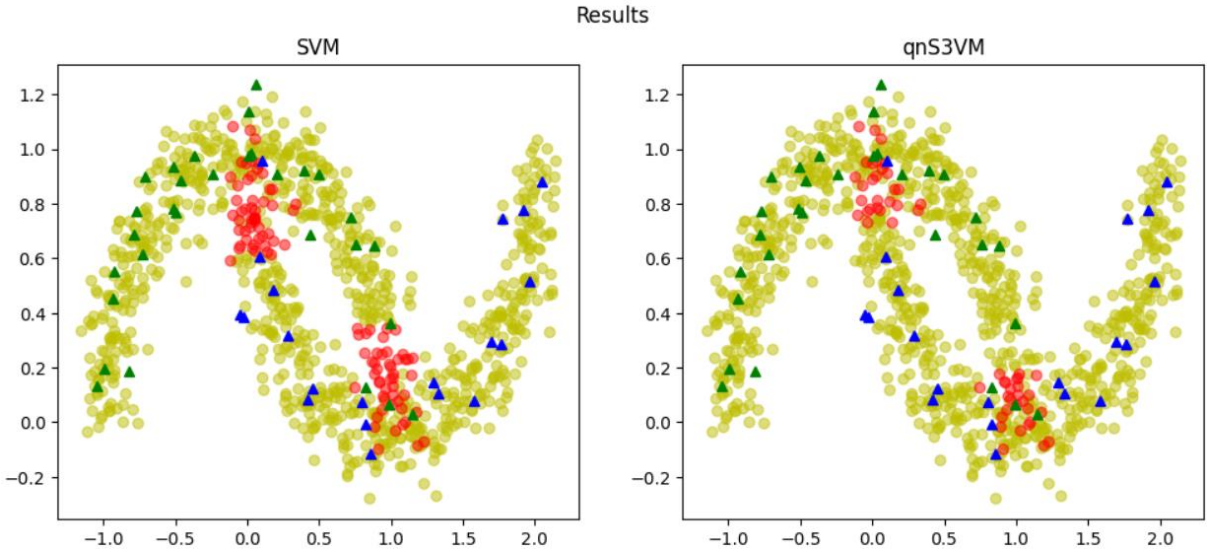


Рисунок 3.6 – Результати класифікації для другого набору даних

Оскільки ця вибірка вже дещо складніша, було додатково проведено дослідження про зміни в точності моделі залежно від кількості розмічених зразків. Отримані результати відображені на наступному графіку.

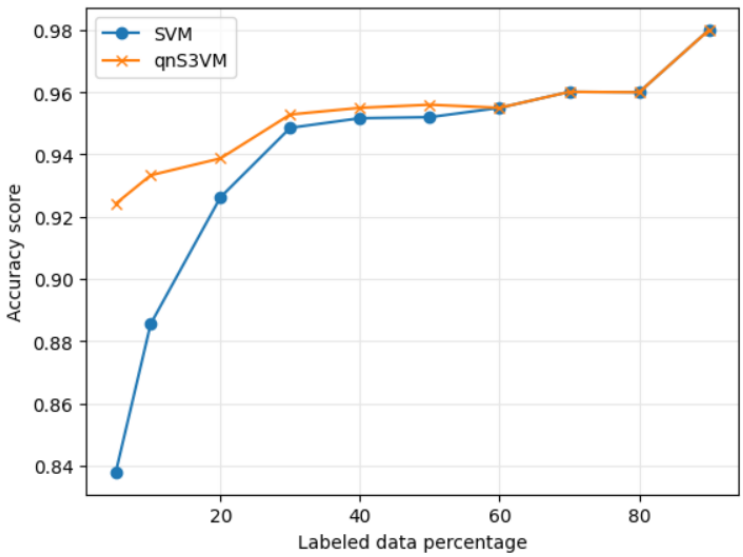


Рисунок 3.7 – Порівняння точності SVM та S3VM на другому наборі даних при різній кількості маркованих елементів

Як можна побачити з графіку, обидві реалізації показують приблизно однакові результати, коли відсоток розмічених даних є високим, а от при його зменшенні стає помітною перевага напівкерованого алгоритму.

3.2.3 Випадок вкладених класів

Наступним для дослідження було взято вибірку, у якій елементи одного класу знаходяться всередині іншого класу.

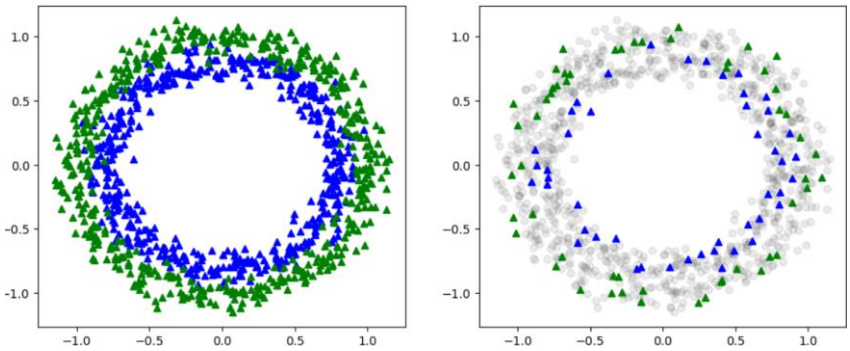


Рисунок 3.8 – Набір даних з вкладеними класами (100% та 10% розмічених елементів)

При використанні RBF ядра класичний SVM метод досить гарно справляється з класифікацією такої вибірки, показуючи точність у 90 та більше відсотків навіть при невеликій кількості (15-20%) розмічених тренувальних даних. При цьому приріст точності за умови використання S3VM сягає не більше одного відсотка. Проте якщо кількість маркованих зразків зменшувати ще більше, то стає помітною різниця в ефективності цих двох методів, як можна побачити на наступному графіку.

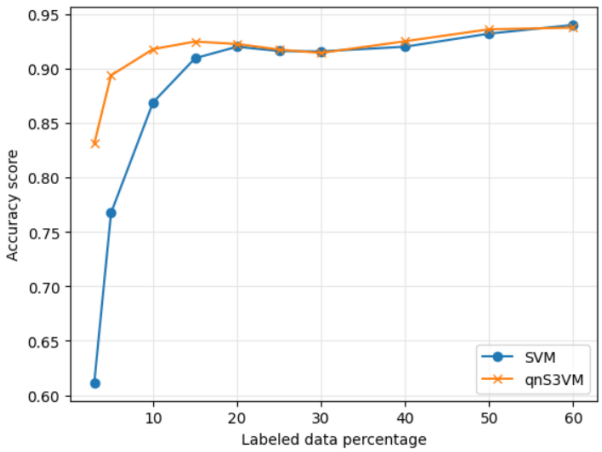


Рисунок 3.9 – Порівняння точності SVM та S3VM на другому наборі даних при різній кількості маркованих елементів

3.2.4 Випадок незбалансованих класів та багатовимірних даних

До цього моменту розглядалися вибірки, у яких кількість елементів у кожному з класів була приблизно однаковою. Більш цікавою є задача незбалансованих класів, оскільки на таких даних SVM зазвичай показує гірші результати.

Крім того, дані у попередніх вибірках мали всього дві ознаки, тоді як у реальних задачах їх кількість часто значно більша.

Для дослідження випадку незбалансованих класів та багатовимірних ознак було вирішено обрати задачу класифікації текстів. Більш конкретно, розглядався набір SMS повідомлень, частина з яких є спамом. На основі нього також досліджувалася швидкодія побудованої S3VM моделі, оскільки цей набір містить в кілька разів більше даних, ніж попередні.

Розподіл даних між класами у цій вибірці можна побачити на наступній діаграмі.

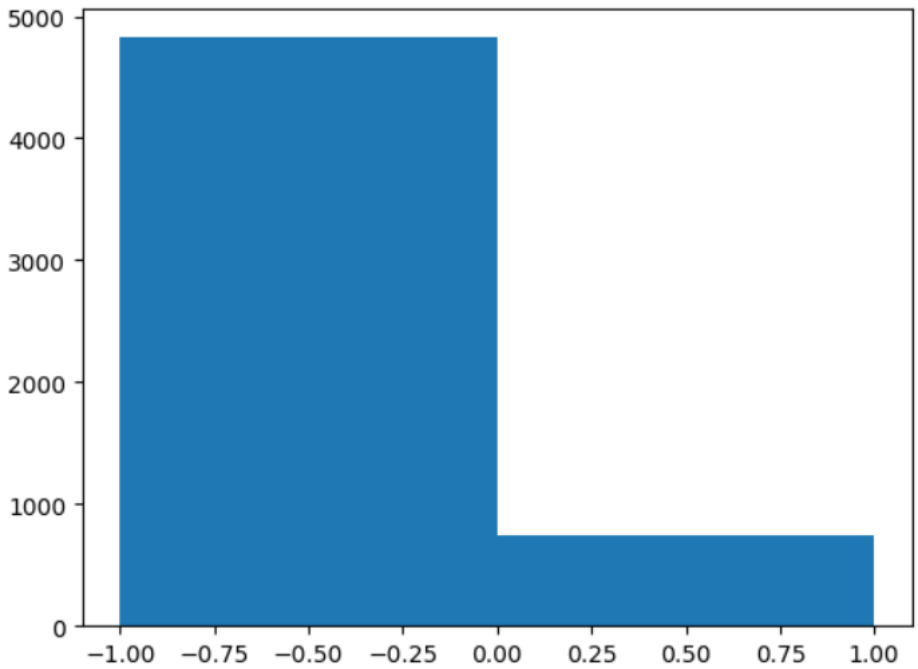


Рисунок 3.10 – Розподіл даних у вибірці для задачі класифікації текстів

Набір даних в початковому варіанті має всього одну ознаку – власне текст повідомлення. Зрозуміло, що метод опорних векторів з такими даними працювати не може, тому було виконане певне перетворення даних за допомогою методів NLP (Natural Language Processing). В результаті кожен елемент модифікованої вибірки було представлено у вигляді вектору ознак, у якому кожне ненульове значення дорівнює значенню TF-IDF (Term Frequency – Inverse Document Frequency) показника для відповідного слова з початкового повідомлення.

Category	Message
ham	Go until jurong point, crazy.. Available only in bugis n grea...
ham	Ok lar... Joking wif u oni...
spam	Free entry in 2 a wkly comp to win FA Cup final tkts 21st May...
ham	U dun say so early hor... U c already then say...
ham	Nah I don't think he goes to usf, he lives around here though
spam	FreeMsg Hey there darling it's been 3 week's now and no word ...
ham	Even my brother is not like to speak with me. They treat me l...
ham	As per your request 'Melle Melle (Oru Minnaminunginte Nurungu...
spam	WINNER!! As a valued network customer you have been selected ...
spam	Had your mobile 11 months or more? U R entitled to Update to ...
ham	I'm gonna be home soon and i don't want to talk about this st...
spam	SIX chances to win CASH! From 100 to 20,000 pounds txt> CSH11...
spam	URGENT! You have won a 1 week FREE membership in our £100,000...
ham	I've been searching for the right words to thank you for this...
ham	I HAVE A DATE ON SUNDAY WITH WILL!!

Рисунок 3.11 – Вигляд набору даних до перетворення

```
(0, 11069)    0.40059731892909933
(0, 10695)    0.20657545787061052
(0, 8587)     0.5043175634668231
(0, 7698)     0.3767229062690876
(0, 3062)     0.2911862458354422
(0, 2449)     0.5619631532224204
```

Рисунок 3.12 – Представлення окремого повідомлення з вибірки після перетворення

Оскільки розглядається випадок незбалансованих класів, для оцінки моделі вже не можна використовувати метрику Accuracy, тому замість неї обчислюється F1-Score. Отримані результати можна побачити у наступній таблиці.

Таблиця 3.3 – Метрики для задачі класифікації текстів

Відсоток розмічених даних	SVM F1-score	qnS3VM F1-score
80	0.911	0.952
70	0.886	0.933
60	0.879	0.931
50	0.871	0.925
40	0.831	0.911
30	0.792	0.889
20	0.721	0.874
10	0.55	0.846
5	0.21	0.802

3.2.5 Випадок багатокласової задачі

Для дослідження випадку, коли кількість класів даних у вибірці більша двох, також було вибрано задачу класифікації текстів. Обраний набір даних складається з відгуків та скарг студентів університету у різних сферах їх студентського життя, як наприклад фінансова підтримка, їжа у їдальнях, кар’єрні перспективи, мешкання у гуртожитках тощо. Усього у вибірці представлено 11 різних класів. Над елементами вибірки було проведено ту ж трансформацію, що і для попереднього набору даних.

Genre	Reports
Career opportunities	The lack of job and internship opp...
Academic Support and Resources	Limited access to technology and s...
Online learning	It's frustrating to have to rely ...
Financial Support	The application process for studen...
Athletics and sports	The university needs to prioritiz...
Athletics and sports	It's disappointing that there are...
Online learning	The online format can be especial...
Food and Cantines	35. "The chips in the cafeteria ar...
Financial Support	The thought of graduating with th...
Housing and Transportation	8. "The nearest grocery store is a...
Activities and Travelling	1. "I've been looking for ways to ...
Academic Support and Resources	Mental health concerns have been a...
Food and Cantines	13. "The food options on campus ar...
Online learning	As an online student, I feel isol...
Financial Support	I'm frustrated that students who ...
Housing and Transportation	11. "I have to commute for over an...
Academic Support and Resources	The academic workload has been rea...
Health and Well-being Support	I'm concerned about the quality of...
Career opportunities	I've applied to numerous internshi...
Food and Cantines	34. "The cantine should offer more...

Рисунок 3.13 – Вигляд набору даних для багатокласової задачі

Розподіл даних за класами зображено на наступній гістограмі.

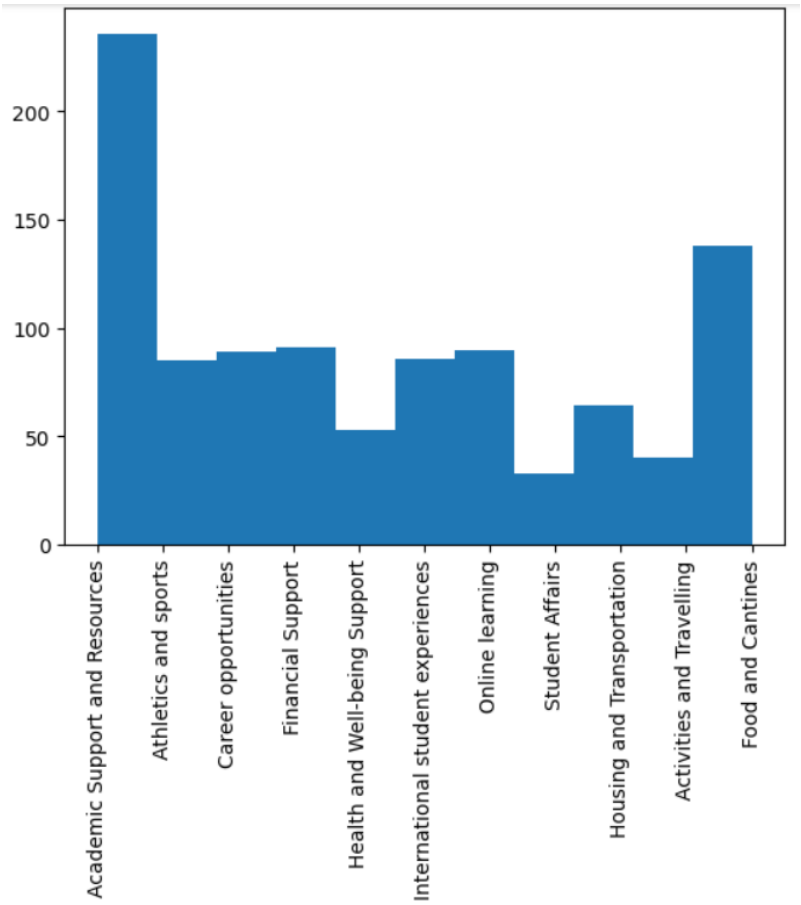


Рисунок 3.14 – Розподіл елементів між класами

Результати роботи моделей оцінювалися метрикою Accuracy.

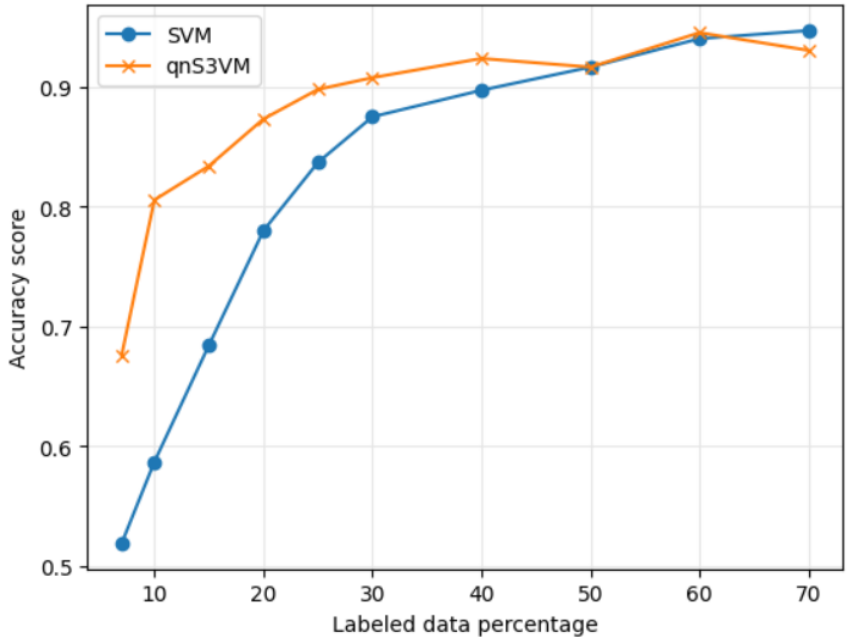


Рисунок 3.15 – Результати роботи моделей для багатокласової задачі

Як можна побачити, для випадку багатокласової класифікації S3VM метод теж показує вищу точність, ніж класичний SVM, при невеликій кількості розмічених елементів вибірки.

3.3 Висновки до розділу 3

У даному розділі було обґрунтовано вибір мови програмування та середовища розробки, а також представлено результати моделі для різних наборів даних. Спочатку було розглянуто кілька різних типів вибірок з двовимірними даними, і представлено результати класифікації на графіках. Потім було досліджено ефективність моделі для більш складних та більших за розміром вибірок у задачах класифікації текстів. Для усіх розглянутих випадків було наведено табличне або графічне порівняння результатів реалізованої S3VM моделі та класичного методу опорних векторів, з якого можна побачити перевагу напівкерованого методу при невеликій кількості наявних розмічених даних.

РОЗДІЛ 4. ФУНКЦІОНАЛЬНО-ВАРТІСНИЙ АНАЛІЗ ПРОГРАМНОГО ПРОДУКТУ

У даному розділі проводиться оцінка основних характеристик програмного продукту, розробленого для дослідження ефективності методу напівкерованої машини опорних векторів як способу вирішення проблеми маркування даних.

Нижче наведено аналіз різних варіантів реалізації продукту з метою вибору оптимальної, з огляду як на економічні фактори, так і на характеристики модулю, що впливають на продуктивність роботи і на його сумісність з апаратним забезпеченням. Для цього застосовувався апарат функціонально-вартісного аналізу.

Функціонально-вартісний аналіз (ФВА) передбачає собою технологію, що дозволяє оцінити реальну вартість продукту чи послуги незалежно від організаційної структури компанії. ФВА проводиться з метою виявлення резервів зниження витрат за рахунок ефективніших варіантів виробництва, кращого співвідношення між споживчою вартістю виробу та витратами на його виготовлення. Для проведення аналізу використовується економічна, технічна та конструкторська інформація.

Алгоритм функціонально-вартісного аналізу включає в себе визначення послідовності етапів розробки продукту, визначення повних витрат (річних) та кількості робочих годин, визначення джерел витрат та кінцевий розрахунок вартості програмного продукту.

4.1 Постановка задачі проектування

У роботі застосовується метод ФВА для проведення техніко-економічного аналізу розробки системи навчання та оцінювання продуктивності напівкерованої машини опорних векторів. Оскільки рішення стосовно

проектування та реалізації компонентів, що розробляється, впливають на всю систему, кожна окрема підсистема має її задовольняти. Тому фактичний аналіз представляє собою аналіз функцій програмного для збору, обробки, проведення аналізу даних для класифікації та відображення отриманих результатів.

Технічні вимоги до програмного продукту є наступні:

- Функціонування на персональних комп'ютерах із стандартним набором компонентів;
- Зручність та зрозумілість для користувача;
- Швидкість обробки даних та доступ до інформації у реальному часі;
- Можливість зручного масштабування та обслуговування;
- Мінімальні витрати на впровадження програмного продукту.

4.2 Обґрунтування функцій програмного продукту

Головна функція F_0 – розробка програмного продукту, який використовує напівкеровану машину опорних векторів для задачі класифікації та дозволяє аналізувати отримані результати. Беручи за основу цю функцію, можна виділити наступні:

- F_1 – вибір мови програмування;
- F_2 – вибір методу впровадження алгоритмів;
- F_3 – вибір середовища розробки.

Кожна з цих функцій має декілька варіантів реалізації.

Функція F_1:

a) мова програмування Python;

b) мова програмування C++;

Функція F_2:

a) використання готових бібліотек та open-source реалізацій;

b) написання алгоритмів вручну;

Функція F_3:

a) Google Colaboratory;

b) IDE Jetbrains.

Варіанти реалізації основних функцій наведені у морфологічній карті системи (рис. 4.1).

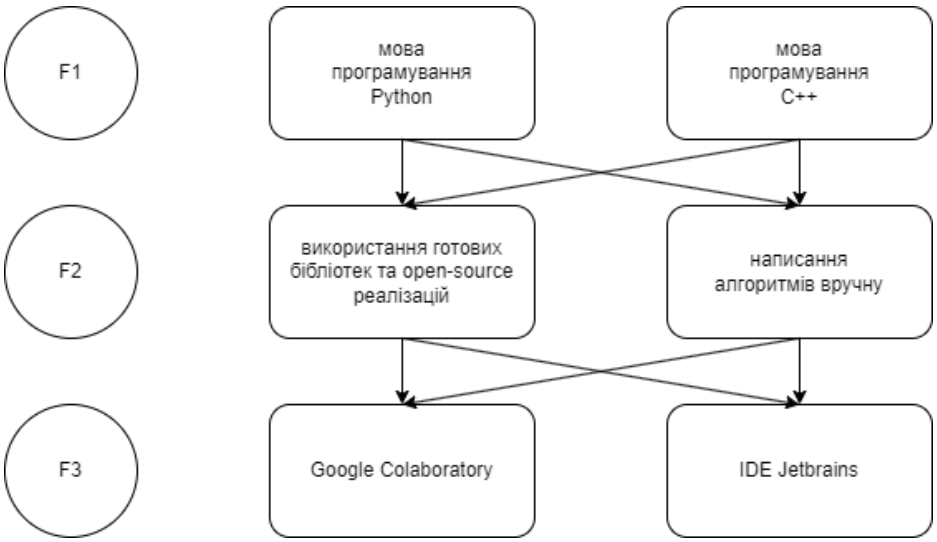


Рисунок 4.1 – Морфологічна карта

Морфологічна карта відображає множину всіх можливих варіантів основних функцій. Позитивно-негативна матриця показана в таблиці 4.1.

Таблиця 4.1 – Позитивно-негативна матриця

Основні функції	Варіанти реалізації	Переваги	Недоліки
F_1	*А*	Швидкість розробки програми, доступність бібліотек, кросплатформеність,	Динамічна типізація, більш повільне виконання коду
	Б	Швидкість роботи програми	Займає багато часу при написанні коду
F_2	*А*	Економія часу розробки, легкість реалізації	Менша гнучкість

	Б	Більша гнучкість, повна відповідність власним потребам	Значно більша кількість часу на реалізацію, можливі помилки
F_3	*А*	Не вимагає інсталяції, дозволяє використовувати хмарні ресурси для швидкого виконання програми на будь-якому ПК	Відсутність можливості дебагу коду, відсутня можливість роботи без інтернету
	Б	Присутня можливість дебагу коду	Необхідна додаткова інсталяція, запуск коду може виконуватися повільніше

На основі аналізу позитивно-негативної матриці робимо висновок, що при розробці програмного продукту деякі варіанти реалізації функцій варто відкинути, тому що вони не відповідають поставленим перед програмним продуктом задачам.

Функція F_1:

Оскільки задача потребує великої частоти зміни коду, витрати часу на його переписування не є бажаними, тому варто віддати перевагу варіанту А і відкинути варіант Б.

Функція F_2:

Існуючі бібліотеки мови Python є надійними, оптимізованими і покривають велику кількість різноманітних задач, тому написання алгоритмів вручну не є оптимальним варіантом, отже відкидаємо варіант Б.

Функція F_3:

Середовище розробки не відіграє велику роль у реалізації даного програмного продукту, тому обидва варіанти можна застосовувати у розробці.

Таким чином, будемо розглядати такі варіанти реалізації програмного продукту:

1) $F_{1A} - F_{2A} - F_{3A}$

2) $F_{1A} - F_{2A} - F_{3Б}$

Для оцінювання якості розглянутих функцій обрана система параметрів, описана нижче.

4.3 Обґрунтування системи параметрів програмного продукту

На основі даних про основні функції, які повинен реалізувати програмний продукт, визначаються основні параметри вибору, які будуть використані для розрахунку коефіцієнту технічного рівня.

Для того, щоб охарактеризувати програмний продукт, будемо використовувати наступні параметри:

- *X1* – швидкодія мови програмування;
- *X2* – об’єм доступної оперативної пам’яті для обчислень та збереження даних;
- *X3* – потенційний об’єм програмного коду.

Гірші, середні і кращі значення параметрів вибираються на основі вимог замовника й умов, що характеризують експлуатацію програмного продукту, як показано у таблиці 4.2.

Таблиця 4.2 – Система параметрів додатку

Найменування параметру	Позначення параметру	Одиниці виміру	Значення параметру		
			Гірші	Середні	Кращі
Швидкодія мови програмування	X1	оп/мс	10000	14000	19000
Об’єм доступної оперативної пам’яті	X2	Гб	6	12	18

Потенційний об’єм програмного коду	X3	кількість рядків коду	2500	1200	500

За даними в таблиці 4.2 будуються графічні характеристики параметрів – (рис. 4.2 – рис. 4.4).

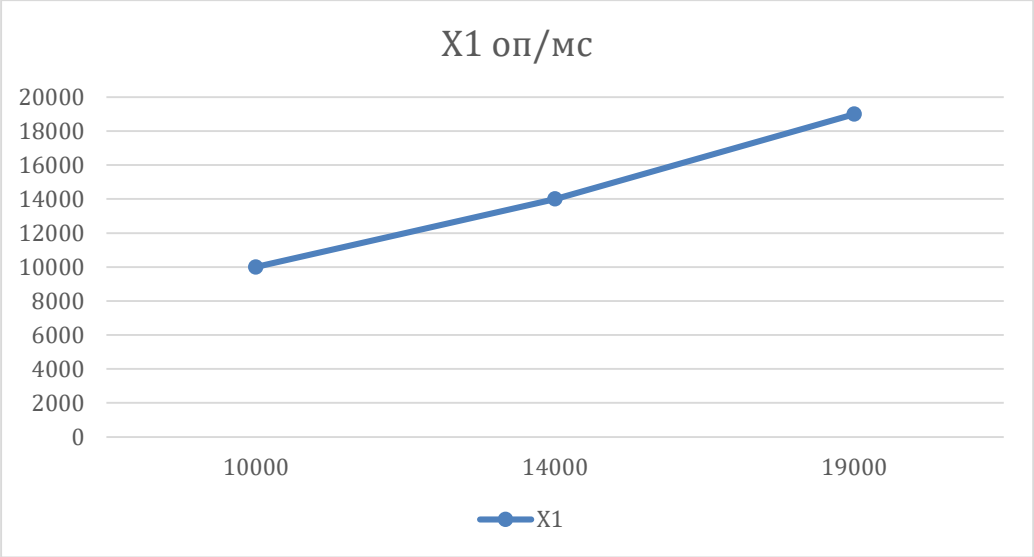


Рисунок 4.2 – X1, швидкодія мови програмування

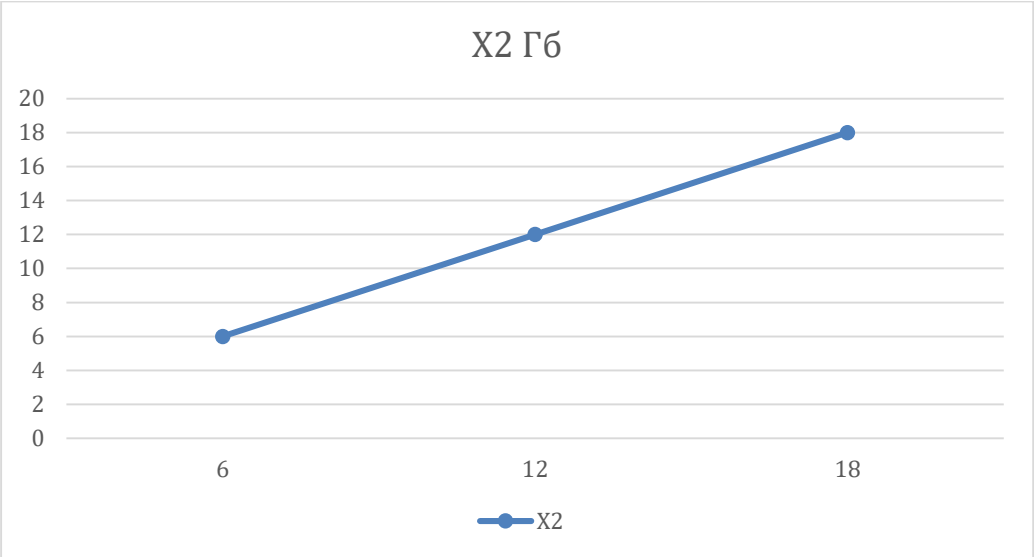


Рисунок 4.3 – X2, об’єм доступної оперативної пам’яті

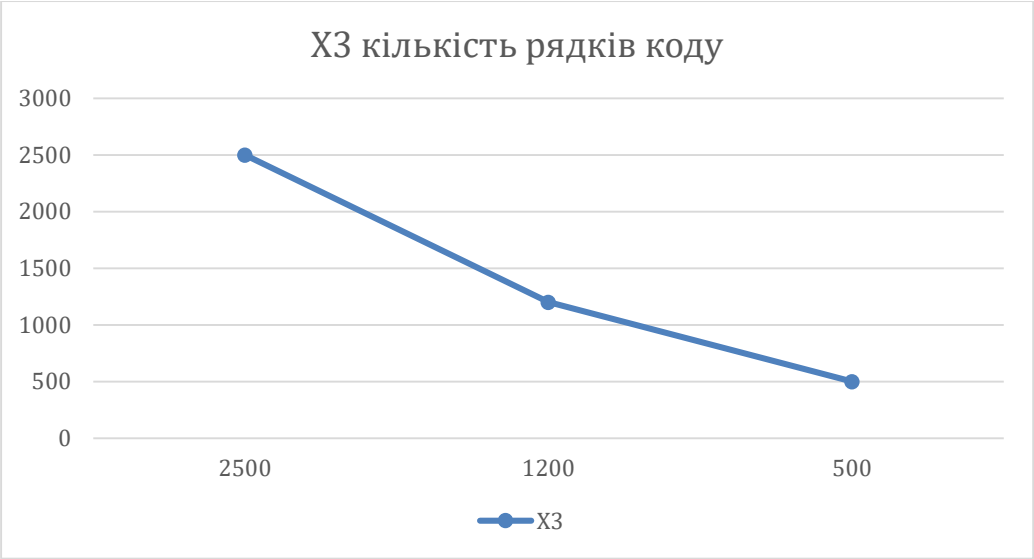


Рисунок 4.4 – X3, потенційний об’єм програмного коду

4.4 Аналіз експертного оцінювання параметрів

Після детального обговорення й аналізу кожен експерт оцінює ступінь важливості кожного параметру для конкретно поставленої цілі – розробки програмного продукту, який дає найкращі результати для побудови S3VM моделі для задачі класифікації.

Значимість кожного параметру визначається методом попарного порівняння. Оцінку проводить експертна комісія із 7 людей. Визначення коефіцієнтів значимості передбачає:

- визначення рівня значимості параметру шляхом присвоєння різних рангів;
- перевірку придатності експертних оцінок для подальшого використання;
- визначення оцінки попарного пріоритету параметрів;
- обробку результатів та визначення коефіцієнту значимості.

Результати експертного ранжування наведені у таблиці 4.3

Таблиця 4.3 – Результати ранжування параметрів

Позначення параметру	Назва параметру	Одиниці виміру	Ранг параметру за оцінкою експерта							Сума рангів R_i	Відхи-лення Δ_i	Δ_i^2
			1	2	3	4	5	6	7			
X1	Швидкодія мови програмування	Оп/мс	2	2	2	3	1	2	2	14	0	0
X2	Об’єм доступної оперативної пам’яті	Гб	3	3	3	2	3	3	3	20	6	36
X3	Потенційний об’єм програмного коду	Кількість рядків коду	1	1	1	1	2	1	1	8	-6	36
	Разом		6	6	6	6	6	6	6	42	0	72

Для перевірки степені достовірності експертних оцінок визначимо наступні параметри:

а) сума рангів кожного з параметрів і загальна сума рангів:

$$R_i = \sum_{j=1}^{N} r_{ij} R_{ij} = \frac{Nn(n+1)}{2} = 42, \quad (4.1)$$

де N – число експертів, n – кількість параметрів;

б) середня сума рангів:

$$T = \frac{1}{n} R_{ij} = 14 \quad (4.2)$$

в) відхилення суми рангів кожного параметра від середньої суми рангів:

$$\Delta_i = R_i - T. \quad (4.3)$$

Сума відхилень по всім параметрам повинна дорівнювати нулю;

г) загальна сума квадратів відхилення:

$$S = \sum_{i=1}^{N} \Delta_i^2 = 72. \quad (4.4)$$

Порахуємо коефіцієнт узгодженості:

$$W = \frac{12S}{N^2(n^3 - n)} = \frac{12 \cdot 72}{7^2(3^3 - 3)} = 0{,}735 > W_k = 0{,}67. \quad (4.5)$$

Ранжування можна вважати достовірним, оскільки знайдений коефіцієнт узгодженості перевищує нормативний, котрий дорівнює 0,67.

Скориставшись результатами ранжирування, проведемо попарне порівняння всіх параметрів і результати занесемо у таблицю 4.4

Таблиця 4.4 – Попарне порівняння параметрів

Параметри	Експерти							Кінцева оцінка	Числове значення
	1	2	3	4	5	6	7		
X1 і X2	<	<	<	>	<	<	<	<	0,5
X1 і X3	>	>	>	>	<	>	>	<	1,5
X2 і X3	>	>	>	>	>	>	>	>	1,5

Числове значення, що визначає ступінь переваги i–го параметра над j–тим, a_{ij} визначається по формулі:

$$a_{ij} = \begin{cases} 1.5 \text{ при } X_i > X_j \\ 1.0 \text{ при } X_i = X_j \\ 0.5 \text{ при } X_i < X_j \end{cases}. \quad (4.6)$$

З отриманих числових оцінок переваги складемо матрицю $A = \|a_{ij}\|$.

Для кожного параметра зробимо розрахунок вагомості $K_{\text{ві}}$ за наступними формулами:

$$K_{\text{ві}} = \frac{b_i}{\sum_{i=1}^{n} b_i} \quad (4.7)$$

$$b_i = \sum_{i=1}^{N} a_{ij} \tag{4.8}$$

Відносні оцінки розраховуються декілька разів доти, поки наступні значення не будуть незначно відрізнятися від попередніх (менше 2%). На другому і наступних кроках відносні оцінки розраховуються за наступними формулами:

$$K_{\text{ві}} = \frac{b_i'}{\sum_{i=1}^{n} b_i'}, \tag{4.9}$$

$$b_i' = \sum_{i=1}^{N} a_{ij} b_j \tag{4.10}$$

Як видно з таблиці 4.5, різниця значень коефіцієнтів вагомості не перевищує 2%, тому більшої кількості ітерацій не потрібно.

Таблиця 4.5 – Розрахунок вагомості параметрів

Параметри X_i	Параметри X_i			Перший крок		Другий крок	
	X1	X2	X3	b_i	K_{bi}	b_i	K_{bi}
X1	1	0,5	1,5	3	0,333	8	0,320
X2	1,5	1	1,5	4	0,444	11,5	0,460
X3	0,5	0,5	1	2	0,222	5,5	0,220
Загалом				9	1	25	1

4.5 Аналіз рівня якості варіантів реалізації функцій

Визначаємо рівень якості кожного варіанту виконання основних функцій окремо.

Абсолютні значення параметрів X1 та X3 відповідають технічним вимогам умов функціонування даного продукту.

Коефіцієнт технічного рівня для кожного варіанту реалізації розраховується так (таблиця 4.6):

$$K_K(j) = \sum_{i=1}^{n} K_{вi,j} B_{i,j}\,, \qquad (4.11)$$

де n – кількість параметрів, $K_{вi}$– коефіцієнт вагомості i–го параметра, B_i – оцінка i–го параметра в балах.

Таблиця 4.6 – Розрахунок показників рівня якості варіантів реалізації

Основні функції	Варіант реалізації	Абсолютне значення параметру	Бальна оцінка параметру	Коефіцієнт вагомості	Коефіцієнт якості
F1	а) X1	14000	14	0,32	4,48
F2	а) X3	1000	16	0,22	3,52
F3	а) X2	12	15	0,46	6,9
	б) X2	7	12	0,46	5,52

За даними таблиці 4.6 за формулою:

$$K_K = K_{\text{ТУ}}[F_{1k}] + K_{\text{ТУ}}[F_{2k}] + \ldots + K_{\text{ТУ}}[F_{zk}], \qquad (4.12)$$

визначаємо рівень якості кожного з варіантів:

$$K_{K1} = 4{,}48 + 3{,}52 + 6{,}9 = 14{,}9;$$

$$K_{K1} = 4{,}48 + 3{,}52 + 5{,}52 = 13{,}52.$$

Як видно з розрахунків, кращим є перший варіант, оскільки він має найбільше значення коефіцієнту технічного рівня.

4.6 Економічний аналіз варіантів розробки програмного продукту

Для визначення вартості розробки програмного продукту спочатку проведемо розрахунок трудомісткості.

Всі варіанти включають в себе два окремих завдання:

1. Розробка проекту програмного продукту;

2. Розробка програмної оболонки.

Завдання 1 за ступенем новизни відноситься до групи А, завдання 2 – до групи Б. За складністю алгоритми, які використовуються в завданні 1 належать до групи 1, а в завданні 2 – до групи 3.

Для реалізації завдання 1 використовується довідкова інформація, а завдання 2 використовує інформацію у вигляді даних.

Проведемо розрахунок норм часу на розробку та програмування для кожного з завдань.

Загальна трудомісткість обчислюється як:

$$\mathrm{T_O} = \mathrm{T_P} \cdot \mathrm{K_П} \cdot \mathrm{K_{СК}} \cdot \mathrm{K_М} \cdot \mathrm{K_{СТ}} \cdot \mathrm{K_{СТ.М}}, \qquad (4.13)$$

де $\mathrm{T_P}$ – трудомісткість розробки програмного продукту, $\mathrm{K_П}$ поправочний коефіцієнт, $\mathrm{K_{СК}}$ – коефіцієнт на складність вхідної інформації, $\mathrm{K_М}$ – коефіцієнт рівня мови програмування, $\mathrm{K_{СТ}}$ коефіцієнт використання стандартних модулів і прикладних програм, $\mathrm{K_{СТ.М}}$ коефіцієнт стандартного математичного забезпечення.

Для першого завдання, виходячи із норм часу для завдань розрахункового характеру степеню новизни А, та групи складності алгоритму 1, трудомісткість дорівнює $\mathrm{T_P} = 45$ людино-днів. Поправочний коефіцієнт, який враховує вид нормативно-довідкової інформації для першого завдання: $\mathrm{K_П} = 1.7$. Поправочний коефіцієнт, який враховує складність контролю вхідної та вихідної інформації для всіх завдань рівний 1: $\mathrm{K_{СК}} = 1$. Оскільки при розробці першого завдання використовуються стандартні модулі, врахуємо це за допомогою коефіцієнту $\mathrm{K_{СТ}} = 0.8$. Тоді загальна трудомісткість програмування першого завдання дорівнює:

$$T_1 = 45 \cdot 1.7 \cdot 0.8 = 61.2 \text{ людино-днів.}$$

Проведемо аналогічні розрахунки для подальших завдань.

Для другого завдання (використовується алгоритм третьої групи складності, степінь новизни Б), тобто $\mathrm{Т_Р} = 26$ людино-днів, $\mathrm{К_П} = 0.9$, $\mathrm{К_{СК}} = 1$, $\mathrm{К_{СТ}} = 0.8$:

$$T_2 = 26 \cdot 0.9 \cdot 0.8 = 18.72 \text{ людино-днів.}$$

Складаємо трудомісткість відповідних завдань для кожного з обраних варіантів реалізації програми, щоб отримати їх трудомісткість:

$$T_{\mathrm{I}} = (61.2 + 18.72 + 4.8 + 18.72) \cdot 8 = 827.52 \text{ людино-годин.}$$

$$T_{\mathrm{II}} = (61.2 + 18.72 + 6.91 + 18.72) \cdot 8 = 844.4 \text{ людино-годин.}$$

Найбільш високу трудомісткість має варіант II.

В розробці беруть участь один програміст з окладом 32000 гривень та аналітик даних з окладом 34000 гривень. Визначимо зарплату за годину за формулою:

$$C_{\text{год}} = \frac{\mathrm{М}}{T_m \cdot t} \text{ грн.,} \tag{4.14}$$

де М – місячний оклад працівників, T_m – кількість робочих днів на місяць, t – кількість робочих годин на день.

$$C_{\text{год}} = \frac{32000+34000}{2 \cdot 21 \cdot 8} = 196.43 \text{ грн.,} \tag{4.15}$$

Тоді розраховуємо заробітну плату за формулою:

$$C_{\text{ЗП}} = C_{\text{год}} \cdot T_i \cdot K_{\text{Д}}, \tag{4.16}$$

де $C_{\text{год}}$ – величина погодинної оплати праці виконавців, T_i – трудомісткість відповідного завдання, $K_{\text{Д}}$ – норматив, який враховує додаткову заробітну плату.

Заробітна плата розробників за варіантами становить:

I. $C_{\text{ЗП}} = 196.43 \cdot 827.52 \cdot 1.2 = 195059{,}7$ грн.

II. $C_{\text{ЗП}} = 196.43 \cdot 844.4 \cdot 1.2 = 199038{,}59$ грн.

Відрахування на соціальний внесок становить 22%

I. $C_{\text{ВІД}} = C_{\text{ЗП}} \cdot 0.22 = 195059{,}7 \cdot 0.22 = 42913{,}134$ грн.

II. $C_{\text{ВІД}} = C_{\text{ЗП}} \cdot 0.22 = 199038{,}59 \cdot 0.22 = 43788{,}49$ грн.

Тепер визначимо витрати на оплату однієї машино-години (C_M).

Так як одна електронно-обчислювальна машина обслуговує одного програміста з окладом 32000 грн., та одного аналітика даних з окладом 34000 грн. з коефіцієнтом занятості 0.2, то для однієї машини маємо:

$$C_{\text{Г}} = 12 \cdot M \cdot K_{\text{З}} = 12 \cdot (32000 + 34000) \cdot 0.2 = 158400 \text{ грн.}$$

З урахуванням додаткової заробітної плати:

$$C_{\text{ЗП}} = C_{\text{Г}} \cdot (1 + K_{\text{З}}) = 158400 \cdot 1.2 = 190080 \text{ грн.}$$

Відрахування на соціальний внесок:

$$C_{\text{ВІД}} = C_{\text{ЗП}} \cdot 0.22 = 41817{,}6 \text{ грн.}$$

Амортизаційні відрахування розраховуємо при амортизації 25% та вартості електронно-обчислювальної машини – 35000 грн.

$$C_A = K_{TM} \cdot K_A \cdot \text{Ц}_{\text{ПР}} = 1.15 \cdot 0.25 \cdot 35000 = 10062.5 \text{ грн.,}$$

де K_{TM} – коефіцієнт, який враховує витрати на транспортування та монтаж приладу у користувача, K_A – річна норма амортизації, $\text{Ц}_{\text{ПР}}$ – ціна приладу.

Витрати на ремонт та профілактику розраховуємо як:

$$C_{\text{Р}} = K_{TM} \cdot \text{Ц}_{\text{ПР}} \cdot K_{\text{Р}} = 1.15 \cdot 35000 \cdot 0.05 = 2012.5 \text{ грн.,}$$

де $K_{\text{Р}}$ – відсоток витрат на поточні ремонти.

Ефективний годинний фонд часу ПК за рік розраховуємо за формулою:

$$\text{Т}_{\text{ЕФ}} = (\text{Д}_{\text{К}} - \text{Д}_{\text{В}} - \text{Д}_{\text{С}} - \text{Д}_{\text{Р}}) \cdot t_{\text{З}} \cdot \text{К}_{\text{В}} = (365 - 104 - 12 - 16) \cdot 8 \cdot 0.7 =$$
$$= 1304.8 \text{ години,}$$

де $Д_К$ – календарна кількість днів у році, $Д_В$, $Д_С$ – відповідно кількість вихідних та святкових днів, $Д_Р$ – кількість днів планових ремонтів устаткування, t – кількість робочих годин на день, $К_В$ – коефіцієнт використання приладу у часі протягом зміни.

Витрати на оплату електроенергії розраховуємо за формулою:

$$С_{ЕЛ} = Т_{ЕФ} \cdot N_C \cdot К_З \cdot Ц_{ЕН} = 1304.8 \cdot 0.6 \cdot 0.2 \cdot 4.799 = 751.4 \text{ грн.},$$

де N_C – середньо-споживча потужність приладу, $К_З$ – коефіцієнт зайнятості приладу, $Ц_{ЕН}$ – тариф за 1 КВт-годин електроенергії.

Накладні витрати рахуємо за формулою:

$$С_Н = Ц_{ПР} \cdot 0.67 = 35000 \cdot 0.67 = 23450 \text{ грн.}$$

Тоді річні експлуатаційні витрати будуть:

$$С_{ЕКС} = С_{ЗП} + С_{ВІД} + С_А + С_Р + С_{ЕЛ} + С_Н, \quad (4.17)$$

$$С_{ЕКС} = 190080 + 41817{,}6 + 10062{,}5 + 2012.5 + 751.4 + 23450 = 268174 \text{ грн.}$$

Собівартість однієї машино-години електронно-обчислювальної машини дорівнюватиме:

$$С_{М-Г} = \frac{С_{ЕКС}}{Т_{ЕФ}} = \frac{268174}{1304.8} = 205.53 \text{ грн/год.}$$

Оскільки в даному випадку всі роботи, які пов’язані з розробкою програмного забезпечення, ведуться на електронно-обчислювальній машині, то витрати на оплату, в залежності від обраного варіанту реалізації, складають:

$$С_М = С_{М-Г} \cdot Т, \quad (4.18)$$

I. $С_М = 205.53 \cdot 827.52 = 170080.19$ грн.

II. $С_М = 205.53 \cdot 844.4 = 173549.53$ грн.

Накладні витрати складають 67% від заробітної плати:

$$С_Н = С_{ЗП} \cdot 0.67 \quad (4.19)$$

I. $С_Н = 195059,7 \cdot 0.67 = 130689.99$ грн.

II. $С_Н = 199038,59 \cdot 0.67 = 133355.86$ грн.

Отже, вартість розробки програмного продукту за варіантами становить:

$$С_{ПП} = С_{ЗП} + С_{ВІД} + С_М + С_Н, \quad (4.20)$$

I. $С_{ПП} = 195059,7 + 42913,134 + 170080.19 + 130689.99 = 538743,01$ грн.

II. $С_{ПП} = 199038,59 + 43788,49 + 173549.53 + 133355.86 = 549732,47$ грн.

4.7 Вибір кращого варіанту з огляду на техніко-економічний рівень

Розрахуємо коефіцієнт техніко-економічного рівня за формулою:

$$К_{ТЕР_j} = \frac{К_{К_j}}{C_{ПП_j}} \quad (4.21)$$

$$К_{ТЕР_1} = \frac{14,9}{538743,01} = 2.77 \cdot 10^{-5}$$

$$К_{ТЕР_2} = \frac{13,52}{549732,47} = 2.46 \cdot 10^{-5}$$

Як бачимо, найбільш ефективним є перший варіант реалізації з коефіцієнтом техніко-економічного рівня $К_{ТЕР_1} = 2.77 \cdot 10^{-5}$.

Цей варіант реалізації має такі параметри:

- мова програмування – Python;
- впровадження алгоритмів з використанням готових бібліотек та open-source реалізацій;
- середовище розробки Google Colaboratory

4.8 Висновки до четвертого розділу

У даному розділі було проведено повний функціонально-вартісний аналіз програмного продукту. Також було знайдено оцінку основних функцій програмного продукту.

В результаті виконання функціонально-вартісного аналізу програмного модулю було визначено та проведено оцінку основних функцій програмного продукту, а також знайдено параметри, які його характеризують.

На основі аналізу вибрано варіант реалізації програмного продукту.

ВИСНОВКИ

У даній роботі було досліджено методи напівкерованого навчання, зокрема напівкеровану машину опорних векторів, як спосіб вирішення проблеми маркування даних.

Була опрацьована наукова література, проаналізовано основні властивості та особливості керованого та напівкерованого навчання, переваги та недоліки відповідних методів. Було досліджено різні підходи до реалізації напівкерованої машини опорних векторів, вибрано оптимальний варіант для програмної реалізації.

Було розроблено програмний продукт, який дозволяє вирішувати задачі бінарної та багатокласової класифікації за допомогою S3VM, та проведено порівняльний аналіз точності побудованої моделі та класичного методу опорних векторів.

На основі отриманих результатів можна стверджувати, що у багатьох випадках при наявності лише невеликої кількості розмічених даних напівкерована машина опорних векторів показує значно вищу точність класифікації, що свідчить про потенційну ефективність використання цього методу для вирішення проблеми маркування даних.

СПИСОК ЛІТЕРАТУРИ

1. IBM: What are Neural Networks? URL: https://www.ibm.com/topics/neural-networks
2. Example of a simple Feed Forward Neural Network. URL: www.researchgate.net/figure/fig1_332049058
3. Explaining Recurrent Neural Networks. URL: https://www.bouvet.no/bouvet-deler/explaining-recurrent-neural-networks
4. The overall architecture of the Convolutional Neural Network (CNN). URL: www.researchgate.net/figure/fig4_331540139
5. Generative Adversarial Network (GAN). URL: https://www.geeksforgeeks.org/generative-adversarial-network-gan/
6. Structure of modular neural network. URL: www.researchgate.net/figure/fig3_328127686
7. Architecture of the RBF neural network. RBF: radial basis function. URL: www.researchgate.net/figure/fig2_333469185
8. Neural Networks, Manifolds, and Topology. URL: https://colah.github.io/posts/2014-03-NN-Manifolds-Topology/
9. A Comprehensive Guide to Convolutional Neural Networks. URL: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53
10. A Gentle Introduction to Generative Adversarial Networks (GANs) URL: https://machinelearningmastery.com/what-are-generative-adversarial-networks-gans/
11. Semi Supervised Learning, Explained with Examples. URL: https://www.altexsoft.com/blog/semi-supervised-learning/
12. IBM: What is supervised learning? URL: https://www.ibm.com/topics/supervised-learning

13. Support Vector Machine: Introduction to Machine Learning Algorithms URL: https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47
14. Support Vector Machine (SVM): A Complete guide for beginners URL: https://www.analyticsvidhya.com/blog/2021/10/support-vector-machinessvm-a-complete-guide-for-beginners/
15. Support Vector Machine (SVM) Algorithm URL: https://www.javatpoint.com/machine-learning-support-vector-machine-algorithm
16. Support Vector Machines: A Simple Tutorial URL: https://svmtutorial.online
17. Недашківська Н.І. Інтелектуальний аналіз даних. Конспект лекцій. 2021 р.
18. Kernel Functions-Introduction to SVM Kernel & Examples. URL: https://data-flair.training/blogs/svm-kernel-functions/
19. Seven Most Popular SVM Kernels. URL: https://dataaspirant.com/svm-kernels/
20. C. M. Bishop. Pattern Recognition and Machine Learning. 2006. 738 с.
21. What is Semi-Supervised Learning. URL: https://machinelearningmastery.com/what-is-semi-supervised-learning/
22. K. P. Bennet, A. Demiriz. Semi-Supervised Support Vector Machines. // Advances in Neural Information Processing Systems 11. 1998.
23. F. Gieseke, A. Airola, T. Pahikkala, O. Kramer. Sparse Quasi-Newton Optimization for Semi-Supervised Support Vector Machines. // Proceedings of the 1st International conference on Pattern Recognition Applications and Methods. 2012. с. 45-54.
24. F. Gieseke, A. Airola, T. Pahikkala, O. Kramer. Fast and Simple Gradient-Based Optimization for Semi-Supervised Support Vector Machines. // Neurocomputing vol. 123. 2014. с. 23-32.
25. X. Zhu. Semi-Supervised Learning Tutorial URL: https://pages.cs.wisc.edu/~jerryzhu/pub/sslicml07.pdf
26. Yu-Feng Li, Zhi-Hua Zhou. Towards Making Unlabeled Data Never Hurt. // IEEE Transactions on Pattern Analysis and Machine Intelligence. 2014.

27. Tijl De Bie, N. Cristianini. Semi-Supervised Learning Using Semi-Definite Programming. // O. Chapelle, B. Scholkopf, A. Zien. Semi-Supervised Learning. 2006. с. 119-135.
28. Tijl De Bie, N. Cristianini. Convex Methods for Transduction. // Advances in Neural Information Processing Systems. 2004.
29. В. Синєглазов, О. Чумаченко. Методи та технології напівкерованого навчання: Курс лекцій. 2022.
30. O. Chapelle, M. Chi, A. Zien. A Continuation Method for Semi-Supervised SVMs. // Proceedings of the 23rd international conference on machine learning. 2006. с. 185-192.
31. A. Singh, R. D. Nowak, X. Zhu. Unlabeled data: Now it helps, now it doesn't. 2009.
32. BFGS in a Nutshell: An Introduction to Quasi-Newton Methods. URL: https://towardsdatascience.com/bfgs-in-a-nutshell-an-introduction-to-quasi-newton-methods-21b0e13ee504
33. Multi-class Classification — One-vs-All & One-vs-One. URL: https://towardsdatascience.com/multi-class-classification-one-vs-all-one-vs-one-94daed32a87b
34. One-versus-All Multi-Class Classification. URL: https://jermwatt.github.io/machine_learning_refined/notes/7_Linear_multiclass_classification/7_2_OvA.html
35. Pypi: semisupervised. URL: https://pypi.org/project/semisupervised/
36. Spam Message Text Classification. URL: https://www.kaggle.com/datasets/team-ai/spam-text-message-classification
37. University Students Complaints & Reports Dataset. URL: https://www.kaggle.com/datasets/omarsobhy14/university-students-complaints-and-reports

ДОДАТОК А ЛІСТИНГ ПРОГРАМИ

```
import nltk
import string
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn import svm, preprocessing
from sklearn.model_selection import train_test_split
from nltk.corpus import stopwords
from sklearn.feature_extraction.text import CountVectorizer, TfidfTransformer
from sklearn.metrics import accuracy_score, f1_score

##########
# Helper functions for
# generating and displaying
# two-dimensional datasets
##########

def make_moons(n_samples: int = 500, *, distance: int = 0.5, noise: float = 0, seed:
int = 0):

    n_samples_out = n_samples // 2
    n_samples_in = n_samples - n_samples_out

    outer_circ_x = np.cos(np.linspace(0, np.pi, n_samples_out))
    outer_circ_y = np.sin(np.linspace(0, np.pi, n_samples_out))
    inner_circ_x = 1 - np.cos(np.linspace(0, np.pi, n_samples_in))
    inner_circ_y = 1 - np.sin(np.linspace(0, np.pi, n_samples_in)) - distance

    X = np.vstack(
        [np.append(outer_circ_x, inner_circ_x), np.append(outer_circ_y, inner_circ_y)]
    ).T
    y = np.hstack(
        [-1*np.ones(n_samples_out, dtype=np.intp), np.ones(n_samples_in,
dtype=np.intp)]
    )
```

```
    generator = np.random.RandomState(seed)
    X += generator.normal(scale=noise, size=X.shape)

    return X, y

def make_circles(n_samples: int = 1000, *, noise: float = 0.08,
                 seed: int =0, factor: float = 0.8):
    if factor >= 1 or factor < 0:
        raise ValueError("'factor' has to be between 0 and 1.")
    n_samples_out = n_samples // 2
    n_samples_in = n_samples - n_samples_out
    # so as not to have the first point = last point, we set endpoint=False
    linspace_out = np.linspace(0, 2 * np.pi, n_samples_out, endpoint=False)
    linspace_in = np.linspace(0, 2 * np.pi, n_samples_in, endpoint=False)
    outer_circ_x = np.cos(linspace_out)
    outer_circ_y = np.sin(linspace_out)
    inner_circ_x = np.cos(linspace_in) * factor
    inner_circ_y = np.sin(linspace_in) * factor

    X = np.vstack(
        [np.append(outer_circ_x, inner_circ_x), np.append(outer_circ_y, inner_circ_y)]
    ).T
    y = np.hstack(
        [-1*np.ones(n_samples_out, dtype=np.intp), np.ones(n_samples_in,
dtype=np.intp)]
    )
    generator = np.random.RandomState(seed)
    X += generator.normal(scale=noise, size=X.shape)

    return X, y

def create_semi_supervised(X, y, percentage_labeled: float = 0.8):
  unlabeled_count = int(len(X) * (1 - percentage_labeled))
  indicies = np.random.choice(X.shape[0], size=unlabeled_count, replace=False)
  uX, uY = X[indicies, :], y[indicies]
  X = np.delete(X, indicies, axis=0)
  y = np.delete(y, indicies, axis=0)
  return (X, y, uX, uY)
```

```
def display_dataset(X, y) -> None:
  plt.plot(X[:, 0][y==1], X[:, 1][y==1], "bs")
  plt.plot(X[:, 0][y==0], X[:, 1][y==0], "g^")

  plt.xlabel(r"$x_1$", fontsize=10)
  plt.ylabel(r"$x_2$", fontsize=10)
  plt.show()

  def semi_supervised(percentage: int):
    xL, yL, xU, yU = create_semi_supervised(X, y, percentage / 100.0)
    plt.plot(xU[:, 0], xU[:, 1], "ko", alpha=0.1)
    plt.plot(xL[:, 0][yL==1], xL[:, 1][yL==1], "bs")
    plt.plot(xL[:, 0][yL==0], xL[:, 1][yL==0], "g^")
    plt.xlabel(r"$x_1$", fontsize=10)
    plt.ylabel(r"$x_2$", fontsize=10)
    plt.show()

  for p in (50, 10, 1, 0):
    plt.cla()
    semi_supervised(p)

def plot_boundary(X, y, estimator):
    x_min, x_max = X[:, 0].min() - 0.1, X[:, 0].max() + 0.1
    y_min, y_max = X[:, 1].min() - 0.1, X[:, 1].max() + 0.1
    xx, yy = np.meshgrid(np.linspace(x_min, x_max, 100), np.linspace(y_min, y_max,
100))
    x_in = np.c_[xx.ravel(), yy.ravel()]
    y_pred = estimator.predict(x_in)
    y_pred = np.round(y_pred).reshape(xx.shape)
    plt.contour(xx, yy, y_pred, alpha=0.3)
    plt.scatter(X[:, 0], X[:, 1], c=y)
    plt.xlim(xx.min(), xx.max())
    plt.ylim(yy.min(), yy.max())
    plt.show()

##############################################################################
############
# Quasi-Newton-S3VM
#
```

```
# MIT License
#
# Permission is hereby granted, free of charge, to any person obtaining a copy
# of this software and associated documentation files (the "Software"), to deal
# in the Software without restriction, including without limitation the rights
# to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
# copies of the Software, and to permit persons to whom the Software is
# furnished to do so, subject to the following conditions:
#
# The above copyright notice and this permission notice shall be included in all
# copies or substantial portions of the Software.
##########################################################################
############

import copy as cp
import numpy as np
from scipy import optimize
from scipy import sparse
import time

class linear_k:

    def __init__(self, issparse):
        self._sparse = issparse

    def compute(self, data1, data2):
        if self._sparse:
            return data1 * data2.T
        else:
            return np.mat(data1) * np.mat(data2).T

class rbf_k:

    def __init__(self, sigma, issparse):
        self._sparse = issparse
        self._sigma = sigma

    def compute(self, mat1, mat2):
        mat1 = np.mat(mat1)
```

```
        mat2 = np.mat(mat2)
        mat1T_mat1 = np.mat([(v * v.T)[0, 0] for v in mat1]).T
        mat2T_mat2 = np.mat([(v * v.T)[0, 0] for v in mat2]).T
        mat1T_mat1 = mat1T_mat1 * np.mat(np.ones((mat2.shape[0], 1),
dtype=np.float64)).T
        mat2T_mat2 = np.mat(np.ones((mat1.shape[0], 1), dtype=np.float64)) *
mat2T_mat2.T
        k = mat1T_mat1 + mat2T_mat2
        k -= 2 * mat1 * mat2.T
        k *= - 1. / (2 * np.power(self._sigma, 2))
        return np.exp(k)

class Quasi_Newton_S3VM:

    def __init__(self, X_l, y, X_u, class_ratio=-1., lam=1., lam_u=1.,
                 sigma=1., kernel="linear", s=3., gamma=20.):
        self._start = time.time()
        self._X_l = X_l
        self._X_u = X_u
        self._y_T = np.mat(y, dtype=np.float64).T
        self._size_labeled = X_l.shape[0]
        self._size_unlabeled = X_u.shape[0]
        self._size_total = self._size_labeled + self._size_unlabeled
        self._lam = lam
        self._lam_u = lam_u
        self._sigma = sigma
        self._kernel = kernel
        self._gamma = gamma
        self._s = s
        if class_ratio == -1.:
            self._b = (1. / y.shape[0]) * np.sum(y)
        else:
            self._b = 2 * class_ratio - 1
        self._numerically_stable_threshold = 500
        if sparse.issparse(self._X_l) and sparse.issparse(self._X_u):
            self._sparse = True
            self._X = sparse.vstack((X_l, X_u))
            self._X_u_mean = self._X_u.mean(axis=0)
            self._X_u_T = X_u.T
```

```
            self._X_l_T = X_l.T
            self._X_T = self._X.T
        else:
            self._sparse = False
            self._X = np.vstack((X_l, X_u))
            if self._kernel == "linear":
                self._kernel = linear_k(self._sparse)
            elif self._kernel == "rbf":
                self._kernel = rbf_k(self._sigma, self._sparse)
            self._K_l = self._kernel.compute(self._X_l, self._X)
            self._K_u = self._kernel.compute(self._X_u, self._X)
            if self._sparse:
                self._K_m_tmp = sparse.bmat([[self._K_l], [self._K_u]])
            else:
                self._K_m_tmp = np.bmat([[self._K_l], [self._K_u]])
            self._K_m = self._K_m_tmp
            self._center_kernel()

    def _center_kernel(self):
        self._K_X_X_u = self._kernel.compute(self._X, self._X_u)
        self._K_X_X_u_or_mean = (1. / self._size_unlabeled) *
self._K_X_X_u.sum(axis=1)
        self._K_X_u_X = self._kernel.compute(self._X_u, self._X)
        self._K_X_u_X_ve_mean = (1. / self._size_unlabeled) *
self._K_X_u_X.sum(axis=0)
        self._K_X_u_X_u = self._kernel.compute(self._X_u, self._X_u)
        self._K_X_u_X_u_mean = (1. / self._size_unlabeled ** 2) *
self._K_X_u_X_u.sum()
        self._K_m_tmp = self._K_m_tmp - self._K_X_X_u_or_mean - \
                    self._K_X_u_X_ve_mean + self._K_X_u_X_u_mean
        self._K_m = self._K_m_tmp
        self._K_l = self._K_m_tmp[range(0, self._size_labeled), :]
        self._K_u = self._K_m_tmp[range(self._size_labeled, self._size_total), :]

    def fit(self):
        self._annealing()

    def get_predictions(self, X, decision_function=False):
        if self._sparse:
```

```
            W = self._X.T * self._c - self._X_u_mean.T * np.sum(self._c)
            predictions = (X * W + self._b).T
        else:
            K_X_t_X = self._kernel.compute(X, self._X)
            K_X_t_X_u = self._kernel.compute(X, self._X_u)
            K_X_t_X_u_or_mean = (1.0 / self._size_unlabeled) *
K_X_t_X_u.sum(axis=1)
            K_X_t_X = K_X_t_X - K_X_t_X_u_or_mean - self._K_X_u_X_ve_mean +
self._K_X_u_X_u_mean
            predictions = (K_X_t_X * self._c + self._b).T
        if decision_function is True:
            min_value = np.min(predictions)
            max_value = np.max(predictions)
            return ((predictions - min_value )/ (max_value - min_value)).tolist()[0]
        else:
            return np.asarray(np.sign(predictions).tolist()[0])

    def predict(self, X, decision_function=False):
        return self.get_predictions(X, decision_function)

    def get_train_time(self):
        return self._train_time

    def _annealing(self):
        c_current = np.zeros(self._size_total, dtype=np.float64)
        for i in [float(self._lam_u * i) for i in [.0, 0.000001, 0.0001, 0.01, 0.1, 0.5, 1.]]:
            self._lam_u = i
            c_current = self._bfgs(c_current)
        self._c = np.mat(c_current).T
        self._train_time = time.time() - self._start

    def _bfgs(self, c):
        if self._sparse:
            return optimize.fmin_l_bfgs_b(self._objective_function_sparse, c, m=50,
                                          fprime=self._objective_function_gradient_sparse,
maxfun=500,
                                          factr=488288000, pgtol=1.0000000000000001e-05,
iprint=-1)[0]
        else:
```

```
        return optimize.fmin_l_bfgs_b(self._objective_function, c, m=50,
                                      fprime=self._objective_function_gradient, maxfun=500,
                                      factr=488288000, pgtol=1.0000000000000001e-05,
iprint=-1)[0]

    def _objective_function(self, c):
        c = np.mat(c).T
        labeled_loss_tmp = self._gamma * (1. - np.multiply(self._y_T, self._K_l * c +
self._b))
        labeled_loss_stable = cp.deepcopy(labeled_loss_tmp)
        mask = labeled_loss_tmp > 1. / self._numerically_stable_threshold
        labeled_loss_stable[mask] = 0
        labeled_loss = np.log(1. + np.exp(labeled_loss_stable))
        np.place(labeled_loss, mask, np.array(labeled_loss_tmp[mask])[0])
        labeled_loss = (1. / (self._gamma * self._size_labeled)) * np.sum(labeled_loss)
        unlabeled_loss = self._K_u * c + self._b
        unlabeled_loss = np.multiply(unlabeled_loss, unlabeled_loss)
        unlabeled_loss = (self._lam_u / self._size_unlabeled) * np.sum(np.exp(-self._s *
unlabeled_loss))
        margin = self._lam * (c.T * self._K_m * c)
        return labeled_loss + unlabeled_loss + margin

    def _objective_function_gradient(self, c):
        c = np.mat(c).T
        a_labeled_tmp = self._gamma * (1. - np.multiply(self._y_T, self._K_l * c +
self._b))
        a_labeled_stable = cp.deepcopy(a_labeled_tmp)
        mask = a_labeled_tmp > 1. / self._numerically_stable_threshold
        a_labeled_stable[mask] = 0
        a_labeled = np.exp(a_labeled_stable)
        a_labeled = np.multiply(a_labeled, 1. / (1. + a_labeled))
        a_labeled[mask] = 1
        a_labeled = (-1. / self._size_labeled) * np.multiply(self._y_T, a_labeled)
        k_a_labeled = a_labeled.T * self._K_l
        a_unlabeled_tmp = (self._K_u * c + self._b)
        a_unlabeled = np.multiply(a_unlabeled_tmp, a_unlabeled_tmp)
        a_unlabeled = np.exp(-self._s * a_unlabeled)
        a_unlabeled = (-2. * self._s * self._lam_u / self._size_unlabeled) \
                * np.multiply(a_unlabeled, a_unlabeled_tmp)
```

```
        k_a_unlabeled = a_unlabeled.T * self._K_u
        margin = (2. * self._lam * (self._K_m * c)).T
        return (k_a_labeled + k_a_unlabeled + margin).T

    def _objective_function_sparse(self, c):
        c = np.mat(c).T
        c_sum = np.sum(c)
        X_t_c = self._X_T * c - self._X_u_mean.T * c_sum
        labeled_loss_tmp = self._gamma * (1.0 - np.multiply(self._y_T,
                                           (self._X_l * X_t_c - self._X_u_mean * X_t_c)
+ self._b))
        labeled_loss_stable = cp.deepcopy(labeled_loss_tmp)
        mask = labeled_loss_tmp > 1. / self._numerically_stable_threshold
        labeled_loss_stable[mask] = 0
        labeled_loss = np.log(1. + np.exp(labeled_loss_stable))
        np.place(labeled_loss, mask, np.array(labeled_loss_tmp[mask])[0])
        labeled_loss = (1. / (self._gamma * self._size_labeled)) * np.sum(labeled_loss)
        unlabeled_loss = (self._X_u * X_t_c - self._X_u_mean * X_t_c) + self._b
        unlabeled_loss = np.multiply(unlabeled_loss, unlabeled_loss)
        unlabeled_loss = (self._lam_u / self._size_unlabeled) * np.sum(np.exp(-self._s *
unlabeled_loss))
        margin = self._lam * c.T * (self._X * X_t_c - self._X_u_mean * X_t_c)
        return labeled_loss + unlabeled_loss + margin

    def _objective_function_gradient_sparse(self, c):
        c = np.mat(c).T
        c_sum = np.sum(c)
        XTc = self._X_T * c - self._X_u_mean.T * c_sum
        a_labeled_tmp = self._gamma * (1.0 - np.multiply(self._y_T,
                                          (self._X_l * XTc - self._X_u_mean * XTc) +
self._b))
        a_labeled_stable = cp.deepcopy(a_labeled_tmp)
        mask = a_labeled_tmp > 1. / self._numerically_stable_threshold
        a_labeled_stable[mask] = 0
        a_labeled = np.exp(a_labeled_stable)
        a_labeled = np.multiply(a_labeled, 1. / (1. + a_labeled))
        a_labeled[mask] = 1
        a_labeled = np.multiply(self._y_T, a_labeled)
        a_labeled = self._X_l_T * a_labeled - self._X_u_mean.T * np.sum(a_labeled)
```

```
        K_a_labeled = (-1. / self._size_labeled) * (self._X * a_labeled - self._X_u_mean
* a_labeled)
        a_unlabeled_tmp = (self._X_u * XTc - self._X_u_mean * XTc) + self._b
        a_unlabeled = np.multiply(a_unlabeled_tmp, a_unlabeled_tmp)
        a_unlabeled = np.exp(-self._s * a_unlabeled)
        a_unlabeled = np.multiply(a_unlabeled, a_unlabeled_tmp)
        a_unlabeled = self._X_u_T * a_unlabeled - self._X_u_mean.T *
np.sum(a_unlabeled)
        K_a_unlabeled = ((-2. * self._s * self._lam_u) / self._size_unlabeled) * \
                    (self._X * a_unlabeled - self._X_u_mean * a_unlabeled)
        margin = 2. * self._lam * (self._X * XTc - self._X_u_mean * XTc)
        return (K_a_labeled + K_a_unlabeled + margin).T

import warnings
warnings.filterwarnings("ignore", category=PendingDeprecationWarning)

def qn_s3vm_predict():
  qn_s3vm = Quasi_Newton_S3VM(X_l=xL, y=yL, X_u=xU, lam=0.00001,
lam_u=10,
                              kernel="linear", sigma=0.5)
  qn_s3vm.fit()
  yP_qns3vm = qn_s3vm.get_predictions(xU)
  return yP_qns3vm

def qn_s3vm_predict_multiclass(classes):
  probs = []
  for classIndex in range(classes):
    multi_yL = np.where(yL == classIndex, 1, -1)
    qn_s3vm = Quasi_Newton_S3VM(X_l=xL, y=multi_yL, X_u=xU, lam=0.00001,
lam_u=10,
                              kernel="linear", sigma=0.8)
    qn_s3vm.fit()
    probs.append(qn_s3vm.get_predictions(xU, True))

  result = np.copy(probs[0])
  for classIndex in range(1, classes):
    result = np.maximum(result, probs[classIndex])
  for classIndex in range(classes):
    result[result == probs[classIndex]] = classIndex
```

```
  return result

##########
# Training models on datasets
##########

percentage = 10
X, y = make_moons(1000, distance = 0.05, noise = 0.1)
# X, y = make_circles(1000, noise=0.07, factor=0.8)
xL, xU, yL, yU = train_test_split(X, y, test_size=(1 - percentage / 100.0),
random_state=7)

fig, (ax1, ax2) = plt.subplots(1, 2)
fig.set_figwidth(12)
fig.suptitle('Dataset')
ax1.plot(X[:, 0][y == 1], X[:, 1][y == 1], "b^")
ax1.plot(X[:, 0][y == -1], X[:, 1][y == -1], "g^")

ax2.plot(xU[:, 0], xU[:, 1], "ko", alpha=0.08)
ax2.plot(xL[:, 0][yL == 1], xL[:, 1][yL == 1], "b^")
ax2.plot(xL[:, 0][yL == -1], xL[:, 1][yL == -1], "g^")

classifier = svm.SVC()
classifier.fit(xL, yL)
yP = classifier.predict(xU)

yP_qns3vm = qn_s3vm_predict()

print(f"SVM Accuracy score: {accuracy_score(yU, yP)}\n")
print(f"qnS3VM Accuracy score: {accuracy_score(yU, yP_qns3vm)}\n")

fig, (ax1, ax2) = plt.subplots(1, 2)
fig.set_figwidth(12)
fig.suptitle('Results')

ax1.plot(xU[:, 0][yP == yU], xU[:, 1][yP == yU], "yo", alpha=0.5)
ax1.plot(xU[:, 0][yP != yU], xU[:, 1][yP != yU], "ro", alpha=0.5)
ax1.plot(xL[:, 0][yL == 1], xL[:, 1][yL == 1], "b^")
ax1.plot(xL[:, 0][yL == -1], xL[:, 1][yL == -1], "g^")
```

```
ax1.title.set_text('SVM')

ax2.plot(xU[:, 0][yP_qns3vm == yU], xU[:, 1][yP_qns3vm == yU], "yo", alpha=0.5)
ax2.plot(xU[:, 0][yP_qns3vm != yU], xU[:, 1][yP_qns3vm != yU], "ro", alpha=0.5)
ax2.plot(xL[:, 0][yL == 1], xL[:, 1][yL == 1], "b^")
ax2.plot(xL[:, 0][yL == -1], xL[:, 1][yL == -1], "g^")
ax2.title.set_text('qnS3VM')

acc_svm = []
acc_s3vm = []
percentages = [60, 50, 40, 30, 25, 20, 15, 10, 5, 3]
for percentage in percentages:

    xL, xU, yL, yU = train_test_split(X, y, test_size=(1 - percentage / 100.0),
random_state=7)

    classifier = svm.SVC()
    classifier.fit(xL, yL)
    yP = classifier.predict(xU)
    yP_qns3vm = qn_s3vm_predict()
    acc_svm.append(accuracy_score(yU, yP))
    acc_s3vm.append(accuracy_score(yU, yP_qns3vm))

plt.plot(percentages, acc_svm, 'o-', label='SVM')
plt.plot(percentages, acc_s3vm, 'x-', label='qnS3VM')
plt.grid(color='0.9')
plt.legend()
plt.xlabel('Labeled data percentage')
plt.ylabel('Accuracy score')
print(acc_svm)
print(acc_s3vm)
plt.show()

###
# Text classification tasks
###

nltk.download('stopwords')
```

```
df = pd.read_csv('./spam_text_messages.csv')

y = df['Category'].values
y = np.where(y == 'ham', -1, 1)
pd.set_option('display.max_colwidth', 65)
print(np.where(y==-1, 1, 0).sum())
print(np.where(y==1, 1, 0).sum())
print(df.head(15))

def text_process(message):
    no_punctuation = [char for char in message if char not in string.punctuation]
    no_punctuation = ''.join(no_punctuation)
    return [word for word in no_punctuation.split() if word.lower() not in
stopwords.words('english')]

bow_transformer = CountVectorizer(analyzer=text_process).fit(df['Message'])
messages_bow = bow_transformer.transform(df['Message'])
print('Shape of Sparse Matrix: ', messages_bow.shape)
print('Amount of Non-Zero occurences: ', messages_bow.nnz)

tfidf_transformer = TfidfTransformer().fit(messages_bow)
tfidf_messages = tfidf_transformer.transform(messages_bow)
X = tfidf_messages
plt.hist(y, bins=2)

metrics_svm = []
metrics_s3vm = []
percentages = [80, 70, 60, 50, 40, 30, 20, 10, 5]
for percentage in percentages:

    xL, xU, yL, yU = train_test_split(X, y, test_size=(1 - percentage / 100.0),
random_state=15)

    classifier = svm.SVC(kernel="linear")
    classifier.fit(xL, yL)
    yP = classifier.predict(xU)
    yP_qns3vm = qn_s3vm_predict()
    metrics_svm.append([percentage, f1_score(yU, yP)])
    metrics_s3vm.append([percentage, f1_score(yU, yP_qns3vm)])
```

```
for values in metrics_svm:
  print(values)
print('########')
for values in metrics_s3vm:
  print(values)

df = pd.read_csv('./students_complaints.csv')
df.drop(['Age','Gpa','Year','Count','Gender'], axis=1, inplace=True)

y = df['Genre'].values
pd.set_option('display.max_colwidth', 38)
print(df.sample(20))

def text_process(message):
    no_punctuation = [char for char in message if char not in string.punctuation]
    no_punctuation = ''.join(no_punctuation)
    return [word for word in no_punctuation.split() if word.lower() not in
stopwords.words('english')]

bow_transformer = CountVectorizer(analyzer=text_process).fit(df['Reports'])
messages_bow = bow_transformer.transform(df['Reports'])
print('Shape of Sparse Matrix: ', messages_bow.shape)
print('Amount of Non-Zero occurences: ', messages_bow.nnz)

tfidf_transformer = TfidfTransformer().fit(messages_bow)
tfidf_messages = tfidf_transformer.transform(messages_bow)
X = tfidf_messages

plt.xticks(rotation='vertical')
plt.hist(y, bins=11)

label_mapper = preprocessing.LabelEncoder()
print(y)
y = label_mapper.fit_transform(y)
print(y)

classes = df['Genre'].nunique()
acc_svm = []
```

```
acc_s3vm = []

percentages = [ 70, 60, 50, 40, 30, 25, 20, 15, 10, 7]
for percentage in percentages:

    xL, xU, yL, yU = train_test_split(X, y, test_size=(1 - percentage / 100.0),
random_state=33)

    classifier = svm.SVC(kernel='linear', decision_function_shape='ovo')
    classifier.fit(xL, yL)
    yP = classifier.predict(xU)
    yP_qns3vm = qn_s3vm_predict_multiclass(classes)
    acc_svm.append(accuracy_score(yU, yP))
    acc_s3vm.append(accuracy_score(yU, yP_qns3vm))

plt.plot(percentages, acc_svm, 'o-', label='SVM')
plt.plot(percentages, acc_s3vm, 'x-', label='qnS3VM')
plt.grid(color='0.9')
plt.legend()
plt.xlabel('Labeled data percentage')
plt.ylabel('Accuracy score')
print(acc_svm)
print(acc_s3vm)
plt.show()
```