math_2020_081

Міністерство освіти і науки України

Чернівецький національний університет

імені Юрія Федьковича

Факультет математики та інформатики
(повна назва інституту/факультету)

Кафедра математичного моделювання
(повна назва кафедри)

Побудова нейронної мережі на основі моделі випадкового лісу

Дипломна робота

Рівень вищої освіти - другий (магістерський)

Виконав:
студент 6 курсу, групи 607
спеціальності 124 – Системний аналіз_
(назва спеціальності)
__Прохніцький Владислав Васильович__
(прізвище, ім'я та по-батькові)
Керівник к.ф.-м.н., доцент Лукашів Т. О.
(науковий ступінь, вчене звання, прізвище та ініціали)

До захисту допущено:

Протокол засідання кафедри № 7

від „ 15 ” грудня 2020 р.

зав. кафедри _________ проф. Черевко І.М.

Чернівці – 2020

Зміст

Вступ 4
Розділ 1. Підготовча частина 5
1.1 Постановка задачі 5
1.2 Огляд літератури...6
1.3. Підбір технологій та середовища розробки 10
Розділ 2. Теоретична частина 15
2.1 Метод дерев рішень...15
2.2 Метод випадкового лісу 25
Розділ 3. Практична частина 31
3.1 Аналізатор сайту погоди 31
3.2 Алгоритм дерев рішень та випадкового лісу. Інформаційна веб сторінка. 36
Висновки 43
Література та використані джерела...45
Додатки 46

Анотація

Дипломна робота присвячена застосуванню алгоритму випадкового лісу для побудови короткострокового прогнозу погоди. Розроблено аналізатор для отримання історичних даних для регіону, інформаційну веб-сторінку для відображення прогнозу погоди, який будується на основі моделі випадкового лісу, що реалізована засобами мови програмування Python.

Вступ

Нам пощастило проживати у надзвичайно цікавий етап життя людства – з року в рік технології все більше і швидше розвиваються. Повсякдене життя людей все більше спрощується із використанням даних досягнень. В результаті, кожен з нас в будь-який момент може дізнатись майже будь-яку інформацію, робити покупки не виходячи з дому, спілкуватись із друзями онлайн, тощо. Що важливіше, технології використовуються на виробництвах – певні процеси автоматизовуються і людині лише потрібно спостерігати як за неї всю роботу виконує роботизований механізм. Значна частина даних можливостей стала реальною завдяки машинному навчанню та нейронним мережам. Даний напрям являється дуже популярним та має багато прихильників. Завдяки йому людство досягає високої ефективності при затраті мінімальної кількості енергії. Не потрібно боятись цього – потрібно познайомитись та зрозуміти, тоді кожен зможе використовувати надані технології з максимальною користю.

Актуальність теми роботи випливає із наступного:

- актуальний напрям розвитку – машинне навчання та нейронні мережі, які активно використовуються у сучасному житті.
- Перспектива отримати програму, які шляхом навчання буде досить точно інформувати людину про погоду.
- Вивчення та застосування на практиці методу випадкового лісу.

Прикладною задачею для нейронної мережі є прогнозування максимальної температури та стану погоди на наступний день.

Розділ 1. Підготовча частина

1.1 Постановка задачі

На даний момент існує дуже багато різних бібліотек із реалізованими функціями навчання та прогнозування. У рамках даного проекту не було використано жодну із бібліотек машинного навчання, тому метод реалізовано вручну. Лише при правильному розумінні алгоритму та різних методів навчання можна реалізувати програму, яка буде видавати результати із високою точністю. Ще однією запорукою успіху являється здобуття досить великої кількості вхідних даних для повноцінного навчання системи. У роботі використано дані прогнозу погоди починаючи з 2003 року. Знайти архівні дані прогнозу погоди для України є доволі важко, тому для того, щоб зібрати усі дані був написаний аналізатор для сайту погоди, який зчитав інформацію із сайту та зберіг його у текстовий файл, який легко піддається маніпуляції.

Основною ідеєю є надання користувачу програми, яка, на основі попередніх даних, зможе передбачити максимальну температуру та стан погоди на наступний день. Тренування даних та прогнозування буде відбуватись автоматично – користувачу лише потрібно відкрити інформаційну сторінку та дочекатись результату. Програма може бути оновлена кожного дня, отримавши новий, дійсний прогноз, який буде використаний для наступних передбачень. Нові дані будуть отримані за допомогою аналізатора.

Отже, мета проекту – це реалізація програми із графічним інтерфейсом користувача, де можна побачити прогноз максимальної температури та стан погоди на наступні дні.

1.2 Огляд літератури

Для виконання магістерської роботи, використано одну навчальну книгу і різні навчальні матеріали у мережі Інтернет.

Книга – “Machine Learning With Random Forests And Decision Trees: A Visual Guide For Beginners” автора Chris Smith являється чудовим початком для вивчення методу випадкового лісу та дерев вибору. Вона містить 18 розділів у яких викладено концепцію методів, використовуючи приклади та візуалізацію, що дуже допомагає у розумінні матеріалу. Також у книзі є кроки по створенню своєї програми машинного навчання методом випадкового лісу, використовуючи мову програмування Python та бібліотеку машинного навчання. Перелічимо та дамо короткий опис найважливіших розділів:

1. Розділ 1 – коротке пояснення трьох типів алгоритму машинного навчання:
    a. алгоритм навчання, який використовує вхідні дані, та набір передбачених результатів для них.
    b. Алгоритм, який використовує тільки вхідні дані і знаходить певну залежність між ними.
    c. Змішаний – навчання алгоритму відбувається як з даними без передбачених результатів, так і з ними.
2. Розділ 2 – пояснення багатьох термінів.
3. Розділ 5 – розгляд п’яти популярних алгоритмів для реалізації дерев вибору.
4. Розділ 7 – викладені математичні відомості, які використовуються у алгоритмах дерев рішень.

5. Розділ 8 – відповіді на найбільш поширенні питання при реалізації алгоритмів. Даний розділ особливо корисний при реалізації алгоритму вручну, без будь-яких бібліотек.
6. Розділ 9 – розгляд найбільш поширених помилок при реалізації.
7. Розділ 13 – пояснення алгоритму випадкового лісу та того, що відбувається насправді під час його виконання.
8. Розділ 16 – опис методів для визначення вагомих атрибутів вибірки даних, які найбільше впливають на передбачення.
9. Розділ 17 – відповіді на найбільш поширенні питання стосовно методу випадкового лісу.

Прочитавши книгу та використавши її для реалізації алгоритму випадкового лісу у програмі, можемо виділити такі її переваги:

- Книга надрукована у 2017 році, тому її матеріал є доволі актуальний. Використанні останні алгоритми та бібліотеки для реалізації на мові Python.
- Доволі серйозна та складна тема пояснена простими словами, що дозволяє ознайомитись із методами навіть людям, які зовсім не знайомі із машинним навчанням та нейронними мережами. Часто використовуються приклади із реального життя.
- Багато візуалізації. Теорія підкріплена також і картинками, які дуже чітко показують те, про що автор розповідає у тексті (рис. 1).

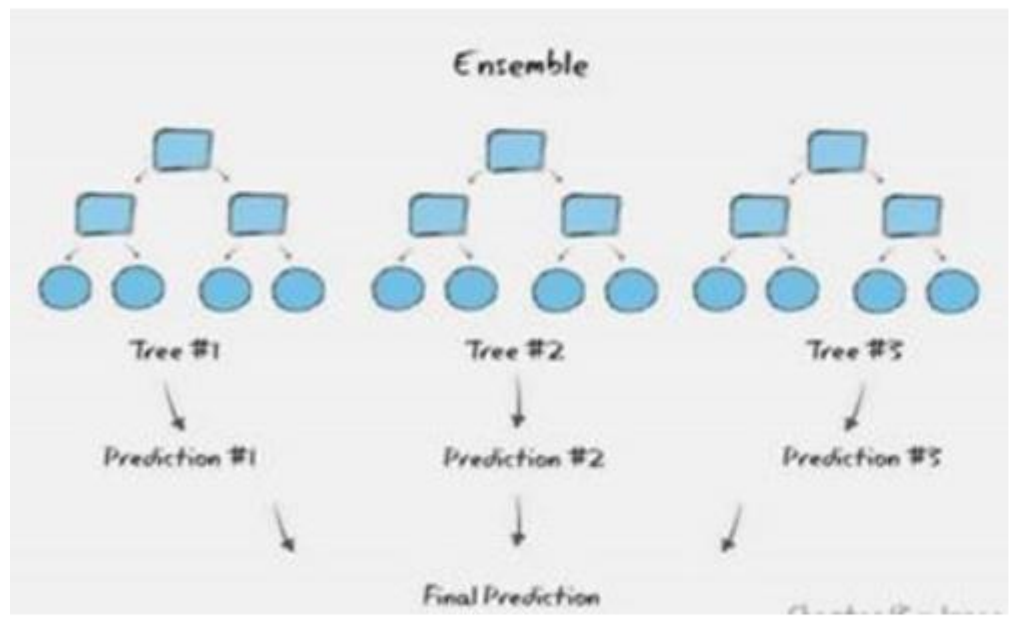

Рис. 1. Візуалізація випадкового лісу у книзі

- Програмна реалізація алгоритмів дуже корисна для швидкого старту використання прочитаного матеріалу у прикладних задачах.

Щодо недоліків, то їх, насправді, важко виділити. Книга написана змістовно і зрозуміло, це підтверджують відгуки експертів у галузі машинного навчання, які також мали нагоду ознайомитись із нею.

Після опрацювання розділів книги отримано достатню кількість знань для початку розробки методу випадкового лісу для прогнозування максимальної температури та стану погоди. Крім книги, були використані такі ресурси:

1. Відеоуроки [2];
2. Документація Python [3], бібліотеки NumPy [4] та фреймворка Flask [5] для розробки веб сторінки.

Документації допомогли більш детально ознайомитись з методами та компонентами, а відеоуроки дозволили побачити реальні приклади, які використовують дані технології та бібліотеку, провести їх аналіз та виділити важливі аспекти для роботи.

1.3. Підбір технологій та середовища розробки

Мова програмування – Python 2.7.16

Python — інтерпретована об'єктно-орієнтована мова програмування високого рівня із строгою динамічною типізацією. Розроблена в 1990 році Гвідо ван Россумом (Guido van Rossum). Структури даних високого рівня разом із динамічною семантикою та динамічним зв'язуванням роблять її привабливою для швидкої розробки програм, а також як засіб поєднання існуючих компонентів. Python підтримує модулі та пакети модулів, що сприяє модульності та повторному використанню коду. Інтерпретатор Python та стандартні бібліотеки доступні як у скомпільованій так і у вихідній формі на всіх основних платформах. В мові програмування Python підтримується кілька парадигм програмування, зокрема: об'єктно-орієнтована, процедурна, функціональна та аспектно-орієнтована.

Серед основних її переваг можна назвати такі:

- для виділення блоків слід використовувати табуляцію, а не фігурні дужки як у С подібних мов програмування.
- Спрощений синтаксис – відомо, що дана мова програмування має низький поріг розуміння.
- стандартний дистрибутив має велику кількість корисних модулів (включно з модулем для розробки графічного інтерфейсу).
- відкритий код – над мовою працюють програмісти із усього світу.

Python має ефективні структури даних високого рівня та простий, але ефективний підхід до об'єктно-орієнтованого програмування. Динамічна обробка типів, а також те, що це інтерпретована мова, роблять її ідеальною для

написання скриптів та швидкої розробки прикладних програм у багатьох галузях на більшості платформ.

Інтерпретатор мови Python і багата стандартна бібліотека можуть бути отримані із офіційного сайту Python [6], і можуть вільно розповсюджуватися. Тут також можна знайти дистрибутиви та посилання на численні модулі, програми, утиліти та додаткову документацію.

Інтерпретатор мови Python може бути розширений функціями та типами даних, розробленими на С чи С++. Python також зручна як мова розширення для прикладних програм, що потребують подальшого налагодження.

Python підтримує динамічну типізацію, тобто, тип змінної визначається лише під час виконання. З базових типів слід зазначити підтримку цілих чисел довільної довжини, комплексних чисел, робота із рядками.

Можливості та особливості реалізації об’єктно орієнтовного програмування у Python:

- Класи є одночасно об'єктами.
- Успадкування, в тому числі множинне.
- Поліморфізм (всі функції віртуальні).
- Інкапсуляція (два рівні — загальнодоступні та приховані методи і поля).
- Спеціальні методи, що керують життєвим циклом об'єкта: конструктори, деструктори, розподільники пам'яті.
- Перевантаження операторів (усіх, крім is, «.», «=» і символьних логічних).
- Властивості (імітація поля за допомогою функцій).

- Управління доступу до полів (емуляція полів і методів, частковий доступ тощо).

Програмне забезпечення (застосунок або бібліотека) на Python оформлюється у вигляді модулів, які у свою чергу можуть бути зібрані в пакунки.

Багата стандартна бібліотека є однією з привабливих сторін Python. Тут є засоби для роботи з багатьма мережевими протоколами та форматами Інтернету, наприклад, модулі для написання HTTP-серверів та клієнтів, для розбору та створення поштових повідомлень, для роботи з XML, тощо. Набір модулів для роботи з операційною системою дозволяє писати крос-платформні застосунки. Існують модулі для роботи з регулярними виразами, текстовими кодуваннями, мультимедійними форматами, криптографічними протоколами та іншим.

Бібліотека NumPy для роботи з великою кількістю даних дозволяє досягти продуктивності наукових розрахунків, порівнянної зі спеціалізованими пакетами. SciPy використовує NumPy і надає доступ до великого спектру математичних алгоритмів.

Python та переважна більшість бібліотек до нього безкоштовні й поставляються у вихідних кодах. Навіть більше, на відміну від багатьох відкритих систем, ліцензія ніяк не обмежує використання Python у комерційних розробках та не накладає ніяких зобов'язань, крім зазначення авторських прав.

Отже, Python – це мова програмування, яку можна використовувати у різних цілях і однією із таких є математичні і статистичні дослідження. Як було згадано дана мова містить у собі потужний інтерпретатор, а відкритий

вихідний код дозволяє отримувати бібліотеки для роботи із даними від людей із усього світу. Такі бібліотеки і використовують для роботи із великими об'ємами даних. Крім Python, серед програмістів, які працюють із статистикою популярним рішенням також є мова R, яка налаштована саме під роботу із статистичними даними. Було обрано саме Python оскільки, на мою думку, дана мова більш потужна, має більше можливостей і рішень серед яких підтримка web-розробки.

Середовище розробки – PyCharm 2019.3.4

PyCharm — одне із найпопулярніших інтегрованих середовищ розробки для мови програмування Python. Надає засоби для аналізу коду, графічний відладжувач, інструмент для запуску юніт-тестів і підтримує веб-розробку на Django, Flask. PyCharm розроблена чеською компанією JetBrains на основі IntelliJ IDEA. Середовище працює під операційними системами Windows, Mac OS X і Linux.

PyCharm був випущений на ринок інтегрованих середовищ розробки для Python щоб створити конкуренцію з PyDev і поширенішим середовищем розробки Komodo IDE. Бета-версія була випущена в липні 2010, версія 1.0 була випущена на три місяцями пізніше. Версія 2.0 вийшла 13 грудня 2011. Версія 3.0 випущена 24 вересня 2013.

PyCharm Community Edition, безкоштовна версія з відкритим вихідним кодом була опублікована 22 жовтня 2013. Крім цієї версії є ще версія PyCharm Professional, яка містить у собі більше можливостей у вигляді підключення різноманітних модулей, що роблять розробки ще легшою. Але це підсилення не є суттєвим і на комфортну роботу можна розраховувати і із звичайною версією середовища.

Можливості:

- Статичний аналіз коду, підсвічування синтаксису і помилок.
- Навігація серед проектів і серцевого коду: відображення файлової структури проекту, швидкий перехід між файлами, класами, методами і використанням методів.
- Редагування коду: перейменування, витяг методу, введення змінної, введення константи, тощо.
- Підтримка систем контролю версій: загальний користувацький інтерфейс для Mercurial, Git, Subversion, Perforce і CVS з підтримкою списків змін та злиття.
- Досить велика кількість додаткових модулів, які можна підключати до основного середовища.

Враховуючи вищенаведені можливості та відгуки користувачів, для розробки обрано саме це середовище.

Розділ 2. Теоретична частина

2.1 Метод дерев рішень

Дерева вибору – одні із найбільш використовуваних моделей прогнозування у машинному навчанні. Це тип моделі контрольованого навчання, тобто для тренування алгоритму на вхід ми повинні надати не тільки дані, але й очікувані результати для наданого набору. Таким чином, перевіривши велику кількість таких груп, алгоритм повинний зрозуміти зв'язок між даними та очікуваним результатом.

Якщо постаратись візуалізувати дерева рішень, то ми отримаємо саме те, що нам підказує інтуїція – певні об'єкти з'єднанні між собою лініями, як гілля дерева (рис. 2).

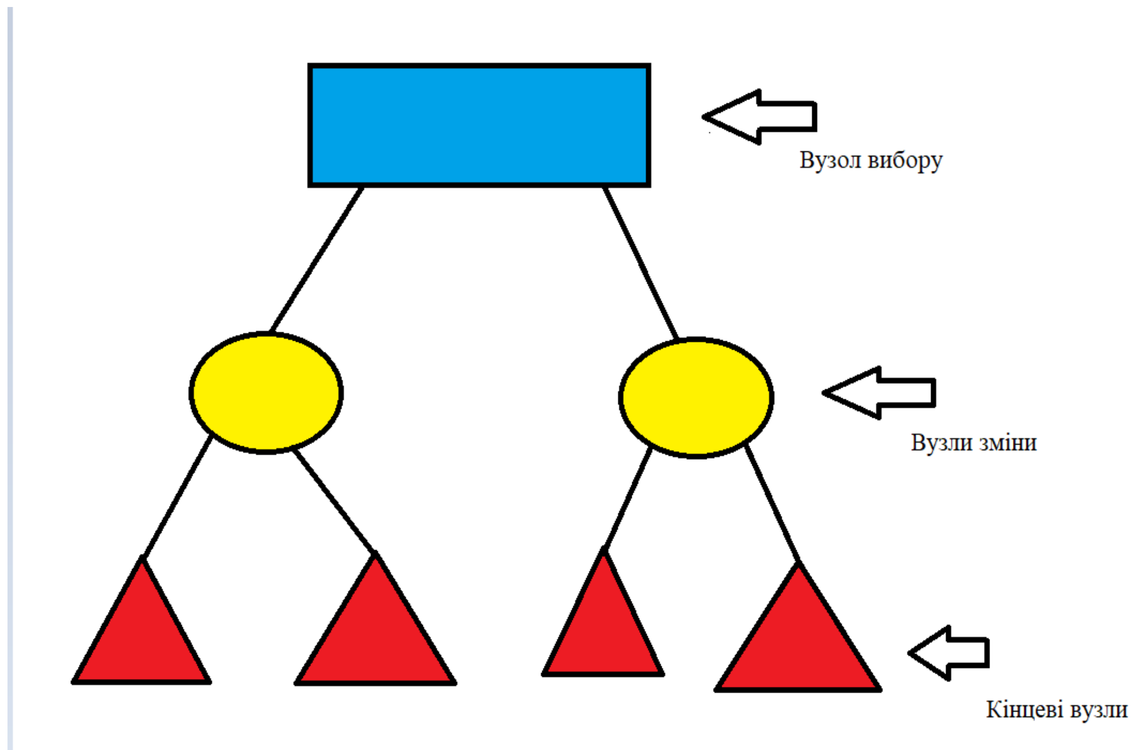

Рис. 2. Візуалізація дерева рішень

Усі об’єкти, які пов’язані лініями називаються вузлами і кожен з них виконує різну функцію у дереві:

1. Вузол вибору – будь-який вузол, який виконує вибір між двома варіантами. Перший такий вузол, який ставить наше досліджуване питання називається корінним, так як алгоритм починається із нього. Часто такі вузли малюють квадратом чи прямокутником.
2. Вузол зміни – вузол, який також робить вибір на поставлене уточнююче запитання. Наприклад, для передбачення стану погоди, питання може звучати так: «Чи було вчора сонячно?».
3. Вузол завершення – це, власне, очікувана відповідь на питання, яке ми досліджуємо за допомогою моделі.

Після того як ми розуміємо базову термінологію, можна більш детально познайомитись із алгоритмом.

Для передбачення стану погоди та максимальної температури на наступний день скористаємось простим планом:

1. Визначення початкового питання.
2. Визначення можливих варіантів та невизначеності з ними.
3. Визначення значень.
4. Визначення ймовірностей.
5. Розрахунок ваг атрибутів.
6. Визначення правильності результату кожного дерева.

У роботі розглянемо два початкові питання:

1. Яка максимальна температура буде наступного дня?
2. Який стан погоди буде наступного дня?

Для початку визначимо варіанти відповіді. Для першого питання варіантів відповіді є доволі багато – це діапазон температур, базований на значеннях, які ми можемо спостерігати попередні дні, із деяким відхиленням. Якщо сьогодні температура коливалась від +5 до +15 градусів цельсія, то взявши відхилення у 5 градусів, ми можемо очікувати максимальну температуру завтра в межах 0 та +20 градусів цельсія. Кожне значення із проміжку і буде нашим можливим рішенням, яке ми очікуємо. Щодо, невизначеності наших очікуваних значень, то це те, що ми не можемо впевнено вибрати якесь одне значення із проміжку, так як є багато природніх факторів, які можуть суттєво вплинути на значення, яке ми шукаємо. Для того, щоб мінімізувати непередбачуваність, використаємо інформацію із історії для тієї дати, яку ми намагаємось передбачити. Тобто, непередбачуваність нейтралізується шляхом запитань (вузлів зміни), які базуються на даних із попередніх років. Узагальнивши, прийдемо до такого списку:

1. Максимальна температура завтра 0 градусів (вузол вибору):
    a. Чи була максимальна температура вчора 0 градусів? (один із вузлів зміни)
    b. Чи була максимальна температура 0 градусів сьогодні ?
    c. Чи близьке значення середнього максимальної температури за останні роки на дату, яка нас цікавить, до 0?

Аналогічні вузли будуються для всіх значень із вибраного діапазону (рис. 3).

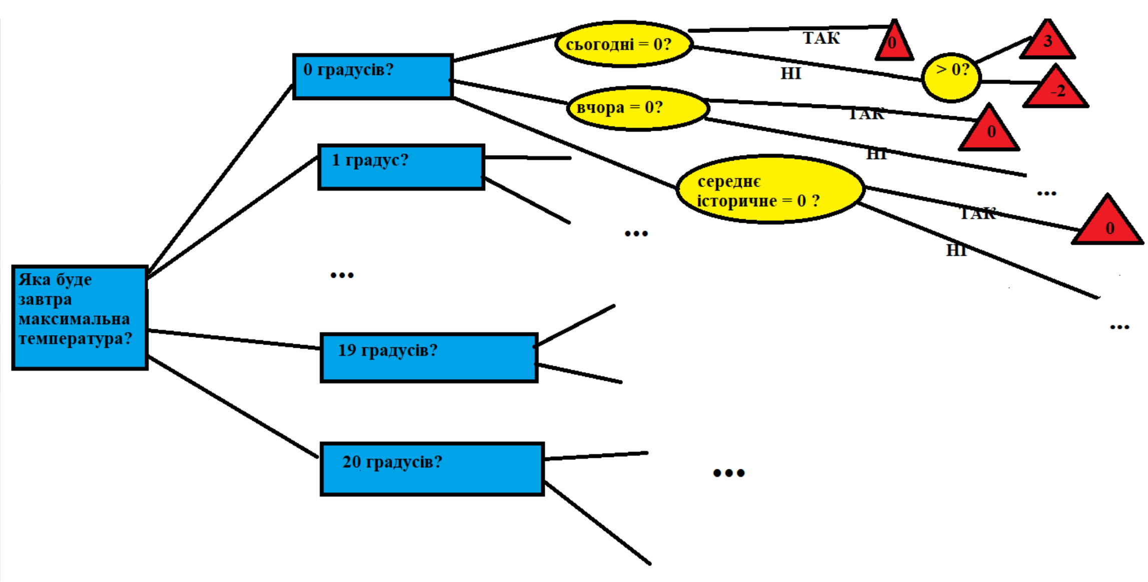

Рис. 3. Приклад вузлів для дерева рішень

Як бачимо головним атрибутом у даному прикладі взято температуру, хоча можна взяти і будь-який інший атрибут із початкової вибірки, наприклад, швидкість вітру, вологість, тощо. Також, крім вузлів вибору, вузли зміни також спрощені, оскільки вони можуть складатись із сотень питань, які допомагають уточнювати фінальне значення. У таких вузлах можуть використовуватись питання, які стосуються інших атрибутів.

Із вищенаведеного випливає те, що алгоритм дерев вибору це алгоритм, який використовує атрибути вибірки у якості вузлів зміни від найважливішого (той, що найсильніше впливає на результат – головний атрибут) до найменш важливого (того, які використовується для вузлах вибору). Для цього алгоритм бере кожен атрибут як початковий та починає будувати питання від нього. Після побудови нам потрібно визначити чи поточне дерево є «чистим». Чисте дерево називається таким, коли ми прийшли до рішення, яке було точним, тобто кожного разу ми отримували відповідь тільки «так» або тільки «ні». Таким чином, ми можемо явно побачити, що ця комбінація атрибутів та вузлів

зміни відповідає на наше питання точно. Буває ситуація, коли ми отримуємо кілька атрибутів, які мають однакову кількість відповідей, тоді виникає ситуація, яку комбінацію краще вибрати для дерева? Відповідь на це запитання допоможе знайти значення ентропії дерева вибору.

Ентропія – це величина, яка дозволяє зрозуміти як багато інформації ми втратили, використовуючи певний набір атрибутів у вузлах. Нехай ми маємо два класи випадкових дерев з різним набором атрибутів. Оскільки ціль методу передбачити якомога точніше температуру, для цього потрібно вибрати ту групу атрибутів у якій менше втраченої інформації – менше вузлів рішень із негативною відповіддю. Співвідношення втраченої інформації вимірюється за допомогою ентропії. Формула за якою ми можемо обчислити дану величину для двох класів атрибутів:

$$E(S) = -P_{(+i)} * log_2 P_{(+i)} - P_{(-i)} * log_2 P_{(-i)}. \qquad (1)$$

$P_{(+i)}, P_{(-i)}$ – відповідають за ймовірності дискретної випадкової величини одного із двох класів атрибуту. Залежна змінна із префіксом плюс означає вузол вибору на який ми отримали позитивну відповідь, відповідно із знаком мінус – ті на які ми отримали негативну відповідь. Для визначення цієї величини, використовуючи кілька класів відповідей, наприклад, проміжок температури, потрібно також враховувати частоту появи того чи іншого атрибуту у інших класах. У такому випадку формула набуває вигляду:

$$E(S) = \sum_{c \in X} P(c) * E(c), \qquad (2)$$

де, с – клас (вузол вибору) із множити класів, $p(\text{c})$ ймовірність появи атрибута у поточному класі, $E(c)$ – значення ентропії для двох інших класів. Формула

(2) використана для передбачення максимальної температури, тоді як попередня (1) – для класифікації стану погоди.

Щоб зрозуміти детальніше значення ентропії, повернемось до задачі. Для початку потрібно вибрати якийсь головний атрибут серед наданих у початковій вибірці даних.

Візьмемо головним атрибутом температуру:

- Згідно із прикладом, кількість можливих значень температури дорівнює 20 (діапазон від 0 до 20).

Очевидно, для кожного вузла вибору, що відповідає одному значенню, ми отримаємо відповідь на останній вузол зміни або «так» або «ні». Нехай, для одного такого вузла, ми отримали передбачення нашого значення ми отримали 15 відповідей «так» і, відповідно, 5 відповідей «ні». У даному випадку кількість позитивних відповідей використовується для обчислення ймовірності $P_{(+i)}$, та кількість негативних відповідей для $P_{(-i)}$. Отже, підставимо результати у формулу (1), щоб знайти ентропію:

$$E(S) = -\frac{15}{20} * log_2(\frac{15}{20}) - \frac{5}{20} * log_2(\frac{5}{20}).$$

Отримаємо

$$E(S) \approx 0.24.$$

Величина ентропії вимірюється від 0 до 1. В окремих випадках, при великій кількості класів дерев рішень, значення може виявитись більше 1, але його прийнято трактувати як 1. Трактувати значення можна дуже просто: чим менше значення, тим менше інформації втрачено, тим «чистішим» дерево є і тим краще підходить даний атрибут як головний. Візуально залежність

значення ентропії від кількості класів відповідей можна зобразити у вигляді такого графіка (рис. 4).

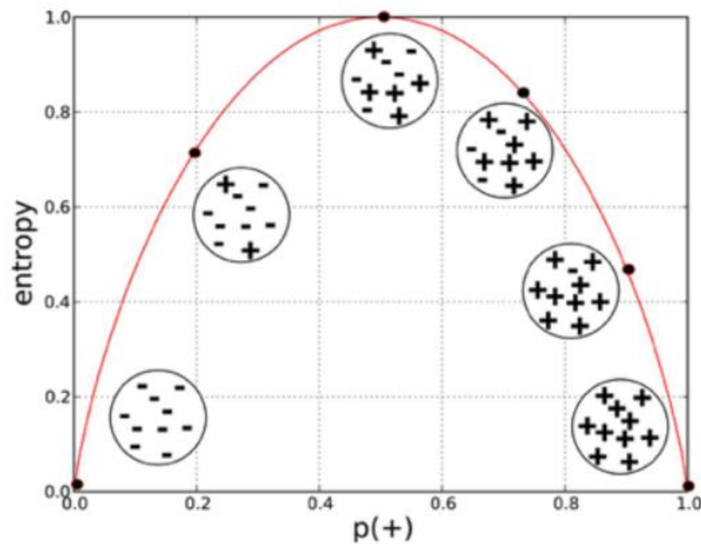

Рис. 4. Значення ентропії від кількості класів

де по осі *Ox* маємо кількість відповідей «так», а по осі *Oy* значення ентропії. Із графіку також видно, що дерево буде із великою кількістю втраченої інформацію, коли маємо однакову кількість відповідей. Таке дерево в комбінації із головним атрибутом використовувати не раціонально, оскільки точність результатів буде низька [1].

Оскільки атрибутів може бути доволі багато для одного і того ж атрибуту, нам потрібно зрозуміти який із них має найбільшу вагу у передбаченні нам потрібно використати інформаційний виграш (information gain). Дане значення дає відповідь на питання як впливає даний атрибут на загальне значення ентропії, чи збільшує воно його чи зменшує. Для цього формула використовує значення ентропії перед тим як пройти вузол вибору для певного атрибуту і після нього, потім порівнює значення. Значення

ентропії обчислене перед атрибутом називається батьківським. Отже для обчислення інформаційного виграшу, спочатку потрібно обчислити саме його, використовуючи стандартну формулу. Наступним кроком потрібно обчислити ентропію після того як пройдений вузол рішення. Таким чином отримаємо просту формулу:

$$IG(Y,X) = E(X) - E(Y|X),$$

де $E(X)$ і $E(Y)$ – це ентропія атрибутів, при умові що обидва існують, $E(Y|X)$ значення ентропії від атрибуту Y, який залежить від атрибуту X.

Ми використовуємо поділ атрибутів на групи та створюємо все нові і нові дерева, обчислюючи для них інформаційний виграш. Постає питання, коли поділ із даною частиною вхідної інформації зупинити і дерево нам підходить? Для такого виділяють кілька варіантів:

1. Шукати дерева поки кожна група є «чистою», тобто має мінімальне значення ентропії або інформаційного виграшу. Дане рішення доволі раціональне і хороше, але воно має мінус – перенавчання.
2. Зупинити пошук після того, як кожен кінцевий вузол буде мати мінімальну ентропію. Дане рішення не є найкращим оскільки сильно зменшує продуктивність та призводить до великого співвідношення помилок.
3. Визначити власне обмеження коли зупиняти пошук.

Проаналізувавши недоліки перших двох методів, реалізовано власне обмеження – зупиняти пошук після того як дерево досягне глибини не більше трьох, тобто максимальна кількість вузлів зміни не повинна перевищувати 3.

Таким чином ми побудували дерева рішень, які згодом використаємо для побудови моделі випадкового лісу. Алгоритм, який використаний для побудови має назву C4.5. Це алгоритм, який використовує «жадібний» пошук, починаючи із головного атрибуту (вузла) і без повернення назад. Для визначення рівня «чистоти» використовуються ентропія та значення інформаційного виграшу. Крім даного атрибуту існують інші, які також часто вживані, основна відмінність у них у способі визначення «чистоти»:

- ID3 – батько алгоритму C3.5 і є дуже схожий із ним, використовує ті ж значення для визначення чистоти.
- CART – використовує індекс Джині (Gini impurity index).

Наведемо переваги та недоліки алгоритму C3.5.

Переваги:

- Зрозумілі правила за рахунок яких виконується передбачення.
- Може працювати і з завданням регресії так і класифікації.
- Може працювати із незавершеними даними (навчатись змішаним методом).

Недоліки:

- Модуль може перенавчатись, якщо використовується маленька вибірка вхідних даних.
- Є нахил до створення великих дерев із великим рівним глибини.

На завершення підведемо підсумки методу дерева рішень та поговоримо про його позитивні та негативні сторони.

Переваги:

- Легко зрозуміти та візуалізувати. Метод зближений із реальним світом, як результат людям легко зрозуміти алгоритм.
- Може передбачувати як числові так і рядкові дані.
- Мінімальна підготовка даних.
- Вибір атрибутів відбувається автоматично, згідно із вибраним алгоритмом.

Щодо недоліків:

- Є тенденція до перенавчання, особливо коли глибина дерева не обмежена.
- Маленька зміна у даних впливає на всю структуру дерева.
- Досить великий час навчання та складність обчислення при зростанні кількості потрібних значень для передбачень.

Отже, як бачимо переваги досить вагомі і найбільшою із них є те, що дерева рішень дуже легко інтерпретувати і створити візуалізацію. На відміну від інших методів, наприклад, лінійної регресії, цей алгоритм є досить простим для розуміння, а отже і для реалізації.

2.2 Метод випадкового лісу

Випадковий ліс – це алгоритм машинного навчання, який реалізований шляхом об’єднання великої кількості дерев рішень. Даний алгоритм є потужним і використовується у багатьох компаніях, як інструмент для прийняття важливих рішень, зменшення ризиків і збільшення ймовірності успіху. Алгоритм може бути використаний як для завдання регресії, так і для завдання класифікації. Передбачення отримується шляхом вибору найчастішого результату з передбачень кожного із дерев рішень для завдання класифікації та обчисленням середнього значення при вирішенні задачі регресії (рис. 5). Очевидно, що результат обробки даних багатьох дерев буде набагато кращий ніж одного. Кожне дерево будується шляхом вибору випадкової групи атрибутів та даних.

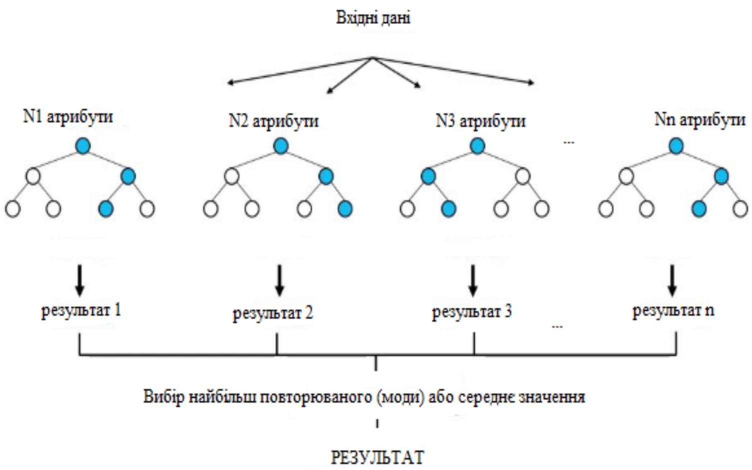

Рис. 5. Візуалізація моделі випадкового лісу

Випадковий ліс хоч і складається із дерев рішень, але дані методи вважаються окремими. Кожне дерево будується шляхом передачі дещо інших

атрибутів та даних. Узагальнивши, можна виділити 4 пункти побудови дерева для випадкового лісу:

1. Створити випадковий набір даних.
2. Вибрати випадкові атрибути.
3. Вибрати найкращий атрибут для початку поділу.
4. Почати поділ на вузли зміни.

У даному алгоритмі кроки 2 – 4 повторюються поки дерево не буде повністю побудоване. Зауважимо, що пункт 3 та 4 покрито у попередньому розділі про створення дерева рішення.

Почнемо із кроку 1 – створення випадкового набору даних із нашої вибірки погоди за минулі роки. Дані, які повинні використовуватись для дерева у випадковому лісі є такими самим, лише додається умова – усі дані мають бути унікальними для навчання. Це досягається за рахунок вибору випадкових даних, для кожного дерева та його перестворення. Даний спосіб називається «завантаження» (bootstrapping), що є статистичним методом для створення набору вибірок. Шляхом такого вибору даних ми досягаємо однієї вагомої речі – деяка частина даних, як мінімум, буде додана або відображена у одному дереві. Зауважимо, що:

- Якщо розмір початкового набору даних дорівнює n, то вибірка, створена випадково також повинна бути такого ж розміру n для кожного дерева.
- Кожна створена вибірка може використовувати один і той же елемент даних кілька разів.

Другий крок – вибір випадкових атрибутів. Після того як ми створили набір випадкових вибірок із даних, нам потрібно вибрати атрибути, із яких

будуть будуватись вузли дерев. Для початку потрібно обмежити максимальну кількість атрибутів, яка може бути використана. Зазвичай таке значення дістають за формулою:

$$q = \sqrt{n},$$

де, n – це загальна кількість атрибутів у початковій вибірці. Отже, якщо у вибірці присутні 9 атрибутів, то ми отримаємо, що максимальна кількість випадково вибраних для деякого дерева рішення дорівнює 3. Після цього потрібно випадково вибрати кілька атрибутів із всіх, але не більше максимальної кількості. Атрибути також можуть повторюватись.

Крок 3 – вибір атрибуту із якого потрібно почати побудову дерева. Даний алгоритм використовує значення ентропії, описане у розділі дерев рішення. Побудова дерева виконується звичайним способом, тут немає особливих способів спеціально для випадкового лісу.

Кроки 2 – 4 повторюються поки дерево повністю не буде побудоване. Під висловом «повністю побудоване» мається на увазі дерево вважається побудованим тоді, коли рівень глибини дерева досягнуто. Даний спосіб використано, оскільки він допомагає уникнути перенавчання алгоритму.

Після того як усі дерева побудовані і ми отримали певний результат для кожного з них, метод випадкового лісу пропонує два способи, які допоможуть вибрати значення серед усіх передбачених деревами:

1. Вибір передбаченого значення, яке найчастіше зустрічається у результатах дерев.
2. Середнє значення від усіх результатів.

Для передбачення максимального значення температури на наступний день використано 2 спосіб, щоб мінімізувати помилку. Перший спосіб використаний для передбачення стану погоди, оскільки даний спосіб підходить задачам класифікації. Отже, якщо для задачі передбачення максимальної температури ми побудуємо n дерев рішень, та отримаємо передбачення від кожного із них $x_1, x_2\ \dots\ x_n$, то щоб знайти передбачення для випадкового лісу нам потрібно використати формулу:

$$T = \frac{\sum_{i=1}^{n} x_i}{n}.$$

Якщо використати спосіб 1 для задачі передбачення стану погоди, потрібно знайти найбільш повторюваний елемент i у масиві передбачень розмірністю n, де $i = 1 \dots n$.

Після знаходження результату природним наступним кроком у задачах машинного навчання є визначення помилки передбачення. Для визначення такого значення у задачах класифікації та регресії можна використати один із двох методів:

1. Обчислення відношення кількості правильно передбачених значень та загальної кількості.
2. Оцінка якості (out of bag error).

У роботі використаний перший спосіб визначення точності, оскільки він є простішим та надає швидкий результат. Для цього використовується формула:

$$A = \frac{C}{N} * 100\%,$$

де, C – кількість правильно передбачених значень, N – загальна кількість передбачень. Оцінка якості обчислюється складнішим методом.

Повертаючись до першого кроку побудування випадкового лісу – складання випадкового набору даних, ми дозволяли повторювання даних і вибірка була обмежена, враховуючи це, природньо, що усі дані із початкової вибірки не можуть потрапити до випадкового лісу. Зазвичай 30% даних не включається до випадкового лісу. Отже, для того, щоб отримати значення помилки оцінки якості, потрібно вищезгадані дані також надати випадковому лісу для передбачення. Порівнявши результати із реальними, отримаємо відсоток неправильних результатів, що і буде оцінкою якості. Дане значення потрібно відняти від 100%, щоб отримати точність передбачення для випадкового лісу.

Виконавши оцінку точності передбачення для прикладної задачі, маємо такі результати:

- Передбачення максимальної температури – 83%.
- Передбачення стану погоди – 94.3%.

Похибка задачі регресії виявилась доволі великою, однією з причин можна виділити велику кількість класів відповідей, що значно ускладнює обчислення ентропії і, як результат, впливає на вибір найкращих атрибутів для побудови дерев.

Отже, метод випадкового лісу – є популярним методом машинного навчання, який часто використовується у різних сферах життя. В основі методи лежить побудова дерев рішень. Переваги випадкового лісу:

- Вирішує обидві проблеми: класифікації і регресії.
- Можливість опрацювання великої кількості даних, шляхом вибору обмеженої кількості даних для кожного дерева.

До недоліків методу можна віднести:

- Низька швидкість у випадку генерації великої кількості дерев рішень.
- Схильність до перенавчання.

Враховуючи, що для нашого завдання потрібно передбачити як числове значення, так і рядкове, даний метод повністю підходить.

Розділ 3. Практична частина

3.1 Аналізатор сайту погоди

Відомо, що однією із найголовніших компонент задач машинного навчання являється початковий набір даних – він повинній бути якомога об’ємний. Також відомо, що найбільш зручним форматом даних для роботи є формат CSV (comma separated values) – це звичайний текстовий файл у якому усі значення розділенні комою. На жаль, готового файла, який би відповідав вищезазначеним критеріям, знайти не вдалось. Натомість було знайдено сайт [7], який містить архівну інформацію прогнозів погоди, починаючи із 2003 року (рис. 6). Було вирішено написати окремий скрипт, який дозволив би проаналізувати потрібно нам інформацію із сайту та зберегти її у файл потрібного типу.

Рис. 6. Сайт погоди із якого отримано архівні дані погоди

Отримати сторінку із потрібною датою сайт дозволяв за допомогою передачі параметру у HTTP GET запиті(рис. 7).

meteo.ua/ua/archive/91/chernovtsyi/2003-01-01

Рис. 7. Адреса та параметр за яким можна отримати доступ до погоди на вказану дату

За допомогою можливостей мови програмування Python створено консольний додаток, який, приймаючи два параметри – дату початку і кінцеву дату, що створювали проміжок для якого нам потрібно отримати інформацію про погоду. Після запуску, аналізатор перевіряє передані дати на валідність, після чого починає алгоритм аналізування та збереження інформації:

1. Побудова адреси, в залежності від міста для якого потрібно отримати інформацію і дати.
2. Аналізатор за допомогою GET запиту звертається до побудованої адреси та отримую об'єкт, який містить інформацію із сторінку сайту.
3. За допомогою HTML тегів, які були отриманні із браузера виконується пошук потрібних елементів на сторінці, отримання даних та збереження їх у об’єкт.
4. Об’єкт із необхідною інформацією записується у файл.
5. Кроки 1 – 4 повторюються, поки дата поточної ітерації не досягне кінцевої.

Під час виконання даних дій, ми можемо побачити поточний статус аналізу у консолі додатку (рис. 8). Статус несе інформацію міста, для якого зараз виконується алгоритм та дату, яка на даний момент часу опрацьовується.

```
Parsing URL: https://meteo.ua/archive/91/chernovtsyi/2020-09-01
Parsing city: chernovtsyi
Parsing date: 2020-09-01
Parsing date: 2020-09-02
Parsing date: 2020-09-03
Parsing date: 2020-09-04
Parsing date: 2020-09-05
Parsing date: 2020-09-06
Parsing date: 2020-09-07
Parsing date: 2020-09-08
Parsing date: 2020-09-09
Parsing date: 2020-09-10
Parsing date: 2020-09-11
Parsing date: 2020-09-12
Parsing date: 2020-09-13
Parsing date: 2020-09-14
Parsing date: 2020-09-15
```

Рис. 8. Вивід поточного статусу аналізу погоди

У результаті отримаємо файл типу CSV із такими значеннями атрибутів випадкового лісу:

- Поточна дата.
- Час.
- Стан погоди.
- Температура.
- Швидкість вітру у метрах в секунду.
- Напрям вітру.
- Атмосферний тиск.
- Вологість у відсотках.
- Місто.

Для методу випадкового лісу було проаналізовано лише місто Чернівці та близько 120-ти тисяч записів, що є достатньою кількістю для виконання

задачі передбачення максимальної температури та стану погоди. Деякі статистичні графіки даних:

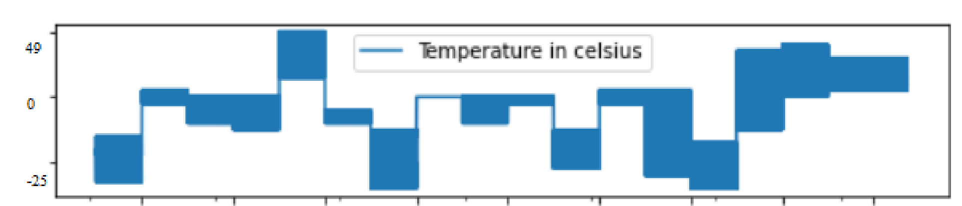

Рис. 9. Градація значень температури повітря з 2003 по 2020 роки.

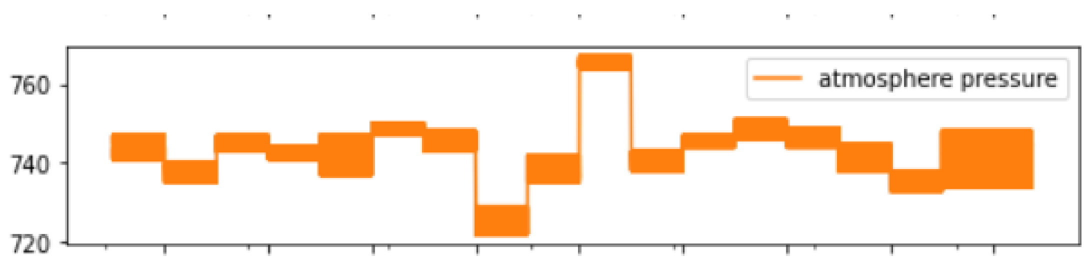

Рис. 10. Значення атмосферного тиску із 2003 по 2020 роки.

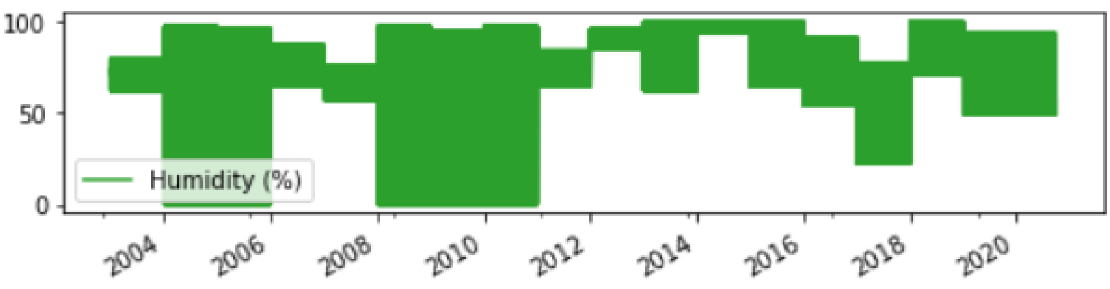

Рис. 11. Значення вологості у відсотках із 2003 по 2020 роки.

Отже, використовуючи окремо написану консольну програму було проаналізовано сайт погоди, починаючи із 2003 року у місті Чернівці і отримано близько 120-ти тисяч записів у вигляді текстового файла. Даний

аналізатор значно спростив пошук та збір, потрібної для методу випадкового лісу, інформації.

3.2 Алгоритм дерев рішень та випадкового лісу. Інформаційна веб сторінка.

Для вирішення задачі передбачення максимальної температури та стану погоди на наступні дні реалізовано метод випадкового лісу та створено інформаційну веб сторінку для користувача. Для реалізації дерев рішень використано алгоритм ID3 та набір даних прогнозу погоди за попередні роки.

Оскільки операція навчання та створення дерев рішень для випадкового лісу є доволі об'ємною, навіть враховуючи обмеження кількості дерев до кілька десятків та глибини, то ініціалізація алгоритму та побудова лісу виконується одразу після запуску сервера. Після чого результат зберігається у кеші та відображається на інформаційній сторінці.

Процес ініціалізації побудований за алгоритмом, який викладений у теоретичній частині роботи. Перед виконанням побудови випадкового лісу, програма виконує попередню підготовку даних, які отримані від аналізатора:

1. Зчитування даних із файла.
2. Заміна усіх пробілів у назвах атрибутів символом «_».
3. Перетворення двох окремих колонок «Дата» і «Час» у одне поле вибірки.
4. Перетворення напряму вітру із слів у градуси.
5. Перетворення полів «Швидкість вітру» та «Напрям вітру» у числове поле «Вектор вітру х» і «Вектор вітру у».
6. Перетворення дати і часу у секунди.
7. Перетворення секунд дати і часу у «сигнали», враховуючи періодичність даного поля.

8. Розділення одного файла на два окремі (температура та стан погоди) із додатковою колонкою, яка містить правильне для передбачення значення.
9. Розділення даних для тренування із правильними результатами і тесту побудованої моделі.

Замінивши усі пробіли на символ «_» ми досягаємо впевненості в тому, що при виконанні алгоритму не виникне непередбачуваних проблем із зчитуванням атрибутів та/або значень із вибірки даних.

Дата та час перетворенні із двох колонок в одну для зручності. Очевидно, що робота програми із значеннями вигляду «01/01/2020 12:00» не є надто ефективною, тому для зручності значення атрибуту були переведені у секунди за допомогою внутрішніх функцій Python. Потрібно зауважити, що кожна дата з'являється із певною періодичністю, щоб забезпечити це, значення переведено у «сигнали» днів та років за допомогою формул:

$$Ds = \sin\left(t * \frac{2*PI}{d}\right),$$

$$Dc = \cos\left(t * \frac{2*PI}{d}\right),$$

де, t – час у секундах, d – тривалість дня у секундах.

Для переведення у «сигнали» років замість тривалості дня у секундах, потрібно використати тривалість року у секундах. Використавши, дані формули ми отримуємо періодичні значення у проміжку [-1, 1] (рис. 12), що відповідають за дні.

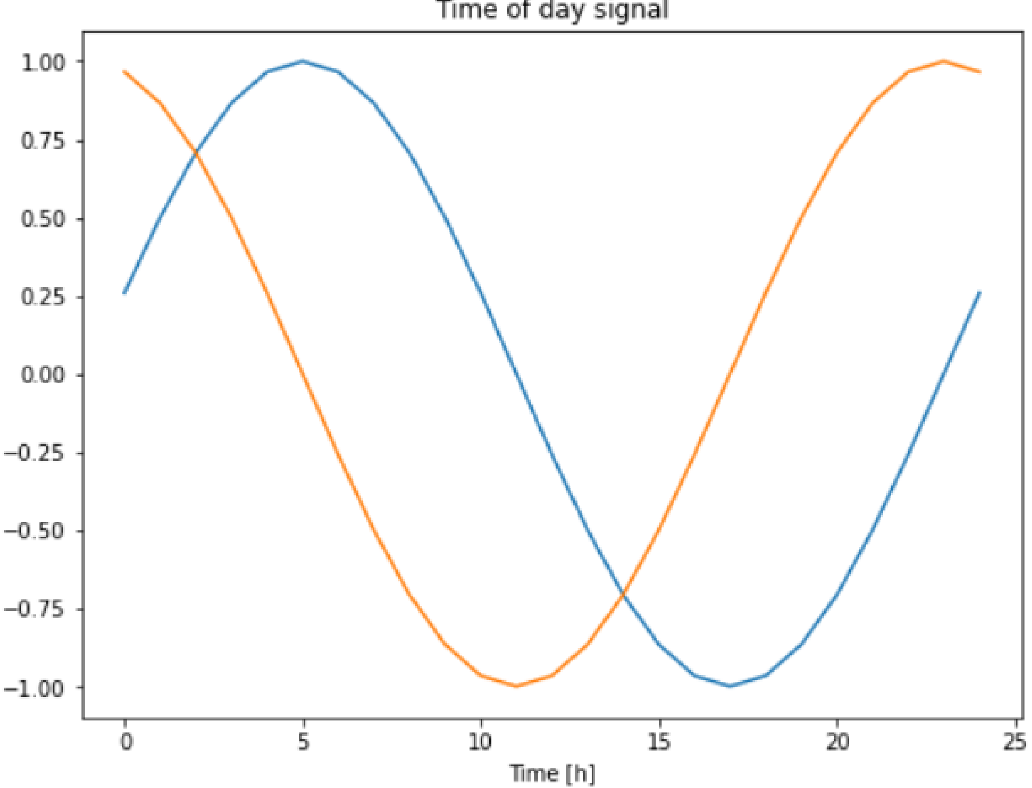

Рис. 12. Інтервал значень та їх періодичність для «сигналів» днів.

Схожу ситуацію ми маємо із полями «Швидкість вітру» та «Напрям вітру». Для початку останню потрібно перевести у градусні значення:

- Північний – 0 градусів.
- Північно-західний – 45 градусів.
- …
- Західно-північний – 315 градусів.

Таким чином отримуємо 360 градусів у сумі, покриваючи усі напрямки у межах кола. Очевидно, що обидва поля є взаємопов’язаними і не мають сенсу окрему у контексті нашого завдання. Наприклад, напрям вітру не є корисним атрибутом, якщо швидкість вітру дорівнює 0 метрів в секунду. Для того, щоб оминути подібні помилки, переведемо обидва поля у вектор, подібно до того,

що ми зробили для дати і часу. Для початку переведемо градуси у радіани, використовуючи формулу:

$$r = \frac{d*PI}{180},$$

після чого, отримаємо вектор помноживши на швидкість вітру:

$$Vx = S * \cos(r),$$

$$Vy = S * \sin(r).$$

Отримавши значення, зберігаємо їх у вибірку для подальшого опрацювання.

Оскільки нам потрібно дати відповідь на два запитання на основі однієї вибірки даних, розділимо вибірку на дві окремі, які матимуть однакові дані, крім останнього поля. Останнє поле – це результат, який має бути передбачений для поточної дати. Дане значення використовується при побудові дерев рішень. Крім цього атрибуту, розділивши дані, ми досягаємо ізольованості даних, тому задача регресії ніяк не зможе повпливати на задачу класифікації або навпаки.

Після підготовки даних відбувається побудова випадкового лісу і дерев рішень. Для початку ми отримуємо випадкову вибірку даних для дерева. У програмі кількість даних обмежена 3000 для обох задач. Після чого відбувається нормалізація даних і початок вибору атрибутів. Кількість випадково вибраних атрибутів також обмежена для обох задач цифрою 3. Після цього відбувається отримання випадкових вибірок із атрибутами, вибір найкращого атрибута шляхом обчислення ентропії та порівняння значення між вузлами зміни атрибуту. Вибравши найкращу групу атрибутів, виконується

побудова дерева із вузлів запитань, які стосуються вибраних атрибутів. Запитання також відрізняються в залежності від типу задачі:

- Перевірка більше/менше/дорівнює у випадку регресії.
- Перевірка на точну відповідність так або ні, якщо працюємо з класифікацією.

Даний процес триває доволі довго і за його проходженням можна спостерігати у консолі сервера, також дані записуються в файл налагодження у випадку майбутньої потреби. Кількість та глибина дерев обмежена, щоб уникнути перенавчання, яке є слабкою стороною дерев рішень. Отже, кількість дерев для задачі класифікації обмежено 20, для задачі регресії – 30, глибина кожного дерева не перевищує 7. Після передбачення результату усіма деревами, усі результати зберігаються у об’єкті випадкового лісу та відбувається визначення остаточного результату. Щоб отримати максимальну температуру, відбувається знаходження середнього із 30 значень передбаченої температури за допомогою дерев рішень. Для стану погоди – знаходиться мода – найбільш часте значення у вибірці передбачень. Ці результати зберігаються у кеші та відображаються на інформаційному сайті при потребі.

Для того, щоб забезпечити зручний перегляд результату для користувача, розроблено також «невеликий» веб сайт, який відображає передбаченні значення та фактичні значення на сьогоднішній день. Даний сайт має лише одну сторінку на якій і відображена уся інформація (рис. 13). Для отримання інформації про погоду, достатньо лише зайти на дану сторінку та переглянути інформацію, яка цікавить.

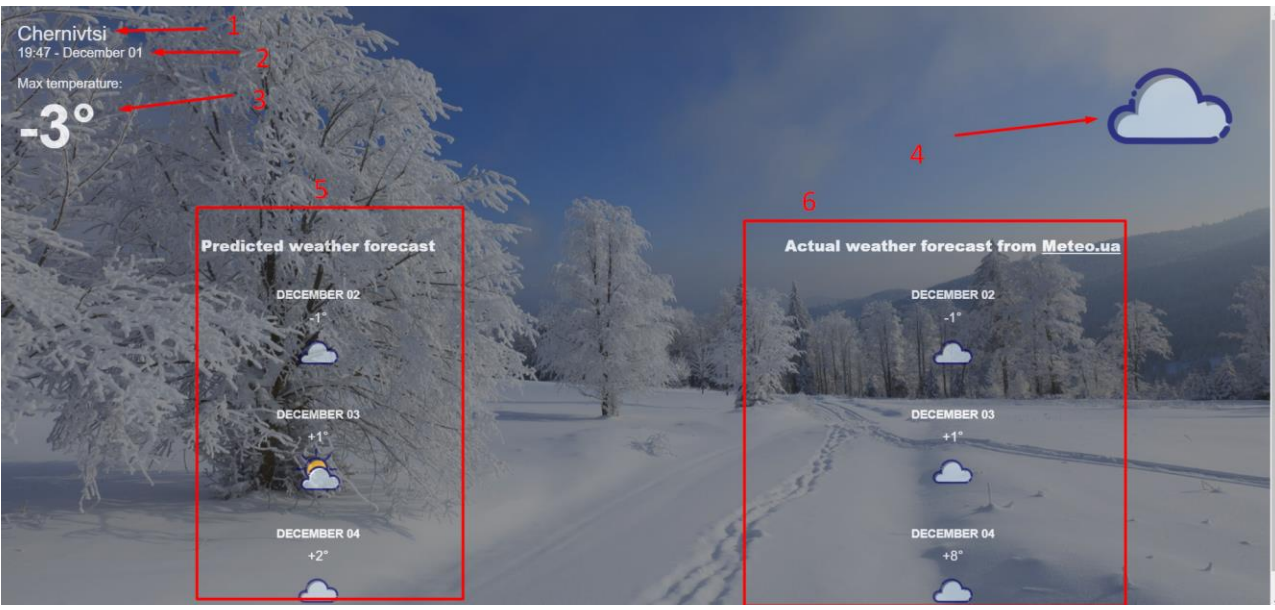

Рис. 13. Зовнішній вигляд інформаційної веб-сторінки

Розглянемо інтерфейс користувача детальніше:

1. Місто, для якого відображена інформація.
2. Поточний час і дата.
3. Максимальна температура станом на час і дату, які зазначені вище.
4. Стан погоди у вигляді картинки.
5. Передбачені за допомогою методу випадкового лісу значення на 3 наступні дні.
6. Прогноз погоди на 3 найближчі дні із сайту погоди [7].

Для відображення температури та стану погоди на сьогодні використовуються фактичні дані із файла проаналізованих даних із зовнішнього сайту погоди.

Передбаченні значення можна порівняти із фактичними даними із сайту погоди. Також, у заголовку колонки 6, розміщено посилання на ресурс із якого був взятий прогноз.

У випадку будь-якої помилки на стороні сервера на сторінці буде відображено відповідне повідомлення. Для того, щоб перевірити систему можна скористатись файлом відлагодження, у який постійно записуються усі дії програми.

Також сайт має картинку заднього фону, яка змінюється в залежності від поточної пори року (рис. 14).

Рис. 14. Можливі варіанти фонового зображення для різних пір року.

Отже, шляхом реалізації методу випадкового лісу та дерева рішень, ми виконали задачу передбачення максимальної температури та стану погоди на наступні дні. За допомогою невеликого веб сайту, користувач може дізнатися передбаченні значення.

Висновки

У роботі реалізовано передбачення максимальної температури та стану погоди на наступні дні методом випадкового лісу. Для відображення інформації, розроблено невелику веб сторінку, яка містить фактичні значення згаданих атрибутів на поточний час та передбаченні значення на наступні дні. Для тренування методу та його побудови використано набір даних, який був отриманий шляхом використання консольного додатку, який, проаналізувавши усі необхідні дані із сайту погоди, зберіг їх у файл.

Дерева рішень – основа методу випадкового лісу і метод, який легко трансформує прикладну задачу у алгоритм передбачення результату. Переваги методу:

- Легко зрозуміти та візуалізувати.
- Може передбачати як числові так і рядкові дані.
- Мінімальна підготовка даних.
- Вибір атрибутів відбувається автоматично.

Щодо недоліків:

- Є тенденція до перенавчання.
- Маленька зміна у даних впливає на всю структуру дерева.
- Досить великий час навчання та складність обчислення при зростанні кількості потрібних значень для передбачень.

Метод випадкового лісу, побудований із багатьох дерев рішень являється ефективним методом машинного навчання для одночасно задач класифікації та регресії, серед його переваг:

- Вирішує обидві проблеми: класифікації і регресії.

- Можливість опрацювання великої кількості даних, шляхом вибору обмеженої кількості даних для кожного дерева.

До недоліків методу можна віднести:

- Низька швидкість у випадку генерації великої кількості дерев рішень.
- Схильність до перенавчання.

Даний метод чудово підходить для вирішення поставлених задач, що і свідчить його часте використання у різноманітних прикладних задачах реального життя.

Використовуючи метод випадкового лісу та дерева рішень для його побудови, досліджено алгоритмічну складову методу та застосував її на практиці, реалізувавши програму, яка передбачає максимальну температуру та стан погоди. Проаналізував його переваги та недоліки, порівняв їх із іншими методами машинного навчання.

Література та використані джерела

1. Smith C., Koning M. Decision Trees and Random Forests: A Visual Introduction for Beginners. Independently published, 2017. 168 p.
2. Відеоуроки про використання та побудову дерев рішень та випадкового лісу. URL: https://www.youtube.com/watch?v=mQaAPoRGOcs (Дата останнього звернення: 13.10.2020).
3. Документація мови програмування Python. Офіційний сайт. URL: https://docs.python.org/3/index.html (Дата останнього звернення: 29.11.2020).
4. Документація бібліотеки NumPy для мови програмування Python. Офіційний сайт. URL: https://numpy.org/doc/ (Дата останнього звернення: 29.11.2020).
5. Документація фреймворку Flask для мови програмування Python. Офіційний сайт. URL: http://flask.pocoo.org/docs/1.0/ (Дата останнього звернення: 02.12.2020).
6. Сайт розповсюдження мови програмування Python. Офіційний сайт. URL: https://www.python.org/ (Дата останнього звернення: 06.02.2020).
7. Прогноз погоди та архівні дані прогнозу попередніх років. URL: https://meteo.ua/ua/ (Дата останнього звернення: 10.12.2020).

Додатки

app_main.py

```
import datetime

from preparation_main import prepare_data
from random_forest_main import predict_weather
from util.helper_functions import update_predictions, update_today_weather
from util.logging import WeatherLogging
from weather_parser.parser_main import run_parser, get_today_forecast
from web.web_config import weather_web_app

PARSE_DAY_START = datetime.datetime.now() - datetime.timedelta(days=-1)


def run():
    WeatherLogging.setup_logger()
    run_parser(PARSE_DAY_START)
    prepare_data()
    update_predictions(predict_weather())
    update_today_weather(get_today_forecast())

    weather_web_app.run()


if __name__ == '__main__':
    run()
```

preparation_main.py

```
import datetime

import matplotlib.pyplot as plt
import numpy as np
import pandas as pd

from web.web_config import LOG


def replace_wind_direction_by_degree(data_frame):
    wind_dir_replace = {"Wind direction": {"North": 0,
                                           "Northeast": 45,
                                           "East": 90,
                                           "Southeast": 135,
                                           "South": 180,
                                           "Southwest": 225,
                                           "West": 270,
                                           "Northwest": 315
                                           }
```

```
                        }
    data_frame.replace(wind_dir_replace, inplace=True)


def data_preparation(data_frame):
    LOG.info(data_frame.describe().transpose())

    date_time = pd.to_datetime(data_frame.pop('Date') + ' ' + data_frame.pop('Time'),
infer_datetime_format=True)
    LOG.info(date_time)

    replace_wind_direction_by_degree(data_frame)
    wind(data_frame)
    time_to_seconds(data_frame, date_time)

    return data_frame


def wind(data_frame):
    wind_speed = data_frame.pop('wind speed (m/s)')
    wind_direction_radian = data_frame.pop('Wind direction') * np.pi / 180

    data_frame['Wx (m/s)'] = wind_speed * np.cos(wind_direction_radian)
    data_frame['Wy (m/s)'] = wind_speed * np.sin(wind_direction_radian)

    plt.hist2d(data_frame['Wx (m/s)'], data_frame['Wy (m/s)'], bins=(50, 50),
vmax=400)
    plt.colorbar()
    plt.xlabel('Wind X [m/s]')
    plt.ylabel('Wind Y [m/s]')
    ax = plt.gca()
    ax.axis('tight')

    plt.show()


def time_to_seconds(data_frame, date_time):
    timestamp_in_seconds = date_time.map(datetime.datetime.timestamp)

    day = 24 * 60 * 60
    year = 365.2425 * day

    data_frame['Day sin'] = np.sin(timestamp_in_seconds * (2 * np.pi / day))
    data_frame['Day cos'] = np.cos(timestamp_in_seconds * (2 * np.pi / day))
    data_frame['Year sin'] = np.sin(timestamp_in_seconds * (2 * np.pi / year))
    data_frame['Year cos'] = np.cos(timestamp_in_seconds * (2 * np.pi / year))

    plt.plot(np.array(data_frame['Day sin'])[:25])
    plt.plot(np.array(data_frame['Day cos'])[:25])
    plt.xlabel('Time [h]')
    plt.title('Time of day signal')

    plt.show()
```

```
def save_to_new_files(processed_df):
    temp_labels = processed_df["Temperature in celsius"]
    weather_condition_labels = processed_df["Weather condition"]
    data = processed_df.drop(["Weather condition", "Temperature in celsius"], axis=1)
    temp_data = data.copy()
    weather_condition_data = data.copy()
    temp_data["label"] = temp_labels
    weather_condition_data["label"] = weather_condition_labels

    weather_condition_data.to_csv("weather_data/weather_condition_data.csv",
index=False)
    temp_data.to_csv("weather_data/temperature_data.csv", index=False)


def prepare_data():
    data_frame = pd.read_csv('weather_data/weather_data.csv')

    processed_df = data_preparation(data_frame)
    save_to_new_files(processed_df)
```

random_forest_main.py

```
import numpy as np
import pandas as pd

from random_forest_algorithm.random_forest import random_forest_algorithm,
random_forest_predictions
from util.helper_functions import calculate_accuracy, DAYS_TO_PREDICT
from web.web_config import LOG


def classify_weather_condition(days_to_predict):
    # rename to avoid spaces
    df = pd.read_csv("weather_data/weather_condition_data.csv")
    df.columns = df.columns.str.replace(' ', '_')

    df["label"] = df.replace(' ', '_', regex=True).label

    # split data
    n = len(df)
    train_df = df[0:int(n * 0.7)]
    val_df = df[int(n * 0.7):int(n * 0.9)]
    test_df = df[int(n * 0.9):]

    # random forest to classify
    forest = random_forest_algorithm(train_df, val_df, n_trees=20, n_bootstrap=3000,
n_features=3, dt_max_depth=7,
                                     ml_task="classification")
    predictions = random_forest_predictions(test_df, forest,
ml_task="classification")
    accuracy = calculate_accuracy(predictions, test_df.label)
```

```python
    LOG.info("Accuracy = {}".format(accuracy))

    return predictions[:days_to_predict]


def max_temperature_regression(days_to_predict):
    # load and replace spaces
    df = pd.read_csv("weather_data/temperature_data.csv")
    df.columns = df.columns.str.replace(' ', '_')
    df.head()

    # split the data
    n = len(df)
    train_df = df[0:int(n * 0.7)]
    val_df = df[int(n * 0.7):int(n * 0.9)]
    test_df = df[int(n * 0.9):]

    forest = random_forest_algorithm(train_df, val_df, n_trees=30, n_bootstrap=3000,
n_features=3, dt_max_depth=7,
                                     ml_task="regression")
    predictions = random_forest_predictions(test_df, forest, "regression")
    rmse = np.sqrt(((predictions - test_df.label) ** 2).mean())
    LOG.info("RMSE = {}".format(rmse))

    return predictions[:days_to_predict]


def predict_weather():
    weather_condition_prediction = classify_weather_condition(DAYS_TO_PREDICT)
    max_temperature_prediction = max_temperature_regression(DAYS_TO_PREDICT)

    return weather_condition_prediction, max_temperature_prediction
```

random_forst_algorithm/decision_tree_functions.py

```python
# coding: utf-8

import random

import numpy as np
import pandas as pd


# 1. Decision Tree helper functions
# 1.1 Data pure?
from web.web_config import LOG


def check_purity(data):
    label_column = data[:, -1]
    unique_classes = np.unique(label_column)
```

```
    if len(unique_classes) == 1:
        return True
    else:
        return False


# 1.2 Classify
def classify_data(data, ml_task):
    label_column = data[:, -1]

    if ml_task == "regression":
        classification = np.mean(label_column)

    # classfication
    else:
        unique_classes, counts_unique_classes = np.unique(label_column,
return_counts=True)
        index = counts_unique_classes.argmax()
        classification = unique_classes[index]

    return classification


# 1.3 Potential splits?
def get_potential_splits(data, random_subspace):
    potential_splits = {}
    _, n_columns = data.shape
    column_indices = list(range(n_columns - 1))  # excluding the last column which is
the label

    if random_subspace and random_subspace <= len(column_indices):
        column_indices = random.sample(population=column_indices, k=random_subspace)

    for column_index in column_indices:
        values = data[:, column_index]
        unique_values = np.unique(values)

        potential_splits[column_index] = unique_values

    return potential_splits


# 1.4 Determine Best Split
def calculate_entropy(data):
    label_column = data[:, -1]
    _, counts = np.unique(label_column, return_counts=True)

    probabilities = counts / counts.sum()
    entropy = sum(probabilities * -np.log2(probabilities))

    return entropy


def calculate_mse(data):
```

```
    actual_values = data[:, -1]
    if len(actual_values) == 0:  # empty data
        mse = 0

    else:
        prediction = np.mean(actual_values)
        mse = np.mean((actual_values - prediction) ** 2)

    return mse


def calculate_overall_metric(data_below, data_above, metric_function):
    n = len(data_below) + len(data_above)
    p_data_below = len(data_below) / n
    p_data_above = len(data_above) / n

    overall_metric = (p_data_below * metric_function(data_below)
                      + p_data_above * metric_function(data_above))

    return overall_metric


def determine_best_split(data, potential_splits, ml_task):
    first_iteration = True
    for column_index in potential_splits:
        for value in potential_splits[column_index]:
            data_below, data_above = split_data(data, split_column=column_index,
split_value=value)

            if ml_task == "regression":
                current_overall_metric = calculate_overall_metric(data_below,
data_above, metric_function=calculate_mse)

            # classification
            else:
                current_overall_metric = calculate_overall_metric(data_below,
data_above,

metric_function=calculate_entropy)

            if first_iteration or current_overall_metric <= best_overall_metric:
                first_iteration = False

                best_overall_metric = current_overall_metric
                best_split_column = column_index
                best_split_value = value

    return best_split_column, best_split_value


# 1.5 Split data
def split_data(data, split_column, split_value):
    split_column_values = data[:, split_column]

    type_of_feature = FEATURE_TYPES[split_column]
```

```python
    if type_of_feature == "continuous":
        data_below = data[split_column_values <= split_value]
        data_above = data[split_column_values > split_value]

    # feature is categorical
    else:
        data_below = data[split_column_values == split_value]
        data_above = data[split_column_values != split_value]

    return data_below, data_above


# 2. Decision Tree Algorithm
# 2.1 Helper Function
def determine_type_of_feature(df):
    feature_types = []
    n_unique_values_treshold = 15
    for feature in df.columns:
        if feature != "label":
            unique_values = df[feature].unique()
            example_value = unique_values[0]

            if (isinstance(example_value, str)) or (len(unique_values) <=
n_unique_values_treshold):
                feature_types.append("categorical")
            else:
                feature_types.append("continuous")

    return feature_types


# 2.2 Algorithm
def decision_tree_algorithm(df, ml_task, counter=0, min_samples=2, max_depth=5,
random_subspace=None):
    # data preparations
    if counter == 0:
        global COLUMN_HEADERS, FEATURE_TYPES
        COLUMN_HEADERS = df.columns
        FEATURE_TYPES = determine_type_of_feature(df)
        data = df.values
    else:
        data = df

        # base cases
    if (check_purity(data)) or (len(data) < min_samples) or (counter == max_depth):
        classification = classify_data(data, ml_task)

        return classification


    # recursive part
    else:
        counter += 1

        # helper functions
```

```
        potential_splits = get_potential_splits(data, random_subspace)
        split_column, split_value = determine_best_split(data, potential_splits,
ml_task)
        data_below, data_above = split_data(data, split_column, split_value)

        # check for empty data
        if len(data_below) == 0 or len(data_above) == 0:
            classification = classify_data(data, ml_task)
            return classification

        # determine question
        feature_name = COLUMN_HEADERS[split_column]
        type_of_feature = FEATURE_TYPES[split_column]
        if type_of_feature == "continuous":
            question = "{} <= {}".format(feature_name, split_value)

        # feature is categorical
        else:
            question = "{} = {}".format(feature_name, split_value)

        LOG.debug("Current question: " + question)

        # instantiate sub-tree
        sub_tree = {question: []}

        # find answers (recursion)
        yes_answer = decision_tree_algorithm(data_below, ml_task, counter,
min_samples, max_depth, random_subspace)
        no_answer = decision_tree_algorithm(data_above, ml_task, counter,
min_samples, max_depth, random_subspace)

        # If the answers are the same, then there is no point in asking the qestion.
        # This could happen when the data is classified even though it is not pure
        # yet (min_samples or max_depth base case).
        if yes_answer == no_answer:
            sub_tree = yes_answer
        else:
            sub_tree[question].append(yes_answer)
            sub_tree[question].append(no_answer)

        return sub_tree


# 3. Make predictions
# 3.1 One example
def predict_example(example, tree):
    # tree is just a root node
    if not isinstance(tree, dict):
        return tree

    question = list(tree.keys())[0]
    feature_name, comparison_operator, value = question.split(" ")

    # ask question
    if comparison_operator == "<=":
```

```
        if example[feature_name] <= float(value):
            answer = tree[question][0]
        else:
            answer = tree[question][1]

    # feature is categorical
    else:
        if str(example[feature_name]) == value:
            answer = tree[question][0]
        else:
            answer = tree[question][1]

    # base case
    if not isinstance(answer, dict):
        return answer

    # recursive part
    else:
        residual_tree = answer
        return predict_example(example, residual_tree)


# 3.2 All examples of a dataframe
def decision_tree_predictions(df, tree):
    if len(df) != 0:
        predictions = df.apply(predict_example, args=(tree,), axis=1)
    else:
        # "df.apply()"" with empty dataframe returns an empty dataframe,
        # but "predictions" should be a series instead
        predictions = pd.Series()

    return predictions


# 3.3 Accuracy
def calculate_accuracy(df, tree):
    predictions = make_predictions(df, tree)
    predictions_correct = predictions == df.label
    accuracy = predictions_correct.mean()

    return accuracy
```

Random_forst_algorithm/random_forest.py

```
# coding: utf-8

import numpy as np
import pandas as pd

from random_forest_algorithm.decision_tree_functions import decision_tree_algorithm,
decision_tree_predictions
```

```python
# Reduced Error Tree Pruning
from web.web_config import LOG


def filter_df(df, question):
    feature, comparison_operator, value = question.split()

    # continuous feature
    if comparison_operator == "<=":
        df_yes = df[df[feature] <= float(value)]
        df_no = df[df[feature] > float(value)]

    # categorical feature
    else:
        df_yes = df[df[feature].astype(str) == value]
        df_no = df[df[feature].astype(str) != value]

    return df_yes, df_no


def determine_leaf(df_train, ml_task):
    if ml_task == "regression":
        return df_train.label.mean()

    # classification
    else:
        return df_train.label.value_counts().index[0]


def determine_errors(df_val, tree, ml_task):
    predictions = decision_tree_predictions(df_val, tree)
    actual_values = df_val.label

    if ml_task == "regression":
        # mean squared error
        return ((predictions - actual_values) ** 2).mean()
    else:
        # number of errors
        return sum(predictions != actual_values)


def pruning_result(tree, df_train, df_val, ml_task):
    leaf = determine_leaf(df_train, ml_task)
    errors_leaf = determine_errors(df_val, leaf, ml_task)
    errors_decision_node = determine_errors(df_val, tree, ml_task)

    if errors_leaf <= errors_decision_node:
        return leaf
    else:
        return tree


def post_pruning(tree, df_train, df_val, ml_task):
```

```
    question = list(tree.keys())[0]
    yes_answer, no_answer = tree[question]

    # base case
    if not isinstance(yes_answer, dict) and not isinstance(no_answer, dict):
        return pruning_result(tree, df_train, df_val, ml_task)

    # recursive part
    else:
        df_train_yes, df_train_no = filter_df(df_train, question)
        df_val_yes, df_val_no = filter_df(df_val, question)

        if isinstance(yes_answer, dict):
            yes_answer = post_pruning(yes_answer, df_train_yes, df_val_yes, ml_task)

        if isinstance(no_answer, dict):
            no_answer = post_pruning(no_answer, df_train_no, df_val_no, ml_task)

        tree = {question: [yes_answer, no_answer]}

        return pruning_result(tree, df_train, df_val, ml_task)


def bootstrapping(train_df, n_bootstrap):
    bootstrap_indices = np.random.randint(low=0, high=len(train_df),
size=n_bootstrap)
    df_bootstrapped = train_df.iloc[bootstrap_indices]

    return df_bootstrapped


def random_forest_algorithm(train_df, val_df, n_trees, n_bootstrap, n_features,
dt_max_depth, ml_task):
    forest = []
    for i in range(n_trees):
        LOG.debug("Decision-Tree # " + str(i + 1))
        df_bootstrapped = bootstrapping(train_df, n_bootstrap)
        tree = decision_tree_algorithm(df_bootstrapped, ml_task=ml_task,
max_depth=dt_max_depth,
                                       random_subspace=n_features)
        tree_pruned = post_pruning(tree, train_df, val_df, ml_task)
        forest.append(tree_pruned)

    return forest


def random_forest_predictions(test_df, forest, ml_task):
    df_predictions = {}
    for i in range(len(forest)):
        column_name = "tree_{}".format(i)
        predictions = decision_tree_predictions(test_df, tree=forest[i])
        df_predictions[column_name] = predictions

    df_predictions = pd.DataFrame(df_predictions)
    if ml_task == "regression":
```

```python
        predictions = df_predictions.mean(axis=1)
    else:
        predictions = df_predictions.mode(axis=1)[0]

    return predictions
```

util/helper_functions.py

```python
# coding: utf-8
import datetime

import numpy as np

WEATHER_ICON_FOLDER = '../static/css/weather-icons/'
WEATHER_ICON_EXTENSION = '.png'

DAYS_TO_PREDICT = 3
WEATHER_STATE_PREDICTION_INDEX = 0
WEATHER_TEMPERATURE_PREDICTION_INDEX = 1

PREDICTED_MAX_TEMPERATURE_VALUES = []
ACTUAL_MAX_TEMPERATURE_VALUES = []

PREDICTED_WEATHER_STATE = []
ACTUAL_WEATHER_STATE = []


# Distinguish categorical and continuous features
def determine_type_of_feature(df):
    feature_types = []
    n_unique_values_treshold = 15
    for feature in df.columns:
        if feature != "label":
            unique_values = df[feature].unique()
            example_value = unique_values[0]

            if (isinstance(example_value, str)) or (len(unique_values) <=
n_unique_values_treshold):
                feature_types.append("categorical")
            else:
                feature_types.append("continuous")

    return feature_types


# 3. Accuracy
def calculate_accuracy(predictions, labels):
    predictions_correct = predictions == labels
    accuracy = predictions_correct.mean()

    return accuracy
```

```
# 4. Regression R-Squared
def calculate_rmse(predictions, labels):
    return np.sqrt(((predictions - labels) ** 2).mean())


def generate_response_object(temperature, weather_state):
    result = {}

    if len(temperature) != len(weather_state):
        pass

    for i in range(DAYS_TO_PREDICT):
        day_offset = datetime.timedelta(days=i + 1)
        date = datetime.datetime.now() + day_offset
        current_day = date.strftime("%B %d")

        result[current_day] = [temperature[i], WEATHER_ICON_FOLDER + weather_state[i]
+ WEATHER_ICON_EXTENSION]

    return result


def update_predictions(weather_prediction):
    global PREDICTED_WEATHER_STATE
    global PREDICTED_MAX_TEMPERATURE_VALUES

    PREDICTED_WEATHER_STATE = weather_prediction[WEATHER_STATE_PREDICTION_INDEX]
    PREDICTED_MAX_TEMPERATURE_VALUES =
weather_prediction[WEATHER_TEMPERATURE_PREDICTION_INDEX]


def update_today_weather(today_weather):
    global ACTUAL_WEATHER_STATE
    global ACTUAL_MAX_TEMPERATURE_VALUES

    ACTUAL_WEATHER_STATE = today_weather[WEATHER_STATE_PREDICTION_INDEX]
    ACTUAL_MAX_TEMPERATURE_VALUES =
today_weather[WEATHER_TEMPERATURE_PREDICTION_INDEX]
```

util/logging.py

```
# coding: utf-8
import datetime

import numpy as np

WEATHER_ICON_FOLDER = '../static/css/weather-icons/'
WEATHER_ICON_EXTENSION = '.png'

DAYS_TO_PREDICT = 3
WEATHER_STATE_PREDICTION_INDEX = 0
```

```
WEATHER_TEMPERATURE_PREDICTION_INDEX = 1

PREDICTED_MAX_TEMPERATURE_VALUES = []
ACTUAL_MAX_TEMPERATURE_VALUES = []

PREDICTED_WEATHER_STATE = []
ACTUAL_WEATHER_STATE = []


# Distinguish categorical and continuous features
def determine_type_of_feature(df):
    feature_types = []
    n_unique_values_treshold = 15
    for feature in df.columns:
        if feature != "label":
            unique_values = df[feature].unique()
            example_value = unique_values[0]

            if (isinstance(example_value, str)) or (len(unique_values) <=
n_unique_values_treshold):
                feature_types.append("categorical")
            else:
                feature_types.append("continuous")

    return feature_types


# 3. Accuracy
def calculate_accuracy(predictions, labels):
    predictions_correct = predictions == labels
    accuracy = predictions_correct.mean()

    return accuracy


# 4. Regression R-Squared
def calculate_rmse(predictions, labels):
    return np.sqrt(((predictions - labels) ** 2).mean())


def generate_response_object(temperature, weather_state):
    result = {}

    if len(temperature) != len(weather_state):
        pass

    for i in range(DAYS_TO_PREDICT):
        day_offset = datetime.timedelta(days=i + 1)
        date = datetime.datetime.now() + day_offset
        current_day = date.strftime("%B %d")

        result[current_day] = [temperature[i], WEATHER_ICON_FOLDER + weather_state[i]
+ WEATHER_ICON_EXTENSION]

    return result
```

```
def update_predictions(weather_prediction):
    global PREDICTED_WEATHER_STATE
    global PREDICTED_MAX_TEMPERATURE_VALUES

    PREDICTED_WEATHER_STATE = weather_prediction[WEATHER_STATE_PREDICTION_INDEX]
    PREDICTED_MAX_TEMPERATURE_VALUES =
weather_prediction[WEATHER_TEMPERATURE_PREDICTION_INDEX]


def update_today_weather(today_weather):
    global ACTUAL_WEATHER_STATE
    global ACTUAL_MAX_TEMPERATURE_VALUES

    ACTUAL_WEATHER_STATE = today_weather[WEATHER_STATE_PREDICTION_INDEX]
    ACTUAL_MAX_TEMPERATURE_VALUES =
today_weather[WEATHER_TEMPERATURE_PREDICTION_INDEX]
```

web/web_config.py

```
# coding=utf-8
import datetime
import logging

from flask import Flask, render_template, session

from util.helper_functions import WEATHER_ICON_FOLDER, WEATHER_ICON_EXTENSION,
generate_response_object, \
    PREDICTED_MAX_TEMPERATURE_VALUES, PREDICTED_WEATHER_STATE, ACTUAL_WEATHER_STATE,
ACTUAL_MAX_TEMPERATURE_VALUES
from util.logging import ROOT_LOGGER_NAME

LOG = logging.getLogger(ROOT_LOGGER_NAME)
INDEX_PAGE = 'index-weather.html'
WEATHER_ENDPOINT = '/weather-predict'
WEATHER_APP_ADDRESS = 'http://127.0.0.1:5000' + WEATHER_ENDPOINT
COMMA_SPACE = ', '
ERROR_CATEGORY = 'error'
INFO_CATEGORY = 'info'

weather_web_app = Flask(__name__)
weather_web_app.secret_key = 'secret_for_session'


@weather_web_app.before_first_request
def _init_app():
    session.pop('_flashes', None)
    LOG.debug("%s - app has been successfully initialized" % datetime.datetime.now())
```

```python
@weather_web_app.route(WEATHER_ENDPOINT)
def weather_index():
    LOG.debug("%s - Request to weather home page" % datetime.datetime.now())

    today_response = ["+5", WEATHER_ICON_FOLDER + "Mostly cloudly" +
WEATHER_ICON_EXTENSION]
    predicted_response = generate_response_object(PREDICTED_MAX_TEMPERATURE_VALUES,
PREDICTED_WEATHER_STATE)
    actual_response = generate_response_object(ACTUAL_MAX_TEMPERATURE_VALUES,
ACTUAL_WEATHER_STATE)

    return render_template(INDEX_PAGE, today_response=today_response,
prediction_response=predicted_response,
                           actual_response=actual_response)
```

web/static/css/weather-style.css

```css
body {
    background: url("background-images/winter.png") no-repeat center center fixed;
    width: 100%;
    height: 100%;
    margin: 0 0;
    padding: 0 0;
    font-family: 'Source Sans Pro', sans-serif;
    font-weight: 200;
    opacity: 75%;
}

h3 {
    font-weight: 800;
}

.city-selected {
    position: relative;
    overflow: hidden;
    min-height: 200px;
    background: rgba(0, 0, 0, 0.5);
}

article {
    position: relative;
    z-index: 2;
    color: #ffffff;
    padding: 20px;

    display: -ms-flexbox;
    display: -webkit-flex;
    display: flex;
    -webkit-flex-direction: row;
    -ms-flex-direction: row;
    flex-direction: row;
```

```css
    -webkit-flex-wrap: wrap;
    -ms-flex-wrap: wrap;
    flex-wrap: wrap;
    -webkit-justify-content: space-between;
    -ms-flex-pack: justify;
    justify-content: space-between;
    -webkit-align-content: flex-start;
    -ms-flex-line-pack: start;
    align-content: flex-start;
    -webkit-align-items: flex-start;
    -ms-flex-align: start;
    align-items: flex-start;
}

.info .city {
    font-size: 24px;
    font-weight: 200;
    position: relative;


    -webkit-order: 0;
    -ms-flex-order: 0;
    order: 0;
    -webkit-flex: 0 1 auto;
    -ms-flex: 0 1 auto;
    flex: 0 1 auto;
    -webkit-align-self: auto;
    -ms-flex-item-align: auto;
    align-self: auto;
}

.city span {
    color: #ffffff;
    font-size: 13px;
    font-weight: bold;

    text-transform: lowercase;
    text-align: left;
}

.temp {
    font-size: 73px;
    display: block;
    position: relative;
    font-weight: bold;
}

.day:hover {
    background: rgba(0, 0, 0, 0.6);
    cursor: pointer;
    -webkit-transition: all 0.5s ease;
    -moz-transition: all 0.5s ease;
    -ms-transition: all 0.5s ease;
    -o-transition: all 0.5s ease;
    transition: all 0.5s ease;
```

```
}

.day {
    padding: 10px 0;
    text-align: center;

}

.day h1 {
    font-size: 14px;
    text-transform: uppercase;
    margin-top: 10px;
}

* {
    box-sizing: border-box;
}

.col {
    float: left;
    width: 50%;
    padding: 10px;
    background: rgba(0, 0, 0, 0.5);

}

.row:after {
    content: "";
    display: table;
    clear: both;
}

.data-header, .day {
    color: white;
    text-align: center;
}

a {
    color: inherit;
    text-decoration: underline;
}
```

web/static/js/date-time-utils.js

```
const MONTHS = ["January", "February", "March", "April", "May", "June", "July",
    "August", "September", "October", "November", "December"];

const WINTER_BACKGROUND = "background-images/winter.jpg";
const SPRING_BACKGROUND = "background-images/spring.jpg";
const SUMMER_BACKGROUND = "background-images/autumn.jpg";
const AUTUMN_BACKGROUND = "background-images/autumn.jpg";
```

```
export default class DateTimeUtils {
    static getMonthName(monthNumber) {
        return MONTHS[monthNumber];
    }

    static formatCurrentDate(dateNumber) {
        if (dateNumber < 10) {
            return '0' + dateNumber;
        }

        return dateNumber.toString();
    }

    static getCurrentDateTime(hours, minutes) {

        if (hours < 10) {
            hours = '0' + hours;
        }

        if (minutes < 10) {
            minutes = '0' + minutes;
        }

        return hours.toString() + ':' + minutes.toString();
    }

    static getSeasonBackgroundByMonthNumber(monthNumber) {
        if (monthNumber === 0 || monthNumber === 1 || monthNumber === 11) {
            return WINTER_BACKGROUND;
        } else if (monthNumber > 1 && monthNumber < 5) {
            return SPRING_BACKGROUND;
        } else if (monthNumber > 4 && monthNumber < 8) {
            return SUMMER_BACKGROUND;
        }

        return AUTUMN_BACKGROUND;
    }
}
```

web/static/js/weather-predict-script.js

```
import DateTimeUtils from "./date-time-utils.js";

export default class WeatherPrediction {
    constructor() {
        this.htmlBody = document.getElementById("htmlBody");
        this.currentDateTimeDiv = document.getElementById("current-time");
    }

    init() {
```

```
        this.showCurrentDateTime();
        this.updateBackground();
    }

    showCurrentDateTime() {
        let currentDate = new Date();

        let currentMonthName = DateTimeUtils.getMonthName(currentDate.getMonth());
        let currentDay = DateTimeUtils.formatCurrentDate(currentDate.getDate());
        let currentTime = DateTimeUtils.getCurrentDateTime(currentDate.getHours(),
currentDate.getMinutes());

        this.currentDateTimeDiv.innerText = currentTime + " - " + currentMonthName +
' ' + currentDay;
    }

    updateBackground() {
        let currentDate = new Date();

        let currentSeasonBackgroundImage =
DateTimeUtils.getSeasonBackgroundByMonthNumber(currentDate.getMonth());

        console.log(currentSeasonBackgroundImage);
    }
}
```

web/static/js/weather-predict-self-invoked.js

```
import WeatherPrediction from "./weather-predict-script.js";

(function () {
   let weatherPrediction = new WeatherPrediction();

   weatherPrediction.init();
})();
```

web/template/index-weather.html

```
<!DOCTYPE html>
<html lang="eu">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link rel="stylesheet" href="{{ url_for('static', filename='css/weather-
style.css') }}">
    <title>Weather forecast</title>
```

```
</head>
<body id="htmlBody">
<div class="container">
    <div class="row">
        <div class="col-sm-12">
            <div id="card">
                <div class="city-selected">
                    <article>
                        <div class="info">
                            <div class="city">Chernivtsi</div>
                            <div class="info" id="current-time"></div>
                            <br>
                            <div>Max temperature: <span
class="temp">{{today_response.0}}°</span></div>
                        </div>

                        <div class="icon text-center">
                            <img src="{{today_response.1}}" alt="Weather State icon"
width="200" height="200">
                        </div>

                    </article>
                </div>
            </div>
        </div>
        <div class="row">
            <div class="col">
                <div class="data-header">
                    <h3>Predicted weather forecast</h3>
                </div>
                {% for current_day, predictions in prediction_response.items() %}
                    <div class="day">
                        <h1>{{current_day}}</h1>
                        <div>{{predictions.0}}°</div>
                        <img width="64" height="64" src="{{predictions.1}}"
alt="Weather State icon">
                    </div>
                {% endfor %}
            </div>
            <div class="col">
                <div class="data-header">
                    <h3>Actual weather forecast from
                        <a target="_blank"
href="https://meteo.ua/ua/91/chernovtsyi">Meteo.ua</a>
                    </h3>
                </div>
                {% for current_day, predictions in actual_response.items() %}
                    <div class="day">
                        <h1>{{current_day}}</h1>
                        <div>{{predictions.0}}°</div>
                        <img width="64" height="64" src="{{predictions.1}}"
alt="Weather State icon">
                    </div>
                {% endfor %}
            </div>
```

```
        </div>
    </div>
</div>
<script type="module" src="{{ url_for('static', filename='js/weather-predict-self-invoked.js') }}"></script>
</body>
```

weather_parser/parser_main.py

```
import datetime
import time

from weather_parser.FileUtils.file_export import WeatherDataFileExport
from weather_parser.Models.city_weather_data import WeatherDataByCity
from weather_parser.Models.country_weather_data import CountryWeatherData
from weather_parser.Parser.html_beauty_soup_provider import HtmlBeautySoupProvider
from weather_parser.Parser.parse_data_provider import ParseDataProvider
from weather_parser.Parser.weather_parser import WeatherParser
from weather_parser.Parser.web_util import get_page, validate_page
from web.web_config import LOG

FILE_EXPORT_PATH_TEMPLATE = "C:\\projects\\{file_name}"

BASE_URL = "https://meteo.ua/archive"
TODAY = datetime.datetime.now()
DEFAULT_END_DATE = TODAY
DELAY_TIME_IN_SECONDS_WHEN_CONNECTION_EXCEPTION = 90


def run_parser(date_start):
    country_weather = get_country_weather_data("Ukraine", ParseDataProvider(BASE_URL,
date_start))
    WeatherDataFileExport(FILE_EXPORT_PATH_TEMPLATE, country_weather).export()


def get_country_weather_data(country, parse_data_provider):
    city_weather_list = []

    for city_code in parse_data_provider.city_codes_from_site:
        parse_data = parse_data_provider.build_parse_data(city_code)
        LOG.debug("Parsing URL: {site_url}".format(site_url=parse_data.page_url))
        LOG.debug("Parsing city: {parse_city}".format(parse_city=parse_data.city))

        city_weather_list.append(get_city_weather_data(parse_data))

    return CountryWeatherData(country, city_weather_list)


def get_city_weather_data(parse_data, current_weather_list=None,
end_date=DEFAULT_END_DATE):
    if current_weather_list is None:
```

```
        current_weather_list = []
    weather_list = current_weather_list

    try:
        while parse_data.date <= end_date:
            LOG.debug("Parsing date:
{current_parse_date}".format(current_parse_date=parse_data.date))
            page = get_page(parse_data.page_url)

            if validate_page(page):
                beauty_soup_provider = HtmlBeautySoupProvider(page)
                weather_parser =
WeatherParser(beauty_soup_provider.get_beauty_soup(), parse_data)
                weather_list.append(weather_parser.parse())
            else:
                LOG.error("Can`t get requested URL: {page_url}.
Skipping...".format(page_url=parse_data.page_url))

            page.close()
            parse_data.next_day()
    except Exception as parse_error:
        LOG.error("Some error just occurred\n. {exp_message} \nTrying again in
{delay_time} seconds".format(
            exp_message=str(parse_error),
delay_time=DELAY_TIME_IN_SECONDS_WHEN_CONNECTION_EXCEPTION))

        time.sleep(DELAY_TIME_IN_SECONDS_WHEN_CONNECTION_EXCEPTION)
        get_city_weather_data(parse_data, weather_list)

    return WeatherDataByCity(parse_data.city, weather_list)


def get_max_temperature(hourly_list):
    max_temp = 0

    for weather_data in hourly_list:
        if max_temp < weather_data.temperature_in_celcius:
            max_temp = weather_data.temperature_in_celcius

    return max_temp


def get_forecast_for_days(days_offset):
    temperature_forecast = []
    weather_state_forecast = []

    parse_data_provider = ParseDataProvider(BASE_URL, TODAY)
    parse_data =
parse_data_provider.build_parse_data(ParseDataProvider.CHERNIVTSI_CITY_CODE)
    weather_data = get_city_weather_data(parse_data, None, TODAY + days_offset)

    for hourly_list in weather_data.hourly_weather_list:
        temperature_forecast.append(get_max_temperature(hourly_list))
        weather_state_forecast.append(hourly_list[0][0].weather_state)
```

```
    return weather_state_forecast, temperature_forecast


def get_today_forecast():
    get_forecast_for_days(0)
```

weather_parser/Parser/base_parser.py

```
from abc import ABC, abstractmethod


class BaseParser(ABC):

    def __init__(self, html_beauty_soup):
        self.html_beauty_soup = html_beauty_soup

    @abstractmethod
    def parse(self):
        pass
```

weather_parser/Parser/html_beauty_soup_provider.py

```
from bs4 import BeautifulSoup


class HtmlBeautySoupProvider:

    SOUP_HTML_PARSE_TYPE = "html.parser"

    def __init__(self, page):
        self.page = page

    def get_beauty_soup(self):
        beauty_soup = BeautifulSoup(self.page.text, self.SOUP_HTML_PARSE_TYPE)
        beauty_soup.encode("utf-8")

        return beauty_soup




from weather_parser.Models.parse_data import ParseData


class ParseDataProvider:
```

```
    CHERNIVTSI_CITY_CODE = 91

    city_codes_from_site = [CHERNIVTSI_CITY_CODE]

    cities = {
        34: "kiev",
        150: "harkov",
        111: "odessa",
        164: "dnepr-dnepropetrovsk",
        169: "donetsk",
        172: "zaporoje",
        44: "lvov",
        134: "simferopol",
        122: "herson",
        112: "nikolaev",
        170: "lugansk",
        97: "kropivnitskiy-kirovograd",
        91: "chernovtsyi",
        83: "ujgorod",
        71: "vinnitsa",
        13: "lutsk",
        23: "sumyi",
        28: "rovno",
        31: "jitomir",
        47: "ternopol",
        49: "hmelnitskiy",
        56: "cherkassyi",
        58: "poltava",
        67: "ivano-frankovsk",
        6: "chernigov"
    }

    DEFAULT_CITY = "kiev"

    def __init__(self, base_url, date_from):
        self.base_url = base_url
        self.date_from = date_from

        self.CURRENT_DATE = date_from

    def build_parse_data(self, city_code):
        city = self.cities.get(city_code, self.DEFAULT_CITY)
        page_url = (self.base_url +
"/{city_code}/{city}/{date}").format(city_code=city_code, city=city,

date=self.CURRENT_DATE)

        return ParseData(page_url, self.date_from, city)
```

weather_parser/Parser/weather_parser.py

```
from bs4 import BeautifulSoup

from weather_parser.Models.weather_data import WeatherData
from weather_parser.Models.weather_data_indexes import WeatherDataIndexes
from weather_parser.Models.weather_state import WeatherState
from weather_parser.Models.wind_data import WindData
from weather_parser.Models.wind_direction import WindDirection
from weather_parser.Parser.base_parser import BaseParser


class WeatherParser(BaseParser):
    def __init__(self, html_beauty_soup, parse_data):
        super().__init__(html_beauty_soup)
        self.parse_data = parse_data
        if isinstance(html_beauty_soup, BeautifulSoup):
            self.html_beauty_soup = html_beauty_soup
        else:
            raise ValueError("[html_beauty_soup] parameter must be type of
BeautifulSoup")

    def parse(self):
        # remove first one since its for table header values
        archive_table_data_list =
self.html_beauty_soup.find(class_="archive_table").find_all("tr")[1:]
        weather_data_list = []

        for tr in archive_table_data_list:
            weather_data_list.append(self.get_weather_data(tr))

        return weather_data_list

    def get_weather_data(self, tr):
        td_list = tr.find_all("td")
        weather_data_list = self.parse_weather_data(td_list)

        start_time = weather_data_list[WeatherDataIndexes.TIME_INDEX]
        weather_state =
self.parse_weather_state(td_list[WeatherDataIndexes.WEATHER_STATE_INDEX])
        temperature_in_celcius = self.parse_temperature_without_celcius(
            weather_data_list[WeatherDataIndexes.TEMPERATURE_INDEX])
        wind_data = self.parse_wind_data(td_list[WeatherDataIndexes.WIND_DATA_INDEX])
        atmosphere_pressure =
weather_data_list[WeatherDataIndexes.ATMOSPHERE_PRESSURE_INDEX]
        humidity = weather_data_list[WeatherDataIndexes.HUMIDITY_INDEX]

        return WeatherData(start_time, weather_state, temperature_in_celcius,
wind_data, atmosphere_pressure, humidity,
                           self.parse_data.date, self.parse_data.city)

    def parse_temperature_without_celcius(self, temperature_value):
        return str(temperature_value).replace(self.parse_data.CELCIUS_SYMBOL, '')

    def parse_weather_data(self, td_list):
        data_list = []
```

```
        for td in td_list:
            data_list.append(td.find(class_="vl_child").span.string)

        return data_list

    def parse_weather_state(self, state_td):
        return
WeatherState.get_weather_state(state_td.find(class_="ov_hide").span.span.string)

    def parse_wind_data(self, wind_td):
        wind_direction = WindDirection.get_wind_direction(wind_td.img["title"])
        wind_speed = wind_td.span.span.next.next  # ignoring first value since it is
the image representing wind arrow

        return WindData(wind_speed, wind_direction)
```

weather_parser/Parser/web_util.py

```
import requests

STATUS_CODE_OK = 200


def get_page(page_url):
    return requests.get(page_url)


def validate_page(page):
    if not isinstance(page, requests.Response):
        return False

    if page.status_code != STATUS_CODE_OK:
        return False

    return True
```

weather_parser/Models/city.py

```
class City:
    def __init__(self, city):
        self.id = None
        self.city = city
```

weather_parser/Models/city_weather_data.py

```
class WeatherDataByCity:
    def __init__(self, city, hourly_weather_list):
        self.hourly_weather_list = hourly_weather_list if hourly_weather_list is not
None else []
        self.city = city

    def add(self, hourly_weather):
        self.hourly_weather_list.append(hourly_weather)
```

weather_parser/Models/country.py

```
class Country:
    def __init__(self, country):
        self.id = None
        self.country = country
```

weather_parser/Models/country_weather_data.py

```
class CountryWeatherData:
    def __init__(self, country, city_hourly_data_list):
        self.city_hourly_data_list = city_hourly_data_list if city_hourly_data_list
is not None else []
        self.country = country

    def add(self, city_weather_data):
        self.city_hourly_data_list.append(city_weather_data)
```

weather_parser/Models/parse_data.py

```
import datetime


class ParseData:

    CELCIUS_SYMBOL = '°C'

    def __init__(self, page_url, date, city):
```

```
        self.page_url = page_url
        self.date = date
        self.city = city

    def next_day(self):
        self.date += datetime.timedelta(days=1)
```

weather_parser/Models/weather_data.py

```
from weather_parser.Models.weather_state import WeatherState


class WeatherData:
    def __init__(self, start_time=None, weather_state=None,
temperature_in_celcius=None,
                 wind_data=None, atmosphere_pressure=None, humidity_in_percent=None,
date=None, city=None):
        self.id = None
        self.start_time = start_time
        self.weather_state = weather_state
        self.temperature_in_celcius = temperature_in_celcius
        self.wind_data = wind_data
        self.atmosphere_pressure = atmosphere_pressure
        self.humidity_in_percent = humidity_in_percent
        self.date = date
        self.city = city

    def get_separated_string(self, separator):
        return str(self.date) + separator + self.start_time + separator \
               + (self.weather_state.value if isinstance(self.weather_state,
WeatherState) else self.weather_state) \
               + separator + self.temperature_in_celcius + separator +
self.wind_data.get_separated_string(separator) \
               + separator + self.atmosphere_pressure + separator +
self.humidity_in_percent + separator + self.city
```

weather_parser/Models/weather_data_indexes.py

```
from enum import IntEnum


class WeatherDataIndexes(IntEnum):
    TIME_INDEX = 0
    WEATHER_STATE_INDEX = 1
    TEMPERATURE_INDEX = 2
    WIND_DATA_INDEX = 3
```

```
    ATMOSPHERE_PRESSURE_INDEX = 4
    HUMIDITY_INDEX = 5
```

weather_parser/Models/weather_state.py

```
from enum import Enum


class WeatherState(Enum):
    SUNNY = "Sunny"
    CLEAR = "Clear"
    RAIN = "Rain"
    LIGHT_RAIN_AND_ICE = "Light rain and ice"
    CLOUDY = "Cloudy"
    PARTLY_CLOUDY = "Partly cloudy"
    PARTLY_SUNNY = "Partly sunny"
    LIGHT_SNOW = "Light snow"
    OVERCAST = "Overcast"
    SNOW = "Snow"
    HAZE = "Haze"
    FOG = "Fog"
    HEAVY_SNOW = "Heavy snow"
    HEAVY_RAIN_WITH_SNOW = "Heavy rain with snow"
    HEAVY_RAIN = "Heavy rain"
    SNOW_AND_FOG = "Snow and fog"
    LIGHT_SNOWSTORM = "Light snowstorm"
    STRONG_SNOWSTORM = "Strong snowstorm"
    ICE = "Ice"
    UNDEFINED = "Undefined"

    @staticmethod
    def get_weather_state(state):
        switcher = {
            "Слабый снег": WeatherState.LIGHT_SNOW,
            "Сплошная облачность": WeatherState.OVERCAST,
            "Небольшая облачность": WeatherState.OVERCAST,
            "Снег": WeatherState.SNOW,
            "Дымка": WeatherState.HAZE,
            "Туман": WeatherState.FOG,
            "Ясно": WeatherState.SUNNY,
            "Переменная облачность": WeatherState.PARTLY_CLOUDY,
            "Сильный снег": WeatherState.HEAVY_SNOW,
            "Ливневый дождь со снегом": WeatherState.HEAVY_RAIN_WITH_SNOW,
            "Ливневый дождь": WeatherState.HEAVY_RAIN,
            "Снег, туман": WeatherState.SNOW_AND_FOG,
            "Слабый дождь, гололед": WeatherState.LIGHT_RAIN_AND_ICE,
            "Слабая метель": WeatherState.LIGHT_SNOWSTORM,
            "Сильная метель": WeatherState.STRONG_SNOWSTORM,
            "Гололед": WeatherState.ICE
        }

        return switcher.get(state, state)
```

weather_parser/Models/wind_data.py

```
from weather_parser.Models.wind_direction import WindDirection


class WindData:
    def __init__(self, wind_speed, wind_direction):
        self.id = None
        self.wind_speed = wind_speed
        self.wind_direction = wind_direction

    def get_separated_string(self, separator):
        return self.wind_speed + separator + self.wind_direction.value if
self.wind_direction is not None \
            else WindDirection.UNKNOWN
```

weather_parser/Models/wind_direction.py

```
from enum import Enum


class WindDirection(Enum):
    NORTH = "North"
    NORTHEAST = "Northeast"
    EAST = "East"
    SOUTHEAST = "Southeast"
    SOUTH = "South"
    SOUTHWEST = "Southwest"
    WEST = "West"
    NORTHWEST = "Northwest"
    UNKNOWN = "Unknown"

    @staticmethod
    def get_wind_direction(wind_direction):
        switcher = {
            "Северный": WindDirection.NORTH,
            "Северо-восточный": WindDirection.NORTHEAST,
            "Восточный": WindDirection.EAST,
            "Юго-восточный": WindDirection.SOUTHEAST,
            "Южный": WindDirection.SOUTH,
            "Юго-западный": WindDirection.SOUTHWEST,
            "Западный": WindDirection.WEST,
            "Северо-западный": WindDirection.NORTHWEST
        }

        return switcher.get(wind_direction, wind_direction)
```

weather_parser/FileUtils/file_export.py

```
import codecs
import datetime
import os

from web.web_config import LOG


class WeatherDataFileExport:
    DEFAULT_SEPARATOR = ','
    NEW_LINE_SEPARATOR = '\n'
    FILE_NAME_TEMPLATE = "weather_data_{current_time}.txt"

    def __init__(self, file_path_template, weather_data):
        self.file_path_template = file_path_template
        self.weather_data = weather_data

    def get_file_path(self):
        current_date_time = datetime.datetime.now().strftime("%Y%m%d_%H%M%S")

        return 
self.file_path_template.format(file_name=self.FILE_NAME_TEMPLATE.format(current_time=
current_date_time))

    def export(self, separator=DEFAULT_SEPARATOR):
        file_path = self.get_file_path()

        if os.path.exists(file_path):
            LOG.debug("File {file_path} already exists, 
skipping...".format(file_path=file_path))
            return
        export_file = None

        try:
            city_weather_list = self.weather_data.city_hourly_data_list
            export_file = codecs.open(file_path, 'w', "utf-8")

            for city_weather in city_weather_list:
                export_file.write(self.get_separated_string(city_weather, separator))
        finally:
            if export_file is not None:
                export_file.close()

    def get_separated_string(self, city_weather, separator):
        result_string = ""

        for daily_weather in city_weather.hourly_weather_list:
            for hourly_weather in daily_weather:
                result_string += hourly_weather.get_separated_string(separator) + 
self.NEW_LINE_SEPARATOR

        return result_string
```

weather_parser/FileUtils/missing_value_export.py

```
import codecs


DEFAULT_FILE_PATH_FOR_EXPORT = "C:\\projects\\missing_weather_values.txt"

missing_state_list = []
missing_direction_list = []


def add_state_if_not_exist(state_value):
    if state_value in missing_state_list:
        return

    missing_state_list.append(state_value)


def add_direction_if_not_exist(direction_value):
    if direction_value in missing_direction_list:
        return

    missing_direction_list.append(direction_value)


def export_missing_values_to_file(file_path=DEFAULT_FILE_PATH_FOR_EXPORT):
    export_file = codecs.open(file_path, "w", "utf-8")
    try:
        if len(missing_state_list) != 0:
            export_missing_values(export_file, missing_state_list, "Missing weather
state values:\n")
        if len(missing_direction_list) != 0:
            export_missing_values(export_file, missing_direction_list, "Missing wind
directions values:\n")
    finally:
        export_file.close()


def export_missing_values(export_file, missing_values, title):
    export_file.write(title)

    for value in missing_values:
        export_file.write(value + "\n")
```