Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя
Факультет комп’ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)
Кафедра комп'ютерних систем та мереж
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

магістр
(назва освітнього ступеня)

на тему: Розподілена комп’ютерна система для прогнозування поширення рослинного покриву з використанням засобів машинного навчання

Виконав(ла): студент(ка) 5 курсу, групи СІд-1
спеціальності 123 “Компютерна інженерія”

(підпис) (прізвище та ініціали)

(підпис) (прізвище та ініціали)

(підпис) (прізвище та ініціали)

(підпис) (прізвище та ініціали)

(підпис) (прізвище та ініціали)

м. Тернопіль – 2021

Міністерство освіти і науки України

Тернопільський національний технічний університет імені Івана Пулюя

(повне найменування вищого навчального закладу)

Факультет комп'ютерно-інформаційних систем і програмної інженерії

Кафедра комп'ютерних систем та мереж

ЗАТВЕРДЖУЮ

Завідувач кафедри *Осухівська Г.М.*

«_____» _____ 2021 р.

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня *магістр*

(назва освітнього ступеня)

за спеціальністю *123 «Комп'ютерна інженерія»*

(шифр і назва спеціальності)

студенту *Соболь Віталій Миколайович*

(прізвище, ім'я, по-батькові)

1. Тема проекту (роботи) *Методи і засоби тестування програмного забезпечення комп'ютерних систем з використанням алгоритмів машинного навчання*

Керівник проекту (роботи) *Луцків Андрій Мирославович, к.т.н., доц.*

(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом ректора від «__» _____ 20__ року № _____

2. Термін подання студентом завершеної роботи _____

3. Вихідні дані до роботи Розподілена комп'ютерна система для прогнозування поширення рослинного покриву з використанням засобів машинного навчання

4. Зміст роботи (перелік питань, які потрібно розробити)АНАЛІЗ ОСОБЛИВОСТЕЙ ПРОЦЕСУ ОБРОБКИ ВЕЛИКИХ ДАНИХ ТА КЛАСИФІКАЦІЯ ОСНОВНИХ АЛГОРИТМІВ

Вступ. 1. Аналіз особливостей процесу обробки великих даних та класифікація основних алгоритмів 2. Опис та вибір методів машинного навчання при обробці великих даних. 3. Вибір та опис методів машинного навчання для обробки великих даних. 4. Охорона праці та безпека в надзвичайних ситуаціях. Висновки

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, слайдів)

1. Актуальність і мета дослідження. 2. Задачі дослідження, , об'єкт і предмет, наукова новизна і практична цінність дослідження. 3. Основна модель даних у розв'язанні поставленої задачі. 4. Візуалізація роботи конвеєра Spark ML. 5. Основні залежності та бази даних, котрі працюють із Spark Framework. 6. Resilient Destributed Dataset або RDD, як інструмент роботи із не структурованими даними. 7. Хешування даних, як основна перевага вибору Apache Spark. 8. Висновки

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Охорона праці та	*Осухівська Г.М.*		
безпека в надзвичайних	*Стручок В.С.*		
ситуаціях			

7. Дата видачі завдання _____

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів курсового проекту	Термін виконання етапів проекту	Примітка
1	*Аналіз особливостей процесу обробки великих даних та класифікація основних алгоритмів*	*28.10.2021 - 12.11.2021*	*виконано*
2	*Опис та вибір методів машинного навчання при обробці великих даних*	*13.11.2021 - 20.11.2021*	*виконано*
3	*Вибір та опис методів машинного навчання для обробки великих даних*	*21.11.2021 – 28.11.2021 –*	*виконано*
4	*Охорона праці та безпека в надзвичайних ситуаціях*	*28.11.2021 - 02.12.2021*	*виконано*
5	*Оформлення пояснювальної записки*	*03.12.2021- 06.12.2021*	*виконано*
6	*Оформлення графічного матеріалу*	*07.12.2021- 11.12.2021*	*виконано*
7	*Попередній захист кваліфікаційної роботи магістра*	*15.12.2021*	*виконано*
8	*Захист кваліфікаційної роботи магістра*		

Студент _____ *Соболь В. М.*
(підпис) (прізвище та ініціали)

Керівник проекту (роботи) _____ *Луцків А.М.*
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Тема кваліфікаційної роботи: “Розподілена комп’ютерна система для прогнозування поширення рослинного покриву з використанням засобів машинного навчання”/ Кваліфікаційна робота// Соболь Віталій Миколайович // Тернопільський національний технічний університет імені Івана Пулюя, факультет комп’ютерно- інформаційних систем та програмної інженерії, група СІд-1 // Тернопіль, 2021 // с. – 99, рис. – 9, лістингів - 14, аркушів А1 – 8, додат. – 2, бібліогр. – 27.

Ключові слова: метод, засіб, машинне навчання, прогнозування, великі дані, Apache Spark, програмне забезпечення, комп’ютерна система, алгоритм, машинне навчання.

Метою роботи є розробка програмного забезпечення та імплементація алгоритмів машинного навчання для прогнозування лісового покриття певної території, зважаючи на різноманітність та унікальність навколишнього середовища, та початкових насаджень на певній території.

У дослідженні проведено аналіз важливих понять, принципів і послідовності виконання процесів, що використовуються при проектуванні комп’ютерних систем та написання програм, та роботі з великими даними, зокрема, термінологічні особливості у процесі імплементації програмного забезпечення на прогнозування, що дало змогу зрозуміти і в подальшому визначити шляхи імплементації методів машинного навчання для підвищення ефективності зелених насаджень на певній території.

Для розв’язання задач дипломного проекту було використано наступні методи: критичний аналіз – при тестуванні програмного забезпечення КС та дослідженні основних методів; машинне навчання – при прогнозуванні дефектів та побудові алгоритмів для класифікації вимог; програмування – при кодуванні алгоритмів машинного навчання; апробація – при отриманні та перевірці результатів алгоритмів машинного навчання.

ABSTRACT

Theme of qualification work: "Distributed computer system for predicting the spread of vegetation using machine learning tools" / Qualification work // Sobol Vitaliy Mykolayovych // Ternopil Ivan Pul'uj National Technical University, Faculty of Computer Information Systems and software engineering, group CId-1 // Ternopil, 2021 // p. - 99, fig. - 9, listing - 14,sheets A1 - 8, appendix. - 2, bibliogr. - 27.

Keywords: method, tool, machine learning, forecasting, big data, Apache Spark, software, computer system, algorithm, machine learning.

The aim of the work is to develop software and implement machine learning algorithms for forecasting the forest cover of a certain area, taking into account the diversity and uniqueness of the environment and the original plantings in a certain area.

The study analyzes important concepts, principles and sequences of processes used in the design of computer systems and program writing, and work with big data, in particular, terminological features in the process of implementing software for forecasting, which allowed to understand further identify ways to implement machine learning methods to improve the efficiency of greenery in a given area. The following methods were used to solve the problems of the diploma project: critical analysis - when testing the software of the COP and researching the basic methods; machine learning - when predicting defects and building algorithms for the classification of requirements; programming - when coding machine learning algorithms; approbation - when obtaining and verifying the results of machine learning algorithms.

ЗМІСТ

ПЕРЕЛІК ОСНОВНИХ УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ І СКОРОЧЕНЬ

БД	База Даних
КС	Комп’ютерна Система
ПЗ	Програмне забезпечення
HPC	Високопродуктивні обчислення
BD	Big data
REPL	Read-eval-print loop — цикл читання- обчислення-друку, мовна оболонка

ВСТУП

Актуальність теми. Проектування сучасних КС, автоматизація процесів, розумні пристрої, наука про дані (Data Science), стає все більш популярною в сучасному світі та дають змогу реалізовувати складні, та водночас ефективні для кінцевого користувача продукти.

Різні університети і курси по Data Science готовлять високоякісних професіоналів. Все завдяки тій кількості інформації, яку компанії отримують майже безупинно: потоки інформації про поведінку користувачів, їхні переваги, дані про покупки та багато іншого. Таку кількість інформації важко проаналізувати з допомогою традиційних методів, та її об’єднують під назвою великі дані, або big data.

Існують два типи даних – традиційні та великі дані. Традиційні дані зберігаються в базах даних, та зберігаються у структурованих таблицях з цифровою, текстовою та іншою інформацією, а великі дані перевищують в кількості традиційні дані, тому такий тип даних розподіляється між комп'ютерами.

Одним із найважливіших завдань обробки великих даних є отримання структурованих даних із структурованих, неструктурованих або слабоструктурованих великих даних.

В свою чергу структурованим типом даних, є тип даних, при якому всі дані є структорованими, що свідчить про те, що вони зберігаються, витягуються, або можуть бути використані в певному форматі.

Об’єми даних увесь час збільшуються, ми отримуємо дані із різних джерел – з площадок електронної торгівлі (наприклад Amazon, eBay), соціальних мереж (Facebook, Instagram, Pinterest, курсів і т. д.). Таким чином у нас в розпорядженні терабайти, петабайти і ексабайти даних.

Очистка даних також використовується для організації обробки інформації. Тобто очистка даних від невідповідностей – пропущених значень або помилки в тексті, що ми спробуємо зробити з нашою вхідною інформацією.

Великі дані використовують в абсолютно різних індустріях – ІТ, охороні здоров’я, фінансах, освіті, електронній комерції, виробництві, сфері розваг і т.д. Великі дані будуть збільшуватися, саме тому аналіз великих даних, як ніколи є актуальним та спеціалісти, які працюють з великими даними є затребувані по всьому світі.

На початок двадцять першого століття вже було зрозуміло, що вирубка дерев, та насаджень у такій кількості має катастрофічні наслідки для екології всієї планети. Якщо не прийняти заходів прямо зараз, може бути надто пізно, та виживання нашого виду, та загалом життя на планеті, буде в катастрофічний загрозі, геть до повного вимирання.

Оскільки ліси є основним постачальником кисню - про них потрібно подбати в першу чергу. Тому тема розповсюдження лісових насаджень, та відновлення лісового покриву є як ніколи актуальною. Значний внесок у збереження та відновлення планети зроблено завдяки розвитку комп’ютерних технологій, зокрема наукою про дані.

Розкриття даної теми у нашій кваліфікаційній роботі є новим кроком в об’єднанні знань про довкілля, з комп’ютерними технологіями в комп’ютерній програмі, яка забезпечить прогнозування поширення лісового покриву на певній території, зважаючи на вхідні дані.

Мета і завдання досліджень. Метою роботи є розробка програмного забезпечення та імплементація алгоритмів машинного навчання для прогнозування лісового покриття певної території, зважаючи на різноманітність та унікальність навколишнього середовища, та початкових насаджень на певній території.

Для досягнення мети поставлені такі **задачі**:

- проведено аналітичний огляд науково-літературних джерел за тематикою досліджуваного об'єкту, надано порівняльну характеристику існуючих вибірок даних, та існуючих систем, вказано їх переваги та недоліки;
- аналіз алгоритмів машинного навчання для впровадження у процес тестування ПЗ;

- аналіз та обробка вхідних даних;
- аналіз та обґрунтування вибору інструментів опрацювання великих даних;
- розроблено програму для обробки великих даних, для отримання прогнозу розповсюдження дерев певного сорту на вказаній території.

Об’єктом дослідження: розробка програмного забезпечення для аналізу великих даних.

Предмет дослідження: алгоритми, методи та моделі машинного навчання, дерево рішень та обробка даних на основі розробленого ПЗ.

Методи дослідження: Для розв’язання задач дипломного проекту використано наступні методи: критичний аналіз – при тестуванні програмного забезпечення КС та дослідженні основних методів; машинне навчання – при прогнозуванні дефектів та побудові алгоритмів для класифікації вимог; програмування – при кодуванні алгоритмів машинного навчання; апробація – при отриманні та перевірці результатів алгоритмів машинного навчання.

Наукова новизна одержаних результатів: Наукова новизна даного дослідження полягає у наступному:

– запропоновано метод класифікації вимог до набору даних, який базується на методах машинного навчання і використовується для отримання прогнозу поширення рослинного покриву на заданій території.

– уперше було порівняно різні підходи для виконання поставленої задачі, та вибір найбільш сттабільної, та найменш вимогливою до переналаштувань при зміні вхідних даних.

Практичне значення одержаних результатів роботи. Результати роботи вносять науковий вклад в збереження та розсаджування дерев на певній території. Розробки даної програми прогнозування поширення рослинного покриву з використанням засобів машинного навчання можуть бути використані у наукових дослідженнях подібної тематики.

Результати даної роботи дозволять забезпечити підвищення ефективності поширення лісового покриву та прогнозування дерев’яних насаджень на

досліджуваній території.

Публікації. Результати кваліфікаційної роботи апробовані на IX міжнародній науково - технічній конференції «Інформаційні моделі, системи та технології» (8 - 9 грудня 2021 р.) Тернопільського національного технічного університету імені Івана Пулюя як тези конференцій.

1. Луцків А.М., Г.А. Абоах, Р.К. Рувімбо, В.М. Соболь. Побудовазахищених хмарних середовищ опрацювання даних. Матеріали IX науково-технічної конференції Тернопільського національного технічного університету імені Івана Пулюя «Інформаційні моделі, системи та технології» (8-9 грудня 2021 року). Тернопіль: ТНТУ. 2021. С. 103.

2. Луцків А.М., Г.А. Абоах, Р.К. Рувімбо, В.М. Соболь. Розв’язання задач машинного навчання у середовищах із розподіленою пам’яттю. Матеріали IX науково-технічної конференції Тернопільського національного технічного університету імені Івана Пулюя «Інформаційні моделі, системи та технології» (8-9 грудня 2021 року). Тернопіль: ТНТУ. 2021. С. 130.

Структура роботи. Кваліфікаційна робота містить розрахунково-пояснювальну записку та графічний матеріал. До складу записки входить - перелік основних умовних позначень, вступу, 4 розділи, загальні висновки і список використаних джерел. Обсяг роботи: розрахунково-пояснювальна записка – 99 арк. формату А4, графічна частина – 8, лістингів 14, аркушів формату А1.

РОЗДІЛ 1
АНАЛІЗ ОСОБЛИВОСТЕЙ ПРОЦЕСУ ОБРОБКИ ВЕЛИКИХ ДАНИХ ТА КЛАСИФІКАЦІЯ ОСНОВНИХ АЛГОРИТМІВ

1.1. Аналіз та основні виклики науки про дані

Існують завдання, які ми не могли виконані 5 – 10 років тому. Наприклад:

- Створення моделі, що виявляє шахрайство з кредитними картками, використовуючи тисячі функцій та мільярди транзакцій;
- Розумно рекомендувати користувачам продукти;
- Оцінювати фінансові ризики за допомогою моделювання портфелів, з мільйонами інструментів;
- Маніпулювати даними тисяч людських геномів, щоб виявити генетичні асоціації із захворюваннями.

Коли говорять, що ми живемо в епоху великих даних, це означає, що у нас є інструменти для збору, зберігання та обробки інформації у нечуваному раніше масштабі. За цими можливостями стоїть екосистема програмного забезпечення з відкритим кодом, яка може керувати кластерами товарних комп'ютерів для обміну величезною кількістю даних.

Наука про дані - це практика перетворення інструментів та необроблених даних на те, що може цікавити вчених науки про великі дані. Раніше розмістити схему над даними та структурувати їх у SQL було достатньо для подальшого використання даних, та іноді нам необхідно більше. SQL все ще може бути основним у підході, але для того, щоб обійти ідіосинкразії в даних або виконати складний аналіз, нам потрібна парадигма програмування, яка буде трохи більш гнучкою і з більш багатими функціональними можливостями в таких областях, як машинне навчання та статистика. Саме такі аналізи ми розглянемо в нашій роботі.

Протягом тривалого часу фреймворки з відкритим кодом, такі як R, PyData стек та Octave, зробили швидкий аналіз та побудову моделі життєздатними для невеликих наборів даних. Маючи менше 10 рядків коду, ми можемо скласти

модель машинного навчання на половині набору даних і використати її для прогнозування міток на другій половині. Доклавши трохи більше зусиль, ми можемо заповнити відсутні дані, експериментувати з кількома моделями, щоб знайти найкращу, або використати результати моделі як вихідні дані для іншої. Та щоб створити еквівалентний процес, який може використовувати кластери комп'ютерів для досягнення однакових результатів щодо величезних наборів даних, можна просто розширити цей фреймворк для роботи на декількох машинах, щоб зберегти їх моделі програмування та переписати їхній код для правильної роботи в розподілених налаштуваннях. Однак проблеми розподілених обчислень вимагають від нас переосмислення багатьох основних припущень, на які ми спираємось в одинарних системах. Наприклад, оскільки дані повинні бути розділені на багато вузлів кластера, алгоритми, які мають широкі залежності від даних, постраждають від того факту, що швидкість передачі даних у мережі на порядок повільніша за доступ до пам'яті. Зі збільшенням кількості машин, які працюють над проблемою, зростає ймовірність невдачі. Ці факти вимагають парадигми програмування, яка є чутливою до характеристик основної системи: такої, яка перешкоджає поганому вибору та полегшує написання коду, який буде використаний в паралельному порядку.

Інструменти, написані для середовищ HPC (високопродуктивні обчислення), часто не можуть відокремити моделі даних в пам'яті від моделей зберігання нижчого рівня. Наприклад, багато інструментів знають лише те, як читати дані з файлової системи POSIX в одному потоці, що ускладнює паралелізацію інструментів або використання інших резервних копій сховищ, таких як бази даних. Останні системи в екосистемі Hadoop пропонують абстракції, які дозволяють користувачам використовувати кластер комп'ютера більш як цілий комп'ютер - автоматично розбивати файли та розподіляти пам'ять на багато машин, розподіляти роботу на менші завдання та виконувати їх у розподіленій манері і відновлювати їх у разі невдач.

Переважна більшість робіт, які використовуються для проведення успішного аналізу, полягають у попередній обробці даних. Дані безладні, а

очищення, змішування, злиття, затирання є передумовами для того, щоб з ними зробити щось корисне. Особливо великі масиви даних, оскільки вони не піддаються прямому дослідженню людьми, можуть вимагати обчислювальних методів, щоб навіть виявити, які кроки попередньої обробки необхідні. Навіть коли настає час оптимізувати продуктивність моделі, типовий набір даних вимагає витрати набагато більше часу на розробку та вибір функцій, ніж на вибір та написання алгоритмів. Наприклад, коли створюється модель для виявлення шахрайських покупок на веб-сайті, дослідник даних повинен вибрати із широкого спектру потенційних можливостей: поля, які користувачеві необхідно заповнити, інформацію про місцезнаходження IP, час входу та журнали натискань користувача на сайті. Кожен з них приходить із своєю проблемою вибору придатних алгоритмів машинного навчання. Система повинна підтримувати більш гнучкі перетворення, ніж перетворення 2D-масиву у математичну модель.

Ітерація є фундаментальною частиною науки про дані. Моделювання та аналіз зазвичай вимагають декількох проходів по тих самих даних. Один із аспектів лежить у алгоритмах машинного навчання та статистичних процедурах. Популярні процедури оптимізації, такі як стохастичний градієнтний спуск та максимізація очікувань, включають повторні сканування над їхніми вхідними даними для досягнення конвергенції. Ітерація також має значення у власному робочому процесі вченого з даних. Коли дослідники даних інстинктивно досліджує і намагаються визначити набір даних, зазвичай результати запиту повідомляють про наступний запит, який має бути виконано.

Вибір правильних функцій, вибір правильних алгоритмів, виконання правильних значущих тестів та пошук правильних гіперпараметрів - все це вимагає експериментів. Фреймворк, який вимагає зчитування одного і того ж набору даних з диска при кожному зверненні, додає затримку, яка може уповільнити процес дослідження та обмежити кількість речей, які ми можемо спробувати.

Завдання не завершено, коли добре побудована модель спроектована та впроваджена. Використання механізмів рекомендацій щодо даних та систем

виявлення шахрайства в реальному часі завершується застосуванням даних. У цьому випадку моделі стають частиною виробничого сервісу і, можливо, їх доведеться періодично або навіть у реальному часі перебудовувати.

Звичайно, кожен міг би заощадити час, якщо б оригінальний код моделювання міг бути дійсно використаний у програмі, для якої він написаний, але такі мови, як R, повільні і не мають інтеграції з більшістю рівнів стека виробничої інфраструктури, а також такі мови, як Java та C ++ - це просто поганий інструмент для дослідницької аналітики. Їм не вистачає середовищ циклу читання-оцінювання-друк (REPL) для інтерактивної гри з даними та потрібна велика кількість коду для вираження простих перетворень. Структура, яка полегшує моделювання, але також добре підходить для виробничих систем, - це велика перемога.

1.2. Порівняння Hadoop і Spark, як основних конкурентів по роботі з Великими даними

Spark — це розширення Hadoop для MapReduce. Основна відмінність між Spark і MapReduce полягає в тому, що Spark обробляє та зберігає дані в пам’яті для наступних кроків, тоді як MapReduce обробляє дані на диску. Як наслідок, для невеликих робочих навантажень швидкість обробки даних Spark до 100 разів вища, ніж у MapReduce.

Крім того, на відміну від двоетапного процесу виконання в MapReduce, Spark створює орієнтований ациклічний графік (DAG) для планування завдань і оркестрування вузлів у кластері Hadoop. Цей процес відстеження завдань забезпечує стійкість до відмов, яка повторно застосовує записані операції до даних з попереднього стану.

Давайте детальніше розглянемо ключові відмінності між Hadoop і Spark у шести критичних контекстах:

— Продуктивність: Spark працює швидше, оскільки використовує оперативну пам’ять (RAM) замість читання та запису проміжних даних на диски. Hadoop зберігає дані з кількох джерел і обробляє їх пакетами за допомогою

MapReduce.

— Вартість: Hadoop працює з нижчою ціною, оскільки для обробки даних покладається на будь-який тип дискового сховища. Spark працює дорожче, оскільки покладається на обчислення в пам’яті для обробки даних у реальному часі, що вимагає від нього використання великої кількості оперативної пам’яті для розгортання вузлів.

— Обробка: хоча обидві платформи обробляють дані в розподіленому середовищі, Hadoop ідеально підходить для пакетної обробки та лінійної обробки даних. Spark ідеально підходить для обробки та обробки неструктурованих потоків даних у реальному часі.

— Масштабування: коли обсяг даних швидко зростає, Hadoop швидко масштабується, щоб задовольнити попит через розподілену файлову систему Hadoop (HDFS). У свою чергу, Spark покладається на відмовостійку HDFS для великих обсягів даних.

— Безпека: Spark покращує безпеку за допомогою аутентифікації через спільний секрет або реєстрацію подій, тоді як Hadoop використовує кілька методів аутентифікації та контролю доступу. Хоча в цілому Hadoop є більш безпечним, Spark може інтегруватися з Hadoop, щоб досягти більш високого рівня безпеки.

— Машинне навчання (ML): Spark є найкращою платформою в цій категорії, оскільки включає MLlib, який виконує ітераційні обчислення ML в пам’яті. Він також включає інструменти, які виконують регресію, класифікацію, збереження, побудову конвеєра, оцінку тощо.

Помилкові уявлення про Hadoop і Spark та Поширені помилкові уявлення про Hadoop наступні

— Hadoop дешевий: хоча він є відкритим вихідним кодом і його легко налаштувати, підтримка сервера може бути дорогою. При використанні таких функцій, як обчислення в пам’яті та мережеве сховище, керування великими даними може коштувати до 5000 доларів США.

— Hadoop — це база даних: хоча Hadoop використовується для зберігання, керування та аналізу розподілених даних, під час отримання даних

немає запитів. Це робить Hadoop сховищем даних, а не базою даних.

— Hadoop не допомагає малому та середньому бізнесу: «Великі дані» не є винятковими для «великих компаній». Hadoop має прості функції, як-от звіти Excel, які дозволяють невеликим компаніям використовувати його потужність. Наявність одного або двох кластерів Hadoop може значно підвищити продуктивність невеликої компанії.

— Hadoop важко налаштувати: хоча керування Hadoop є складним на вищих рівнях, існує багато графічних інтерфейсів користувача (GUI), які спрощують програмування для MapReduce.

Поширені помилкові уявлення про Spark:

— Spark — це технологія в пам’яті: хоча Spark ефективно використовує алгоритм, який використовується найменше нещодавно (LRU), сам по собі він не є технологією на основі пам’яті.

— Spark завжди працює в 100 разів швидше, ніж Hadoop: хоча Spark може працювати до 100 разів швидше, ніж Hadoop для невеликих робочих навантажень, згідно з Apache, він зазвичай працює лише до 3 разів швидше для великих.

— Spark впроваджує нові технології в обробку даних: хоча Spark ефективно використовує алгоритм LRU і конвеєри обробки даних, ці можливості раніше існували в базах даних з масовою паралельною обробкою (MPP). Однак те, що відрізняє Spark від MPP, — це його орієнтація на відкритий код.

1.3. Обгрунтування вибору Apache Spark як основного фреймворка роботи

Apache Spark – це фреймворк з відкритим вихідним кодом, який поєднує механізм розповсюдження програм у кластерах машин з елегантною моделлю для написання програм поверх нього. Spark, яка виникла в AMPLab UC Berkeley і з тих пір була внесена до Фонду програмного забезпечення Apache, є першим програмним забезпеченням з відкритим кодом, яке робить розподілене програмування дійсно доступним для вчених з даних.

Одним із яскравих способів зрозуміти Spark є його прогрес у порівнянні з

його попередником - MapReduce від Apache Hadoop. MapReduce революціонізував обчислення величезних наборів даних, запропонувавши просту модель для написання програм, які можуть виконуватися паралельно на сотнях, тисячах машин. MapReduce досягає майже лінійної масштабованості - із збільшенням розміру даних ми можемо добавляти до нього більше комп’ютерів і бачити, як завдання виконуються за той самий проміжок часу і стійкий до збоїв, які трапляються напрочуд рідко на одній машині, трапляються весь час на кластерах тисяч машин. Він розбиває роботу на невеликі завдання і може витончено подолати невдачі, не завдаючи шкоди роботі загального процесу.

Spark підтримує лінійну масштабованість і відмовостійкість MapReduce, але розширює її трьома важливими способами. По-перше, замість того, щоб покладатися на жорсткий формат, його механізм може виконувати більш загальний ациклічний графік (DAG) операторів. Це означає, що в ситуаціях, коли MapReduce повинен записувати проміжні результати у розподілену файлову систему, Spark може передати їх безпосередньо на наступний крок виконання програми. Таким чином, він схожий на Dryad, нащадка MapReduce, що виникла в компанії Microsoft Research. По-друге, він доповнює цю можливість багатим набором перетворень, які дозволяють користувачам проводити обчислення більш природно. Він має сильний фокус розробників та спрощений API, який може представляти складні обчислення у кількох рядках коду.

По-третє, Spark розширює своїх попередників обробкою в пам'яті. Його абстракції Dataset і DataFrame дозволяють розробникам матеріалізувати будь яку точку в конвеєрі обробки в пам'ять по всьому кластеру, а це означає, що майбутні кроки, які хочуть мати справу з тим самим набором даних, не потребують повторного обчислення або перезавантаження з диска. Ця можливість відкриває варіанти використання, до яких раніше не могли підійти механізми розподіленої обробки. Spark добре підходить для високотехнологічних алгоритмів, які вимагають багаторазового проходження набору даних, а також для реактивних програм, які швидко відповідають на запити користувачів шляхом сканування великих наборів даних в пам'яті.

Можливо, найголовніше, що Spark добре вписується у вищезгадані жорсткі істини науки про дані, визнаючи, що найбільшим вузьким місцем у створенні програм для даних є не процесор, диск або мережа, а продуктивність аналітиків. Можливо, не можна переоцінити, наскільки згортання повного конвеєра - від попередньої обробки до оцінки моделі - до єдиного середовища програмування може прискорити розвиток. Упаковуючи ексклюзивну модель програмування з набором аналітичних бібліотек відповідно до REPL, Spark уникає повторних переходів до IDE, необхідних для таких фреймворків, як MapReduce, та проблем підвибірки та переміщення даних вперед-назад із розподіленої файлової системи Hadoop (HDFS), що вимагається такими структурами, як R. Чим швидше аналітики можуть дослідити свої дані, тим більша ймовірність зробити з ними щось корисне.

Що стосується доцільності модифікації та ETL, Spark прагне бути чимось ближчим до Python великих даних, ніж MATLAB великих даних. Як механізм обчислень загального призначення, його основні API забезпечують міцну основу для перетворення даних незалежно від будь-яких функцій статистики, машинного навчання чи матричної алгебри. Його API Scala і Python дозволяють програмувати виразними мовами загального призначення, а також мати доступ до існуючих бібліотек.

Кешування в пам'яті Spark робить його ідеальним для ітерації як на мікро, так і на макрорівні. Алгоритми машинного навчання, які здійснюють багаторазовий прохід через свій навчальний набір, можуть кешувати його в пам’яті. Коли вони вивчають набір даних, вчені можуть зберігати дані в пам’яті під час виконання запитів, а також легко кешувати перетворені версії цього файлу, не потребуючи при цьому запису даних на диск.

Нарешті, Spark охоплює розрив між системами, призначеними для дослідницької аналітики, та системами, призначеними для оперативної аналітики. Часто зазначається, що вчений з даних - це той, хто кращий в інженерії, ніж більшість статистиків і кращий у статистиці, ніж більшість інженерів. Насамкінець, Spark є кращою в якості операційної системи, ніж більшість

дослідницьких систем, і краще для дослідження даних, ніж технології, які зазвичай використовуються в операційних системах. Він створений для забезпечення продуктивності та надійності з нуля. Розташований на вершині JVM, він може скористатися багатьма операційними засобами та засобами налагодження, створеними для стека Java.

Spark може похвалитися міцною інтеграцією з різноманітними інструментами в екосистемі Hadoop. Він може читати та записувати дані у всіх форматах даних, підтримуваних MapReduce, що дозволяє йому взаємодіяти з форматами, які зазвичай використовуються для зберігання даних у Hadoop, такими як Apache Avro та Apache Parquet (і старий добрий CSV). Він може читати та записувати в бази даних NoSQL, такі як Apache HBase та Apache Cassandra. Його бібліотека обробки потоків, Spark Streaming, може постійно отримувати дані з таких систем, як Apache Flume та Apache Kafka. Його бібліотека SQL, SparkSQL, може взаємодіяти з Apache Hive Metastore, а ініціатива Hive on Spark дозволила використовувати Spark як базовий механізм виконання для Hive, як альтернативу MapReduce. Він може працювати всередині YARN, планувальника та менеджера ресурсів Hadoop, що дозволяє йому динамічно обмінюватися ресурсами кластера та керуватися тими ж процесами, що й інші механізми обробки, такі як Map Reduce та Apache Impala.

1.4. Швидкий перехід до регресії

Понад століття статистики і з моменту появи сучасного машинного навчання та науки про дані, ми все ще говоримо про ідею прогнозування значення на основі інших значень як регресію, хоча це не має нічого спільного з сповзанням назад до середнього значення. Методи регресії також відносяться до методів класифікації. Як правило, регресія відноситься до передбачення числової величини, наприклад розміру, доходу або температури, тоді як класифікація відноситься до передбачення мітки або категорії, як-от «спам» або «фото кота».

Загальним зв’язком регресії та класифікації є те, що обидва передбачають одне (або більше) значень з урахуванням одного (або кількох) інших значень. Для

цього обидва вимагають сукупності вхідних і вихідних даних, на яких можна вчитися. Їх потрібно годувати як питаннями, так і відомими відповідями. З цієї причини вони відомі як типи навчання керівництвом.

Класифікація та регресія є найстарішими та найбільш добре вивченими типами прогнозної аналітики. Більшість алгоритмів, з якими ви, ймовірно, зіткнетеся в аналітичних пакетах і бібліотеках, — це методи класифікації або регресії, як-от машини опорних векторів, логістична регресія, наївний байєс, нейронні мережі та глибоке навчання. Рекомендації, тема розділу 3, були порівняно більш інтуїтивними для введення, але вони також є відносно недавньою та окремою підтемою в машинному навчанні.

У цьому розділі ми зосередитися на популярному та гнучкому типі алгоритму як для класифікації, так і для регресії: дерева рішень і розширення алгоритму, ліси випадкових рішень. Цікаве в цих алгоритмах те, що, при всій повазі до пана Бора, вони можуть допомогти передбачити майбутнє — або, принаймні, передбачити те, чого ми ще не знаємо напевно, наприклад, ймовірність того, що ви придбаєте автомобіль на основі на вашу поведінку в Інтернеті, незалежно від того, чи є електронний лист спамом, зважаючи на його слова, ймовірно, буде вирощуватися найбільше сільськогосподарських культур, враховуючи їхнє розташування та хімічний склад ґрунту.

1.5. Вектори та особливості

Щоб пояснити вибір набору даних і алгоритму, та почати пояснювати, як працюють регресія та класифікація, необхідно коротко визначити терміни, які описують їх вхідні і вихідні дані.

Подумайте про те, щоб передбачити високу температуру на завтра з огляду на сьогоднішню погоду. У цій ідеї немає нічого поганого, але «сьогоднішня погода» — це невимушене поняття, яке вимагає структурування, перш ніж його можна буде використати в алгоритмі навчання.

Справді, певні особливості сьогоднішньої погоди можуть передбачити завтрашню температуру, наприклад:

— Сьогодні висока температура.

— Сьогодні низька температура.

— Сьогоднішня середня вологість.

— Незалежно від того, чи буде сьогодні хмарно, дощ чи ясно.

— Кількість прогнозів синоптиків про похолодання на завтра.

Ці ознаки також іноді називають вимірами, предикторами або просто змінними. Кожну з цих ознак можна кількісно оцінити. Наприклад, високі та низькі температури вимірюються в градусах Цельсія, вологість може бути виміряна як частка від 0 до 1, а тип погоди можна позначити як хмарна, дощова або ясна. Кількість прогнозистів – це, звичайно, ціле число. Тому сьогоднішню погоду можна скоротити до списку значень, як-от 13.1, 19.0, 0.73, cloudy, 1.

Ці п’ять ознак разом, відомі як вектор ознак і можуть описати погоду будь-якого дня. Це використання має певну схожість із використанням терміна вектор у лінійній алгебрі, за винятком того, що вектор у цьому сенсі може концептуально містити нечислові значення і навіть не мати деяких значень.

Не всі ці функції є однаковими. Перші дві характеристики вимірюються в градусах Цельсія, а третя безодинна (менша одиниці), частка. Четверте — це зовсім не число, а п’яте — це число, яке завжди є цілим невід’ємним числом.

Ми розглянемо ознаки лише у двох широких групах: категоріальні ознаки та числові ознаки. У цьому контексті числові ознаки — це ті, які можуть бути кількісно визначені числом і мають значущий порядок. Наприклад, має сенс сказати, що сьогоднішній максимум був 23ºC, і що це вище, ніж вчорашній максимум 22ºC. Усі вищезгадані функції є числовими, крім типу погоди. Такі терміни, як чистота неба, не є числами і не мають порядку. Безглуздо говорити, що хмарність більше, ніж ясність. Це категоріальна ознака, яка замість цього приймає одне з кількох дискретних значень.

1.6. Тренувальні приклади

Алгоритм навчання повинен тренуватися на даних, щоб робити прогнози. Це вимагає великої кількості вхідних даних та відомих правильних виходів із

історичних даних. Наприклад, у цій задачі алгоритму навчання буде зазначено, що одного дня погода була між 12º і 16ºC, з 10% вологістю, ясна, без прогнозу похолодання; а наступного дня висока температура становила 17,2ºC. Маючи достатньо таких прикладів, алгоритм навчання може навчитися прогнозувати високу температуру наступного дня з певною точністю.

Вектори ознак забезпечують організований спосіб опису вхідних даних для алгоритму навчання (тут: 12.5, 15.5, 0.10, clear, 0). Вихід, або ціль, передбачення також можна розглядати як ознаку. Ось це числова ознака: 17.2. Нерідко просто включати ціль як іншу функцію у вектор ознак. Весь тренінг приклад можна розглядати як 12.5, 15.5, 0.10, clear, 0, 17.2 . Збір усіх цих прикладів відомий як навчальний набір.

Зауважте, що проблеми регресії — це лише ті, де ціль є числовою ознакою, а проблеми класифікації — це ті, де ціль є категоричною. Не кожен алгоритм регресії або класифікації може обробляти категоріальні ознаки або категоріальні цілі; деякі обмежуються числовими ознаками.

1.7. Дерева рішень та ліси

Виявляється, що сімейство алгоритмів, відоме як дерева рішень, може природно обробляти як категоріальні, так і числові ознаки. Побудова одного дерева може бути виконана паралельно, а багато дерев можна побудувати одночасно. Вони стійкі до викидів у даних, а це означає, що кілька екстремальних і, можливо, помилкових точок даних можуть взагалі не вплинути на прогнози. Вони можуть споживати дані різних типів і в різних масштабах без необхідності попередньої обробки або нормалізації.

Дерева рішень узагальнюються в більш потужний алгоритм, який називається лісами випадкових рішень. Їхня гнучкість робить ці алгоритми доцільними для вивчення, та реалізації Spark MLlib DecisionTree і RandomForest, який застосовується до набору даних.

Алгоритми на основі дерева рішень мають додаткову перевагу в тому, що вони відносно інтуїтивно зрозумілі для розуміння та міркування. Насправді, ми всі,

ймовірно, використовуємо ті самі міркування, втілені в деревах рішень, неявно, у повсякденному житті. Я, наприклад, сідаю випити ранкової кави з молоком. Перш ніж взяти це молоко і додати його в каву, я хочу передбачити: чи молоко зіпсоване? Я можу перевірити, чи минув термін придатності. Якщо ні, я передбачаю, ні, це не зіпсовано. Якщо дата пройшла, але це було три або менше днів тому, я ризикую і прогнозую ні, це не зіпсовано. Інакше я нюхаю молоко. Якщо воно не приємно пахне, я передбачаю так, воно зіпсоване, а інакше ні.

Ця серія рішень так/ні, які ведуть до прогнозу, є те, що втілює дерева рішень. Кожне рішення призводить до одного з двох результатів, який є або прогнозом, або іншим рішенням, як показано на рис. 1.1. У цьому сенсі природно думати про процес як про дерево рішень, де кожен внутрішній вузол дерева є рішенням, а кожен листовий вузол є остаточною відповіддю.

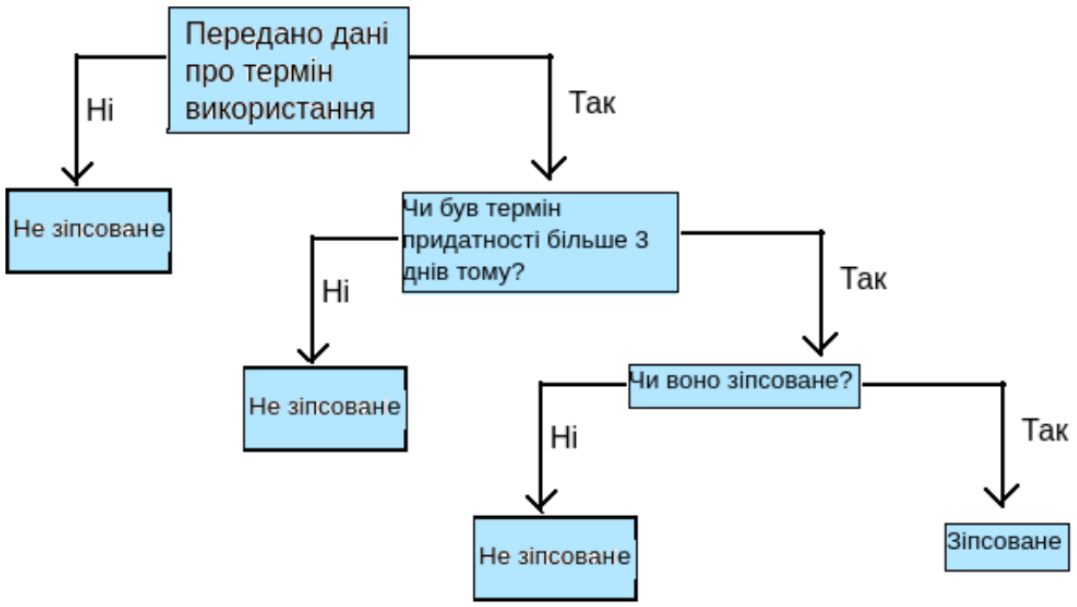

Рис. 1.1. Дерево рішень: чи воно зіпсоване?

Це спрощене дерево рішень, яке не було створено з будь-якою суворістю.

Детальніше про дерево рішень та вибір саме цієї моделі ми розглянемо у розділі 2.

1.8. Набір даних лісового покриття

Набір даних фіксує типи вкритих лісом ділянок землі в Колорадо, США. Це лише збіг обставин, що набір даних стосується реальних лісів! Кожен приклад

містить кілька ознак, що описують кожну земельну ділянку, наприклад її висоту, схил, відстань до води, тінь і тип ґрунту, а також відомий тип лісу, що покриває землю. Тип лісистості слід передбачити за рештою ознак, яких є всього 54. Детальнішу інформацію можна знайти за посиланням, у списку використаної літератури [2].

В даному наборі даних представлені 12 показників, та 54 стовпці даних (10 кількісних змінних, 4 бінарні зони дикої природи та 40 бінарних змінних типу ґрунту), в яких представлені такі показники: Висота в метрах, аспект у градусах по азимуту, нахил у градусах, горизонтальна відстань до найближчих поверхневих водних об’єктів, вертикальна відстань до найближчих поверхневих водних об’єктів, горизонтальна відстань до найближчої проїжджої частини, індекс пагорбів о 9 ранку, літнє сонцестояння, індекс відтінку пагорбів опівдні, літнє сонцестояння , Індекс затінку о 15:00, літнє сонцестояння, Горизонтальна відстань до найближчих точок займання лісових пожеж, Позначення території дикої природи, Позначення типу ґрунту, Позначення типу лісового покриву; також кліматичні зони, геологічні зони, та наявні типи дерев: ялина/ялиця, сосна хрестовина, сосна пондероза, бавовник/верба, осика, дугласова ялиця, крумгольц. Цей набір даних використовувався в дослідженнях і навіть у конкурсі Kaggle. Це цікавий набір даних для вивчення в цьому розділі, оскільки він містить як категоріальні, так і числові ознаки. У наборі даних є 581012 прикладів, які не зовсім кваліфікуються як великі дані, але є достатньо великими, щоб ними можна було керувати як приклад і все ще підкреслюють деякі проблеми масштабу.

1.9. Висновки до розділу

У даному розділі було розглянуто перелік основних умов процесу обробки великих даних, та обґрунтування вибору основного фреймворка. Було надано за і проти вибору того чи іншого фреймворка, вступ до основної моделі, яка буде розглянута у даній науковій роботі. Поверхневий огляд даних, та наведення змінних, які впливатимуть на кінцевий результат. Також, був наведений приклад для вступу у машинне навчання.

РОЗДІЛ 2
ОПИС ТА ВИБІР МЕТОДІВ МАШИННОГО НАВЧАННЯ ПРИ ОБРОБЦІ ВЕЛИКИХ ДАНИХ

2.1. Попередня обробка даних та аналіз даних

Проведення обробки даних здебільшого ґрунтується, на методах інтелектуального аналіз. Вона є однією з технік інтелектуального аналізу даних, яка передбачає перетворення не оброблених («сирих») даних в більш доступний формат. Реальні дані часто не повні, суперечливі або містять помилки та пропущені значення. Попередня обробка даних готує для подальшої обробки вихідні дані.

Методи багатовимірного (статистичного) аналізу даних (такі як факторний, регресійний, кластерний аналіз та ін.) вимагають відсутності пропусків у аналізованих ними даних. Відсутність значення в даних (що являється пропуском) означає, що дані були певний час не доступні або не придатні в той момент, або подія, що призвела до появи цього значення, просто не відбулася.

Аналіз даних, в яких містяться пропуски, як і просте виключення розгляду об’єктів із пропусками, дає не повну картинку, а їхні висновки не є надійними,тобто являються помилковими. Отже, рішення, прийняте, по результатам даного аналізу, матиме фатальні наслідки.

Такі дані називають називають «брудними», тобто вони являються не придатними для продовження аналізу та мають бути попередньо оброблені. Групи типів «брудних даних»:

- які можуть бути автоматично виявлені та очищені;
- про появу яких можна бути попередженим;
- котрі є не придатними для автоматичного виявлення та очищення;
- появу яких не можливо запобігти.

Потрібно розуміти, що засоби очищення не можуть впоратися із всіма видами “брудних” даних.

Найпоширеніші види брудних даних:

— дублікати даних;

— пропущені значення;

— шуми і викиди.

2.1.1 Пропущені значення

Пропущеними значеннями, або Missing Values є значення, які були бути пропущені через:

− дані не були зібрані;

− дані були не доступні або не придатні для збору;

− помилки при внесенні даних;

− не всі ознаки можуть використовуватися для характеристики певних об'єктів.

Існують наступні шляхи вирішення даної проблеми:

– Заповнення з допомогою обчислених значень - коли відсутні значення бувають використані з допомогою обчислення середнього, медіани або режиму спостережуваних значень.

– Проігнорувати пропущені значення - є ефективним, та у свою чергу найпростіший спосіб при обробці відсутніх даних. Даний метод не повинен виконуватися при величезній кількості відсутніх значень.

– Заміна (підміна) пропущених значень на значення, що витікають з показників ознак інших об’єктів у вибірці.

– Ручне заповнення пропущених значень - є одним із найкращих методів. В свою чергу, він має деякі обмеження - коли набір даних великий, та відсутня значна кількість значень, то рахується неефективним, через трудомісткість завдання.

2.1.2. Дублювання даних

Дублікати у наборі даних, або є Duplicate Data. Дублікатами є об’єкти, у яких значеннями атрибутів являються однаковими.

Наявність дублікатів є одним із способів підвищення значущості записів.

Дана необхідність виникає коли потрібно особливо виділити певні записи із набору даних. Та у більшості випадків, дублікати являються результатом помилок.

Відомо два способи усунення наявних дублікатів:

1. Видалення всієї групи об’єктів, які містять дублікати. Використовується тоді, коли дублікати викликають не довіру до інформації, тобто повністю її знецінюють.
2. Заміні групи дублікатів унікальним записом.

2.1.3. Шуми та викиди

Викидами вважаються окремі ознаки об’єктів (або ж навіть цілі записи), які вибиваються із загальної картини. Не точність однакового вимірювання і точки даних буває прийнятною, і пов'язана з технічною помилкою приладу для вимірювання. Очищення даних має зосереджуватися на помилках, що є за межами малих технічних відхилень та є зрушенням в межах (або за межами) розподілу вибірки. Очищенням даних має базуватися знанням технічних помилок і прогнозований діапазон нормальних значень.

Із даною проблемою, у сфері аналізу даних, часто стикаються. Викиди можуть бути окремі спостереження, або спостереженнями об'єднаними в групи. Завданням спеціаліста є не тільки виявити, але й оцінити ступінь впливу їх на результати аналізу. Якщо викиди являються інформативною частиною набору даних, який аналізується, тоді використовуються робастні методи і процедури.

Поширеним являється підхід, в якому передбачають проведення двоетапного аналізу даних – з викидами та без викидів, а після порівнюють отримані результати.

Методи інтелектуального аналізу даних мають різну чутливість до викидів, це потрібно враховувати. Деякі інструменти Data Mining (Добування даних) мають вбудовані методи, що допомагають очищенню шумів та викидів. Наглядним способом виявлення являється візуалізація. Приклад викидів можна побачити на діаграмі розсіювання - рис. 2.1. На рисунку спостерігаються кілька спостережень, що значною мірою відрізняються від інших (котрі перебувають на далеко від

інших спостережень).

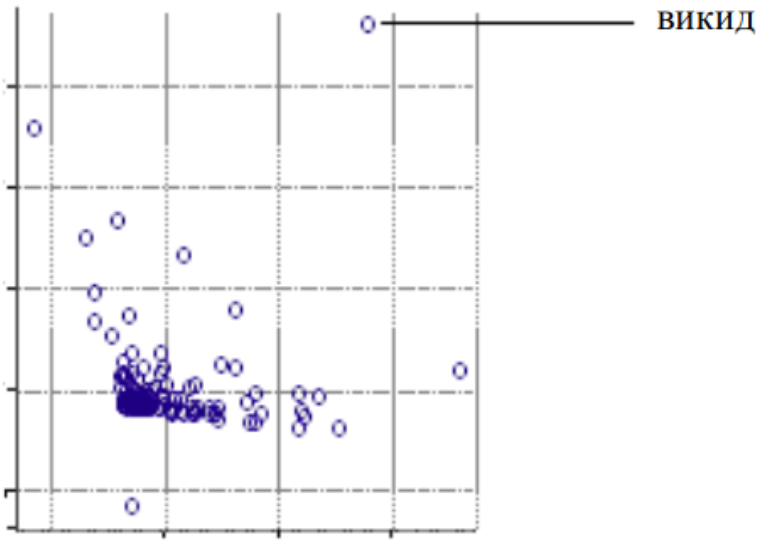

Рис. 2.1. Приклад набору даних з викидами

Окрема проблема полягає у помилкових даних, які внесені не вірно, але знаходяться у межах очікуваного діапазону (не можуть помічатися як викиди). Ці “inliers” являються підозрілими, коли розглядати їх у відношенні до інших змінних, та використовуючи ділянки розсіювання, перевірки або регресійний аналіз послідовності. Іноді їх можна виявити, коли вивчають історію кожної точки даних, або шляхом повторного вимірювання, та такого типу дослідження є рідко можливим. Замість цього, можна оцінити або змінити вибірку об’єктів помилок для самої оцінки частоти помилок.

Очевидно, що такого типу висновки, зроблені з допомогою аналізу брудних даних, не є достатньо надійними, щоб на їхній основі приймати рішення. Та наявність таких даних, не обов'язково означає їх необхідно очищувати або ж запобігати появу. Завжди має бути вибір між наявністю брудних даних та вартістю або часом, який буде витрачено на очищення даних.

2.1.4. Очищення даних

Очищення даних (data cleansing, data cleaning або scrubbing) займається виявленням, видаленням помилок та несумісностей в даних з метою покращення їх якості.

Проблеми з якістю часто зустрічаються в деяких наборах даних – таких як

бази даних та файли. Коли інтеграції підпадає безліч джерел даних (для прикладу у Сховищах, або інтегрованих системах, або ж глобальних інформаційних Інтернет-системах), необхідність очищення значно зростає. Це тому, що дані джерела даних зачасту включають розрізнені дані у різному представленні.

Спеціальними засобами очищення із конкретними областями – такими як адреси і імена – або із виключенням дублікатів. Дані перетворення даних забезпечуються користувачем у інтерактивному режимі, або ж у формі бібліотеки правил. Перетворення даних можуть бути автоматично отримані завдяки узгодження схеми[4].

На сьогоднішній день цілий ряд дослідницьких груп займається проблемами, пов'язаними із очищенням даних, зі специфічними підходами до отримання даних та перетворенню даних зіставляючи схеми.

2.1.5. Методи нормування даних

Усі вибірки вихідних даних, які завантажуються в аналітичний додаток, характеризуються набором властивостей, котрі в свою чергу можуть впливати на ефективність роботи моделі та знизити вірогідність результатів. Навіть коли дані очищені від факторів, які погіршують якість, такі як дублікати, шуми, протиріччя, пропуски, аномальні значення, та інше, вони досі можуть не відповідати методиці та цілям аналізу.

Дані бувають не впорядковані, або ж представлені у форматах, із якими не хоче працювати певний алгоритм. Щоб вирішити цю проблему використовують трансформацію даних (перетворення до певного формату, або виду тощо). Трансформацією являється набір підходів і алгоритмів, з допомогою яких можна оптимізувати формати і представлення даних для поставленої задачі, її цілей та ефективно використати дані.

До методів трансформації відносяться:

- дискретизацію або квантування даних;
- злиття;
- сортування;

- нормалізація;
- налаштування;
- додавання атрибутів, які обчислені на основі наявних даних;

Нормалізація дозволяє підвести всі дані до одного і того ж діапазону. Найчастіше використовують логарифмування (стиснення) всіх значень до інтервалу [0; 1]. Крім цього використовується десяткове масштабування, використання стандартного відхилення, мін-максну нормалізацію та поелементних перетворень.

2.1.6. Методи заповнення пропусків

Проблема пропущених значень є досить актуальною, наприклад для соціології. При прийнятті стратегічних рішень, які стосуються ринку праці, користуються результатами соціологічних досліджень, які в свою чергу засновані на масових опитуваннях.

Причинами неповноти цих даних можуть служити багато факторів: помилки у анкеті, неуважність респондента, відмінність у даних анкет і т.д. Як результат, на етапі аналізу ми маємо неповний масив даних. Тому для таких випадків призначені методи відновлення даних із пропусками.

Зараз є безліч методик, що дозволяють відновлювати дані із пропусками, та у кожної методики є свої переваги та недоліки: розміщення спостережень з пропущеними, заповнення середнім значенням, розрахунок значення, яке є можливе за допомогою регресійної моделі та інші. Серед них:

- Виключення рядків пропусками. Легко реалізовується, та необхідне проходження вимогу MCAR (missing completely at random), тобто пропуски у даних по змінним мають бути повністю випадковими. Крім цього, він зазвичай застосовується тільки при не значній кількості пропусків, інакше отримана таблиця даних стає не представницькою. Головним недоліком такого підходу є втрата інформації при виключенні не повних даних.
- Заповнення пропусків середніми значеннями по стовпчику. Цей метод легко реалізувати, та його застосування має сенс лише в разі задоволення даних

умові MCAR (missing completely at random), коли пропуски у даних випадкові та сам механізм не істотний. До недоліків методу також відносять зменшення дисперсії та спотворення розподілу даних.

Регресійне моделювання пропусків (заповнення пропусків) за допомогою регресійних моделей здійснюється у два етапи:

1. За сукупністю повних спостережень виконується регресійна модель, та оцінюються коефіцієнти в рівнянні, в якій у якості залежної змінної являється цільова змінна – пропущені значення по котрій необхідно відновити;

2. Потім з допомогою отриманого рівняння, підставляються відомі значення не залежних змінних предикторів, та для кожного об'єкта розраховується значення, яке відсутнє по залежній цільовій змінній. У разі абсолютних та інтервальних змінних розраховується конкретне значення, тоді як для порядкових та номінальних змінних передбачається категорія (з певною ймовірністю), до якої має бути віднесений об'єкт.

3. Метод Бартлетта - це ще один підхід до вирішення проблеми наявності пропусків. У даному методі пропущеному значенню присвоюється початкове (наприклад нуль або середнє арифметичне стовпчика), пізніше будується рівняння багатомірне лінійне регресійне із використанням початкового значення пропуску. Кінцеве ж значення присвоюється, виходячи із результатів аналізу регресії.

4. Метод максимізації очікувань (EM – expectation maximization) - дозволяє відновлювати пропущені значення із використанням двоетапного ітеративного алгоритму, та оцінювати середні значення, кореляційні та коваріаційні матриці для кількісних змінних. EM являє собою алгоритм, ціллю якого є вирішення задачі оптимізації функціоналу, із пошуком екстремуму цільової функції.

2.2. Вибір базових класифікаторів

2.2.1. Загальна постановка задачі класифікації

Розрізняють навчання із вчителем, навчання без вчителя,

напів-автоматичне та навчання з підкріпленням (рис 2.2) .

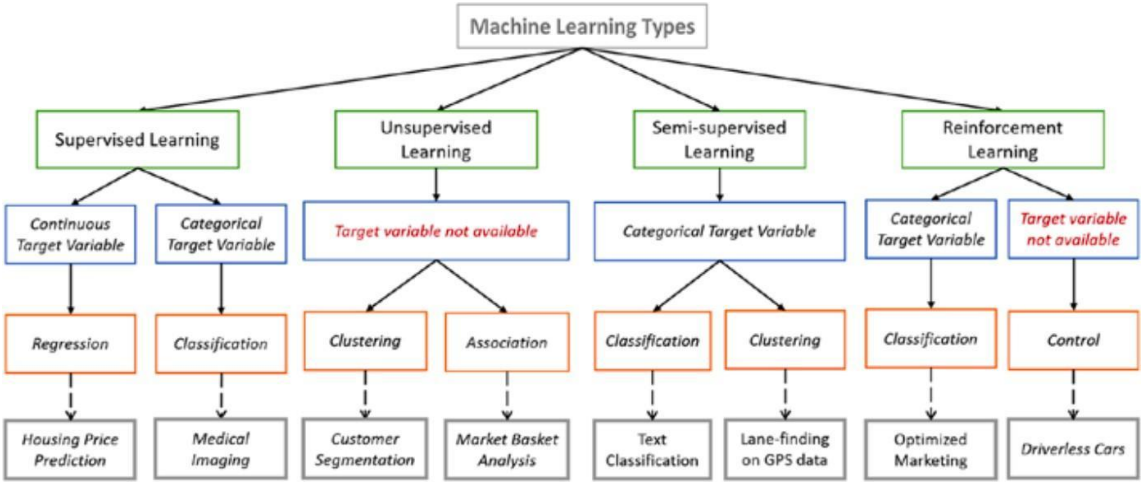

Рис. 2.2 – Типи задач машинного навчання

Спираючись на матеріали лекцій із машинного навчання Воронцова К.В.[5], то задача навчання за прецедентами, має наступний вигляд:

Нехай X – множина об’єктів, Y – множина відповідей, y: $X \to Y$ – це невідома залежність (target function).

Дано:

$\{ x_1, \dots , x_l\} \subset X$ - навчальна вибірка (training sample);

$y_i = y(x_i)$, $i = 1, \dots , l$ - відомі відповіді. Знайти:

a: $X \to Y$ – це алгоритм, функція прийняття рішення (decision function), яка приближує y по всій множині X.

Розглянемо детальніше, що саме представлятимуть об’єкти і відповіді.

f_j: $X \to D_j$, $j = 1, \dots , n$ – це ознаки об’єктів (features).

Маємо наступні типи ознак:

- $D_j = \{0,1\}$ – це бінарна ознака f_j;
- $|D_j| < \infty$ - це номінальна ознака f_j;
- $|D_j| < \infty$, Dj впорядкована – це порядкова ознака f_j;
- $D_j = R$ – це кількісна ознака f_j;

Цей вектор ($f_j(x), \dots, f_n(x)$) також називають вектором ознак об’єкта x.

Розглянемо матрицю «об’єкти-ознаки» (feature data)

$$F = \begin{pmatrix} f_1(x_1) & \cdots & f_n(x_1) \\ \vdots & \ddots & \vdots \\ f_1(x_l) & \cdots & f_n(x_l) \end{pmatrix}$$

Відповіді для задач класифікації (classification):

- $Y = \{-1; +1\}$ – класифікація на 2 класи.
- $Y = \{1, \ldots, M\}$ – на M класів, які не перетинаються.
- $Y = \{0,1\}^M$ – на M класів, які можуть перетинатися. Для задач регресії (regression):
- $Y = R$ або $Y = R^m$.

Для задач ранжування (ranking):

- Y – є скінченною впорядкованою множиною.

Метод навчання (learning algorithm) це відображення вигляду

$\mu: (X \times Y)^l \to A$,

де будь-яка вибірка X^l ставить в відповідність певний алгоритм $a \in A$.

Задачі навчання по прецедентах завжди складаються із двох етапів: навчання і тестування.

У якості функціоналів якості користуються поняттям функції втрат (loss function) $\mathcal{L}(a, x)$ – величина помилки алгоритму $a \in A$ на об'єкті $x \in X$.

Для задач класифікації використовують $\mathcal{L}(a, x) = [a(x) \neq y(x)]$ – індикатор помилки. Для вирішення задач регресії користуються абсолютним значенням помилки ($\mathcal{L}(a, x) = |a(x) - y(x)|$) або квадратичною похибкою ($\mathcal{L}(a, x) = (a(x) - y(x))^L$).

Тоді задача приводиться до задачі оптимізації через введення функції емпіричного ризику, як певного функціоналу якості алгоритму a на X^l:

$$Q(a, X^l) = \frac{1}{l}\sum_{i=1}^{l} \mathcal{L}(a, x_i),$$

$$\mu(X^l) = \arg\min_{a \in A} Q(a, X^l)$$

Тобто, класифікація – є одним із розділів машинного навчання, який

вирішує дану задачу: нехай маємо певну множину об’єктів, які, в свою чергу, розділені на класи за певною ознакою. Крім цього, ця задана скінченна множина об’єктів, та відомо, яким класам вони належать. Ця множина являється навчальною вибіркою. Та до якого класу відносяться решта об’єктів нам не відомо.

Нам необхідно зробити алгоритм, який в свою чергу буде в змозі класифікувати будь- який із об’єктів вихідної множини. Класифікація об’єкта - це відповідь алгоритму, що вказує номер класу (до якого він відноситься).

Задача класифікації у машинному навчанні є однією із задач навчання із вчителем. Які поділяють за такими ознаками:

- Типи вхідних даних: описи ознак, зображення тощо, часові ряди матриця відстаней.
- Типи класів: дво класова класифікація, що перетинаються, класи, що не перетинаються, багато класова, класи, та нечіткі класи.

Алгоритми класифікації широко застосовуються у всіх сферах життя. Найбільш розповсюдженими є застосування у скорингових моделях (оцінки кредитоспроможності), маркетингу, медицині, мови та символів, розпізнавання образів, пошук спаму і навіть у геології.

2.2.2. Лінійні класифікатори

Розглянемо задачу кредитного скорингу (рис 2.3). Ми хочемо збудувати класифікатор, який розрізняє позичальника, котрий є потенційним банкрутом. Характеристика прибутку і кредитний рейтинг визначити потенційних дефолтерів.

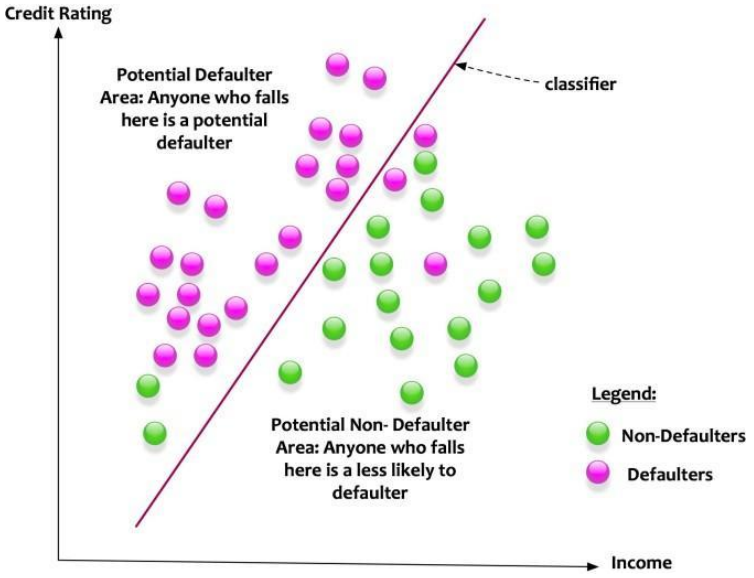

Рис. 2.3. Задача кредитного скорингу

Ця діаграма ілюструє сценарій. Вважаємо, що простір ознак є перетином доходу і кредитного рейтингу. Зелені точки - це надійні позичальники, рожеві – це банкрути. Класифікатор навчається з допомогою вхідних даних і створює лінію, яка розділяє усі об’єкти на дві групи. Результатом роботи даного алгоритму є модель, яка класифікує клієнтів з допомогою наступного принципу:

- Усі, хто зліва від лінії – це потенційні банкрути.
- Усі, хто справа від лінії – це потенційні не банкрути.

Подібні класифікатори, котрі розбивають простір ознак за допомогою лінії (або площиною у просторі, розмірність якого є більше двох), називаються лінійними.

Професор Воронцов К. В. формально описує задачу лінійної класифікації наступним чином. [5]

Нехай ці об’єкти описуються n-числовими ознаками f_8: $X \to R$. Тоді простір ознакових описів об’єкту - це $X = R^n$. Нехай Y – скінченна множина маркерів класів.

В випадку бінарної класифікації ($Y = \{-1; +1\}$) лінійним класифікатором є алгоритм класифікації $a: X \to Y$ вигляду:

$$a(x,w) = sign\left(\sum_{j=1}^{n} w_j f_j(x) - w_0\right) = sign\langle x,w\rangle,$$

де w_j – вага j-тої ознаки, w_0 – поріг прийняття рішення, $\langle x, w\rangle$ – скалярний добуток.

За допомогою методу мінімізації емпіричного ризику зазвичай проводиться навчання лінійних класифікаторів.

Налаштування класифікатора цим методом полягає у тому, що на навчальній вибірці пар «об'єкт, відповідь» потрібно побудувати алгоритм, який мінімізуватиме функціонал емпіричного ризику, що має наступний вигляд:

$$Q(w) = \sum_{i=1}^{m} [a(x_i, w) \neq y_i] \rightarrow \min_w$$

У випадку бінарної класифікації дуже зручно визначати для довільного об'єкта із навчальної вибірки $x_i \in X^m$ величину відступу (margin):

$M(x_i) = y_i \langle x_i, w\rangle$

У випадку довільного числа класів, відступ визначається наступним чином

$$M(x_i) = \langle x_i, w_{y_i}\rangle - \max_{y \in Y, y \neq y_i} \langle x_i, w\rangle$$

Відступ можна розуміти як «ступінь занурення» об'єкта у свій клас. Чим менше відступ, тим ближче об'єкт до розділяючої поверхні та ймовірність не вірної класифікації підвищується. Відступ $M(x_i)$ набуває від'ємних значень лишн тоді коли алгоритм $a(x)$ припускається помилки на об'єкті x_i. Це спостереження дозволяє нам записати функціонал емпіричного ризику у наступному вигляді:

$$Q(w) = \sum_{i=1}^{m} [M(x_i) < 0]$$

Під знаком суми - порогова функція втрат, а тому функціонал $Q(w)$ не є випуклим, та не є навіть неперервним; це є причиною незручності мінімізації та потребує апроксимації функції втрат.

Лінійні методи класифікації різняться між собою, вибором функції L(M), не перервною апроксимацією порогової функції втрат, способом регуляризації і вибором чисельного методу оптимізації функціоналу.

Регуляризація – являється одним з основних способів боротьби із, так

названих, феноменів перенавчання.

2.2.2.1. Лінійний дискримінант Фішера

Відповідає квадратичній апроксимації з умовою, що від кожного вектору ознак попередньо віднімається вектор центрів класів:

$$[M < 0] \leq (1 - M)^2]$$

Оптимізація проводиться методом найменших квадратів.

2.2.2.2. Одношаровий персептрон

Використовується апроксимація сигмоїдною або логіт-функцією:

$$[M < 0] \leq \frac{2}{1 + e^{\alpha M}}$$

де параметр α - деякий параметр, що задається за замовчуванням. В цьому випадку є необхідним використанням додаткових евристик для зменшення перенавчання - регуляризація та стохастичні методи для вибивання із локальних мінімумів.

2.2.2.3. Логістична регресія

Логістична регресія (логістична функція, або сигмоїдна функція) (рис 2.4) була розроблена статистами, щоб описувати ріст популяції у екології. Вона приймає форму літери S і може бути будь яким значенням між 0 і 1, але ніколи не дорівнює їм.

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

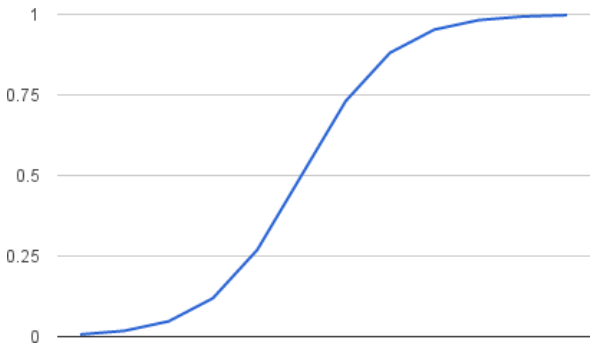

Рис. 2.4 – Логістична або сигмоїдна функція

Якщо розглянути введене вище поняття відступу, ми помітимо порогову функцію втрат, які апроксимують не перервною оцінкою зверху наступним чином:

$$[M < 0] \leq \log_2(1 + e^{-M})$$

Навчання даного логарифму проводяться з допомогою методів градієнтного спуску другого порядку.

2.2.2.4. Метод опорних векторів

Метод опорних векторів, вони ж Support Vector Machine or SVM. Кусково-лінійній апроксимації є одним із найпоширеніших методів навчання з учителем:

$$[M < 0] \leq (1 - M)_+$$

Тут ми використовуємо регуляризації із квадратичною нормою, яке є обов'язковим. Навчання виконують завдяки спеціалізованих методів квадратичного програмування.

До переваг даного методу належить його швидкість, та гарантована наявність єдиного розв'язку (що є наслідком зведення до задачі квадратичного програмування).

Також існують нелінійні модифікації вектору опорних векторів. Лінійність або нелінійність алгоритму визначається типом ядра.

Для прикладу розглянемо метод, який запропонований компанією Майкрософт, під назвою - Two-Class Locally Deep Support Vector Machine, програмна реалізація даного методу являється частиною Azure Machine Learning Studio. Особливістю якого є не типова функція ядра, яка є розробленою для максимально зменшення часу навчання, при умові відсутності погіршення точності алгоритму класифікації.

2.2.3. Метод *k* найбільших сусідів

Являє собою *Узагальнений метричний класифікатор.* Розглянемо наступний приклад - для довільного $x \in X$ відроджуємо об'єкти $x1, \ldots xl$:

$\varrho(x, x^{(1)}) \leq \varrho(x, x^{(2)}) \leq \cdots \leq \varrho(x, x^{(l)})$,

$x^{(i)}$ – i-тий сусід об'єкта x серед $x1, \ldots xl$;

$y^{(i)}$ – відповідь на i-тому сусіді об'єкта x.

Тоді метричний алгоритм класифікації шукаємо в наступному вигляді:

$$a(x; X^l) = \arg\max_{y \in Y} \sum_{i=1}^{l} [y^{(i)} = y]\, w(i, x)$$

$w(i, x)$ – ваговий коефіцієнт, який вказує степінь важливості i-того сусіда об'єкта x, невід'ємний і не зростає по і.

$\sum_{i=1}^{l} [y^{(i)} = y]\, w(i, x)$ – оцінка близькості об'єкта x до класу y.

У випадку, коли $w(i, x) = [i \leq 1]$, маємо наступний метод найближчого сусіда.

Якщо ж $w(i, x) = [i \leq k]$ – є методом k найближчих сусідів.

Тобто, об'єкт є частиною певного класу, залежно від належності більшості його найближчих сусідів певному класу.

До переваг методу відносяться:

- простота реалізації;
- також, чудово працює в якості першочергового вирішення задачі;
- обравши метрику чи ядро, можна адаптувати під конкретну задач.

Серед недоліків алгоритму зазначають:

- метод вважається швидким, але в реальних задачах число сусідів, що використовуються для класифікації, буде великим (100-150), що сповільнює роботу алгоритму, тоді вибір падає на, скажімо, дерева рішень, який ми використовуємо в нашій роботі;
- при великому просторі ознак об'єкта важко підібрати відповідні вагові коефіцієнти та визначити не важливість ознак для вирішення даної задачі;
- залежність обраної метрики виміру відстані між об'єктами (вибір за замовчуванням евклідової відстані являється не обґрунтованим).

2.2.4. Наївний байєсівський класифікатор

Класифікатор Байєса використовує концепцію «ймовірності». Тобто, це класифікатор, який використовує Теорему Байєса і визначає можливість належності об'єкта до того чи іншого класу, як ймовірність належності, прорахована для кожного класу. У результаті об'єкт являється частиною класу, показник ймовірності у якого вищий.

Класифікацію за даним методом проводять у припущенні, що кожен з об'єктів $x \in X$, та описується n незалежними ознаками. Що означає, що дана функція правдоподібності класів розпадається на добуток.

Професор Воронцов стверджує [5], що припущення про незалежність ознак спрощує задачу, адже оцінити n одномірних щільностей простіше, ніж n-мірну щільність. На жаль, воно рідко справджується на практиці, що й спонукало таку назву – «наївний».

Класифікатор може бути параметричним та непараметричним, в залежності від метода відновлення одновимірних щільностей.

До переваг відносять просту реалізацію і не високі обчислювальні витрати для навчання та класифікації. У випадках, коли ознаки взаємо незалежні, наївний класифікатор байєса – (майже) оптимальний.

Основний недолік – низька точність класифікації у більшості реальних задач. Найчастіше ним користуються щоб порівняти різні моделі алгоритмів або структурну одиницю в складніших ансамблях алгоритмів.

2.2.5. Дерева рішень

У нашій роботі ми використовуємо Decision Trees або дерева рішень - це один із логічних методів класифікації. Професор Воронцов зазначає у своїх лекціях [6], що деревом називається скінченний зв'язний граф із множиною вершин V, який не містить циклів та має виділену вершину $v0 \in V$, яка не входить ні на одне ребро. Ця вершина є коренем дерева. Вершина, яка не має ребер, називається термінальною або ж «листом». Інші ж вершини називаються внутрішніми. А дерево називають «бінарним», коли із будь якої його внутрішньої

вершини виходить 2 ребра (рівно).

Бінарне дерево рішень - це класифікатор, який задається бінарним деревом, та у кожній його вершині прописаний предикат βv : X → {0, 1}, а кожному його «листку» відповідає мітка класу.

Використовується поняття ентропії або кількості інформації за Шенноном, та визначається вона наступним чином:

$$S = -\sum_{i=1}^{N} p_i \log_2 p_i,$$

де *pi* – ймовірність знаходження системи у стані *i*.

Основою поширених алгоритмів побудови дерев рішень є принцип жадібної максимізації приросту інформації. Тобто, на кожному кроці обирається ознака, при розбитті котрої приріст ентропії виявляється найбільшим.

Після цього процедура повторюється до тих пір, поки кількість інформації буде більша нуля або стане не меншою заданого числа. У Різних алгоритмах використовують різні евристики для усічення, для того, щоб не трапилось перенавчання.

Іншими критеріями якості розбиття в задачах класифікації виділяють:

− Невизначенність Джині (Gini impurity): $G = 1 - \sum_Ç (p_Ç)^L$. Під його максимізацією розуміють максимізацію числа пар об’єктів одного класу, що опиняються в одному під-дереві. [7]

− Помилку класифікації (misclassification error):

$$E = \max_k p_k$$

На практиці майже не використовується, а ентропія та не визначеність Джині дають майже однакові результати роботи.

Перевагами використання дерев рішень є:

− інтуїтивно зрозумілий алгоритм, який, в свою чергу, легко представити у графічному вигляді і інтерпретувати;

− може працювати як з числовими, так із категоріальними ознаками;

− вимагають невеликої попередньої обробки даних із боку користувача;

– не лінійні залежності між параметрами, майже не впливають на якість роботи.

Недоліками являються:

– нестійкість до перенавчання;

– нестійкість навіть до невеликих відхилень у даних (для їх усунення рекомендується використання дерев, як частини ансамблю алгоритмів, таких як беггінг та бустинг).

2.3. Використання ансамблів моделей класифікації, як більш ефективного алгоритму

2.3.1. Беггінг

Для початку розглянемо поняття бутстрепінгу. В машинному навчанні до методу бутстреп (bootstrap) відноситься випадковий вибір під наборів даних із заміною. Цей взірець називається ресамплером. Що дозволяє моделі або ж алгоритму краще зрозуміти різноманітні особливості та відхилення, які існують в ресамплері. Приймаючи взірець даних, *resample* містить різні характеристики, які могли б міститися у цілому датасеті. На рис. 2.5 , продемонстровано,як кожна вибірка населення може мати різні частини та жоден з них не ідентичний. Що вплине на стандартне відхилення, загальне середнє значення та інші описові показники. В свою чергу, цей підхід може допомогти розвинути більш надійні моделі.

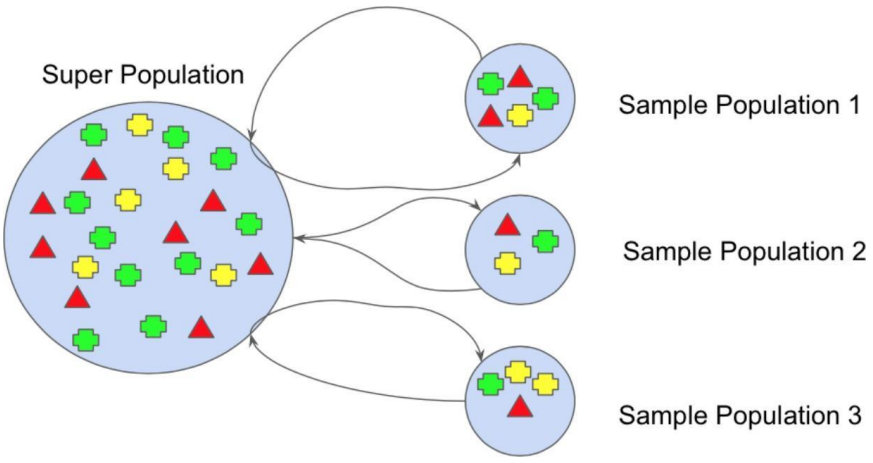

Рис. 2.5. Іллюстрація бутстрепу

Бутстрепінг чудово підходить наборам даних не великого об’єму, що мають

тенденцію до перенавчання. Причиною для використання методу бутстрепу є перевірка стабільності рішення, використання декількох наборів даних, а пізніше тестування декількох моделей, що може підвищити надійність.

Причиною популярності даного підходу являється збільшення обчислювальної потужності. Що дозволяє робити більше перестановок із різними під-наборами. Бутстрепінг використовується у Беггінгу, та у Бустингу.

Беггінг (Begging or Bootstrap Aggregating) – є одним із основних методів агрегації у машинному навчанні, що використовує паралельне навчання класифікаторів. Процес беггінгу можна описати наступними кроками:

1. Із множини вихідних даних, випадковим чином підбирається кілька підмножин, які містять у собі кількість об’єктів, що в свою чергу відповідає кількості об’єктів початкового набору даних.

2. Варто пам’ятати, що відбір реалізується випадковим чином, тому набір об’єктів завжди буде різним (тобто деякі приклади будуть у декількох підмножинах, а деякі – в жодній.

3. Стосовно кожної вибірки будується класифікатор.

4. Результати їх роботи (класифікаторів) агрегуються.

Агрегація результатів, відбувається через усереднення або голосування. При тому, що перший варіант використовується у задачі регресії, а другий у задачі класифікації.

Використання беггінгу допомагає зменшити відхилення. Класичним алгоритмом класифікації, у основі якого лежить бегінг являється Випадковий Ліс (Random Forest).

Випадковий ліс - алгоритм, який базується на деревах рішень, що включає створення декількох дерев, і після цього поєднує їх результати для покращення можливостей узагальнення моделі. Цей метод відомий як метод ансамблю. Ансамбль - це поєднання слабких учнів (окремих дерев), щоб створити сильного учня. Випадкові ліси стійкіші до перенавчання. Кількість дерев можна збільшувати, швидкість роботи все ще буде високою.

Випадковий ліс працює таким чином: навчальна вибірка складається

шляхом відбору зразків (samples) із заміною (для поточного дерева), близько третини випадків залишаються за зразком. Отримані дані, які мають назву “out-of-bag”, використовуються для отримання не упередженої оцінки помилок класифікації, коли нові дерева додаються. Цей підхід ще використовують для отримання оцінок важливості змінних.

Після побудови кожного дерева усі дані пропускаються вниз по дереву, тоді як близькість обчислюється для кожної пари випадків. У випадку, коли два випадки потрапили у одну і ту ж кінцеву вершину (лист), їх значення близькості збільшується на одиницю. В кінці пробігу алглритму, показники близькості нормалізуються шляхом ділення на кількість дерев.

Внутрішньо вона оцінюється наступним чином: кожне із дерев конструюється із допомогою іншого взірця бутстреп із вихідних даних. Близько третини випадків залишаються поза набором і не використовуються для побудови k-го дерева. Кожен об’єкт, який не був використаний для побудові дерева k-го дерева, пропускається алгоритмом, для отримання класифікації. Таким чином, класифікація об’єктів вибірки проводиться у середньому третиною дерев. В кінці, для кожного об’єкта обирають j -тий клас, що отримав більшу кількість голосів кожного разу. Частка випадків, в яких j не дорівнює класу n, усереднена по всіх випадках, що дає нам оцінку помилки.

В кожному дереві, вирощеному у лісі, відкладемо випадки out-of-bag та підрахуємо голоси, «віддані» за правильний клас. Далі випадково перемішаємо значення змінної m та пропустимо через у дерево. Віднімемо кількість голосів за правильний клас у змінних даних out-of-bag від кількості голосів за правильний клас на недоторканих даних. Середнє значення цього числа в всіх деревах лісу є значенням важливості для змінної m.

Якщо значення даного показника від дерева до дерева не залежні, то стандартна помилка може обчислюватися за допомогою стандартного обчислення. Кореляції отриманих оцінок між деревами є обчисленими для ряду наборів даних та виявилися досить низькими, саме тому ми вираховуємо стандартні похибки, ділимо оцінку даного рядку за стандартною помилкою, для отримання z-балу, та

призначати рівень значущості до z-оцінки припускаючи нормальність.

Якщо кількість змінних є дуже великою, ліси можуть бути запущені один раз із усіма змінами, а потім запустити знову, використовуючи лише найважливіші змінні із першого запуску.

Окрім цього, іноді використовують метод усічення (pruning), який видаляє вузли нижчих рівнів.

Окремим розширенням ідеї цього методу є ротаційний ліс (Rotation Forest), який був вперше запропонований у роботі [8], основна особливість його полягає у підготовці даних для навчання простих класифікаторів.

2.3.2. Бустинг

Так само, як беггінг, основним завданням якого є перетворення набору слабких класифікаторів (тобто таких, які припускаються багатьох помилок на тестовій виборці) у один більш сильний. Але, на відміну від беггінгу, тут навчання відбувається послідовно та кожний наступний класифікатор має на меті компенсацію недоліків попереднього.

Фінальний результат це зважена лінійна комбінація відповідей усіх використаних алгоритмів.

Перелік алгоритмів, у основі яких знаходиться принцип бустингу, є дуже великим:

- Stochastic Gradient Boosting – у основі лежить ідея використання методу градієнтного спуску, і кожний новий алгоритм налаштовувати на випадковому піднаборі даних. Детальний опис алгоритму наведений у роботі [9].
- LPBoost: Linear Programming Boosting – бустинговий алгоритм класифікації, який максимізує відступ.
- BrownBoost: підвищує надійність алгоритму на наборах даних, які містять багато шуму, відкидаючи об’єкти, які постійно класифікуються помилково.
- LogitBoost: використовує логарифмічна функція втрат.
- AdaBoost – частковий випадок градієнтного бустинга, у якому в

якості функції втрат використовується експоненційна. AdaBoost являється першим теоретично дослідженим варіантом бустинга, для якого вперше була доведена основна теорема із аналітично виведеними оптимальними b_m на кожному кроці. Початкова версія алгоритму повертала тільки номер класу у бінарній класифікації; пізніше він був узагальнений та тепер являється можливість отримати ймовірність належності відповідному класу.

− CatBoost – технологія, яка була запропонована Яндексом минулого року. В основі алгоритму лежить градієнтний бустинг. На відміну від більшості існуючих алгоритмів, цей здатний ефективно працювати із категоріальними даними. Яндекс використовує його щоб вирішувати задачі ранжування, прогнозування і побудови рекомендацій.

− XGBoost: повна назва - “Extreme Gradient Boosting”, бібліотека, яка реалізує градієнтний бустинг над деревами [10].

− LightGBM: у основі також лежить градієнтний бустинг над деревами, але на відміну, від XGBoost, побудова дерева відбувається горизонтальним шляхом.

Різницю між двома останніми алгоритмами (XGBoost та LightGBM) наглядно демонструє наступні схеми (рис 2.6 – 2.7).

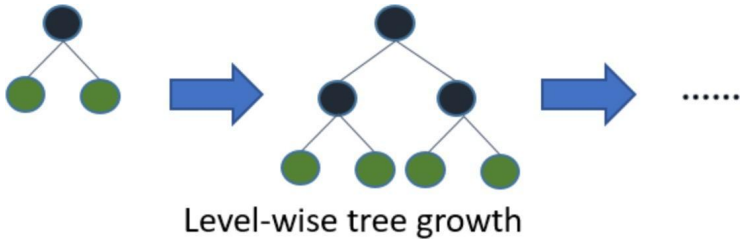

Рис. 2.6. Нарощування листків з огляду на рівень в (XGBoost)

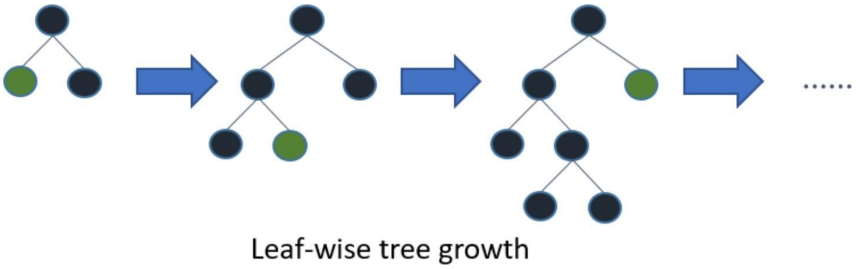

Рис. 2.7. Горизонтальне нарощування (LightGBM)

У роботі [11] також досліджують проблему кореляції між простими

класифікаторами і пропонують 4 шляхи вирішення, серед яких:

- Навчити кожен класифікатор на окремих піднаборах із даних, які не перетинаються.
- Генерація нової навчальної вибірки, з допомогою «усічення»: відкидання надлишкової інформації з вхідних даних.
- Використання ресемплінгу.
- Зважене комбінування і розбиття на піднабори через використання певного принципу, а не випадковим чином, як у попередніх методах.

2.4. Метрики оцінки якості роботи класифікаторів

Наступним кроком є перевірка ефективності моделі з допомогою спеціалізованих метрик і тестового набору даних. У залежності від типу задачі, аналітик даних має можливість обрати, якими метриками користуватися у тому чи іншому випадку.

Метрики якості, котрі ви обираєте, щоб оцінки модель машинного навчання є дуже важливими. Вибір метрики має безпосередній вплив на алгоритми оцінювання і порівняння між собою. Для цього необхідно розуміти природу досліджуваної проблеми.

2.4.1. Правильність (Accuracy)

Так як правильність класифікації визначається співвідношенням між вірними відповідями і загальною кількістю відповідей.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

Accuracy - ефективна метрика якості, коли ми маємо справу з збалансованими класами. Та, якщо кількість об’єктів одного класу на багато перевищує інший, тоді точність покаже хороші результати, а це насправді неправдиво.

2.4.2. Точність (Precision)

$$Precision = \frac{TP}{TP + FP}$$

Точність є мірою, яка вказує яка частка даних, потрапила у нашу вибірку за вказаними критеріями.

2.4.3. Повнота (Recall) або Чутливість (Sensitivity)

$$Recall = \frac{TP}{TP + FN}$$

Повнота дає нам інформацію про роботу класифікатора із огляду на *FN* (тих значень, що алгоритм «не вгадав»), тоді як Точність бере до уваги *FP*.

Отже загалом, все залежить від природи задачі і ціни помилки. Якщо ми маємо за мету мінімізувати кількість «невгаданих» значень, тоді намагатимемось наблизити Точність якомога ближче до 100 відсотків, намагаючись при цьому мати не найгірше значення повноти. І навпаки, якщо важливіше мінімізувати кількість *False positives*, то основний фокус має бути на підвищенні значення Точності.

2.4.4. Специфічність (Specificity)

$$Specificity = \frac{TN}{TN + FP}$$

Вказує на частку «негативних» прикладів, які були класифіковані вірно.

2.4.5. F - міра

Міра дозволяє нам враховувати відразу дві метрики - Точність та Повноту. Отож маємо:

$$F = \frac{2 * precision * recall}{precision + recall}$$

2.4.6. Log-loss (logarithmic loss)

Використовується в випадку, як на виході алгоритму маємо не клас, а ймовірність, із якою об’єкт відноситься до певного класу. Log-loss - це "м'яке" вимірювання точності, яке інкорпорує ідею довірчого інтервалу.

$$log - loss = -\frac{1}{N}\sum_{i=1}^{N} y_i * \log(p_i) + (1 - y_i)\log(1 - p_i),$$

де N – об’єм вибірки, y_i – реальна мітка класу, p_i – відповідь алгоритму на *i*-тому об’єкті.

2.4.7. ROC крива (Receiver Operating Characteristics Curve)

Статистична ROC крива це графік, який ілюструє ефективність бінарної системи класифікації залежно від заданого порогу для ймовірності належності класу. ROC крива дає детальну інформацію про поведінку класифікатора. Вона є результатом зображення на графіку взаємозв’язку *True Positive rate (TPR)* і False Positive rate (FPR) залежно від порогу.

Так ROC крива ілюструє чутливість класифікатору, візуалізуючи скільки правильно класифікованих об’єктів можна отримати, дозволяючи все більше і більше випадків FP.

ROC крива (рис 2.8) для якісної моделі має велику площу (бо частка правильно класифікованих об’єктів *(TPR)* зростає до 100% дуже швидко). Тому графік кривої для поганого класифікатора залишає під собою дуже маленьку площу. Ідеальна модель досягатиме стовідсоткової точності моментально, та такого ніколи не трапляється на практиці.

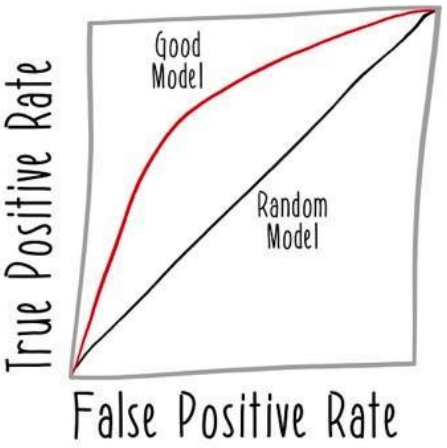

Рис. 2.8 – ROC-крива

В роботі [12] згадується, що площа ROC кривою (AUC) – дуже розповсюджена міра якості виконаної роботи алгоритмів класифікації. З іншого боку, AUC має також і недоліки, деякі з них дуже добре відомі дослідникам.

У роботі [12] математично обґрунтовано недоліки використання AUC і запропоновано альтернативу – H-міру. Автори стверджують - якщо співвідношення вартості не визначено дослідником, тоді неминуче виникає неконтрольований аспект, пов'язаний із будь-якою метрикою, яка інтегрується над таким розподілом. Для вимірювання H це полягає в виборі бета-розподілу та рівних значень параметрів. Проте ця незручність набагато менш тривожна, ніж внутрішня неоднорідність, спричинена мірою AUC, що використовує різні розподіли для оцінки різних класифікаторів.

Крім цього використовують криву Precision-Recall, яка зображує точність і повноту при різних порогових значеннях. Крива precision-recall ілюструє компроміс точності і моделі за різними порогами.

2.5. Висновки до розділу

У даному розділі було розглянуто постановку задачі машинного навчання і, зокрема, бінарної класифікації. Було вивчено поширені моделі класифікаціїї, як байєсівський класифікатор, логістична регресія, одношаровий персептрон, метод k найближчих сусідів, метод опорних векторів, дерева рішень та інші, детально були розглянуті переваги і недоліки кожного. Були розглянуті методи створення

ансамблів класифікаторів, описана ідея кожного і виконаний огляд найновітніших розробок у цій сфері. Також, було розглянуто методи оцінки якості роботи моделей, такі як повнота, точність, крива roc-auc, та інші. Була пояснена концепція кросс-валідації, як підходу, який знижує варіативність результатів.

РОЗДІЛ 3
ВИБІР ТА ОПИС МЕТОДІВ МАШИННОГО НАВЧАННЯ ДЛЯ ОБРОБКИ ВЕЛИКИХ ДАНИХ

3.1. Підготовка вхідних даних та обробка файлу CSV

Фактичний тип лісового покриву для даної осередку розміром 30 x 30 метрів був визначений на основі даних Системи інформації про ресурси регіону 2 лісової служби США (USFS). Потім незалежні змінні були отримані з даних, отриманих від Геологічної служби США та USFS. Дані в необробленому вигляді (не масштабовані) і містять двійкові стовпці даних для якісних незалежних змінних, таких як території дикої природи та тип ґрунту. Загальний опис вхідних даних був наведений у розділі 1, тому повертатися до нього немає змісту; деталі можуть бути знайдені у списку використаних джерел[2].

Дані у простому форматі CSV і не вимагають особливого очищення чи іншої підготовки для використання зі Spark MLlib. Файл covtype.data слід розпакувати та скопіювати в HDFS. У цьому розділі буде вважатися, що файл доступний за адресою /user/ds/. Запустити Spark-shell. Можливо, вам знову буде корисно надати оболонці достатній обсяг пам’яті для роботи, оскільки створення рішень може бути ресурсомістким. Якщо у вас є вільна оперативнапам’ять, вкажіть --driver-memory 8g або аналогічний.

Файли CSV містять принципово табличні дані, організовані в рядки стовпців. Назви стовпців наведено у супровідному файлі covtype.info. Концептуально кожен стовпець файлу CSV також має тип — число, рядок.

Природно аналізувати ці дані як фрейм даних, оскільки це абстракція Spark для табличних даних із визначеною схемою стовпців, включаючи імена та типи стовпців. Фактично, Spark має вбудовану підтримку для читання даних CSV(лістинг 1.1).

```
val dataWithoutHeader = spark.read.
  option("inferSchema", true).
  option("header", false).
  csv("hdfs:///user/ds/covtype.data")

...
org.apache.spark.sql.DataFrame = [_c0: int, _c1: int ... 53 more fields]
```

Лістинг 3.1. CSV рідер

Цей код зчитує вхідні дані як CSV і не намагається розібрати перший рядок як заголовок імен стовпців. Він також вимагає, щоб тип кожного стовпця був визначений шляхом вивчення даних. Це правильний висновок, що всі стовпці є числами, а точніше, цілими числами. На жаль, він може називати лише стовпці «_c0» тощо.

Дивлячись на назви стовпців, стає ясно, що деякі функції справді є числовими. «Elevetion» — це висота в метрах; «Slope» вимірюється в градусах. Однак «Wilderness_Area» — це щось інше, оскільки кажуть, що він охоплює чотири стовпці, кожен з яких дорівнює 0 або 1. Насправді «Wilderness_Area» є категоріальним значенням, а не числовим.

Ці чотири стовпці насправді є кодуванням «*одна гаряча*» (“*one-hot*”) або «1 з n», в якому одна категоріальна ознака, яка приймає N різних значень, стає N числовими ознаками, кожна з яких приймає значення 0 або 1. Рівно одне з N значень має значення 1, а інші — 0. Наприклад, категоріальна ознака погоди, яка може бути хмарною, дощ або ясною, стане трьома числовими ознаками, де хмарно позначається 1,0,0; дощовий на 0,1,0 ; і так далі. Ці три числові характеристики можна розглядати як функції *is_cloudy*, *is_rainy* та *is_clear*. Так само, 40 стовпців насправді є однією категоріальною ознакою Soil_Type.

Це не єдиний можливий спосіб кодування категорійної ознаки як числа. Інше можливе кодування просто призначає окреме числове значення кожному можливому значенню категорійного ознаки. Наприклад, хмарно може стати 1,0, дощове 2,0 тощо. Сама ціль, "Cover_Type", є категоріальним значенням, закодованим як значення від 1 до 7.

Отже, ми бачимо обидва типи кодування категоріальних ознак. Можливо,

було б простіше і зрозуміліше не кодувати такі функції (причому двома способами, не менше), а натомість просто включити їх значення безпосередньо, як «Зона дикої природи Рава». Це може бути артефактом історії; набір даних був випущений в 1998 році. З міркувань продуктивності або щоб відповідати формату, очікуваному бібліотеками того часу, які були створені більше для проблем регресії, набори даних часто містять дані, закодовані таким чином.

У будь-якому випадку, перш ніж продовжити, корисно додати імена стовпців до цього DataFrame, щоб полегшити роботу(Лістинг 3.2).

```
val colNames = Seq(
    "Elevation", "Aspect", "Slope",
    "Horizontal_Distance_To_Hydrology", "Vertical_Distance_To_Hydrology",
    "Horizontal_Distance_To_Roadways",
    "Hillshade_9am", "Hillshade_Noon", "Hillshade_3pm",
    "Horizontal_Distance_To_Fire_Points"
  ) ++ ( ❶
    (0 until 4).map(i => s"Wilderness_Area_$i")
  ) ++ (
    (0 until 40).map(i => s"Soil_Type_$i")
  ) ++ Seq("Cover_Type")

val data = dataWithoutHeader.toDF(colNames:_*).
  withColumn("Cover_Type", $"Cover_Type".cast("double"))

data.head
...
org.apache.spark.sql.Row = [2596,51,3,258,0,510,221,232,148,6279,1,0,0,0,...
```

❶ ++ concatenates collections

Лістинг 3.2. Імена стовпців DataFrame

Стовпці, що стосуються дикої природи та ґрунту, мають назви «Wilderness_Area_0», «Soil_Type_0», і трохи Scala може генерувати ці 44 назви без необхідності вводити їх усі. Нарешті, цільовий стовпець «Cover_Type» наперед перетворюється на подвійне значення, тому що насправді необхідно буде використовувати його як подвійне, а не int у всіх API Spark MLlib. Це стане очевидним пізніше.

Ви можете викликати *data.show(),* щоб побачити деякі рядки набору даних, але дисплей настільки широкий, що його буде важко прочитати взагалі *data.head* відображає його як необроблений об'єкт *Row*, який у цьому випадку буде більш

читабельним.

3.2. Перше дерево рішень (Decision Tree)

Оскільки ми не знаємо, як скласти новий опис нової земельної ділянки в Колорадо з 54 елементами або якого лісового покриву очікувати від такої ділянки.

Замість цього ми повинні відразу перейти до надання деяких даних для оцінки отриманої моделі. Раніше метрика AUC використовувалася для оцінки узгодження між даними прослуховування і прогнозами з рекомендацій. Принцип тут той самий, хоча показник оцінки буде іншим: точність. Більшість — 90% — даних знову буде використано для навчання, а пізніше ми побачимо, що підмножина цього навчального набору буде затримана для перехресної перевірки (набір CV). Інші 10% витриманих тут насправді є третьою підмножиною, належним набором тестів(Лістинг 3.3).

```
val Array(trainData, testData) = data.randomSplit(Array(0.9, 0.1))
trainData.cache()
testData.cache()
```

Лістинг 3.3.Тестове навчання

Для використання з класифікатором у Spark MLlib дані потребують трохи більше підготовки. Вхідний DataFrame містить багато стовпців, кожен з яких містить одну функцію, яка може бути використана для прогнозування цільового стовпця. Spark MLlib вимагає, щоб усі вхідні дані були зібрані в один стовпець, значення якого є вектором. Цей клас є абстракцією для векторів у сенсі лінійної алгебри і містить лише числа. Для більшості цілей вони працюють як простий масив подвійних значень (чисел з плаваючою комою). Наразі ми не звернемо увагу на цей момент і повернемося до нього пізніше.

Клас VectorAssembler може виконати таку роботу(Лістинг 3.4):

```
import org.apache.spark.ml.feature.VectorAssembler

val inputCols = trainData.columns.filter(_ != "Cover_Type")
val assembler = new VectorAssembler().
  setInputCols(inputCols).
  setOutputCol("featureVector")

val assembledTrainData = assembler.transform(trainData)
assembledTrainData.select("featureVector").show(truncate = false)

...
+---------------------------------------------------------------- ...
|featureVector                                                   ...
+---------------------------------------------------------------- ...
|(54,[0,1,2,3,4,5,6,7,8,9,13,15],[1863.0,37.0,17.0,120.0,18.0,90.0,2 ...
|(54,[0,1,2,5,6,7,8,9,13,18],[1874.0,18.0,14.0,90.0,208.0,209.0,135. ...

|(54,[0,1,2,3,4,5,6,7,8,9,13,18],[1879.0,28.0,19.0,30.0,12.0,95.0,20 ...
...
```

Лістинг 3.4. Клас VectorAssembler

Його ключовими параметрами є стовпці, які потрібно об’єднати у вектор ознак, і назва нового стовпця, що містить вектор ознак. Тут усі стовпці — окрім — цілі, звичайно — включені як вхідні функції. Отриманий DataFrame має новий стовпець «featureVector», як показано.

VectorAssembler є прикладом Transformer в рамках поточного API «Pipelines» Spark MLlib. Він перетворює інший DataFrame у DataFrame і компонується з іншими перетвореннями в конвеєр. Далі в цій главі ці перетворення будуть об’єднані в справжній конвеєр. Тут перетворення просто викликається безпосередньо, чого достатньо для побудови моделі класифікатора першого дерева рішень(Лістинг 3.5).

```
import org.apache.spark.ml.classification.DecisionTreeClassifier
import scala.util.Random

val classifier = new DecisionTreeClassifier().
  setSeed(Random.nextLong()). ❶
  setLabelCol("Cover_Type").
  setFeaturesCol("featureVector").
  setPredictionCol("prediction")

val model = classifier.fit(assembledTrainData)
println(model.toDebugString)

...
DecisionTreeClassificationModel (uid=dtc_29cfe1281b30) of depth 5 with 63 nodes
  If (feature 0 <= 3039.0)
   If (feature 0 <= 2555.0)
    If (feature 10 <= 0.0)
     If (feature 0 <= 2453.0)
      If (feature 3 <= 0.0)
       Predict: 4.0
      Else (feature 3 > 0.0)
       Predict: 3.0
       ...
```

❶ Use random seed

Лістинг 3.5. Клас DesisionTreeClassifier

Знову ж таки, основна конфігурація класифікатора складається з імен стовпців: стовпця, що містить вхідні вектори ознак, і стовпця, що містить цільове значення для прогнозування. Оскільки модель пізніше буде використовуватися для прогнозування нових значень цілі, їй присвоюється ім’я стовпця для зберігання передбачень.

Так само, як зламаний годинник є правильним двічі на день, випадкове вгадування класифікації для кожного прикладу також іноді дасть правильну відповідь. Ми могли б побудувати такий випадковий «класифікатор», вибравши випадковий клас пропорційно його поширеності в навчальному наборі. Наприклад, якби 30% навчального набору були покриттям типу 1, то випадковий класифікатор вгадав би «1» у 33% випадків. Кожна класифікація буде правильною пропорційно її поширеності в тестовому наборі. Якби 40% тестового набору були покриттям типу 1, то вгадати «1» було б правильним у 40% випадків. Тоді тип обкладинки 1 буде вгаданий правильно 30% x 40% = 12% часу і внесе 12% у загальну точність. Тому ми можемо оцінити точність шляхом підсумовування цих добутків ймовірностей(Лістинг 3.6).

```
import org.apache.spark.sql.DataFrame

def classProbabilities(data: DataFrame): Array[Double] = {
  val total = data.count()

  data.groupBy("Cover_Type").count(). ❶
    orderBy("Cover_Type"). ❷
    select("count").as[Double]. ❸
    map(_ / total).
    collect()
}

val trainPriorProbabilities = classProbabilities(trainData)
val testPriorProbabilities = classProbabilities(testData)
trainPriorProbabilities.zip(testPriorProbabilities).map { ❹
  case (trainProb, cvProb) => trainProb * cvProb
}.sum

...
0.3771270477245849
```

Лістинг 3.6. Сума добутків

1. Підрахунок за категорією.
2. Впорядкування сум по категоріям
3. Передача в набір даних
4. Сума добутків

Випадкове вгадування досягає 37% точності. Але цей результат був досягнутий за допомогою гіперпараметрів за замовчуванням. Ми можемо зробити ще краще, дослідивши, що вони насправді означають для процесу побудови дерева.

3.3. Гіперпараметри дерева рішень

Гіперпараметри, що керують тим, як вибираються рішення дерева, будуть досить різними: максимальна глибина, максимальні бункери, міра домішки та мінімальний приріст інформації.

Максимальна глибина просто обмежує кількість рівнів у дереві рішень. Це максимальна кількість ланцюжкових рішень, яку прийме класифікатор для класифікації прикладу. Це корисно обмежити, щоб уникнути переобладнання навчальних даних, як показано на малюнку раніше в прикладі зоомагазину.

Алгоритм дерева рішень відповідає за створення потенційних правил прийняття рішень на кожному рівні, наприклад, вага >= 100 або вага >= 500 рішень у прикладі зоомагазину. Рішення завжди мають однакову форму: для числових ознак рішення мають вигляд ознака >= значення ; а для категоріальних ознак вони мають форму ознака в (значення1, значення2, …) . Отже, набір правил прийняття рішень, які потрібно спробувати, насправді є набором значень, які потрібно підключити до правила прийняття рішення. У реалізації Spark MLlib вони називаються «кошками». Більша кількість ящиків вимагає більше часу обробки, але може призвести до пошуку більш оптимального правила прийняття рішення.

Інтуїтивно, гарне правило значуще розрізняло б приклади за значенням цільової категорії. Наприклад, правило, яке розділяє набір даних Covtype на приклади лише з категоріями 1–3, з одного боку, і 4–7 з іншого, було б чудовим, оскільки воно чітко відокремлює одні категорії від інших. Правило, яке призвело до приблизно однакового поєднання всіх категорій, які є в усьому наборі даних, не здається корисним. Дотримання будь-якої гілки такого рішення призводить до приблизно однакового розподілу можливих цільових значень, і тому насправді не просувається до впевненої класифікації.

Іншими словами, хороші правила ділять цільові значення навчальних даних на відносно однорідні або «чисті» підмножини. Вибір найкращого правила означає мінімізацію домішок двох підмножин, які воно викликає. Існує дві зазвичай використовувані міри домішки: домішка Джіні та ентропія.

Домішка Джині безпосередньо пов'язана з точністю класифікатора випадкових здогадок. У підмножині це ймовірність того, що випадково вибрана класифікація випадково вибраного прикладу (обидва відповідно до розподілу класів у підмножині) є неправильною. Це сума добутків пропорцій класів, але з самими собою і віднімається від 1. Якщо підмножина має N класів, а p і — це частка прикладів класу і, то її домішка Джині задається в рівнянні домішок Джіні:

$$I_G(p) = 1 - \sum_{i=1}^{N} p_i^2$$

Якщо підмножина містить лише один клас, це значення дорівнює 0, оскільки воно повністю «чисте». Якщо в підмножині є N класів, це значення більше 0 і є найбільшим, коли класи зустрічаються однакову кількість разів — максимально нечисті.

Ентропія — ще одна міра домішки, запозичена з теорії інформації. Його природу складніше пояснити, але він фіксує, скільки невизначеності має збір цільових значень у підмножині щодо прогнозів для даних, які потрапляють до цієї підмножини. Підмножина, що містить один клас, говорить про те, що результат для підмножини є повністю певним і має 0 ентропії — невизначеності. З іншого боку, підмножина, що містить один із можливих класів, свідчить про велику невизначеність щодо прогнозів для цієї підмножини, оскільки дані спостерігалися з усіма видами цільових значень. Це має високу ентропію. Отже, низька ентропія, як і низька домішка Джині, є хорошою річчю. Ентропія визначається рівнянням ентропії:

$$I_E(p) = \sum_{i=1}^{N} p_i \log\left(\frac{1}{p}\right) = -\sum_{i=1}^{N} p_i \log\left(p_i\right)$$

Одна чи інша міра може бути кращою метрикою для вибору правил прийняття рішень у даному наборі даних. Вони в чомусь схожі. Обидва включають середньозважене значення: суму значень, зважених p і . За замовчуванням у реалізації Spark є домішка Джіні.

Нарешті, мінімальний інформаційний приріст — це гіперпараметр, який встановлює мінімальний приріст інформації або зменшення домішки для правил прийняття рішень. Правила, які недостатньо покращують домішки підмножин, відхиляються. Як і менша максимальна глибина, це може допомогти моделі протистояти переобладнанню, оскільки рішення, які ледве допомагають розділити навчальні дані, насправді можуть взагалі не корисно розділити майбутні дані.

3.4. Налаштування дерев рішень

З огляду на дані не стає очевидним, яка міра домішок веде до кращої точності, або яка максимальна глибина чи кількість бункерів є достатньою, але не надмірною. На щастя, легко дозволити Spark спробувати декілька комбінацій цих значень і повідомити результати.

По-перше, необхідно налаштувати конвеєр, який інкапсулює ті самі два кроки, які описано вище. Створення VectorAssembler і DecisionTreeClassifier і об'єднання цих двох Transformer разом призводить до єдиного об'єкта Pipeline, який представляє ці дві операції разом як одну операцію(Лістинг 3.7).

```
import org.apache.spark.ml.Pipeline

val inputCols = trainData.columns.filter(_ != "Cover_Type")
val assembler = new VectorAssembler().
  setInputCols(inputCols).
  setOutputCol("featureVector")

val classifier = new DecisionTreeClassifier().
  setSeed(Random.nextLong()).
  setLabelCol("Cover_Type").
  setFeaturesCol("featureVector").
  setPredictionCol("prediction")

val pipeline = new Pipeline().setStages(Array(assembler, classifier))
```

Лістинг 3.7. Об'єднання в єдиний pipeline

Тепер ми також можемо визначити комбінації гіперпараметрів, які слід перевірити, використовуючи вбудовану підтримку Spark ML API, ParamGridBuilder. Також настав час визначити метрику оцінки, яка буде використовуватися для вибору «найкращих» гіперпараметрів, і це знову MulticlassClassificationEvaluator тут(Лістинг 3.8).

```
import org.apache.spark.ml.tuning.ParamGridBuilder

val paramGrid = new ParamGridBuilder().
  addGrid(classifier.impurity, Seq("gini", "entropy")).
  addGrid(classifier.maxDepth, Seq(1, 20)).
  addGrid(classifier.maxBins, Seq(40, 300)).
  addGrid(classifier.minInfoGain, Seq(0.0, 0.05)).
  build()

val multiclassEval = new MulticlassClassificationEvaluator().
  setLabelCol("Cover_Type").
  setPredictionCol("prediction").
  setMetricName("accuracy")
```

Лістинг 3.8. Клас MulticlassClassificationEvaluator

Це означає, що модель буде побудована та оцінена для двох значень чотирьох гіперпараметрів. Це 16 моделей. Вони будуть оцінені за багатокласовою точністю. Нарешті, TrainVa lidationSplit об’єднує ці компоненти — конвеєр, який створює моделі, метрики оцінки моделі та гіперпараметри, які можна спробувати — і можна виконувати оцінку навчальних даних(Лістинг 3.9).

```
import org.apache.spark.ml.tuning.TrainValidationSplit

val validator = new TrainValidationSplit().
  setSeed(Random.nextLong()).
  setEstimator(pipeline).
  setEvaluator(multiclassEval).
  setEstimatorParamMaps(paramGrid).
  setTrainRatio(0.9)

val validatorModel = validator.fit(trainData)
```

Лістинг 3.9. Оцінка навчальних даних

Це займе хвилини або більше, залежно від вашого обладнання, оскільки воно створює та оцінює багато моделей. Зверніть увагу, що параметр коефіцієнта поїздів встановлено на 0,9. Це означає, що навчальні дані фактично додатково поділяються TrainValidationSplit на 90%/10% підмножини. Перший використовується для навчання кожної моделі. Решта 10% введених даних зберігаються як набір перехресної перевірки для оцінки моделі. Якщо він уже надає деякі дані для оцінки, то чому ми виділили 10% вихідних даних як тестовий набір?

Якщо метою набору CV було оцінити параметри, які відповідають навчальному набору, то метою тестового набору є оцінка гіперпараметрів, які «підходять» до набору CV. Тобто тестовий набір забезпечує неупереджену оцінку точності кінцевої обраної моделі та її гіперпараметрів.

Скажімо, що найкраща модель, обрана цим процесом, має 90% точність у наборі CV. Здається розумним очікувати, що в майбутніх даних він буде демонструвати 90% точність. Однак у тому, як створюються ці моделі, є елемент випадковості. Випадково ця модель і оцінка могли вийти незвично вдалими. Топ-модель і результат оцінки могли виграти, якщо пощастило, тому її точність, ймовірно, буде трохи оптимістичною. Іншими словами, гіперпараметри також можуть переповнюватися.

Щоб дійсно оцінити, наскільки добре ця найкраща модель буде працювати на майбутніх прикладах, нам потрібно оцінити її на прикладах, які не використовувалися для її навчання. Але ми також повинні уникати прикладів у наборі резюме, які були використані для його оцінки. Тому була проведена третя підмножина, тестова.

Результат валідатора містить найкращу знайдену модель. Це само по собі представляє найкращий загальний конвеєр, який він знайшов, тому що ми надали екземпляр конвеєра для запуску. Щоб запитати параметри, вибрані DecisionTreeClassifier, необхідно вручну витягти DecisionTreeClassificationModel з результуючого PipelineModel, який є останнім етапом конвеєра(Лістинг 3.10).

```
import org.apache.spark.ml.PipelineModel

val bestModel = validatorModel.bestModel
bestModel.asInstanceOf[PipelineModel].stages.last.extractParamMap

...
{
        dtc_9136220619b4-cacheNodeIds: false,
        dtc_9136220619b4-checkpointInterval: 10,
        dtc_9136220619b4-featuresCol: featureVector,
        dtc_9136220619b4-impurity: entropy,
        dtc_9136220619b4-labelCol: Cover_Type,
        dtc_9136220619b4-maxBins: 40,
        dtc_9136220619b4-maxDepth: 20,
        dtc_9136220619b4-maxMemoryInMB: 256,
        dtc_9136220619b4-minInfoGain: 0.0,
        dtc_9136220619b4-minInstancesPerNode: 1,
        dtc_9136220619b4-predictionCol: prediction,
        dtc_9136220619b4-probabilityCol: probability,
        dtc_9136220619b4-rawPredictionCol: rawPrediction,
        dtc_9136220619b4-seed: 159147643
}
```

Лістинг 3.10. Ручне формування даних

Це містить багато інформації про підібрану модель, але також говорить нам, що «ентропія», найкраще працювала як міра домішки і що максимальна глибина 20 була не дивно кращою за 1. Може бути дивним, що найкращою моделлю була вміщається лише з 40 ящиками, але це, ймовірно, ознака того, що 40 було «багато», а не «краще», ніж 300. Нарешті, жодне мінімальне приріст інформації не було кращим, ніж невеликий мінімум, що може означати, що модель більш схильна до недостатності ніж переобладнання.

Ви можете задатися питанням, чи можна побачити точність, яку досягла кожна з моделей для кожної комбінації гіперпараметрів. Гіперпараметри, а також оцінки надаються за допомогою getEstimatorParamMaps і validationMetrics відповідно. Їх можна об’єднати, щоб відобразити всі комбінації параметрів, відсортовані за значенням показника(Лістинг 3.11).

```
val validatorModel = validator.fit(trainData)

val paramsAndMetrics = validatorModel.validationMetrics.
  zip(validatorModel.getEstimatorParamMaps).sortBy(-_._1)

paramsAndMetrics.foreach { case (metric, params) =>
    println(metric)
    println(params)
    println()
}

...
0.9138483377774368
{
        dtc_3e3b8bb692d1-impurity: entropy,
        dtc_3e3b8bb692d1-maxBins: 40,
        dtc_3e3b8bb692d1-maxDepth: 20,
        dtc_3e3b8bb692d1-minInfoGain: 0.0
}

0.9122369506416774
{
        dtc_3e3b8bb692d1-impurity: entropy,
        dtc_3e3b8bb692d1-maxBins: 300,
        dtc_3e3b8bb692d1-maxDepth: 20,
        dtc_3e3b8bb692d1-minInfoGain: 0.0
}
...
```

Лістинг 3.11. Об’єднання всіх комбінацій параметрів

Якої точності досягла ця модель на наборі CV? І нарешті, якої точності досягає модель на тестовому наборі(Лістинг 3.12)?

```
validatorModel.validationMetrics.max
multiclassEval.evaluate(bestModel.transform(testData)) ❶

...
0.9138483377774368
0.9139978718291971
```

❶ bestModel is a complete pipeline.

Лістинг 3.12. Точність моделі

Обидва результати становлять близько 91%. Буває, що для початку оцінка з набору резюме була непогана. Насправді, рідко коли тестовий набір показує інший

результат.

Це цікавий момент, коли варто повернутися до питання переобладнання. Як обговорювалося раніше, можна побудувати дерево рішень настільки глибоко і складно, щоб воно дуже добре або ідеально відповідало наведеним навчальним прикладам, але не вдавалося узагальнювати на інші приклади, оскільки воно надто точно відповідало особливостям і шуму навчальних даних. Це проблема, спільна для більшості алгоритмів машинного навчання, а не лише для дерев рішень.

Коли дерево рішень має перевірку, воно буде демонструвати високу точність при запуску на тих самих навчальних даних, до яких воно відповідає моделі, але низьку точність на інших прикладах. Тут точність остаточної моделі становила близько 91% на інших, нових прикладах. Точність можна так само легко оцінити за тими ж даними, на яких навчалася модель, trainData. Це дає точність близько 95%.

Різниця невелика, але свідчить про те, що дерево рішень певною мірою переповнило навчальні дані. Кращим вибором може бути менша максимальна глибина.

3.5. Переглянуто категорійні характеристики

Поки що в прикладах коду всі вхідні функції неявно розглядалися як числові (хоча «Cover_Type», незважаючи на те, що він закодований як числовий, насправді правильно розглядався як категоріальне значення.) Це не зовсім неправильно, тому що категоріальні ознаки тут одноразово закодовані як кілька двійкових значень 0/1. Виявляється, що розглядати ці окремі об’єкти як числові можна, оскільки будь-яке правило прийняття рішення щодо «числових» об’єктів вибирає пороги від 0 до 1, і всі вони еквівалентні, оскільки всі значення дорівнюють 0 або 1.

Звичайно, таке кодування змушує алгоритм дерева рішень розглядати значення основних категорійних ознак окремо. Оскільки такі ознаки, як тип ґрунту, розбиваються на багато ознак, а дерева рішень розглядають особливості

окремо, важче зв’язати інформацію про пов’язані типи ґрунту.

Наприклад, дев'ять різних типів ґрунту насправді є частиною сімейства Leighcan, і вони можуть бути пов'язані таким чином, що дерево рішень може використовувати. Якби тип ґрунту був закодований як одна категоріальна ознака з 40 значеннями ґрунту, то дерево могло б виражати правила на кшталт «якщо тип ґрунту є одним із дев’яти типів сімейства Лейтонів». Однак, якщо дерево закодовано як 40 об’єктів, то для того, щоб зробити те ж саме, дереву потрібно було б вивчити послідовність з дев’яти рішень щодо типу ґрунту. Ця виразність може призвести до кращих рішень та більш ефективних дерев.

Однак наявність 40 числових ознак, що представляють одну категоріальну ознаку з 40 значеннями, збільшує використання пам’яті та сповільнює роботу.

Як щодо скасування одноразового кодування? Це замінить, наприклад, чотири стовпці, що кодують тип дикої природи, одним стовпцем, який кодує тип дикої природи у вигляді числа від 0 до 3, наприклад «Cover_Type»(Лістинг 3.13).

```
import org.apache.spark.sql.functions._

def unencodeOneHot(data: DataFrame): DataFrame = {
  val wildernessCols = (0 until 4).map(i => s"Wilderness_Area_$i").toArray

  val wildernessAssembler = new VectorAssembler().
    setInputCols(wildernessCols).
    setOutputCol("wilderness")

  val unhotUDF = udf((vec: Vector) => vec.toArray.indexOf(1.0).toDouble) ❶

  val withWilderness = wildernessAssembler.transform(data).
    drop(wildernessCols:_*). ❷
    withColumn("wilderness", unhotUDF($"wilderness")) ❸

  val soilCols = (0 until 40).map(i => s"Soil_Type_$i").toArray

  val soilAssembler = new VectorAssembler().
    setInputCols(soilCols).
    setOutputCol("soil")

  soilAssembler.transform(withWilderness).
    drop(soilCols:_*).
    withColumn("soil", unhotUDF($"soil"))
}
```

❶ Note UDF definition

❷ Drop one-hot columns; no longer needed

❸ Overwrite column with numeric one of same name

Лістинг 3.13. Перепис даних з одинаковими назвами

Тут VectorAssembler розгорнуто, щоб об’єднати 4 та 40 стовпців дикої природи та типу ґрунту в два стовпці Vector. Усі значення в цих векторах дорівнюють 0, за винятком одного розташування, яке має 1. Для цього не існує простої функції DataFrame, тому ми повинні визначити нашу власну UDF, яку можна використовувати для роботи зі стовпцями. Це перетворює ці два нові стовпці на числа саме того типу, який нам потрібен.

Звідси майже той самий процес, що й вище, можна використовувати для налаштування гіперпараметрів моделі дерева рішень, побудованої на цих даних, а також для вибору й оцінки найкращої моделі. Однак є одна важлива відмінність. У двох нових числових стовпцях немає нічого, що вказує на те, що вони насправді є кодуванням категорійних значень. Розглядати їх як числа було б неправильно, оскільки їх упорядкування безглузде. Однак це вдалось би мовчки; інформація в цих функціях була б майже втрачена.

Внутрішньо Spark MLlib може зберігати додаткові метадані про кожен стовпець. Деталі цих даних, як правило, приховані від абонента, але містять таку інформацію, як кодування стовпця категоріального значення та скільки різних значень воно приймає. Щоб додати ці метадані, необхідно помістити дані через VectorIn dexer. Його робота полягає в тому, щоб перетворити вхідні дані в належним чином позначені стовпці категорій. Хоча ми вже зробили значну частину роботи, щоб перетворити категоріальні функції на значення, індексовані 0, VectorIndexer подбає про метадані.

Нам потрібно додати цей етап до конвеєра(Дістинг 3.14).

```
import org.apache.spark.ml.feature.VectorIndexer

val inputCols = unencTrainData.columns.filter(_ != "Cover_Type")
val assembler = new VectorAssembler().
  setInputCols(inputCols).
  setOutputCol("featureVector")

val indexer = new VectorIndexer().
  setMaxCategories(40). ❶
  setInputCol("featureVector").
  setOutputCol("indexedVector")

val classifier = new DecisionTreeClassifier().
  setSeed(Random.nextLong()).
  setLabelCol("Cover_Type").
  setFeaturesCol("indexedVector").
  setPredictionCol("prediction")

val pipeline = new Pipeline().setStages(Array(assembler, indexer, classifier))
```

❶ >= 40 because soil has 40 values

Лістинг 3.14. Обробка метаданих

Підхід передбачає, що навчальна множина містить усі можливі значення кожної з категорійних ознак принаймні один раз. Тобто він працює правильно, лише якщо всі 4 значення ґрунту та всі 40 значень дикої природи з'являються у навчальному наборі, щоб усі можливі значення отримали відображення. Тут це дійсно так, але може не бути для невеликих навчальних наборів даних, у яких деякі мітки з'являються дуже рідко. У таких випадках може знадобитися вручну створити та додати VectorIndexerModel з повним відображенням значень, наданим вручну.

Крім того, процес такий же, як і раніше. Ви повинні виявити, що він вибрав аналогічну найкращу модель, але точність тестового набору становить близько 93%. Розглядаючи категоріальні ознаки як фактичні категоріальні ознаки, класифікатор підвищив свою точність майже на 2%.

1.10. Висновки до розділу

У даному розділі було розглянуто пзадачу машинного навчання і, зокрема, розвязання задачі прогнозування поширення рослинного покриву. Було створено та протестовано декілька прикладів розвязки поставленої задачі, та було вибрано

результат з найбільш правдивими прогнозованими даними, також детально були розглянуті переваги і недоліки кожного вирішення поставленого завдання. Були розглянуті методи створення коду, описана ідея кожного і виконаний огляд найновітніших розробок у цій сфері. Також, було розглянуто методи оцінки якості роботи, такі як точність, та інші.

РОЗДІЛ 4
ОХОРОНА ПРАЦІ ТА БЕЗПЕКА В НАДЗВИЧАЙНИХ СИТУАЦІЯХ

4.1. Охорона праці

Тема кваліфікаційної роботи магістра пов’язана із дослідженням методів і засобів тестування програмного забезпечення комп’ютерних систем з використанням алгоритмів машинного навчання. Такі роботи передбачають використання комп’ютерної техніки на етапах формування пояснювальної записки, налаштування засобів автоматизації процесу управління тестуванням з використанням алгоритмів машинного навчання. Тому, при виконанні робіт з тестування програмних складових комп’ютерних систем необхідно враховувати вимоги з охорони праці при експлуатації комп’ютерної техніки.

В Україні діють ряд законів, нормативних документів та актів, які регулюють процеси забезпечення та управління охороною праці у різних галузях народного господарства. До них належать: Конституція України, Закони України "Про охорону праці", "Про охорону здоров'я", "Про пожежну безпеку", "Про забезпечення санітарного та епідемічного благополуччя населення", "Про загальнообов'язкове державне соціальне страхування від нещасного випадку на виробництві та професійного захворювання, які спричинили втрату працездатності", Кодекс законів про працю України (КЗпП).

Однією з основних вимог до приміщень, де робочі місця обладнані комп’ютерною технікою і планується використання програмного комплексу для забезпечення процесу тестування ПЗ, є вимоги щодо площі, яка відводиться на один ПК. При проектуванні автоматизованих робочих місць тестувальників програмного забезпечення необхідно дотримуватись вимог щодо розміщення комп’ютерів. На один ПК передбачено площу 6 $м^2$ та об’єм 20 $м^3$.

Однак робота з комп’ютером включає різні завдання, які об'єднуються такими загальними чинниками, як те, що робота проводиться в сидячому положенні і вимагає уважного, неперервного та іноді тривалого спостереження.

Перше правило, якого варто дотримуватись тестувальникам програмного забезпечення стосується правильного облаштування робочого столу. При цьому слід передбачити наступні його параметри: фіксована висота – 720 мм, забезпечення необхідного простору для рук по висоті, ширині і глибині, в області сидіння не повинно бути шухляд.

Друге правило визначає облаштування робочого стільця: можливість регулювання висоти стільця, забезпечення обертання конструкції стільця. У приміщеннях з ПК, на яких планується виконання задач з тестування програмних складових комп'ютерних систем, яскравість знаків і яскравість фону дисплею повинна бути спроектована таким чином, щоб не було великої відмінності з яскравістю навколишнього середовища, але знаки повинні чітко розпізнаватися на відстані читання. Характеристики освітлення, зокрема у приміщеннях, де експлуатується ПК, повинні відповідати ДБН В.2.5-28-2006 "Природнє і штучне освітлення". Основні вимоги даного нормативного документу стосуються забезпечення наступних вимог:

- освітлення з лівої сторони;
- рівномірне освітлення всього робочого простору;
- комп'ютерна техніка встановлюється у місцях, віддалених від вікон;
- встановлення непрямого штучного освітлення;
- світло, що поступає через вікна, «пом'якшують» за допомогою штор;
- робоче місце організовується так, щоб напрям погляду був паралельним фронту вікон.

Ще одне правило, якого слід дотримуватись тестувальникам програмного забезпечення, передбачає оптимальний метод роботи, що полягає у передбачені зміни завдань і навантажень, дотримання перерви в роботі: 5 хвилин через 1 годину роботи біля дисплея або 10 хвилин після 2-х годин роботи біля дисплея. Вимоги цього правила регламентовані нормативним документом «Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин».

При створенні сприятливих умов для підвищення продуктивності і

зменшення напруги значну роль грають чинники, що характеризують стан навколишнього середовища: мікроклімат приміщення, рівень шуму і освітлення.

Рекомендована величина відносної вологості, яка повинна бути забезпечена у приміщеннях з експлуатації програмного комплексу поведінкового тестування програмних складових комп'ютерних систем, повинна відповідати НПАОП 0.00-7.15-18 і становити 65 – 70%. При цьому робоче місце повинно бути добре вентильованим.

У даний час з погляду шумового навантаження досягнуто значного прогресу. Рівень шуму в приміщеннях (приблизно 40 Дб) не перевищує допустимого рівня, незалежно від кількості використовуваного обладнання. Для приміщень, в яких експлуатується програмний комплекс підтримки запропонованих методів поведінкового тестування, потрібно забезпечити виконання вимог пожежної безпеки, які визначені Правилами пожежної безпеки в Україні, НПАОП 0.00-7.15-18 «Вимоги щодо безпеки та захисту здоров'я працівників під час роботи з екранними пристроями» [24].

Будівлі і ті їх частини, в яких розташовуються ПК можуть належати до II ступеня вогнестійкості. Над та під приміщеннями, де розташовуються ПК, а також у суміжних з ними приміщеннях не дозволяється розташування приміщень категорій А і Б за вибухопожежною небезпекою. Приміщення категорії В повинні бути відділеними від приміщень з ПК протипожежними стінами.

Таким, чином при дослідженні методів і засобів поведінкового тестування програмних складових комп'ютерних систем, встановлено, що найбільш повним нормативним документом щодо охорони праці користувачів ПК, до яких належать тестувальники програмного забезпечення, є НПАОП 0.00-7.15-18 «Вимоги щодо безпеки та захисту здоров'я працівників під час роботи з екранними пристроями». Дотримання вимог, які неведені у цьому документі, сприяє зниженню негативного впливу ПК, його компонентів та інших зовнішніх пристроїв на тестувальників, які проводять роботи щодо перевірки правильності функціонування програмних складових комп'ютерних систем.74

4.2. Підвищення стійкості роботи об'єктів господарської діяльності у воєнний час

На основі вивчення факторів, які впливають на стійкість роботи об'єктів господарської діяльності, та оцінки стійкості елементів і галузей виробництва проти уражаючих факторів ядерної, хімічної і біологічної зброї, стихійних лих і виробничих аварій, необхідно завчасно організувати і провести організаційні, інженерно-технічні й технологічні заходи для підвищення стійкості роботи.

Здійснення організаційних заходів передбачає завчасну підготовку всіх структур цивільного захисту, служб і формувань до надзвичайних ситуацій, в тому числі і військових дій. Вжиттям технологічних заходів підвищується стійкість роботи об'єктів шляхом змінювання технологічних процесів, режимів, можливих в умовах різних надзвичайних ситуацій.

Інженерно-технічні заходи мають забезпечити підвищену стійкість виробничих споруд, технологічних ліній, устаткування, комунікацій об'єкта до впливу уражаючих факторів під час військових дій.

При проведенні цих заходів необхідно враховувати конкретні умови об'єкта народного господарства. Проте є загальні організаційні інженерно-технічні заходи, які мають проводитись на всіх об'єктах.

Одним з найбільш важливих завдань в умовах воєнного часу і надзвичайних ситуацій є забезпечення захисту людей та їх життєдіяльності.

Для підвищення стійкості об’єктів господарювання та захисту людей необхідно:

– створити на об'єкті надійну систему оповіщення про загрози нападу противника, радіоактивне забруднення, хімічне і біологічне зараження, загрозу стихійного лиха і виробничої аварії.

– організувати розвідку і спостереження за радіоактивним забрудненням, хімічним і біологічним зараженням.

– організувати гідрометеорологічне спостереження за рівнем води, напрямком і швидкістю вітру, рухом і поширенням хмари радіоактивного

забруднення, сильнодіючих отруйних речовин і отруйних речовин.

– створити фонд захисних споруд ЦО, запасів засобів індивідуального захисту і забезпечення своєчасної видачі їх населенню.

– завчасно підготуватись до масової санітарної обробки населення і знезаражування одягу.

– організувати взаємодію з установами охорони здоров'я для медичного обслуговування населення в умовах воєнного часу.

Також в умовах воєнного часу необхідно провести підготовку до евакуації населення, розміщеного в зонах можливих руйнувань і катастрофічного затоплення. Це передбачає завчасну підготовку місць евакуації, організацію прийому евакуйованого населення на територію населених пунктів.

Окрім цього, необхідно забезпечити постачання продуктів харчування, питної води, предметів першої необхідності та провести заходи щодо морально-психологічної підготовки населення до виживання в умовах воєнного часу, забезпечити процес чіткого інформування про обстановку та правила дій і поведінки населення в надзвичайних ситуаціях воєнного часу.

Для забезпечення стійкості роботи об'єктів повинні проводитись інженерно-технічні заходи на мережах комунального господарства з метою захисту джерел тепла із заглибленням у ґрунт комунікацій. Котельні слід розміщувати в спеціальному окремо розміщеному приміщенні.

Якщо об'єкт одержує тепло з міської теплоцентралі, необхідно провести заходи для забезпечення стійкості трубопроводів і розподільних пристроїв, підведених до об'єкта.

Теплова мережа має будуватися за кільцевою системою з прокладанням труб у спеціальних каналах зі з'єднанням паралельних ділянок. Для відключення пошкоджених ділянок мають бути встановлені запірно-регулюючі засувки, вентилі та ін. Ці пристосування необхідно розміщувати в оглядових колодязях, на території, що не завалюється при руйнуванні будівель.

Система каналізації має будуватись окремо: одна для дощових, друга для промислових і господарських вод. На об'єкті має бути не менше двох виводів з

підключенням до міських каналізаційних колекторів, а також виводи і колодязі з аварійними засувками на об'єктових колекторах з інтервалом 50 м на території, що не завалюється, для аварійного скидання неочищеної води в найближчі штучні та природні заглиблення.

На деяких промислових об'єктах є системи для забезпечення технології виробництва: для подання кисню, аміаку, стиснутого повітря та інших рідких і газових реактивів. Для цих систем розробляють заходи для попередження виникнення вторинних факторів зброї, стихійних лих та виробничих аварій і катастроф.

Створення резерву енергетичних потужностей за рахунок автономних пересувних електростанцій, а також місцевих джерел електроенергії. Підготовка автономних електростанцій до роботи за спеціальним режимом (графіком) для забезпечення технологічних процесів виробництва, для яких неможливі тривалі перерви в електропостачанні.

З метою попередження аварій на електричних мережах необхідно установити автоматичну систему відключення при виникненні перенапруги. Повітряні лінії електропостачання замінити на підземно-кабельні.

Створення необхідних запасів (резервів) паливно-мастильних матеріалів та інших видів палива й організація їх безпечного зберігання.

Щоб не допустити зупинки підприємства через дефіцит палива, необхідно підготуватись для роботи на різних видах палива: нафта, вугілля, газ.

Для підвищення стійкості забезпечення водою слід провести такі заходи. Необхідно створити основні і резервні джерела водопостачання. Як резервне джерело краще мати артезіанську свердловину, яку необхідно підключити до системи водопостачання. Крім того, воду можна брати з близько розміщеної природної водойми або спорудити штучну водойму чи резервуари з обладнанням пристроїв для збору і перекачування води.

Всі ділянки водопостачання повинні бути заглиблені в ґрунт з обладнанням пожежних гідрантів і пристроїв для відключення пошкоджених ділянок. Локальні мережі водопостачання окремих великих підприємств варто з'єднати із

загальноміською системою водопостачання в єдине кільце.

Підвищенню стійкості забезпечення водою сприяє подавання води безпосередньо в мережу поза водонапірними баштами, спорудження обвідних ліній для подання води поза пошкодженими спорудами.

Завчасне вжиття заходів захисту вододжерел, водопровідних споруд, свердловин і шахтних колодязів від забруднення радіоактивними речовинами, зараження хімічними і біологічними засобами.

Підготовка меліоративних, гідротехнічних та іригаційних споруд і систем до експлуатації в надзвичайних умовах.

Для забезпечення виробництва продукції необхідні електроенергія, паливо, мастила, засоби захисту рослин, міндобрива, профілактичні й лікувальні препарати ветеринарної медицини, запасні частини, сировина та інші матеріально-технічні засоби. Забезпечення об'єктів цими ресурсами дасть можливість випускати необхідну продукцію в надзвичайних умовах мирного і воєнного часу. Тому повинні проводитись такі заходи, які б забезпечили стійкість постачання і сприяли підвищенню захисту мережі електро-, водо-, газопостачання, транспортних комунікацій і джерел постачання всім необхідним для забезпечення функціонування галузей сільського господарства в надзвичайних умовах.

З метою попередження аварій на електричних мережах необхідно встановити автоматичну систему відключення перенапруги. Повітряні лінії електропостачання слід замінити на підземно-кабельні.

Газ використовується як паливо і на хімічних підприємствах у технологічному процесі. Для безперебійного забезпечення газом, газові мережі необхідно підводити до об'єкта з двох напрямків, які мають бути з'єднані в єдине кільце з обладнанням для можливого дистанційного автоматичного управління й у разі необхідності відключення пошкоджених ділянок.

На великих підприємствах необхідно мати підземні ємності із закачаним резервним газом.

На підприємствах, де використовується пара, необхідно захистити джерела

його постачання, заглибити в ґрунт комунікації паропостачання і встановити запірні пристосування.

Запас резервних матеріалів необхідно розраховувати на такі строки роботи підприємства, за які можливе відновлення регулярного постачання.

Передбачити, на випадок перебоїв в постачанні підприємствами-суміжниками, створення місцевих матеріалів, сировини для виготовлення комплектуючих виробів і інструментів силами свого підприємства. Для підвищення стійкості та забезпечення збереження (відновлення) будівель і споруд в умовах воєнного часу необхідно:

– провести оцінку можливих ступенів руйнування будівель і споруд господарства населеного пункту, визначити обсяг невідкладних ремонтних робіт, потреби в будівельних матеріалах.

– створити і підготувати спеціальні формування для ремонтно-відновних, будівельних та інших робіт на об'єкті.

– розробити комплекс протипожежних заходів, які виключали б можливість виникнення масових пожеж.

Для забезпечення надійності системи управління і зв'язку потрібно організувати захищений пункт управління, забезпечити його засобами зв'язку, які б дали можливість швидко доводити сигнали ЦЗ до всіх виробничих підрозділів і населення у місцях проживання. При цьому необхідно здійснити планування збору даних про обстановку, передачу команд і розпоряджень в умовах впливу на об'єкт уражаючих факторів. Для підвищення стійкості системи управління і зв’язку в умовах воєнного часу необхідно організувати використання радіозасобів, засобів телефонного зв'язку, а також забезпечити зв’язок із колонами евакуйованого населення, що перебувають у дорозі, і відповідальними особами, які супроводжують їх під час евакуації, забезпечити дублювання ліній і каналів зв'язку.

4.3. Висновки до розділу

Отже, підвищення стійкості роботи об'єктів господарської діяльності у

воєнний час можна забезпечити шляхом організації і проведення сукупності заходів, які включають організаційні, інженерно-технічні й технологічні заходи. Надзвичайно важливим є планування та організація захисту комплексів критичної інфраструктури, зокрема водопостачання, водовідведення, газопостачання та зв’язку.

ВИСНОВКИ

Основні наукові та практичні результати полягають в наступному.

1. Проведено аналіз важливих понять, принципів та послідовності виконання процесів, які використовуються при проектуванні комп'ютерних систем, зокрема, термінологічні особливості у процесі імплементації програмного забезпечення для обробки великих даних та по прогнозуванню поширення рослинного покрову, що дає змогу розуміти та в подальшому визначати шляхи розсаджування дерев з допомогою методів машинного навчання та прогнозування їх поширення на конкретній території для підвищення ефективності стадій життєвого циклу.

2. Проаналізовано методи та моделі великих даних, що дало можливість врахувати їх особливості та вибрати найбільш підходячий інструмент для вирішення поставленої задачі.

3. На основі аналізу та класифікації методів та видів великих даних, визначено необхідність імплнментації методу машинного навчання щодо розсадження рослинного покриву згідно з вимогами до вхідних даних, що в перспективі дозволить використовувати їх при плануванні зелених посаджень на заданій території.

4. Було розглянуто перелік основних умов процесу обробки великих даних, та обґрунтування вибору основного фреймворка.

5. Було надано за і проти вибору того чи іншого фреймворка, вступ до основної моделі. Поверхневий огляд даних, та наведення змінних, які впливатимуть на кінцевий результат.

6. Розглянуто постановку задачі машинного навчання і, зокрема, бінарної класифікації. Було вивчено поширені моделі класифікаціїї, як байєсівський класифікатор, логістична регресія, одношаровий персептрон, метод k найближчих сусідів, метод опорних векторів, дерева рішень та інші, детально були розглянуті переваги і недоліки кожного.

7. Були розглянуті методи створення ансамблів класифікаторів, описана ідея кожного і виконаний огляд найновітніших розробок у цій сфері.

8. Досягнуто точності виведених даних у 93% в середньому без прив'язки до конкретного набору даних, на мові програмування Scala та Apache Spark.

ПЕРЕЛІК ПОСИЛАНЬ

1. Apache Spark офіційна сторінка: https://spark.apache.org/downloads.html

2. Повний набір даних: https://archive.ics.uci.edu/ml/machine-learning-databases/covtype/

3. Sandy Ryza, Uri Laserson, Sean Owen, & Josh Wills “Advanced Analytics with Spark, second edition”

4. Фоурино Р. Электронное качество данных: скрытая перспектива очистки данных / Фоурино Р.: http://www.iso.ru/print/rus/document5820.phtml

5. Воронцов К.В. Машинное обучение: курс лекций / Воронцов К.В. — Режим доступа: http://www.machinelearning.ru/wiki/index.php?title=%D0%9C%D0%B0%D1%88%D0

%B8%D0%BD%D0%BD%D0%BE%D0%B5_%D0%BE%D0%B1%D1%83%D1%87%D0%B5%D0%BD%D0%B8%D0%B5_%28%D0%BA%D1%83%D1%80%D1%81_%D0%BB%D0%B5%D0%BA%D1%86%D0%B8%D0%B9%2C_%D0%9A.%D0%92.%D0%92%D0%BE%D1%80%D0%BE%D0%BD%D1%86%D0%BE%D0%B2%29

6. Воронцов К.В. Лекции по логическим алгоритмам классификации / Воронцов К.В.: http://www.ccas.ru/voron/download/LogicAlgs.pdf

7. Курс "Машинное обучение" на ФКН ВШЭ : https://github.com/esokolov/ml-course-hse

8. Rodrıguez J.J. Rotation Forest: A New Classifier Ensemble Method / Rodrıguez J.J., Kuncheva L.I., Alonso C.J. // IEEE Transactions on Pattern Analysis and Machine Intelligence. ВШЭ : https://ieeexplore.ieee.org/abstract/document/1677518/

9. Дьяконов А. Введение в анализ данных и машинное обучение / Дьяконов А.: https://alexanderdyakonov.files.wordpress.com/2017/06/book_boosting_pdf.pdf

10. Chen T. XGBoost: A Scalable Tree Boosting System/ Chen T., Guestrin C.: https://arxiv.org/abs/1603.02754

11. Tumer K.A Error Correlation and Error Reduction in Ensemble Classiers / Tumer K.A.: https://www.tandfonline.com/doi/abs/10.1080/095400996116839

12. Hand D.J. Measuring classifier performance: a coherent alternative to the area under the ROC curve / Hand D.J. – Springer Science+Business Media, LLC 2009: https://link.springer.com/content/pdf/10.1007%2Fs10994-009-5119-5.pdf

13. Dean Wampler, Alex Payne Programming Scala, 2nd edition

14. Alven Alexander Scala Cookbook

15. Hastie, Trevor. Tibshirani, Robert. Friedman, Jerome. The Elements of Statistical Learning: Data Mining, Inference, and Prediction. Springer, New York, NY, 2009.

16. Elfwing, Stefan; Uchibe, Eiji; Doya, Kenji (2017-11-01). "Sigmoid-Weighted Linear Units for Neural Network Function Approximation in Reinforcement Learning".

17. Noel, Mathew Mithra; L, Arunkumar; Trivedi, Advait; Dutta, Praneet (2021-09-04). "Growing Cosine Unit: A Novel Oscillatory Activation Function That

Can Speedup Training and Reduce Parameters in Convolutional Neural Networks".

18. Tolles, Juliana; Meurer, William J (2016). "Logistic Regression Relating Patient Characteristics to Outcomes". *JAMA*. **316** (5): 533–4.

19. M. Strano; B.M. Colosimo (2006). "Logistic regression analysis for experimental determination of forming limit diagrams". *International Journal of Machine Tools and Manufacture*. **46** (6): 673–682.

20. Murphy, Kevin P. (2012). *Machine Learning – A Probabilistic Perspective*. The MIT Press. pp. 245pp.

21. Rodríguez, G. (2007). *Lecture Notes on Generalized Linear Models*. pp. Chapter 3, page 45

22. Piryonesi, S. Madeh; El-Diraby, Tamer E. (2020-03-01). "Data Analytics in Asset Management: Cost-Effective Prediction of the Pavement Condition Index". *Journal of Infrastructure Systems*

23. Piryonesi, S. Madeh; El-Diraby, Tamer E. (2021-02-01). "Using Machine Learning to Examine Impact of Type of Performance Indicator on Flexible Pavement Deterioration Modeling". *Journal of Infrastructure Systems*.

24. Lalchand, Vidhi (2020). "Extracting more from boosted decision trees: A high energy physics case study".

25. McCallum, Andrew. "Graphical Models, Lecture2: Bayesian Network Representation" (PDF). Retrieved 22 October 2019

26. Piryonesi, S. Madeh; El-Diraby, Tamer E. (2020-06-01). "Role of Data Analytics in Infrastructure Asset Management: Overcoming Data Size and Quality Problems". *Journal of Transportation Engineering, Part B: Pavements*.

27. Metsis, Vangelis; Androutsopoulos, Ion; Paliouras, Georgios (2006). *Spam filtering with Naive Bayes—which Naive Bayes?*. Third conference on email and anti-spam (CEAS).

Додаток А

Тези конференцій

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ТЕРНОПІЛЬСЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
ІМЕНІ ІВАНА ПУЛЮЯ

М А Т Е Р І А Л И

ІХ НАУКОВО-ТЕХНІЧНОЇ КОНФЕРЕНЦІЇ

«ІНФОРМАЦІЙНІ МОДЕЛІ, СИСТЕМИ ТА ТЕХНОЛОГІЇ»

8–9 грудня 2021 року

ТЕРНОПІЛЬ
2021

СЕКЦІЯ 3. КОМП'ЮТЕРНІ СИСТЕМИ ТА МЕРЕЖІ

УДК 004.4

А.М. Луцків канд. техн. наук, доцент, Г.А. Абоах, Р.К. Рувімбо, В.М. Соболь
(Тернопільський національний технічний університет імені Івана Пулюя, Україна)

ПОБУДОВА ЗАХИЩЕНИХ ХМАРНИХ СЕРЕДОВИЩ ОПРАЦЮВАННЯ ДАНИХ

UDC 004.4

A.M. Lutskiv PhD, Assoc. Prof., H.A. Aboah, R.K. Ruwimbo, V.M. Sobol

DEVELOPMENT OF SECURED CLOUD DATA PROCESSING ENVIRONMENTS

Nowadays the main part of data processing is held in different cloud services. These cloud services could be based on private, public or hybrid clouds. Usually government organizations trying to use private clouds which caused by national laws and government regulations. Secure cloud should guarantee CIA triad: confidentiality, integrity and accessibility.

To build really secured cloud solution cloud engineers should use different meanings and tools on all of the phases of design, development, deployment and usage. All details should be taken into account:

- the physical and geographical location of the data center;
- servers' hardware peculiarities;
- computer networks in all layers of TCP/IP stack;
- operating systems and all system software components and utilities;
- applied and server software;
- underlying services of third-party organizations;

and also the most important human factor.

To not forget about all of these details Cloud Security Alliance designed document[1] which could be treated as a check-list for security engineers and contains the list of all mentioned factors in details. Engineers should follow best practices and recommendations while developing and support cloud solution. These practices usually written by different government and non-profit international alliances. Very important to underline that these guidelines are dictated by practice and not by some business interests.

Important to understand that one of the highest is a risk which could be caused by human factor. This factor risk prevention can be achieved by different technical and legal measures. Also we have to understand that these measures decrease comfortable usability of the system: complicated passwords, two-factor authentication, different limitation etc. While developing software parts important to acknowledge with the Open Web Application Security Project guidelines and best practices. OWASP also suggests different tools to audit security of designed software system.

In a few last years arose the problem with the human privacy. Multiple software giants collect, transfer and share private users' information collected by their applications. By this reason different countries has their own regulations and laws to prevent sharing and using private information. Especially important is General Data Protection Regulation which used in Europe and should be followed by all private enterprises and government organizations in the region. GDPR restrictions should be taken into account by Engineers too.

References.

1. Cloud Controls Matrix v3.0.1. Release Date: 08/03/2019. URL: https://cloudsecurityalliance.org/artifacts/cloud-controls-matrix-v3-0-1/.
2. Open Web Application Security Project. URL: https://owasp.org/.

УДК 004.4

А.М. Луцків канд. техн. наук, доцент, Г.А. Абоах, Р.К. Рувімбо, В.М. Соболь
(Тернопільський національний технічний університет імені Івана Пулюя, Україна)

РОЗВ'ЯЗАННЯ ЗАДАЧ МАШИННОГО НАВЧАННЯ У СЕРЕДОВИЩАХ ІЗ РОЗПОДІЛЕНОЮ ПАМ'ЯТТЮ

UDC 004.4

A.M. Lutskiv PhD, Assoc. Prof., H.A. Aboah, R.K. Ruwimbo, V.M. Sobol

RESOLVING MACHINE LEARNING TASKS IN DISTRIBUTED MEMORY ENVIRONMENT

Resolving of analytical tasks such as building recommendation or predictive analysis systems involves using of Machine Learning (ML) methods. Usually these methods are implemented in software libraries. The most well-tested ML-methods are implemented in Python libraries. Unfortunately these libraries and solutions can be used only in shared memory environments which are horizontally scale limited. Apache Spark is a distributed memory parallel data processing system which is well horizontally scaled. Other feature of Apache Spark is ability to run locally on a single computer. This feature allows to develop and test ML approaches without having an access to large cluster and also embed Spark into non-distributed applications.

Spark does not offer large amount of ML methods but the most commonly used are implemented in its ml and mllib packages. Among these methods there are different methods to extract features of different types, to classify features, to calculate regression. For ML problems solving in distributed environment Spark offers distributed data types (e.g. DistributedMatrix).

Spark allows to combine resolving of Big Data engineering and Data Science tasks by building pipelines. Spark can be deployed into different environments: physical server or in Kubernetes cloud as a cluster or can be executed as local application on local machine.

Spark has Python API, so data scientist can build solutions by combining Spark Distributed approach with ML-libraries from other tools.

Problems which arise in these solutions related to incompatibility of data formats: usually with Python Pandas library and in Spark RDD, DataFrames and DataSets are used. This autumn Apache Spark developers released version 3.2 [1] which resolves this issue by embedded support of Koalas library (Koalas, the Spark implementation of the popular Pandas library).

Other peculiarities of new Spark version are:

- using Hadoop 3.3.1 libraries (especially performance improvement in AWS S3 object storage support);
- SQL queries using Adaptive Query Execution which improves performance;
- DataSource V2 optimizations related to aggregate pushdown (improvements of operations count, sum, min, max and average);
- Spark Streaming improvement based on using of RocksDB;

and also a few Kubernetes improvements.

Unfortunately not all public cloud providers offer latest version of Apache Spark, so can be used only self-deployed version. For research purposes decided to use helm chart packaged by Bitnami[2] which will be deployed into private Kubernetes cluster.

References.

1. Apache Spark 3.2.0 Documentation. URL: https://spark.apache.org/docs/latest/
2. Apache Spark packaged by Bitnami. URL: https://bitnami.com/stack/spark/helm

Додаток Б

Повний код програми

```
package com.cloudera.datascience.rdf

import org.apache.spark.ml.{PipelineModel, Pipeline}
import org.apache.spark.ml.classification.{DecisionTreeClassifier,
  RandomForestClassifier, RandomForestClassificationModel}
import org.apache.spark.ml.evaluation.MulticlassClassificationEvaluator
import org.apache.spark.ml.feature.{VectorAssembler, VectorIndexer}
import org.apache.spark.ml.linalg.Vector
import org.apache.spark.ml.tuning.{ParamGridBuilder, TrainValidationSplit}
import org.apache.spark.mllib.evaluation.MulticlassMetrics
import org.apache.spark.sql.{DataFrame, SparkSession}
import org.apache.spark.sql.functions._
import scala.util.Random

object RunRDF {

  def main(args: Array[String]): Unit = {
        val spark = SparkSession.builder().getOrCreate()
        import spark.implicits._

        val dataWithoutHeader = spark.read.
        option("inferSchema", true).
        option("header", false).
        csv("hdfs:///user/ds/covtype.data")

        val colNames = Seq(
        "Elevation", "Aspect", "Slope",
        "Horizontal_Distance_To_Hydrology", "Vertical_Distance_To_Hydrology",
        "Horizontal_Distance_To_Roadways",
        "Hillshade_9am", "Hillshade_Noon", "Hillshade_3pm",
        "Horizontal_Distance_To_Fire_Points"
        ) ++ (
        (0 until 4).map(i => s"Wilderness_Area_$i")
        ) ++ (
        (0 until 40).map(i => s"Soil_Type_$i")
        ) ++ Seq("Cover_Type")

        val data = dataWithoutHeader.toDF(colNames:_*).
        withColumn("Cover_Type", $"Cover_Type".cast("double"))

        data.show()
        data.head
```

```
        // Split into 90% train (+ CV), 10% test
        val Array(trainData, testData) = data.randomSplit(Array(0.9, 0.1))
        trainData.cache()
        testData.cache()

        val runRDF = new RunRDF(spark)

        runRDF.simpleDecisionTree(trainData, testData)
        runRDF.randomClassifier(trainData, testData)
        runRDF.evaluate(trainData, testData)
        runRDF.evaluateCategorical(trainData, testData)
        runRDF.evaluateForest(trainData, testData)

        trainData.unpersist()
        testData.unpersist()
  }

}

class RunRDF(private val spark: SparkSession) {

  import spark.implicits._

  def simpleDecisionTree(trainData: DataFrame, testData: DataFrame): Unit = {

        val inputCols = trainData.columns.filter(_ != "Cover_Type")
        val assembler = new VectorAssembler().
        setInputCols(inputCols).
        setOutputCol("featureVector")

        val assembledTrainData = assembler.transform(trainData)
        assembledTrainData.select("featureVector").show(truncate = false)

        val classifier = new DecisionTreeClassifier().
        setSeed(Random.nextLong()).
        setLabelCol("Cover_Type").
        setFeaturesCol("featureVector").
        setPredictionCol("prediction")

        val model = classifier.fit(assembledTrainData)
        println(model.toDebugString)

        model.featureImportances.toArray.zip(inputCols).
        sorted.reverse.foreach(println)
```

```
        val predictions = model.transform(assembledTrainData)

        predictions.select("Cover_Type", "prediction", "probability").
        show(truncate = false)

        val evaluator = new MulticlassClassificationEvaluator().
        setLabelCol("Cover_Type").
        setPredictionCol("prediction")

        val accuracy = evaluator.setMetricName("accuracy").evaluate(predictions)
        val f1 = evaluator.setMetricName("f1").evaluate(predictions)
        println(accuracy)
        println(f1)

        val predictionRDD = predictions.
        select("prediction", "Cover_Type").
        as[(Double,Double)].rdd
        val multiclassMetrics = new MulticlassMetrics(predictionRDD)
        println(multiclassMetrics.confusionMatrix)

        val confusionMatrix = predictions.
        groupBy("Cover_Type").
        pivot("prediction", (1 to 7)).
        count().
        na.fill(0.0).
        orderBy("Cover_Type")

        confusionMatrix.show()
}

def classProbabilities(data: DataFrame): Array[Double] = {
        val total = data.count()
        data.groupBy("Cover_Type").count().
        orderBy("Cover_Type").
        select("count").as[Double].
        map(_ / total).
        collect()
}

def randomClassifier(trainData: DataFrame, testData: DataFrame): Unit = {
        val trainPriorProbabilities = classProbabilities(trainData)
        val testPriorProbabilities = classProbabilities(testData)
        val accuracy = trainPriorProbabilities.zip(testPriorProbabilities).map {
        case (trainProb, cvProb) => trainProb * cvProb
        }.sum
        println(accuracy)
```

```
}

def evaluate(trainData: DataFrame, testData: DataFrame): Unit = {

        val inputCols = trainData.columns.filter(_ != "Cover_Type")
        val assembler = new VectorAssembler().
        setInputCols(inputCols).
        setOutputCol("featureVector")

        val classifier = new DecisionTreeClassifier().
        setSeed(Random.nextLong()).
        setLabelCol("Cover_Type").
        setFeaturesCol("featureVector").
        setPredictionCol("prediction")

        val pipeline = new Pipeline().setStages(Array(assembler, classifier))

        val paramGrid = new ParamGridBuilder().
        addGrid(classifier.impurity, Seq("gini", "entropy")).
        addGrid(classifier.maxDepth, Seq(1, 20)).
        addGrid(classifier.maxBins, Seq(40, 300)).
        addGrid(classifier.minInfoGain, Seq(0.0, 0.05)).
        build()

        val multiclassEval = new MulticlassClassificationEvaluator().
        setLabelCol("Cover_Type").
        setPredictionCol("prediction").
        setMetricName("accuracy")

        val validator = new TrainValidationSplit().
        setSeed(Random.nextLong()).
        setEstimator(pipeline).
        setEvaluator(multiclassEval).
        setEstimatorParamMaps(paramGrid).
        setTrainRatio(0.9)

        val validatorModel = validator.fit(trainData)

        val paramsAndMetrics = validatorModel.validationMetrics.
        zip(validatorModel.getEstimatorParamMaps).sortBy(-_._1)

        paramsAndMetrics.foreach { case (metric, params) =>
        println(metric)
        println(params)
        println()
        }
```

```
        val bestModel = validatorModel.bestModel

        println(bestModel.asInstanceOf[PipelineModel].stages.last.extractParamMap)

        println(validatorModel.validationMetrics.max)

        val testAccuracy = multiclassEval.evaluate(bestModel.transform(testData))
        println(testAccuracy)

        val trainAccuracy = multiclassEval.evaluate(bestModel.transform(trainData))
        println(trainAccuracy)
}

def unencodeOneHot(data: DataFrame): DataFrame = {
        val wildernessCols = (0 until 4).map(i => s"Wilderness_Area_$i").toArray

        val wildernessAssembler = new VectorAssembler().
        setInputCols(wildernessCols).
        setOutputCol("wilderness")

        val unhotUDF = udf((vec: Vector) => vec.toArray.indexOf(1.0).toDouble)

        val withWilderness = wildernessAssembler.transform(data).
        drop(wildernessCols:_*).
        withColumn("wilderness", unhotUDF($"wilderness"))

        val soilCols = (0 until 40).map(i => s"Soil_Type_$i").toArray

        val soilAssembler = new VectorAssembler().
        setInputCols(soilCols).
        setOutputCol("soil")

        soilAssembler.transform(withWilderness).
        drop(soilCols:_*).
        withColumn("soil", unhotUDF($"soil"))
}

def evaluateCategorical(trainData: DataFrame, testData: DataFrame): Unit = {
        val unencTrainData = unencodeOneHot(trainData)
        val unencTestData = unencodeOneHot(testData)

        val inputCols = unencTrainData.columns.filter(_ != "Cover_Type")
        val assembler = new VectorAssembler().
        setInputCols(inputCols).
        setOutputCol("featureVector")
```

```
    val indexer = new VectorIndexer().
    setMaxCategories(40).
    setInputCol("featureVector").
    setOutputCol("indexedVector")

    val classifier = new DecisionTreeClassifier().
    setSeed(Random.nextLong()).
    setLabelCol("Cover_Type").
    setFeaturesCol("indexedVector").
    setPredictionCol("prediction")

    val pipeline = new Pipeline().setStages(Array(assembler, indexer, classifier))

    val paramGrid = new ParamGridBuilder().
    addGrid(classifier.impurity, Seq("gini", "entropy")).
    addGrid(classifier.maxDepth, Seq(1, 20)).
    addGrid(classifier.maxBins, Seq(40, 300)).
    addGrid(classifier.minInfoGain, Seq(0.0, 0.05)).
    build()

    val multiclassEval = new MulticlassClassificationEvaluator().
    setLabelCol("Cover_Type").
    setPredictionCol("prediction").
    setMetricName("accuracy")

    val validator = new TrainValidationSplit().
    setSeed(Random.nextLong()).
    setEstimator(pipeline).
    setEvaluator(multiclassEval).
    setEstimatorParamMaps(paramGrid).
    setTrainRatio(0.9)

    val validatorModel = validator.fit(unencTrainData)

    val bestModel = validatorModel.bestModel

    println(bestModel.asInstanceOf[PipelineModel].stages.last.extractParamMap)

    val testAccuracy = multiclassEval.evaluate(bestModel.transform(unencTestData))
    println(testAccuracy)
}

def evaluateForest(trainData: DataFrame, testData: DataFrame): Unit = {
    val unencTrainData = unencodeOneHot(trainData)
    val unencTestData = unencodeOneHot(testData)
```

```
val inputCols = unencTrainData.columns.filter(_ != "Cover_Type")
val assembler = new VectorAssembler().
setInputCols(inputCols).
setOutputCol("featureVector")

val indexer = new VectorIndexer().
setMaxCategories(40).
setInputCol("featureVector").
setOutputCol("indexedVector")

val classifier = new RandomForestClassifier().
setSeed(Random.nextLong()).
setLabelCol("Cover_Type").
setFeaturesCol("indexedVector").
setPredictionCol("prediction").
setImpurity("entropy").
setMaxDepth(20).
setMaxBins(300)

val pipeline = new Pipeline().setStages(Array(assembler, indexer, classifier))

val paramGrid = new ParamGridBuilder().
addGrid(classifier.minInfoGain, Seq(0.0, 0.05)).
addGrid(classifier.numTrees, Seq(1, 10)).
build()

val multiclassEval = new MulticlassClassificationEvaluator().
setLabelCol("Cover_Type").
setPredictionCol("prediction").
setMetricName("accuracy")

val validator = new TrainValidationSplit().
setSeed(Random.nextLong()).
setEstimator(pipeline).
setEvaluator(multiclassEval).
setEstimatorParamMaps(paramGrid).
setTrainRatio(0.9)

val validatorModel = validator.fit(unencTrainData)

val bestModel = validatorModel.bestModel

val forestModel = bestModel.asInstanceOf[PipelineModel].
stages.last.asInstanceOf[RandomForestClassificationModel]
```

```
println(forestModel.extractParamMap)
println(forestModel.getNumTrees)
forestModel.featureImportances.toArray.zip(inputCols).
sorted.reverse.foreach(println)

val testAccuracy = multiclassEval.evaluate(bestModel.transform(unencTestData))
println(testAccuracy)

bestModel.transform(unencTestData.drop("Cover_Type")).select("prediction").show()
  }

}
```