НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
Інститут прикладного системного аналізу
Кафедра математичних методів системного аналізу

До захисту допущено
Завідувач кафедри
_____ Оксана ТИМОЩУК
«07» 06 2021 р.

Дипломна робота

на здобуття ступеня бакалавра
за освітньо-професійною програмою «Системний аналіз і управління»
спеціальності 124 "Системний аналіз"
на тему: «Система для аналізу багатовимірних часових рядів»

Виконав:
Студент IV курсу, групи КА-77
Буханевич Родіон Михайлович _____

Керівник:
професор, д.т.н.,
Бідюк Петро Іванович _____

Консультант з економічного розділу:
доцент кафедри ТПЕ
к.е.н., доцент Надія Василівна Рощіна _____

Консультант з нормоконтролю:
доцент кафедри ММСА
к.т.н., доцент Анатолій Єпіфанович Коваленко _____

Рецензент:
професор, д.т.н.,
Архипов Олександр Євгенович _____

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

Київ-2021

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Інститут прикладного системного аналізу
Кафедра математичних методів системного аналізу

Рівень вищої освіти – перший (бакалаврський)
Спеціальність – 124 "Системний аналіз"
Освітньо-професійна програма «Системний аналіз і управління»

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____Оксана ТИМОЩУК
«___»_____20__ р.

ЗАВДАННЯ
на дипломну роботу студенту
Буханевичу Родіону Михайловичу

1. Тема роботи «Система для аналізу багатовимірних часових рядів», керівник роботи д.т.н., професор Бідюк Петро Іванович затверджені наказом по університету від «___»_____ 2021 р. №_____
2. Термін подання студентом роботи 07.06.2021 р.
3. Вихідні дані до роботи:
Моделі прогнозування багатовимірних часових рядів, коінтегровані часові ряди, що описують рух взаємопов'язаних процесів.
4. Зміст роботи:
Аналітичний огляд задачі прогнозування та властивостей багатовимірних процесів, застосування методів регресійного аналізу та нейронних мереж для побудови прогнозів, порівняльний аналіз оцінок прогнозів.
5. Перелік ілюстративного матеріалу:
Презентація до захисту роботи.
6. Консультанти розділів роботи:

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Економічний	Рощіна Н.В., доцент кафедри ТПЕ		

7. Дата видачі завдання: 01.02.2021р

Календарний план

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітка
1	Формулювання тематики (напрямку) дослідження.	15.01.2021 – 31.01.2021	виконано
2	Огляд літератури за темою	01.02.2021 – 28.02.2021	виконано

	досліджння.		
3	Формулювання задач дослідження, розробка плану роботи.	01.03.2021 – 15.03.2021	виконано
4	Аналіз відомих результатів стосовно тематики дослідження.	15.03.2021 – 01.04.2021	виконано
5	Збір статистичних даних.	01.04.2021 – 15.04.2021	виконано
6	Затвердження теми дипломної роботи.	15.04.2021	виконано
7	Розробка програмного продукту.	01.04.2021 – 01.05.2021	виконано
8	Виконання обчислювальних експериментів, аналіз та оформлення результатів.	01.05.2021 – 14.05.2021	виконано
9	Оформлення пояснювальної записки	14.05.2021 – 25.05.2021	виконано

Студент Родіон БУХАНЕВИЧ

Керівник Петро БІДЮК

РЕФЕРАТ

Дипломна робота: 132 с., 22 рис., 10 табл., 18 джерел, 3 додатки.

ПРОГНОЗУВАННЯ БАГАТОВИМІРНИХ ЧАСОВИХ РЯДІВ, МОДЕЛЬ ВАР, НЕЙРОННІ МЕРЕЖІ, АНСАМБЛІ ДЕРЕВ РІШЕНЬ, МЕРЕЖА LSTM, БАГАТОВИМІРНІ ПРОЦЕСИ.

Мета роботи – аналіз та порівняння результатів прогнозів багатовимірних часових рядів, які є взаємопов’язаними процесами.

В роботі наведено огляд відомих методів прогнозування. Описано статистичні тести, що використовуються для класифікації процесів.

Досліджено та побудовано моделі регресійного аналізу та моделі на базі машинного навчання та нейронних мереж, що використовуються для прогнозування часових рядів, описано критерії якості оцінок прогнозів.

Розроблено програмний продукт на мові програмування Python 3.8. у середовищі Visual Code, що дозволяє будувати прогнози будь-яких багатовимірних часових рядів за допомогою моделі ВАР, ансамблю дерев рішень, нейронної мережі LSTM.

Проведено порівняльний аналіз результатів, отриманих за допомогою різних методів.

ABSTRACT

Bachelor thesis: 132 p., 22 figures, 10 tables, 18 sources, 3 appendices.

FORECASTING MULTIVARIATE TIME SERIES, VAR MODEL, NEURAL NETWORKS, ENSEMBLE DECISION TREES, LSTM NETWORK, MULTIVARIATE PROCESSES.

The purpose of the work is to analyze and compare the results of forecasts of multidimensional time series, which are cointegrated processes.

The paper provides an overview of known forecasting methods. Describes the statistical tests used to classify processes.

Regression analysis models and models based on machine learning and neural networks used for time series prediction have been studied and built, and quality criteria for forecast estimates have been described.

A software product has been developed in the Python 3.8 programming language in the Visual Code environment, which allows you to build predictions of any multidimensional time series using the VAR model, the ensemble of decision trees, the neural network LSTM.

A comparative analysis of the results obtained using different methods is carried out.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

АКФ – автокореляційна функція;

АР – авторегресія;

АРКС – авторегресія з ковзним середнім;

МНК – метод найменших квадратів;

ЧАКФ – часткова автокореляційна функція;

ЧР – часовий ряд;

AIC – інформаційний критерій Акайке;

LSTM – архітектура нейронної мережі Довга короткочасна пам’ять;

XGBOOST – модель градієнтного бустінгу над деревами рішень;

LGBM – полегшена модель градієнтного бустінгу над деревами рішень;

MAPE – середня абсолютна похибка у відсотках;

ME – середня похибка;

MPE – середня похибка у відсотках;

MSE – середнє квадратів похибок;

RNN – клас рекурентних нейронних мереж;

RSME – середньоквадратична похибка;

R^2 – коефіцієнт детермінації.

ВСТУП

Існують різні підходи та методи побудови прогнозів. Науково-технічний розвиток сприяє поглибленню знань про методи аналізу та передбачення часових рядів. Сфера застосування методів прогнозування часових рядів є дуже широкою, приведемо декілька прикладів задач, які потребують застосування прогнозу:

а) рух цін акцій на фондових ринках;
б) попит товарів;
в) врожайність;
г) кількість несправностей обладнання;

Наведені вище задачі передбачають прогнозування одразу декількох показників для досягнення найбільшої якості. Наприклад, дослідження щомісячного рівня захворюваності на рак у Сполучених Штатах за останні 10 років може включати багато сотень або тисяч часових рядів, залежно від того, чи досліджуємо ми рівень захворюваності на рак у штатах, містах або округах. Для цього потрібні методи аналізу багатовимірних часових рядів. Ці методи відрізняються від стандартної статистичної теорії і методів, заснованих на випадковій вибірці, припускаючи незалежність показників.

В рамках даної роботи ми обмежимось лише дослідженням класичних методів прогнозування багатовимірних часових рядів та сучасними методами дослідження задач регресії, які можна застосувати до передбачення часових рядів.

Перший розділ присвячено аналізу властивостей багатовимірних часових рядів різних процесів, огляду інструментів для класифікації цих процесів, методів прогнозування.

У другому розділі описано методику побудови моделей процесів на базі статистичних методів, моделей машинного навчання та нейронних мереж, розглянуто архітектури штучних нейронних мереж, які можуть використовуватись для прогнозування багатовимірних часових рядів.

У третьому розділі сформульовано вимоги до програмного забезпечення, розроблено функціональну схему програми, детально описано побудову прогнозів досліджуваних процесів.

У четвертому розділі проведено функціонально-вартісний аналіз програмного продукту.

РОЗДІЛ 1 ФОРМАЛІЗОВАНЕ ПРЕДСТАВЛЕННЯ ВЕКТОРНИХ ПРОЦЕСІВ

1.1 Особливості задачі прогнозування

Часовий ряд – набір послідовних вимірів значень певної змінної, як правило, часові інтервали між вимірами є однаковими. Окремі спостереження у часовому ряді також називають рівнями цього ряду. Таким чином фактично часовий ряд складається з двох компонент – час та рівень ряду в цей момент часу. Процес чи модель багатовимірного часового ряду характеризується своїм середнім вектором, функцією кореляційної матриці і частковою кореляційною матричною функцією. В аналізі часових рядів виділяють такі основні компоненти:

а) Тренд – тенденція зміни показників часового ряду. Тренди можуть бути описані різними функціями - лінійними, ступінчастими, експоненціальними і т.д. Тип тренду встановлюють на основі даних тимчасового ряду за допомогою оцінок показників динаміки ряду.

б) Сезонність – періодичні коливання, які спостерігаються на часових рядах. Сезонність характерна для економічних часових рядів. В економіці багато явищ характеризуються періодично повторюваними сезонними ефектами. Наприклад, роздрібні продажі як правило ростуть з наближенням новорічних свят, а після них показують спад. Відповідно тимчасові ряди, що відображають ці сезонні ефекти, містять періодичні коливання.

1.2 Тести на стаціонарність

N-вимірний векторний процес $Z_t = [Z_{1,t}, Z_{2,t}, \ldots, Z_{n,t}]$ є стаціонарним процесом, якщо кожна з його складових - часових рядів є одновимірним стаціонарним процесом. Стаціонарний процес задовольняє наступним умовам:

$$E(y(k)) = E(y(s)) = \text{const.}$$
$$E[y(k) - E(y(k)]^2 = E[y(s) - E(y(s))]^2 = \text{const.}$$
$$\text{cov}[y(s),y(k)] = E[y(k) - E(y(k))][(y(s) - E(y(s))] = \text{const.}$$

Критерій Діккі-Фуллера перевірки наявності одиничних коренів. Задача перевірити основну гіпотезу H0: a = 1 в моделі авторегресії першого порядку, яка виражається наступною формулою:

$$y_t = ay_{t-1} + \varepsilon_t.$$

де y_t – вимір відповідної змінної в момент часу *k;*

ε_t – вплив збурень та випадкових коливань, які ми не можемо виміряти.

Таким чином, гіпотеза про стаціонарність часового ряду y_t полягає в перевірці основної гіпотези виду H0: a = 1. На наступному етапі оцінюється модель після переходу до перших різниць $\Delta y_t=\delta y_{t-1}+\varepsilon_t$, де $\delta=a-1$. Перевірка основної гіпотези виду H0: a = 1 для моделі авторегресії першого порядку аналогічна перевірці гіпотези H0: δ = 0 для отриманої моделі, тобто процес не стаціонарний. Перевірка цієї гіпотези може здійснюватися для трьох типів регресійних рівнянь:

– $\Delta y_t=\delta y_{t-1}+\varepsilon_t$;

– $\Delta y_t = a + \delta y_{t-1} + \varepsilon_t$;

– $\Delta y_t = a + \delta y_{t-1} + \beta t + \varepsilon_t$;

Перша модель є моделлю випадкового тренда, у другій моделі вільний член a є коефіцієнтом випадкового тренда. У третю модель включені і коефіцієнт випадкового тренда, і коефіцієнт лінійного часового тренда βt. Спостережуване значення t-критерію для перевірки основної гіпотези виду H0: β = 0 розраховують за формулою:

$$t_{спост} = \frac{\delta *}{\omega(\delta *)}$$

де $\omega(\delta *)$ – середнє квадратичне відхилення оцінки δ^*.

Однак критичне значення t-критерію в даному випадку не можна визначити по таблиці розподілу Стьюдента. Провели дослідження, в результаті яких визначили критичні значення t-критерію для перевірки гіпотези H0: δ = 0 в залежності від виду моделі регресії і обсягу вибірки.

1.3 Кореляційні і часткові кореляційні матричні функції

Значення часового ряду можуть мати залежність з попередніми значеннями. Така кореляція називається автокореляція, вона представляє часову динаміку ряду. Автокореляційна функція вимірює кореляцію між даними ряду в різних точках. Наприклад, маємо часовий ряд X_1, X_2, …, X_n. Тоді за формулою:

$$\gamma(h) = \frac{1}{n}\sum_{t=1}^{n-|h|} (X_{t+|h|} - X')(X_t - X') \text{ , } -n < h < n$$

де $x' = \frac{1}{n}\sum_{t=1}^{n} X_t$.

Автокореляційна функція буде дорівнювати:

$$P(h) = \frac{\gamma(h)}{\gamma(0)} .$$

Часткова кореляція - це кореляція між двома змінними за припущенням, що ми знаємо і враховуємо значення деяких інших наборів змінних. ЧАКФ обчислюється рекурентно за наступною формулою:

$$\text{ЧАКФ}(X_i, k) = \frac{\text{Cov}\left([X(i)|X(i-1),X(i-2),\dots,X(i-k+1)],[X(i-k)|X(i-1),X(i-2),\dots,X(i-k+1)]\right)}{\delta\left([X(i)|X(i-1),X(i-2),\dots,X(i-k+1)]\right)\ \delta\left([X(i-k)|X(i-1),X(i-2),\dots,X(i-k+1)]\right)}$$

ЧАКФ та АКФ будують для визначення порядку авторегресійної частини моделі та порядку ковзного середнього потрібно відповідно. Для того, щоб визначити порядок авторегресійної частини по корелограммі АКФ та ЧАКФ потрібно порахувати максимальну кількість лагів, які виходять за певний рівень інтервалу. Покажемо приклад АКФ та ЧАКФ коррелограми.

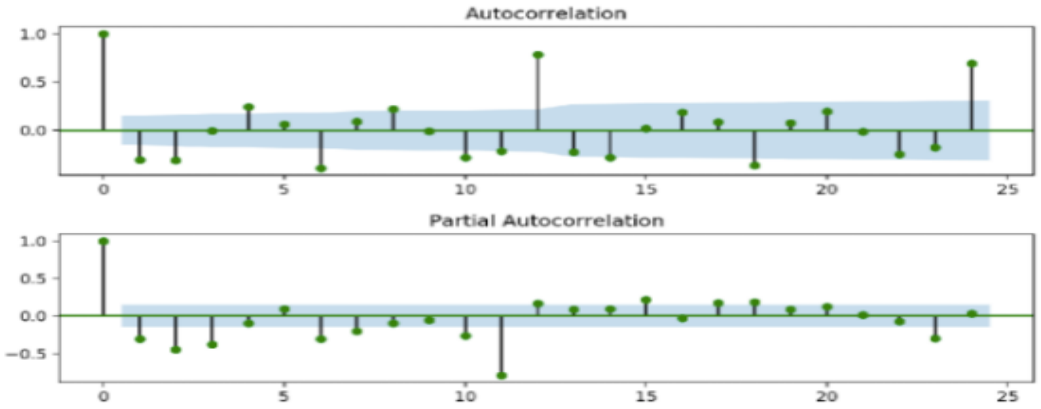


Рисунок 1. АКФ та ЧАКФ функції стаціонарного процесу

Щоб оцінити кількість параметрів авторегресійної частини, нам потрібно глянути на графік ЧАКФ. По-перше, значення із зсувом 0 завжди відображатиме кореляцію 1, оскільки ми оцінюємо кореляцію між поточним значенням із самим собою. На графіку є синя область, що представляє довірчий інтервал. Рахуємо скільки значень функції знаходиться вище або нижче довірчого інтервалу, перш ніж наступне значення знову буде в межах синьї області. Отже, дивлячись на графік ЧАКФ вище, ми можемо порахувати, що порядок авторегресійної частини нашої моделі дорівнює 3, оскільки лаги 1, 2 і 3 виходять за межі довірчого інтервалу, а зсув 4 – в межах довірчого інтервалу області. Порядок ковзного середнього нашої моделі дорівнює 2, бо значення, які відповідають зсувам 1 та 2 виходять за межі довірчого інтервалу, а значення за лагом 3 знаходиться в межах області.

Нехай $Z_t = [Z_{1,t}, Z_{2,t}, \ldots, Z_{n,t}]$, $t = 0, \pm 1, \pm 2, \ldots$ - n-вимірний стаціонарний вектор, який містить дійсні числі. Такий, що $E(Z_{i,t}) = \mu_i$, де μ_i – константа для всіх $i = 1, 2, ..., n$ та перехресні коваріації між $Z_{i,t}$ та $Z_{i,s}$, s, для всіх $i = 1, 2, \ldots,$ m та $j = 1, 2, \ldots,$ m та $j = 1, 2, ...,n$, є функціями різниць часу $(s – t)$. Маємо вектор математичного сподівання $E(Z_t) = \mu = (\mu_1, \mu_2, \ldots, \mu_n)$ і матрицю коваріації k-го лагу:

$$\Gamma(k) = \mathrm{Cov}\{ Z_t, Z_{t+k} \} = E[(Z_t - \mu)(Z_{t+k} - \mu)^T]$$

де $\mu = E(Z_t)$

Функція Γ(k) називається матричною функцією коваріації векторного процесу Z. Матрична функція коваріації має такі властивості:

- $\Gamma(k) = \Gamma(-k)$
- $|\gamma_{i,j}(k)| \leq [\gamma_{i,i}(0)\gamma_{j,j}(0)]^{1/2}$, для всіх i, j = 1, …, m
- Матрична функція коваріації є позитивно визначеною.

1.3 Векторні авторегресійні процеси

Застосування VAR-моделей з високою розмірністю потенційно здатне зробити більш коректним якість прогнозу. Часто висловлюються припущення про те, що ряди, які представляють собою вектор деяких змінних, є стохастичними процесами. Таким чином, модель VAR описує якусь спільну взаємодію змінних за обраний період часу. Одне з найпростіших визначень, яке можна дати VAR-моделі - це модель, яка одночасно описує значення спільно залежних змінних через зміну значень попередніх значень і значень інших спільно залежних змінних.

1.4 Нестаціонарні векторні авторегресійні процеси

У випадку одновимірного часового ряду нестаціонарний часовий ряд зводиться до стаціонарного. Вони все ще можуть бути використані у багатовимірній моделі часового ряду. Однак слід зазначити, що ці перетворення потрібно застосовувати до окремої компоненти, оскільки не всі

серії компонентів можна звести до стаціонарних однаковою кількістю перетворень. Для зручності після використання належних перетворень до серії компонентів ми використаємо наступну презентацію для нестаціонарної векторної моделі часових рядів:

$$\Phi_p(B)\, D(B)\, Z_t = \Theta_p(B)\, a_t ,$$

де $D(B) = \begin{matrix} (1-B)^{d_1} & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & (1-B)^{d_m} \end{matrix}$;

тест Діккі і Фуллера на одиничні корні може бути використаний для визначення порядку d_i.

1.5 Коінтегровані часові ряди. Тести на коінтегрованість

Перед тим як застосовувати до декількох пар часових рядів ту чи іншу модель VAR чи нейронну мережу ми повинні перевірити чи дійсно ці ряди пов’язані між собою. Першою в голову приходить ідея перевірити ряди на кореляцію. Але цей метод не працює тому що кореляція не дає нам достатньо інформації про взаємозв’язок рядів в довгостроковій перспективі. Тому для ідентифікації зв'язаних між собою пар використовується термін коінтегрованності. Нехай у нас є два часових ряди x_t та y_t та їх лінійна комбінація $y_t - \beta x_t$ є стаціонарним рядом 0 порядку. Тоді ці ряди називаються коінтегрованими. Іншими словами коінтегрованність - це регресія нестаціонарних рядів. Перевірити два ряди на коінтегрованність можливо за допомогою двох тестів: тест Енгла-Гренжера та тест Йохансена. Щоб зрозуміти чи є два ряди коінтегрованими за тестом Енгла-Гренжера нам потрібно провести тестування нульової гіпотези:

$$H_0 : \varepsilon_t \text{ - не стаціонарний}$$

$$H_1 : \varepsilon_t \text{ - стаціонарний,}$$

де $\varepsilon_t = y_t - \beta x_t$;

Спочатку за допомогою МНК оцінюється компонента β, потім перевіряється ε_t на стаціонарність. Розглянемо алгоритм роботи тесту Йохансена. Якщо тести мають однаковий порядок інтегрованості беруться залишки моделі VAR(p), які можна представити у вигляді формул:

$$y_t = \sum_{i=1}^{p} A_i y_{t-i} + Bx_t + \varepsilon_t$$

$$\Delta y_t = c + \Pi y_{t-1} + \Gamma_1 \Delta y_{t-1} + \ldots + \Gamma_p \Delta y_{t-p} + \varepsilon_t$$

$$\Pi = \sum_{i=1}^{p} A_i - I$$

$$\Gamma_i = - \sum_{i=1}^{p} A_i$$

Послідовна процедура Йохансена заключається в тому, що ранг коінтеграції дорівнює рангу матриці і використовується тест відношення правдоподібності від рангу 0 до к-1. Якщо гіпотеза не відхилюється для рангу 0, то коінтегрованності нема.

1.6 Висновки та постановка задачі

Визначено місце задачі прогнозування в практичній діяльності людини та актуальність прогнозування багатовимірних процесів. Окреслено предмет

досліджень даної роботи, визначені інструменти для дослідження процесів, проаналізовано на основі літератури та електронних джерел особливості та властивості побудови прогнозів. Наведено класифікацію процесів, проведено огляд статистичних тестів для класифікації процесів. Наведено класифікацію методів прогнозування.

Основні результати дослідження:

- Задача прогнозування виникає практично у всіх сферах діяльності, прогнозування багатовимірних процесів один із основних факторів, що визначає ефективність стратегічного планування.

- Більшість процесів. як правило. є нестаціонарними та нелінійними, що значно ускладнює задачу побудови прогнозів, звужує спектр методів, що можуть застосовуватись.

- Не існує ідеальних методів прогнозування, та одного підходу до прогнозування всіх багатовимірних процесів. Кожен випадок потребує власного аналізу та підходу.

- Перспективним напрямом дослідження є застосування комбінованих моделей, що дозволяє частково нівелювати недоліки різних методів та зберегти їх переваги.

Метою даної роботи є побудова математичних моделей для побудови прогнозів багатовимірних часових рядів, що передбачає наступне:

- Вибір статистичних даних, що описують багатовимірні процеси.
- Вибір декількох моделей різних класів для побудови прогнозів, зокрема регресійних моделей та моделей на основі нейронних мереж.
- Попередню обробку даних, використання статистичних критеріїв для дослідження характеристик процесу.
- Формулювання вимог до програмного забезпечення та розробка програмного забезпечення для побудови досліджуваних моделей та прогнозів.
- Аналіз якості отриманих прогнозів та інтерпретацію результатів, порівняння результатів, отриманих за допомогою різних підходів.

- Аналіз результатів проведеного дослідження та визначення можливих напрямків подальших досліджень.

РОЗДІЛ 2 Багатовимірні регресійні моделі і прогнозування на їх основі

2.1 Багатовимірні моделі VAR і VARMA

Нехай $X_т = (x_{1т}, x_{2т}, \dots, x_{пт})^T$ - n-вимірний вектор значень часових рядів. Векторний авторегресійний процес або модель порядку p, скорочена до VAR(p), задається формулою:

$$X_t = c + \Phi_1 X_{t-1} + \dots + \Phi_p X_{t-p} + \varepsilon_t$$

де ε_t - n-вимірний векторний процес, який є білим шумом

Φ_i – матриця коефіцієнтів n x n.

Наприклад, модель VAR(2) буде мати вигляд:

$$\begin{pmatrix} x_{1т} \\ x_{2т} \end{pmatrix} = \begin{pmatrix} c_1 \\ c_2 \end{pmatrix} + \begin{pmatrix} \phi_{11}^1 & \phi_{12}^1 \\ \phi_{21}^1 & \phi_{22}^1 \end{pmatrix} \begin{pmatrix} x_{1т-1} \\ x_{2т-1} \end{pmatrix} + \begin{pmatrix} \phi_{11}^2 & \phi_{12}^2 \\ \phi_{21}^2 & \phi_{22}^2 \end{pmatrix} \begin{pmatrix} x_{1т-2} \\ x_{2т-2} \end{pmatrix} + \begin{pmatrix} c_{1т} \\ c_{2т} \end{pmatrix},$$

де $\Phi(L) = I_n - \Phi_1 L - \dots - \Phi_p L^p$;

Хоча векторні авторегресійні моделі із ковзним середнім (VARMA) мають теоретичні та практичні переваги порівняно з простішими векторними авторегресивними моделями (VAR), моделі VARMA рідко використовуються в прикладній макроекономічній роботі. Однією з ймовірних причин є те, що вони набагато складніші, і немає сенсу жертвувати часом обчислення заради отриманого результату. Процес з використанням регресійного ковзного середнього VARMA(p, q) задається формулою:

$$X_t = c + \Phi_1 X_{t-1} + \dots + \Phi_p X_{t-p} + \varepsilon_t - \Theta_1 \varepsilon_{t-1} - \dots - \Theta_q \varepsilon_{t-q}$$

де ε_t - n-вимірний векторний процес, який є білим шумом

$$\Phi_p(L) = I - \Phi_1 L - \ldots - \Phi_2 L^p \ , \Theta_p(L) = I - \Theta_1 L - \ldots - \Theta_p L^p$$

2.2 Прогнозування часових рядів методами машинного навчання

Класичні методи прогнозування часових рядів та багатовимірних часових рядів загалом не дають найкращих результатів прогнозування, тим паче методи VAR та VARMA потребують дуже великої кількості обчислень, тому частіше ці методи використовуються як допоміжні до методів машинного та глибинного навчання, задля досягнення кращого результату. Щоб використовувати моделі машинного та глибинного навчання перш за все ми маємо перетворити наш багатовимірний процес до задачі навчання з вчителем. Шляхом зсуву на n – лагів, чи відокремлення додаткових даних з індексу часу, як наприклад, день, місяць, година, і так далі. Нехай маємо часовий ряд $T = \begin{matrix} t_1 \\ \vdots \\ t_n \end{matrix}$ щоб перетворити його в формат, який можна використовувати для моделі машинного навчання ми маємо застосувати формулу:

$$T_n = T_{n-m}$$

де n – порядковий номер елемента часового ряду

m – довжина зсуву.

Таким чином матимемо матрицю розміром (n x k*m) , де n – довжина часового ряду, k – кількість взаємопов'язаних часових рядів, m – розмір вікна зсуву. Таким чином ми перейдемо від задачі передбачення часового ряду до задачі регресії. Тобто нашою навчальною вибіркою буде матриця

$$X = \begin{pmatrix} t_1^{1,1} & t_1^{1,n} & \cdots & t_1^{k,m-1} & t_1^{k,m} \\ t_2^{1,1} & t_2^{1,n} & \cdots & t_2^{k,m-1} & t_2^{k,m} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ t_{n-1}^{1,1} & t_{n-1}^{1,n} & \cdots & t_{n-1}^{k,m-1} & t_{n-1}^{k,m} \\ t_n^{1,1} & t_n^{1,n} & & t_{n-1}^{k,m-1} & t_{n-1}^{k,m} \end{pmatrix}$$

а цільовою змінною буде

$$У = \begin{matrix} t_1^k \\ t_2^k \\ \vdots \\ t_{n-1}^k \\ t_n^k \end{matrix}$$

де k є індексом часового ряду в багатовимірному векторному процесі.

Тепер для роботи з отриманими даними можемо використовувати як лінійні моделі так і дерева рішень. Останім часом моделі, які базується на деревах рішень заголом показують кращі результати ніж лінійні моделі. Розглянемо структуру моделі дерева рішень. Воно складається з таких частин:

1. Вузол - точка в дереві між двома гілками, в якій знаходиться правило по розподіленню
2. Кореневий вузол - перший вузол у дереві
3. Гілки - стрілка, що з'єднує один вузол з іншим
4. Листовий вузол - кінцевий вузол у дереві, точка, в якій точці даних присвоюється мітка.
5. Рівень - номер, присвоєний кожному набору вузлів, починаючи з корня

6. Коефіцієнт розгалуження - коефіцієнт розгалуження B на рівні l дорівнює кількості розгалужень до вузлів на рівні l + 1.

Дерева рішень є для класифікації та регресії. В випадку класифікації листові вузли містять дискретні значення, у випадку регрессії неперервні значення. Математика в деревах рішень починається на етапі навчання. На вхід ми передаємо дані {X, Y}, по яким ми будуємо структуру дерева та правила прийняття рішень на кожному вузлі. Кожен вузол розділить набір даних на дві або більше непересічних підмножини D(l, i) , де l - номер рівня, а і позначає кожну окрему підмножину. Якщо всі наші мітки в цій підмножині відносяться до одного класу, підмножина вважається пустою, і цей вузол буде листом, і тоді ми досягли . Якщо ні, то продовжуємо поділ. У більш складних наборах даних, щоб дістатися до моменту, коли кожен листовий вузол вже не можна розділити, можуть знадобитися надзвичайно глибокі дерева рішень, що призведе до перенавчання. Тому потрібно зупинитись до того, як дійдемо до чистих листових вузлів. Для цього є спеціальні критерії, які допомагають нам це зробити: ентропійний критерій та критерій Джинні. Ентропійний критерій допомагає нам створити дерево рішень для вибору найкращої умови для розподілення. Ентропію можна визначити як міру правильності підрозщеплення. Ентропія завжди лежить від 0 до 1. Ентропію будь-якого сплітування можна обчислити за цією формулою:

$$H(s) = \sum -P(class_i) * log_2 P(class_i)$$

де i – кількість унікальних значень нашого цільового вектора.

Наприклад для тестового набору X = {a, a, a, b, b, b, b, b} ентропія буде дорівнювати:

$$H(X) = -\left[\left(\frac{3}{8}\right) * log_2 \frac{3}{8} + \left(\frac{5}{8}\right) log_2 \frac{5}{8}\right]$$

Робота критерія Джинні дуже схожа до ентропійного критерію в дереві рішень. Критерій Джинні обчислюється за формулою:

$$GI = 1 - \sum_{i=1}^{n} p_i^2$$

де p_i = (кількість значень > порогового значення) певного классу

Робота обох методів дуже схожа, і обидва використовуються для обчислення ознаки розбиття після кожного нового. Але якщо порівняти обидва методи, то критерій Джині є більш ефективним, ніж ентропія, з точки зору обчислювальної потужності. Як ми можемо бачити на графіку ентропії, вона спочатку збільшується до 1, а потім починає зменшуватися. А у випадку Джині вона знаходиться в межах 0,5, а потім починає зменшуватися, тому для обчислення потрібно менше обчислювальної потужності. Звідси ми можемо зробити висновок, що критерій Джині є кращим порівняно з ентропією для вибору найкращих ознак для позбиття.

Дерева рішень мають такі слабкі та сильні сторони. Недоліки:

1. Дерева рішень дуже швидко та сильно перенавчаються
2. Дерева рішень чутливі до незбалансованих класів в задачі класифікації та можуть видавати зміщенні результати.

Переваги:

1. Легко інтерпретувати
2. Легко візуалізувати процесс побудови дерева

3. Стійкі до викидів в даних

Перейдемо до більш складних моделей заснованих на деревах рішень таких як випадковий ліс та градієнтний бустінг. Фундаментальним поняттям, за рахунок чого випадковий ліс показує результати краще ніж окреме дерево рішень полягає в великій кількості відносно некорельованих моделей (дерев), які працюють як ансамбль, і перевершують будь-яку з окремих складових моделей. Ансамблювання моделей передбачає поєднання декількох моделей машинного навчання в єдину нова модель, яка повинна краще робити передбачення, ніж будь-яка самостійна модель дерева рішень. Більш точно, ансамбль узагальнює передбачення декількох базових моделей, задля зменшення помилки. Для набору незалежних спостережень, кожен з яких має дисперсією σ^2, стандартна похибка вибіркового середнього задана як $\frac{\sigma}{n}$. Іншими словами, усереднення по більшому набору спостережень зменшує дисперсію. На практиці ми зазвичай не маємо багато різних наборів навчальних даних. На допомогу приходить алгоритм бегінгу. Дерева рішень дуже чутливі до даних, на яких вони навчаються - невеликі зміни в тренувальній вибірці можуть призвести до суттєвої різниці в побудові дерев рішень. Бегінг (бутстреп) дозволяє кожному окремому дереву випадково з поверненням брати вибірки даних з нашого датасету і будувати по ним дерева. Бегінг збільшує точність прогнозування, але зменшує інтерпретуємість моделі. Бегінг зменшує дисперсію узагальнюючої властивості дерев рішень за рахунок:

- Усереднення прогнозів
- Випадкова побудова дерев

Зверніть увагу, що в бегінгу ми не ділимо навчальні дані на менші вибірки і тренуємо кожне дерево на цій вибірці. Якщо у нас є датасет розміру N, ми будуємо кожне дерево на навчальному наборі розміру N. Наприклад, якщо наші навчальні дані такі X = [1, 2, 3, 4, 5, 6], тоді ми могли б взяти таку

випадкову навчальну вибірку для побудови нашого дерева x_1 = [1, 2, 2, 3, 6, 6]. Обидві вибірки довжиною шість, двійка і шістка повторюються в випадково вибраних множинах. Модель випадкового лісу приємає на вхід такі гіперпараметри

1. Кількість дерев - визначає кількість дерев, для побудови ансамблю. Чим більше дерев в випадковому лісі – тим краще, але і час на їх будівництво збільшується.
2. Максимальна кількість фіч – параметр, який відповідає за розмір випадкової вибірки, доступної для побудови нового дерева. Чим нижче значення, тим менша кореляція дерев, дисперсія ансамблю, але зростає зміщення.
3. Мінімальний розмір для сплітування – мінімальній розмір вибірки, потрібний для сплітування вузла дерева.

Перейдемо до іншої моделі, яка будується на основі дерев рішень – градієнтний бустінг. Бустінг дещо відрізняється від бегінгу, оскільки він не передбачає взяття випадкових вибірок і одночасної побудови моделей. Натомість дерева будуються послідовно і враховують помилки інших. Основна ідея алгоритму, схожа на звичайний алгоритм градієнтного спуску - навчання базових моделей для отримання градієнта функції втрат, наприклад метод найменших квадратів, і оновлення поточного наближення функції . Формула за якою відбувається навчання алгоритму:

$$H_m(x) = H_{m-1}(x) + \gamma_m h_m(x) = H_{m-1}(x) + argmin_{\gamma,h} \sum_{i=1}^{N} L(y_i, H_{m-1}(x_i) + h(x))$$

де $H_{m-1}(x)$ – поточний ансамбль;

$\gamma_m h_m(x)$ – нова модель в ансамблі;

$L(y_i, H_{m-1}(x_i) + h(x))$ – функція втрат;

Іншими словами, за ітерації m алгоритм обчислює градієнт функції втрат для кожного спостереження, будує базове дерево рішень, як регресію на псевдо-залишках. Знаходить оптимальний коефіцієнт при $h(x)$, відносно початкової функції витрат та оновлює її поточне наближення. Ми також можемо модифікувати функцію втрат для нашої регресійної задачі.

1. $L(y,f) = (y-f)^2$ – функція помилок яка передбачає, що наші дані розподілені нормально.
2. $L(y,f) = |\, y-f \,|$ - функція, яка не так сильно штрафує сильні відхилення, як попередня.
3. $L(y,f) = \begin{matrix} \frac{(y-f)^2}{2}, |\, y-f \,| \le \sigma \\ \sigma\,|\, y-f \,| - \frac{1}{2} * \sigma^{\,2} \;\;, \text{ в інших випадках} \end{matrix}$ - функія втрат Хубера менш чувтлива до викидів в даних ніж квадратична функція помилок.

Отже, градієнтний бустінг – це не просто якийсь конкретний алгоритм, а загальна методологія, як будувати ансамблі дерев рішень.

Є ще один алгоритм який базується на деревах рішень, він працює набагато швидше за градієнтний бустінг, і видає порівняно непогані результати. Цей алгоритм має назву «LightGBM». На відміну від звичайного алгоритму градієнтного бустінгу, який будує ансамбль по всіх елементах навчальної вибірки, алгоритм полегшеного градієнтного бустінгу бере фокус лише на тих навчальних прикладах, які призводять до більшого градієнту. Ця модифікація виключає значну частину екземплярів даних з маленькими градієнтами і бере інші для оцінки приросту інформації.

Отже, методи побудовані над деревами рішень, такі як випадкові ліси чи градієнтний бустінг, частіше всього дадуть гарний результат прогнозу в

задачах регресії та класифікації. Дуже часто найкращий результат досягається поєднанням цих методів в один ансамбль чи зважаним середнім. Але у цих алгоритмів є і мінуси: вони легко перенавчаються, їх складно інтерпретувати, тож є деякі задачі, в яких вони частіше всього не використовуються.

2.2 Прогнозування часових рядів методами глибинного навчання

Основною перевагою моделі рекурентної нейронної мережі є те, що кожен вихід моделі є функцією, яка залежить як від попереднього виходу так і від поточного входу. Рекурентна нейронна мережа є узагальненням нейронної мережі з внутрішньою пам’яттю. За допомогою цих внутрішніх станів мережа може обробляти послідовності даних. Рекурентні мережі використовуються в задачах обробки тексту та передбачення наступного слова, розшифровки рукописного тексту, та розпізнавання голосу. В звичайних нейронних мережах всі входи незалежні один з одним, а в рекурентній вони пов’язані. На рис. 2 зображена схема архітектури рекурентної мережі.

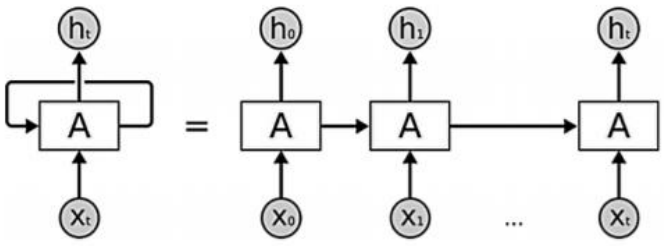


Рисунок 2. Приклад архітектури рекурентної мережі

Спочатку вона бере на вхід перший елемент послідовності x_0, і виводить h_0, який разом із x_1 є входом для наступного кроку. Таким чином формула для поточного стану:

$$h_t = f(h_{t-1}, x_{t)} = \tanh(W_{hh} h_{t-1} + W_{xh} x_t),$$

де W – ваги;

h – схований стан;

W_{hh} - ваги на попередньому схованому стані;

W_{xh} – ваги в поточному вхідному стані;

tanh - це нелінійна функція активації, яка перетворює значення до діапазону [-1,1].

Вихід рахується за формулою:

$$y_t = W_{hy} h_t$$

Архітектури рекурентних нейронних мереж поділяються за типами:

- Один-до-одного: $T_x = T_y = 1$. Використовується як звичайна нейронна мережа.
- Один-до-багатьох: $T_x = 1$, $T_y > 1$. Використовується в генерації музики.
- Багато-до-одного: $T_x > 1$, $T_y = 1$. Використовується в аналізі тональності тексту.
- Багато-до-багатьох: Tx = Ty. Задача розпізнавання іменованих сутностей.

– Багато-до-багатьох: Tx # Ty. Машинний переклад.

Наведемо схематичні зображення різних типів рекурентних мереж на рисунках 3, 4, 5, 6, 7.

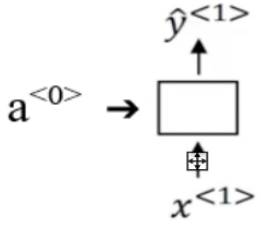


Рисунок 3. Тип зв’язку один-до-одного

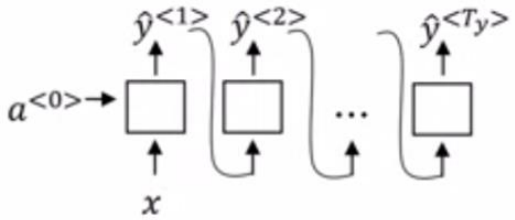


Рисунок 4. Тип зв’язку один-до-багатьох

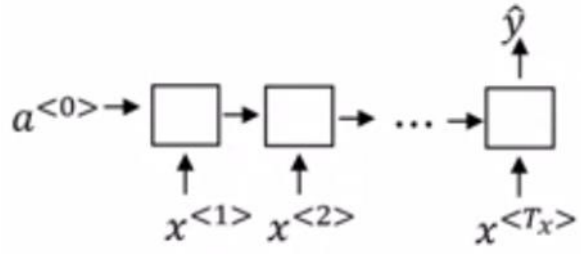


Рисунок 5. Тип зв’язку багато-до-одного

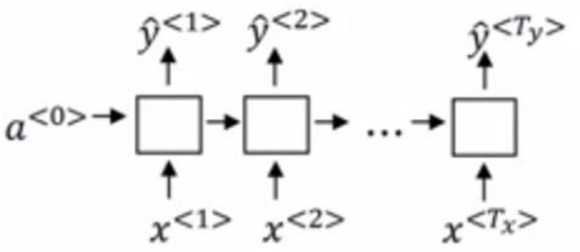


Рисунок 6. Тип зв’язку багато-до-багатьох для задачі розпізнавання іменованих сутностей

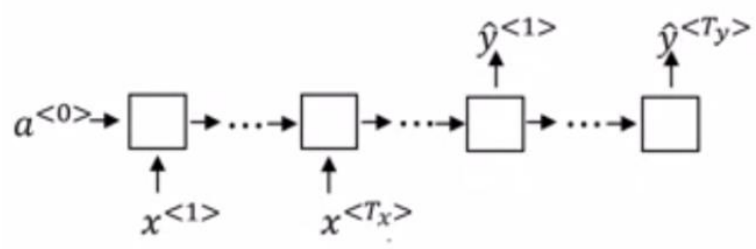


Рисунок 7. Тип зв’язку багато-до-багатьох для задачі машинного перекладу

Розглянемо роботу рекурентної нейронної мережі на прикладі. Нехай, у нас є словник чи тренувальна вибірка, яка містить літери [h,e,l,o]. На вхід ми подаємо літеру h, на виході маємо отримати слово hello. Зобразимо схематично роботу мережі на рис. 8.

Цільові символи:	«e»	«l»	«l»	«o»
Вихідний шар	1 2.2 −3 4.1	0.5 0.3 −1 1.2	0.1 0.5 1.9 −1.1	0.2 −1.5 −0.1 2.2
	↑	↑	↑	↑
Схований шар	0.3 −0.1 0.9	1 0.3 0.1	0.1 −0.5 −0.3	−0.3 0.9 0.7
	↑	↑	↑	↑
Вхідний шар	1 0 0 0	0 1 0 0	0 0 1 0	0 0 1 0
Вхідні символи	«h»	«e»	«l»	«l»

Рисунок 8. Схематична робота рекурентної мережі

Після кожного шару ми маємо застосувати функцію активації. Функція активації – це нелінійна, неперервна, диференційована функція, яка визначає вихідний стан нейрону. Найчастіше використовуються такі функції активації:

- $f(z) = \frac{1}{1+e^{-z}}$ - сігмоїда. Головний її недолік – це те, що при наближенні до кінців сигмоїди значення У слабко реагують на змінну значень Х. Це означає, що градієнт приймає маленькі значення і наша мережа або перестає навчатись, або робить це занадто повільно. Ця проблема називається зникаючий градієнт.
- $f(z) = \tanh(z) = \frac{2}{1+e^{-2z}} - 1$ – гіперболічний тангенс трохи вирішує проблему зникаючого градієнту. Але в основному гіперболічний тангенс використовують коли нема необхідності в нормалізації даних. Через те, що область визначення тангенсу центрована навколо нуля.
- f(z) = max(0, z) – ReLu. Найбільш часто використовувана функція активації в глибокому навчанні. По-перше –це нелінійна функція, для якої дуже швидко рахується похідна: для від'ємних значень 0, для додатніх 1. В сітках з великою кількістю нейронів використання попередньо визначених функцій тягне за собою активацію всіх нейронів, що тормозить навчання моделі. ReLu розріджує та робить мережу легшою.
- $f(z) = \begin{cases} \alpha x, & \text{якщо } x < 0 \\ x, & \text{інакше} \end{cases}$ - Leaky ReLu. Вирішує проблему затухаючого градієнту на відміну від звичайної ReLu. Однак і вона має свої недоліки. Складніше рахувати похідну. Коефіцієнт α – є гіперпараметром, який потрібно налаштовувати.

Рекурентні нейронні мережі мають недоліки. Інколи для виконання задачі нам необхідна лише нещодавня інформація. Наприклад, якщо у нас є

довге речення в якому ми маємо передбачити останнє слово і це передбачення буде залежати від слова, яке знаходиться на початку речення, тоді розрив між актуальною інформацією і точкою її застосування може стати великим. Чим більший цей розрив, тим гірше звичайна RNN працює. Виникає дві проблеми: або градієнти функції помилок зменшуються настільки, що помилки в кінці її послідовності перестають впливати на її початок, або градієнти збільшуються і процес розходиться. Задля вирішення цих проблем була придумана нова архітектура нейронної мережі, яка має назву long short-term memory скорочено LSTM. В LSTM було додано внутрішній стан S_t , за рахунок якого зникла проблема згасаючих градієнтів. Також в були додані вхідні та вихідні ворота, які відповідають за те, що з вхідних даних дасть вплив на внутрішній стан клітини, і що з внутрішнього стану повинно відправитися на вихід. На рис. 9 зображено схематично архітектура однієї LSTM клітини.

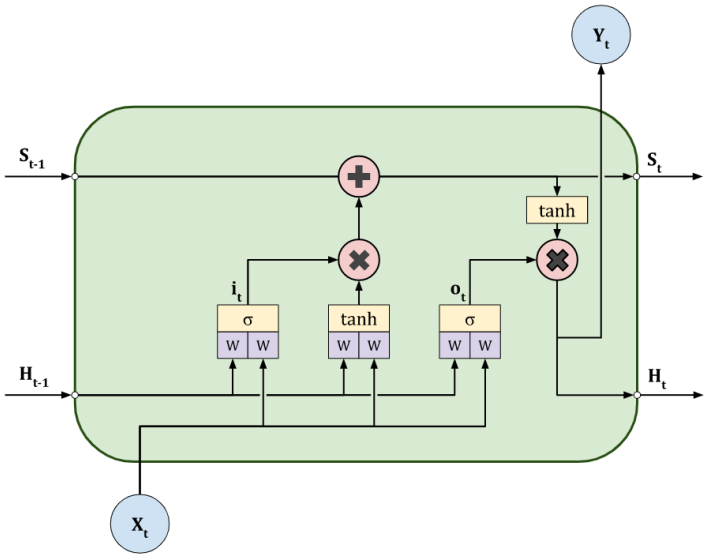


Рисунок 9. Схематичне зображення LSTM клітини

Формально роботу можна описати такими формулами:

$$g^{(t)} = \varphi(W^{gx} X^{(t)} + W^{gh} H^{(t-1)} + b_g)$$

$$i^{(t)} = \sigma(W^{ix} X^{(t)} + W^{ih} H^{(t-1)} + b_i)$$

$$o^{(t)} = \sigma(W^{ox} X^{(t)} + W^{oh} H^{(t-1)} + b_o)$$

$$S^{(t)} = g^{(t)} \odot i^{(t)} + S^{(t-1)}$$

$$H^{(t)} = \varphi(S^{(t)}) \odot o^{t}$$

де $i^{(t)}$ та $o^{(t)}$ вхідні та вихідні ворота відповідно

W^{gx}, W^{gh}, W^{ix}, W^{ih}, W^{ox}, W^{oh} –матриці вагів

$\odot$ - операція поелементного перемноження.

$i^{(t)}$ та $o^{(t)}$ - це вектори, які складаються з нулів та одиниць, які пропускають чи не пропускають якісь компоненти вектору через себе. Наступним кроком були додано ворота, які відповідають за забування інформації, так звані forget gate. На рис. 10 зображено схематично архітектура однієї LSTM клітини з клапаном забування.

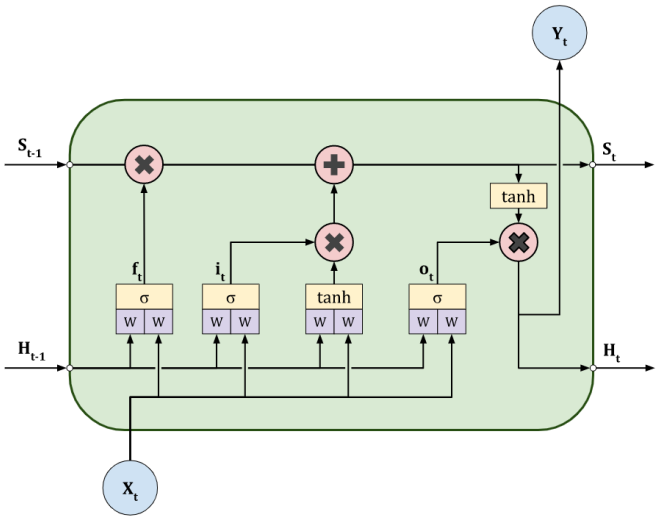


Рисунок 10. Схематичне зображення LSTM клітини з forget gate

Роботу алгоритму можна описати такими формулами:

$$g^{(t)} = \varphi(W^{gx} X^{(t)} + W^{gh} H^{(t-1)} + b_g)$$

$$i^{(t)} = \sigma(W^{ix} X^{(t)} + W^{ih} H^{(t-1)} + b_i)$$

$$o^{(t)} = \sigma(W^{ox} X^{(t)} + W^{oh} H^{(t-1)} + b_o)$$

$$f^{(t)} = \sigma(W^{fx} X^{(t)} + W^{fh} H^{(t-1)} + b_f)$$

$$S^{(t)} = g^{(t)} \odot i^{(t)} + S^{(t-1)} \odot f^{(t)}$$

$$H^{(t)} = \varphi(S^{(t)}) \odot o^{t}$$

де $i^{(t)}$ та $o^{(t)}$ вхідні та вихідні клапани відповідно;

$f^{(t)}$ – клапан забування;

W^{gx}, W^{gh}, W^{ix}, W^{ih}, W^{ox}, W^{oh}, W^{fx}, W^{fh} –матриці вагів;

$\odot$ - операція поелементного перемноження;

Також було вигадано варіант клітини схожий на LSTM, але з меншою кількістю воріт, отже із меншою кількістю параметрів, але такою же якістю прогнозу. Називається такий шар Gated Recurrent Unit. На рис. 11 зображено схематично архітектура однієї GRU клітини.

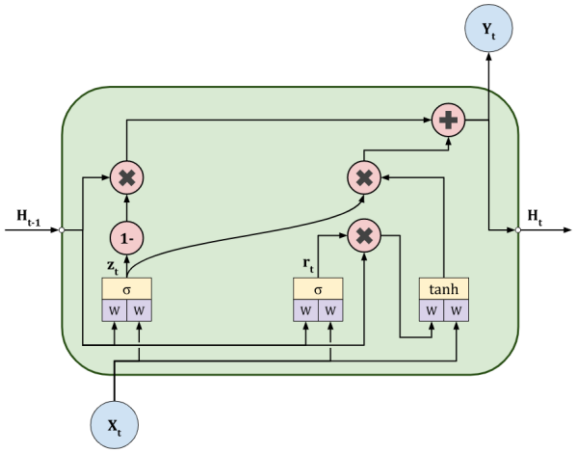


Рисунок 11. Схематичне зображення GRU клітини.

Роботу алгоритму можна описати такими формулами:

$$z^{(t)} = \sigma\left(W^{zx} X^{(t)} + W^{zh} H^{(t-1)} + b_z\right)$$

$$r^{(t)} = \sigma\left(W^{rx} X^{(t)} + W^{rh} H^{(t-1)} + b_r\right)$$

$$H^{*(t)} = \varphi\left(W^{h*x} X^{(t)} + W^{h*h} \left(r^{(t)} \odot H^{(t-1)}\right) + b_{h*}\right)$$

$$H^{(t)} = \left(1 - z^{(t)}\right) \odot H^{(t-1)} + z^{(t)} \odot H^{*(t)}$$

Також ми можемо зробити багатошарову LSTM задня покращення результатів. На рис. 12 зображено схематично архітектура багатошарової LSTM мережі

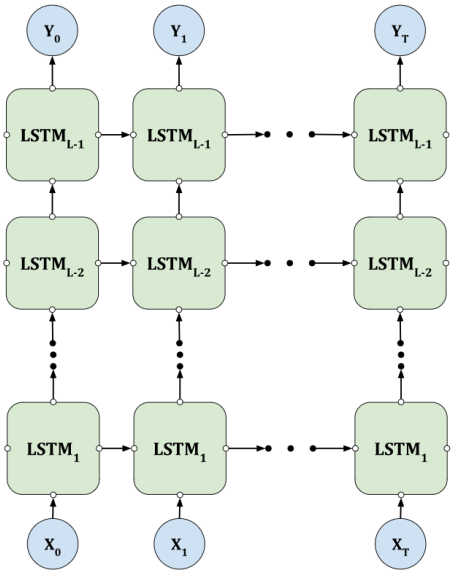


Рисунок 12. Схематичне зображення багатошарової LSTM мережі

2.4 Оцінювання гіперпараметрів моделі методом байєсівської оптимізації

Кожна модель машинного навчання має свій набір гіперпараметрів, які найкраще всього підходять для вирішення тієї чи іншої задачі. Разом з тренуванням моделі та визначенням вагів, ми маємо підібрати найкращі гіперпараметри задля досягнення найбільшої точності нашої моделі, будь то дерева рішень чи рекурентна мережа. Постановка задачі: є простір гіперпараметрів $\lambda = \lambda_1 \times \lambda_2 \times \ldots \times \lambda_n$, де λ_i - простір і-го гіперпараметра. Маємо тренувальні дані X. Потрібно знайти $\lambda^* = \text{argmin}\ E_{X_{train}, X_{valid}} L(M_\lambda, X_{\text{train}}, X_{\text{valid}})$, де $L -$ функція втрат моделі M_λ , навченої на X_{train} – тренувальній вибірці при гіперпараметрах λ та провалідованої на X_{valid}. Одними із найбільш вживаних підходів є GridSearch, RandomSearch. Це просто перебір всіх значень сітки параметрів. Вони є дуже неоптимальними та займають дуже багато часу. Більш оптимальним способом підбору гіперпараметрів баєсівським підходом. Цей підхід є модифікацією випадкового пошуку. Він бере тільки ті множини параметрів навколо тих точок, які дають найбільший приріст в якості прогнозу. Тобто для вибору найкращих гіперпараметрів треба дивитися на історію вже перевірених точок. Цей метод оптимізації отримав назву Sequential Model-Based Optimization, що в перекладі означає послідовна оптимізація на основі моделі. Цей алгоритм складається з двух частин функції вибору та сурогатної моделі. Сурогатна модель на кожній ітерації навчається по всім отриманим раніше виходам цільової функції, а функція вибору використовує отримане розподілення сурогатної моделі оцінює користь від наступних точок. Алгоритм можна описати таким виразом:

$$EI_{y_*}(\lambda) = \int_{-\infty}^{y_*} (y_* - y)\, p_M(y|\lambda, H)\, dy$$

де y_* - поріг якості передбачення;

$p_M(y|\lambda, \mathrm{H})$ – сурогатна модель для отримання у при вибраних гіперпараметрах λ та історії прогнозу H.

Зазвичай в якості сурогатної моделі використовують два найбільш популярні процеси: парзеновські дерева та гауссівський процес. Коли ми використовуємо гауссівський процес цільова функція апроксимується в якості апостеріорного розподілення значень при вибраних параметрах та історії якості передбачень з нормальним розподіленням в якості апріорного. На відміну від попереднього процесу стратегія парзеновських дерев напряму моделює функцію правдоподібності замість апостеріорної вірогідності.

2.5 Критерії якості оцінок прогнозів

Аналіз якості побудованих прогнозів є вирішальним етапом, що визначає чи можна застосовувати побудовану модель або розроблений алгоритм для прогнозування процесів та дозволяє обрати кращу модель з множини альтернатив. Розглянемо найбільш поширені критерії:

- Середньоквадратична похибка (RSME)

$$\mathrm{RSME} = \sqrt{\frac{1}{N}\sum_{k=1}^{T}(y(k) - \hat{y}(k))^2},$$

де y(k) – значення змінної в момент k;

$\hat{y}(k)$ – прогнозоване значення змінної в момент k;

N – довжина ряду.

Дана характеристика дозволяє оцінити діапазон відхилення прогнозованих значень від реальних

1. Середня абсолютна похибка у відсотках (MAPE). Перевага цієї метрики в тому, що вона дозволяють оцінювати якість прогнозу не прив'язуючись до масштабу даних.

$$MAPE = \frac{1}{N}\sum_{k=1}^{N}\frac{|y(k)-\hat{y}(k)|}{|y(k)|}\times 100\% .$$

2. Середня похибка (ME) однаково штрафує як великі відхилення так і маленькі.

$$ME = \frac{1}{N}\sum_{k=1}^{N}(y(k)-\hat{y}(k)) .$$

3. Середнє квадратів похибок (MSE). Даний критерій більше штрафує за великі відхилення.

$$\mathrm{M}SE = \frac{1}{N}\sum_{k=1}^{N}(y(k)-\hat{y}(k))^2 .$$

4. Метрика Хубера. Є менш чутливою до викидів в даних.

$$Hubber = \begin{matrix} \frac{(y-f)^2}{2}, | y-f | \le \sigma \\ \sigma | y-f | -\frac{1}{2}*\sigma^2 \quad , \text{ в інших випадках} \end{matrix}$$

2.6 Висновки

Розглянуто різні підходи, що застосовуються для побудови прогнозів багатовимірних часових рядів даних різної природи. Проведено короткий огляд процедур побудови моделей процесів на базі регресійного аналізу. Детально розглянуто етапи побудови алгоритму вирішення задач прогнозування часових рядів на основі задач машинного та глибинного навчання, визначено переваги та недоліки підходів, основні принципи:

- аналіз та попередня обробка даних. Окреслено завдання, які вирішуються на даному етапі, розглянуто методи попередньої обробки даних;
- оцінювання структури моделі. Описано процес вибору структури за допомогою використання інструментів кореляційного аналізу, статистичні критерії, що можуть використовуватись для оцінювання порядку рівнянь моделі. Описано процес переходу до задачі навчання з вчителем.
- діагностика моделі: наведено підходи для оцінювання якості моделі, покращення якості прогнозу шляхом оптимізації параметрів моделі.

Розглянуто процес побудови прогнозів за допомогою моделі VARMA(p,q). Описано загальні принципи функціонування таких моделей машинного навчання, як дерева рішень та алгоритми, які базуються на них. Принципи функціонування нейронних мереж в цілому та рекурентних нейронних мереж, архітектури популярних моделей, таких як LSTM-мережа. Розглянули методи оптимізації та підбору гіперпараметрів на основі випадкових процесів задля отримання кращого результату прогнозу.

РОЗДІЛ 3 СИСТЕМА ДЛЯ ПРОГНОЗУВАННЯ БАГАТОВИМІРНИХ ЧАСОВИХ РЯДІВ

3.1 Структурна схема (архітектура) системи

В рамках дипломної роботи було розроблено программу на мові программування Python, інтерфейс програми був побудований за допомогою PyQT5. Основними вимогами до продукту є наступні:

- Можливість зчитування даних багатовимірних часових рядів з файлу;
- Можливість використання статистичних критеріїв, описових статистик, побудови корелограм, та її візуалізаця;
- Можливість побудови моделі VAR(p,q) та її модифікацій при різних параметрах p, q;
- Можливість побудови моделі бустінгу на деревах рішень та LSTM з вибором та підбором оптимальних гіперпараметрів моделі, що визначають структуру, проведення процедури навчання;
- Обчислення критеріїв адекватності моделей, якості побудованих прогнозів;
- Наявність інструментів для візуалізації передбачень.

Враховуючи вимоги до програми на рис. 13 розроблено наступну функціональну схему архітектури програми:

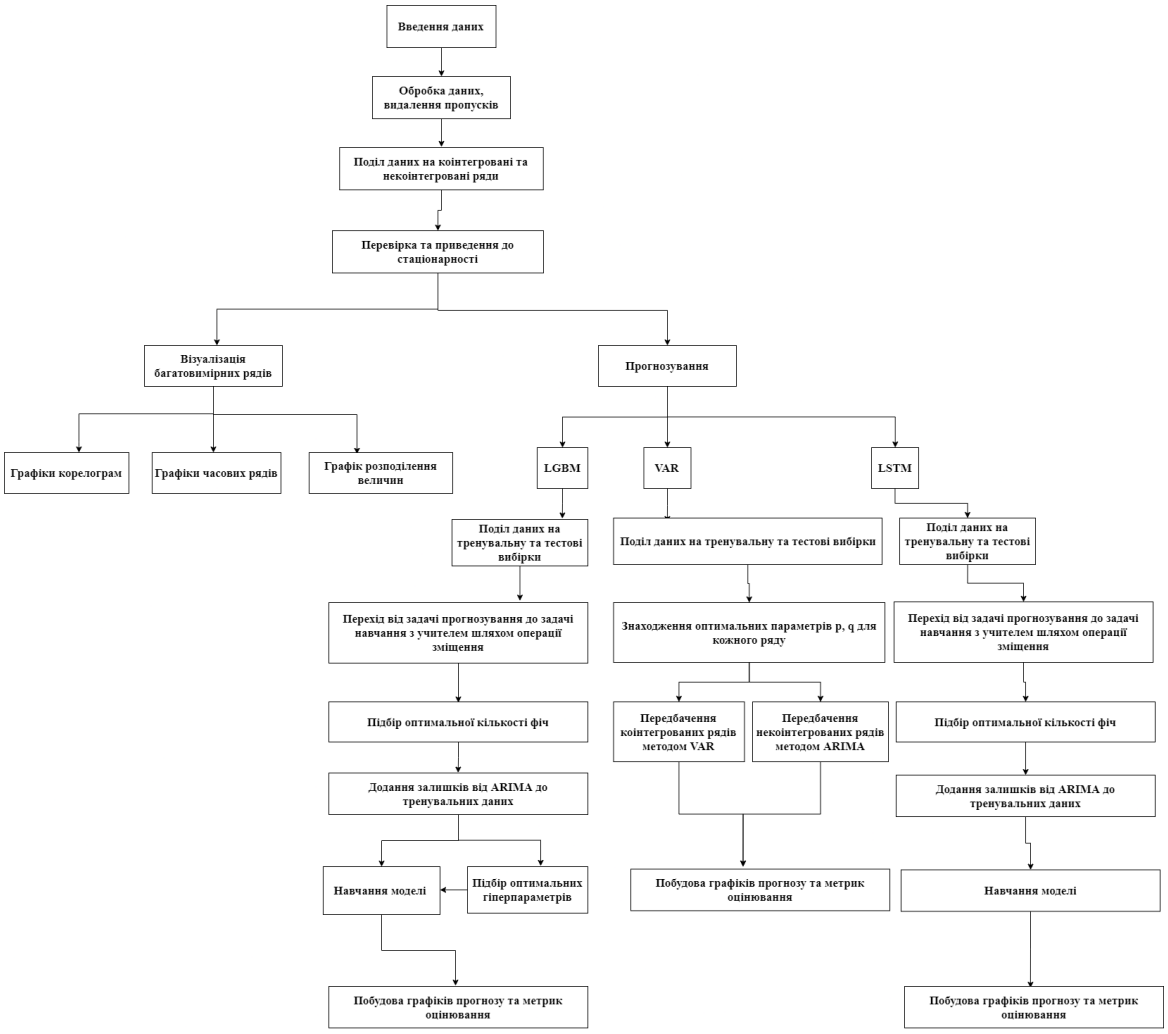


Рисунок 13. Функціональна схема програми

3.2 Результати обчислювальних експериментів: порівняння точності оцінок прогнозів та швидкодії алгоритмів

В якості вхідних даних був обраний датасет, який містить кількісний зміст різних хімічних часток в повітрі, таких як: PT08, NO, C6H6 та інших. На рис. 14 зображені різні статистичні характеристики хімічної частки C6H6.

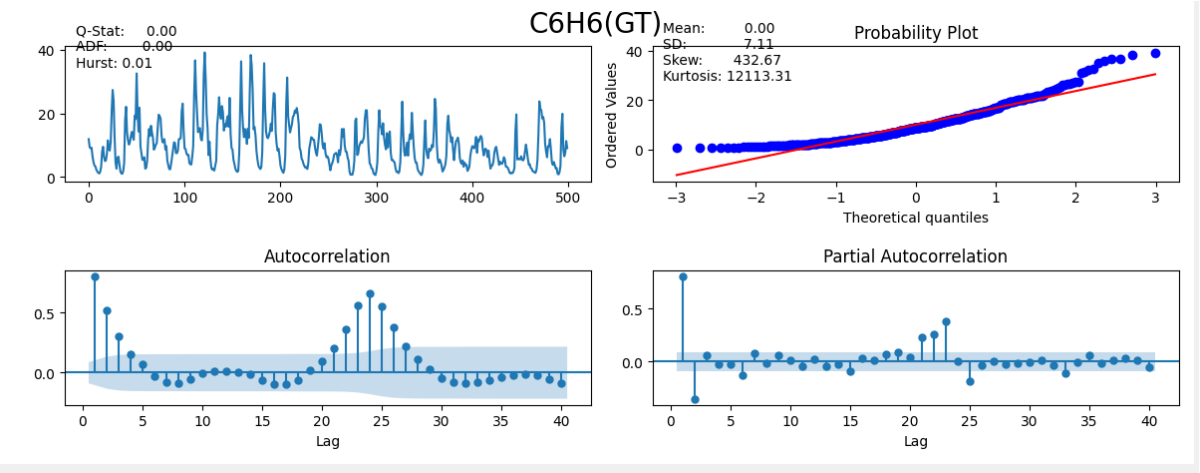


Рисунок 14. Графіки різних статистичних показників частки C6H6

Вхідні дані поділимо на навчальну та тестову вибірку співвідношенням 7:3. До кожного передбачення наведемо значення метрик оцінювання, час тренування моделі та візуалізацію передбачень. Синім кольором позначені справжні значення, зеленим передбачені. По графікам часових рядів можемо зробити висновки, що майже всі вони не стаціонарні. Розглянемо передбачення методом VAR. На рис. 15 зображенні графіки передбачень методом VAR.

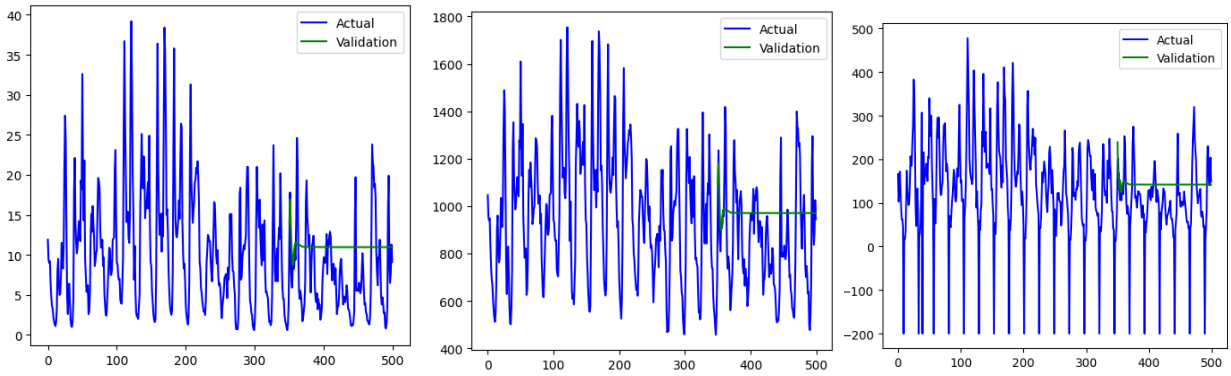


Рисунок 15. Графіки передбачень часток C6H6, PT08, NOx методом VAR

В таблиці 3.1 наведено метрики побудованого прогнозу методом VAR

Таблиця 3.1

	MAE	MSE	R^2
$C6H6$	5.17	36.16	-0.36
PT08	194.01	55177.45	-0.26
NOx	64.54	8524.05	-0.21

Розглянемо передбачення методом Light Gradient Boosting Machine. На рис. 16 зображенні графіки передбачень методом LGBM.

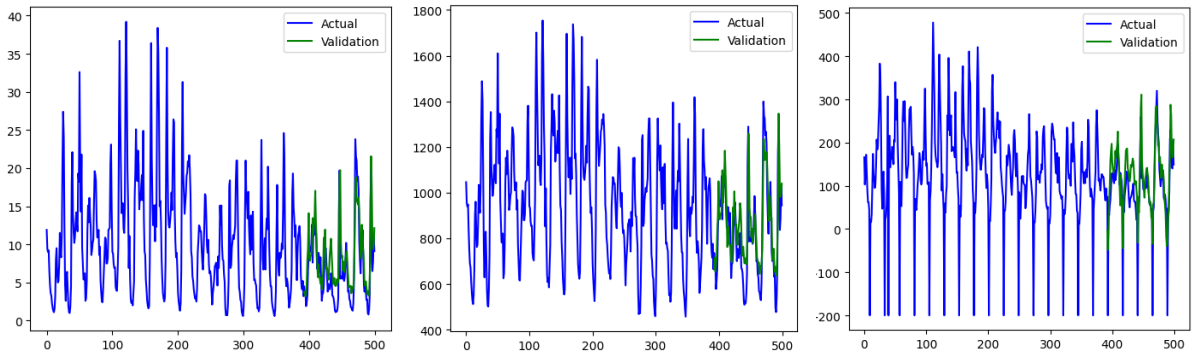


Рисунок 16. Графіки передбачень часток C6H6, PT08, NOx методом Light Gradient Boosting Machine

В таблиці 3.2 наведено метрики побудованого прогнозу методом LGBM

Таблиця 3.2

	MAE	MSE	R^2
$C6H6$	2.54	10.54	0.58
PT08	97.90	14793.79	0.66
NOx	42.82	3304.65	0.56

Як ми можемо побачити по візуалізації результати LGBM набагато кращі за результати методу VAR. Розглянемо передбачення за допомогою нейронної мережі LSTM. На рис. 17 зображенні графіки передбачень методом LSTM.

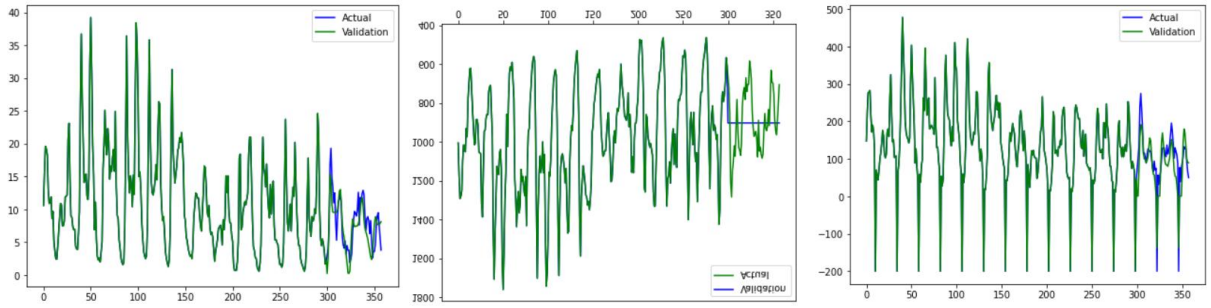


Рисунок 17. Графіки передбачень часток C6H6, PT08, NOx методом LSTM

В таблиці 3.3 наведено метрики побудованого за допомогою нейронної мережі LSTM

Таблиця 3.3

	MAE	MSE	R^2
$C6H6$	2.13	6.47	0.56
PT08	166.64	41007.324	-0.62
NOx	36.83	2251.15	0.56

Ми можемо побачити, що на другому часовому ряді модель не вивчає просторові зміни в графіку частки PT08. Але по двом іншим часовим рядам вона показала кращі результати за дерева рішень як візуально так і по метрикам.

Порівняємо в таблиці 3.4 час тренування моделей на трьох коінтегрованих часових рядах, кожен з яких має 500 спостережень.

Таблиця 3.4

	VAR	LGBM	*LSTM*
Час в секундах	230	55	360

Отже, LSTM мережа тренується довше всього, але показує лише трохи кращі метрики за LGBM. Тому найоптимальнішим вибором для передбачення багатовимірних часових рядів є моделі, які базуються на деревах рішень, зокрема LGBM. Адже вона працює найшвидше з усіх бустінгових моделей і показує не гірші метрики якості.

3.3 Висновки

Отже, ми побудували прогноз для трьох взаємопов'язаних часових рядів. Найбільш якісним в сенсі відношення якості прогнозу до часу тренування моделі виявився прогноз за допомогою моделі LGBM, яка базується на побудові дерев рішень.

РОЗДІЛ 4 ФУНКЦІОНАЛЬНО-ВАРТІСНИЙ АНАЛІЗ ПРОГРАМНОГО ПРОДУКТУ

Даний розділ присвячено оцінюванню основних характеристик програмного продукту, призначеного для побудови прогнозів динаміки розвитку фінансово-економічних процесів, представлених за допомогою одновимірних часових рядів, різними методами, зокрема за допомогою регресійного аналізу та нейронних мереж. Для вибору варіанта реалізації з врахуванням як економічних факторів так і характеристик продукту будемо використовувати аппарат функціонально-вартісного аналізу.

4.1 Постановка задачі проектування

Застосувати метод ФВА для проведення техніко-економічного аналізу розробки програмного продукту для побудови прогнозів динаміки розвитку фінансово-економічних процесів, представлених за допомогою одновимірних часових рядів.

Застосування методу ФВА для проведення техніко-економічного аналізу розробленого продукту передбачає вибір системи показників якості програмного продукту. Продукт повинен відповідати наступним технічним вимогам:

- Функціонування на персональних комп’ютерах під управлінням операційних систем MacOS, Linux, Windows;
- Швидкодія та можливість обробки великого об’єму даних;
- Зручність роботи з програмою;
- Мінімальні витрати на реалізацію та впровадження програмного продукту.

4.2 Обґрунтування функцій програмного продукту

Головна функція F_0 – розробка програмного продукту, який надає інструменти для проведення попереднього аналізу та обробки вхідного часового ряду та побудови прогнозу за допомогою методів регресійного аналізу та нейронних мереж. Беручи за основу цю функцію, можна виділити наступні:

F_1 – вибір мови програмування;

F_2 – вибір засобів побудови моделей;

F_3 – вибір середовища розробки .

Кожна з цих функцій має декілька варіантів реалізації:

Функція F_1:

а) Python;

б) R;

в) Java

Функція F_2:

а) Реалізація алгоритмів вручну;

б) Використання готових бібліотек;

Функція F_3:

а) PyCharm;

б) Microsoft Visual Code.

Варіанти реалізації наведених функцій наведено у морфологічній карті, зображеній на рис. 18.

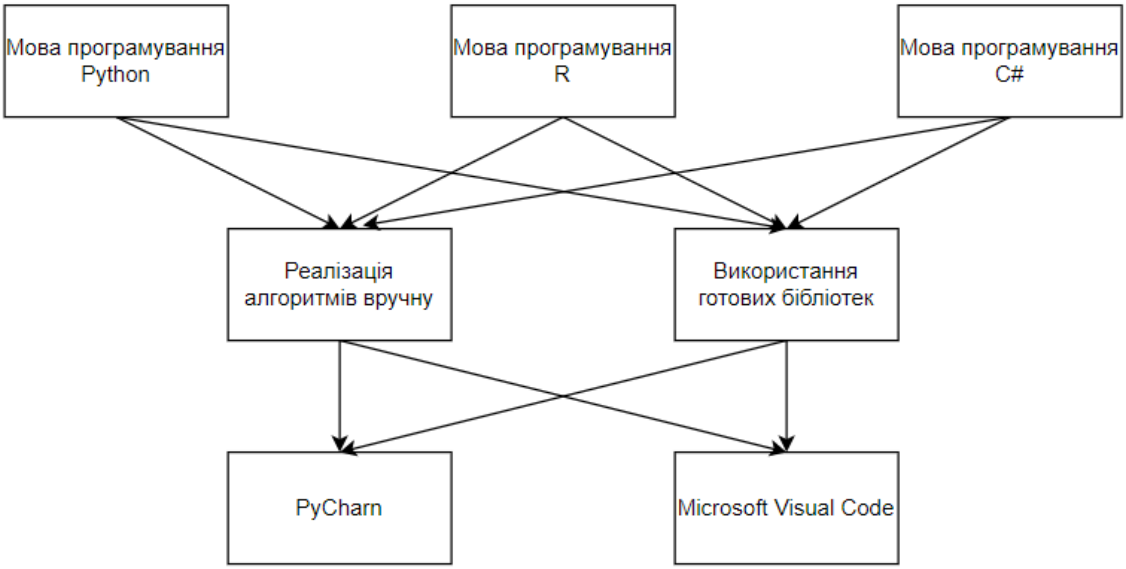


Рисунок 18. Морфологічна карта

Побудована морфологічна карта відображає всі можливі комбінації варіантів реалізації наведених вище функцій.

На базі морфологічної карти побудуємо позитивно-негативну матрицю, наведену у таблиці 4.1.

Таблиця 4.1

Функції	Варіанти реалізації	Переваги	Недоліки
F_1	*а*	Наявність великої кількості доступних бібліотек, легка в опануванні мова	Низька швидкість роботи при обробці великої кількості даних
	б	Наявність великої кількості потужних бібліотек для аналізу даних	Специфічний синтаксис, складність опанування мови
	в	Висока продуктивність роботи програм	Порівняно невелика кількість готових бібліотек для роботи з даними

Функції	Варіанти реалізації	Переваги	Недоліки
F_2	*а*	Можливість створення більш зручних методів для вирішення задачі	Тривалий час реалізації, тестування, висока ймовірність помилок
	б	Суттєве зменшення часу розробки, зручність використання	Необхідність завантаження додаткових модулів
F_3	*а*	Кросплатформенність, зручний вбудований інтерфейс, багато складних особливостей для розробки коду	Довгий запуск та сильне навантаження комп’ютера під час роботи, неможливість використання ipynb файлів та інших мов програмування
	б	Відсутність великого навантаження на процесор комп’ютера, можливість використовувати ipynb файли, та змінювати мову програмування.	Відсутність багатої кількості додаткових фіч для розробки

Кінець таблиці 4.1

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

- Функція F_1. Виключаємо варіант б, оскільки зручність використання мови є важливим фактором. Виключаємо варіант в, оскільки відстуність готових бібліотек значно ускладнює розробку програми.
- Функція F_2. Виключаємо варіант а.

Таким чином, будемо розглядати наступні варіанти реалізації програмного продукту:

- F_1а – F_2а – F_3а
- F_1а – F_2а – F_3б

4.3 Обґрунтування системи параметрів програмного продукту

На підставі вимог до програмного продукту, визначимо основні параметри вибору, які будуть використані для розрахунку коефіцієнта технічного рівня.

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

- *X1* – швидкодія мови програмування;
- *X2* – мінімальний об’єм оперативної пам’яті;
- *X3* – час реалізації програмного продукту;
- *X4* – потенційний об’єм програмного коду.

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

Таблиця 4.2

Назва Параметра	Умовні позначач	Одиниці виміру	Значення параметра		
			гірші	середні	кращі

	ня				
Швидкодія мови програмування	X1	оп/мс	8700	12000	15500
Мінімальний об’єм оперативної пам’яті	X2	Гб	4	2	1
Час реалізації програмного продукту	X3	Год.	86	56	30
Потенційний об’єм програмного коду	X4	кількість рядків коду	3000	1500	1000

За даними таблиці 4.2 побудуємо графічні характеристики параметрів, що наведені на рис. 19, рис. 20, рис. 21, рис. 22.

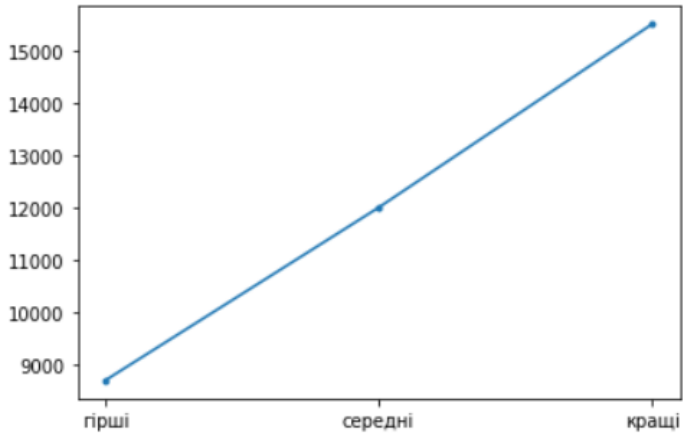


Рисунок 19. Швидкодія мови програмування

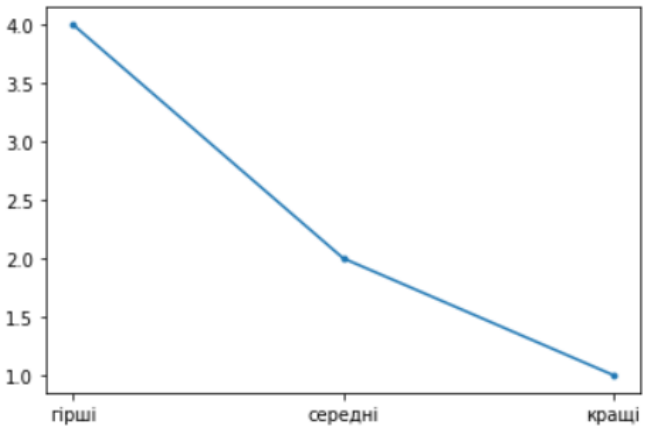


Рисунок 20. Мінімальний об’єм оперативної пам’яті

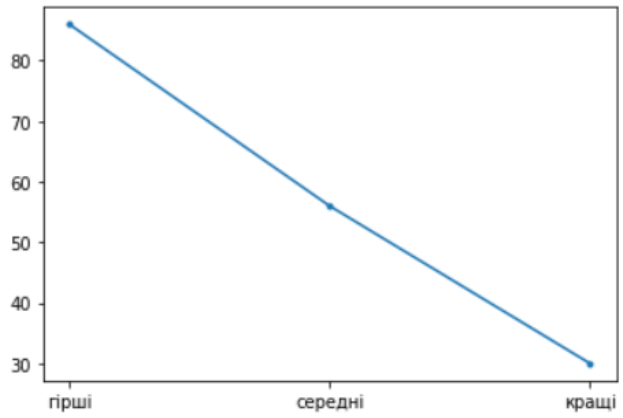


Рисунок 21. Час реалізації програмного продукту

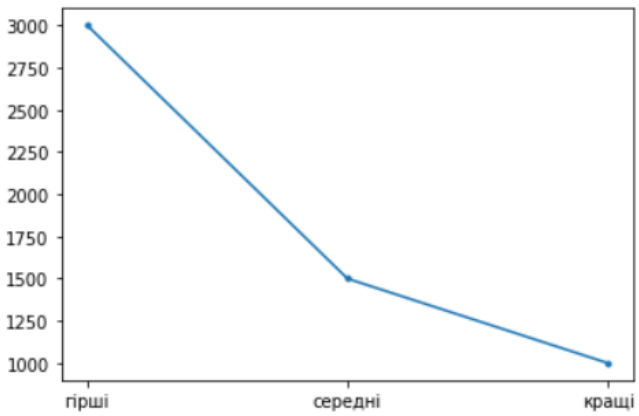


Рисунок 22. Час реалізації програмного продукту

4.4 Аналіз експертного оцінювання параметрів

Значимість кожного параметра визначається за допомогою методу попарного порівняння. Оцінку проводить експертна комісія, що складається з 7 експертів. Кожний експерт оцінює ступінь важливості кожного параметру. Процедура визначення коефіцієнтів значимості передбачає виконання наступних кроків:

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

Результати експертного ранжування наведені у таблиці 4.3

Таблиця 4.3

Позначення параметра	Назва параметра	Одиниці виміру	Ранг параметра за оцінкою експерта							Сума рангів R_i	Відхи-лення Δ_i	Δ_i^2
			1	2	3	4	5	6	7			
X1	Швидкодія мови програмування	Оп/мс	2	3	2	3	2	1	1	14	-3,500	12,250
X2	Мінімальний об’єм оперативної пам’яті	Гб	3	1	4	1	2	3	1	15	-2,500	6,250

X3	Час реалізації програмного продукту	Год.	4	5	3	4	4	4	5	29	11,500	132,250
X4	Потенційний об'єм програмного коду	Кількість рядків коду	1	1	1	2	2	2	2	11	-6,500	42,250
	Разом		10	10	10	10	10	10	10	70	0	193,000

Для перевірки степені достовірності експертних оцінок, визначимо наступні параметри:

- сума рангів кожного з параметрів і загальна сума рангів:

$$R_i = \sum_{j=1}^{N} r_{ij}$$

$$R = \sum_{i=1}^{n} R_i = \frac{Nn(n+1)}{2} = 70,$$

де R_i – сума рангів показника і;

r_{ij} – ранг і-го параметра, визначений j-м експертом;

N – число експертів,

n – кількість параметрів.

- середня сума рангів:

$$T = \frac{1}{n}R = 17{,}5$$

де R – сума рангів кожного з параметрів;

n – кількість параметрів.

- відхилення суми рангів кожного параметра від середньої суми рангів:

$$\Delta_i = R_i - T,$$

де R_i – сума рангів показника і;

- T – середня сума рангів.

Сума відхилень по всім параметрам повинна дорівнювати 0;

- загальна сума квадратів відхилення:

$$S = \sum_{i=1}^{N} \Delta_i^2 = 193$$

де Δ_i – відхилення суми рангів параметра і від середньої суми рангів:

N – число експертів.

Порахуємо коефіцієнт узгодженості за наступною формулою:

$$W = \frac{12S}{N^2(n^3 - n)} = \frac{12 \cdot 193}{7^2(4^3 - 4)} = 0{,}788 > W_k = 0{,}67,$$

де S – загальна сума квадратів відхилення;

N – число експертів;

n – кількість параметрів.

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

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

Таблиця 4.4

Параметри	Експерти							Кінцева оцінка	Числове значення
	1	2	3	4	5	6	7		
X1 і X2	<	>	<	>	=	<	=	<	0.5
X1 і X3	<	<	<	<	<	<	<	<	0.5
X1 і X4	>	>	>	>	=	<	<	>	1,5
X2 і X3	<	<	>	<	<	<	<	<	0,5
X2 і X4	>	=	>	<	=	>	<	>	1,5
X3 і X4	>	>	>	>	>	>	>	>	1,5

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

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

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

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

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

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

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

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

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

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

Таблиця 4.5

Параметриx_i	Параметриx_j				Перша ітер.		Друга ітер.	
	X1	X2	X3	X4	b_i	$K_{\text{ві}}$	b_i^1	$K_{\text{ві}}^1$
X1	1	0,5	0,5	1,5	3,50	0,22	15,25	0,23
X2	0,5	1	0,5	1,5	3,50	0,22	15,25	0,23
X3	0,5	0,5	1	1,5	3,50	0,22	15,25	0,23
X4	1,5	1,5	1,5	1	5,50	0,34	21,25	0,32
Всього:					16,00	1,00	67,00	1,00

4.5 Аналіз рівня якості варіантів реалізації функції

Коефіцієнт технічного рівня для кожного варіанта реалізації програмного продукту розраховується за наступною формулою:

$$K_K(j) = \sum_{i=1}^{n} K_{\text{в}i,j} B_{i,j} ,$$

де n – кількість параметрів;

$K_{вi,j}$– коефіцієнт вагомості і-го параметра якості в сукупності взятих для розгляду параметрів якості;

$B_{i,j}$ – оцінка і-го параметра якості j-го варіанта виробу (в балах);

Результати розрахунків наведено у таблиці 4.6.

Таблиця 4.6

Основні функції	Варіант реалізації функції	Параметри	Абсолютне значення параметра	Бальна оцінка параметра	Коефіцієнт вагомості параметра	Коефіцієнт рівня якості
F1	а	X1	8000	7	0,23	1,61
F2	а	X2	4	8	0,23	1,84
F3	а	X3	25	8	0,23	1,84
	а	X4	2500	6	0,32	1,92
	б	X3	50	6	0,23	1,38
	б	X4	5000	9	0,32	2,88

За даними з таблиці 4.6 визначаємо рівень якості кожного з варіантів за формулою :

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

визначаємо рівень якості кожного з варіантів:

$$К_{К1} = 1,61 + 1,84 + 1,84 + 1,38 = 6,67;$$

$$К_{К2} = 1,61 + 1,84 + 1,92 + 2,88 = 8,25.$$

Як видно з розрахунків, кращим є другий варіант, для якого коефіцієнт технічного рівня має найбільше значення.

4.6 Економічний аналіз варіантів розробки програмного продукту

Для визначення вартості розробки ПП спочатку проведемо розрахунок трудомісткості.

Всі варіанти включають в себе два окремих завдання:

а) розробка проекту програмного продукту;

б) розробка програмної оболонки.

При цьому кожний з варіантів має різні додаткові завдання, які є реалізацією розгалужень варіантів розробки:

в) використання вбудованого інтерфейсу;

г) власна реалізація інтерфейсу.

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

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

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

Загальна трудомісткість обчислюється як

$$T_O = T_P \cdot K_П \cdot K_{СК} \cdot K_М \cdot K_{СТ} \cdot K_{СТ.М},$$

де T_P – трудомісткість розробки ПП;

$K_П$ – поправочний коефіцієнт;

$K_{СК}$ – коефіцієнт на складність вхідної інформації;

$K_М$ – коефіцієнт рівня мови програмування;

$K_{СТ}$ – коефіцієнт використання стандартних модулів і прикладних програм;

$K_{СТ.М}$ – коефіцієнт стандартного математичного забезпечення

Для першого завдання, виходячи із норм часу для завдань розрахункового характеру степеню новизни А та групи складності алгоритму 1, трудомісткість дорівнює: T_P =95 людино-днів. Поправочний коефіцієнт, який враховує вид нормативно-довідкової інформації для першого завдання: $K_П$ = 1.5. Поправочний коефіцієнт, який враховує складність контролю вхідної та вихідної інформації для всіх семи завдань рівний 1: $K_{СК}$ = 1. Оскільки при розробці першого завдання використовуються стандартні модулі, врахуємо це за допомогою коефіцієнта $K_{СТ}$ = 0.8. Тоді загальна трудомісткість програмування першого завдання дорівнює:

$$T_1 = 95 * 1{,}5 * 0{,}75 = 106{,}9 \text{ людино-днів.}$$

Проведемо аналогічні розрахунки для подальших завдань.

Для другого завдання (використовується алгоритм другої групи складності, степінь новизни Б), тобто T_P =40 людино-днів, $K_П$ =0.8,$K_{СК}$ = 1, $K_{СТ}$ = 0.75:

$$T_2 = 40 \cdot 0{,}8 \cdot 0{,}75 = 24 \text{ людино-днів.}$$

Для третього завадання (використовується алгоритм третьої групи складності, степінь новизни Б)), тобто $Т_Р$ =22 людино-днів, $К_П$ =0.8,$К_{СК}$ = 1,$К_{СТ}$ =0.7:

$$Т_З = 22 \cdot 0,8 \cdot 0,7 = 12,32 \text{ людино-днів.}$$

Для четвертого завадання (використовується алгоритм третьої групи складності, степінь новизни Г)), тобто $Т_Р$ = 12 людино-днів, $К_П$ =0.8,$К_{СК}$ = 1,$К_{СТ}$ =0.65:

$$Т_4 = 12 \cdot 0,8 \cdot 0,65 = 6,24 \text{ людино-днів.}$$

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

$$Т_І = (106,9 + 24 + 12,32) \cdot 8 = 1145,76 \text{ людино-годин.}$$

$$Т_{ІІ} = (106,9 + 24 + 6,24) \cdot 8 = 1097,12 \text{ людино-годин.}$$

Найбільш високу трудомісткість має варіант І.

В розробці беруть участь один програміст з окладом 22000 грн., один аналітик в області даних з окладом 25000 . Визначимо середню зарплату за годину за формулою:

$$С_Ч = \frac{М}{T_m \cdot t},$$

де М – місячний оклад працівників;

T_m – кількість робочих днів тиждень;

t – кількість робочих годин в день.

$$С_{ч} = \frac{22000 + 25000}{2 \cdot 21 \cdot 8} = 139{,}9 \text{ грн.}$$

Тоді, розрахуємо заробітну плату за формулою:

$$С_{ЗП} = С_{ч} \cdot Т_{i} \cdot К_{Д},$$

де $С_{ч}$– середня заробітна плата за годину;

$Т_{i}$ – трудомісткість відповідного завдання;

$К_{Д}$ – норматив, який враховує додаткову заробітну плату.

Зарплата розробників за варіантами становить:

I. $С_{ЗП} = 139{,}9 \cdot 1145{,}76 \cdot 1.2 = 192\,350{,}2$ грн.

II. $С_{ЗП} = 139{,}9 \cdot 1097{,}12 \cdot 1.2 = 184184{,}5$ грн.

Відрахування на єдиний соціальний внесок становить 22%:

I. $С_{ВІД} = С_{ЗП} \cdot 0.22 = 192\,350{,}2 \cdot 0.22 = 42317$ грн.

II. $С_{ВІД} = С_{ЗП} \cdot 0.22 = 184184{,}5 \cdot 0.22 = 40520{,}6$ грн.

Тепер визначимо витрати на оплату однієї машино-години. ($С_{М}$)

Так як одна ЕОМ обслуговує одного програміста з окладом 22000 грн., з коефіцієнтом зайнятості 0,2 то для однієї машини отримаємо:

$$С_{Г} = 12 \cdot М \cdot К_{З} = 12 \cdot 22000 \cdot 0{,}2 = 52800 \text{ грн.}$$

З урахуванням додаткової заробітної плати:

$$С_{ЗП} = С_{Г} \cdot (1 + К_{З}) = 52{,}800 \cdot (1 + 0.2) = 63360 \text{ грн.}$$

Відрахування на соціальний внесок:

$$С_{ВІД}= С_{ЗП} \cdot 0.22 = 63360 \cdot 0,22 = 13939,2 \text{ грн.}$$

Амортизаційні відрахування розраховуємо при амортизації 25% та вартості ЕОМ – 22500 грн.

$$С_А = К_{ТМ} \cdot К_А \cdot Ц_{ПР} = 1.15 \cdot 0.25 \cdot 22500 = 6468,75 \text{ грн.},$$

де $К_{ТМ}$– коефіцієнт, який враховує витрати на транспортування та монтаж приладу у користувача;

$К_А$– річна норма амортизації;

$Ц_{ПР}$ – договірна ціна приладу.

Витрати на ремонт та профілактику розраховуємо як:

$$С_Р = К_{ТМ} \cdot Ц_{ПР} \cdot К_Р = 1.15 \cdot 22500 \cdot 0.05 = 1293,75 \text{ грн.},$$

де $К_Р$– відсоток витрат на поточні ремонти.

Ефективний годинний фонд часу ПК за рік розраховуємо за формулою:

$$Т_{ЕФ} =(Д_К – Д_В – Д_С – Д_Р) \cdot t_З \cdot К_В = (365 – 104 – 8 – 16) \cdot 8 \cdot 0.9 = 1706,4 \text{ годин},$$

де $Д_К$ – календарна кількість днів у році;

$Д_В$, $Д_С$ – відповідно кількість вихідних та святкових днів;

$Д_Р$ – кількість днів планових ремонтів устаткування;

t –кількість робочих годин в день;

$К_В$– коефіцієнт використання приладу у часі протягом зміни.

Витрати на оплату електроенергії розраховуємо за формулою:

$$С_{ЕЛ} = Т_{ЕФ} \cdot N_С \cdot К_З \cdot Ц_{ЕН} = 1706,4 \cdot 0,5 \cdot 0,85 \cdot 3,51547 = 2549,5 \text{ грн.},$$

Де $N_С$ – середньо-споживча потужність приладу;

$К_З$– коефіцієнтом зайнятості приладу;

$Ц_{ЕН}$ – тариф за 1 КВт-годин електроенергії.

Накладні витрати розраховуємо за формулою:

$$С_Н = Ц_{ПР} \cdot 0.67 = 22500 \cdot 0,67 = 15075 \text{ грн.}$$

Тоді, річні експлуатаційні витрати будуть:

$$С_{ЕКС} = С_{ЗП} + С_{ВІД} + С_А + С_Р + С_{ЕЛ} + С_Н,$$

$$С_{ЕКС} = 63360 + 13939,2 + 6468,75 + 1293,75 + 2549,5 + 15075 = 102\,686,2 \text{ грн.}$$

Собівартість однієї машино-години ЕОМ дорівнюватиме:

$$С_{М\text{-}Г} = С_{ЕКС} / Т_{ЕФ} = 102\,686,2 / 1706,4 = 60 \text{ грн/год.}$$

Оскільки в даному випадку всі роботи, які пов‘язані з розробкою програмного продукту ведуться на ЕОМ, витрати на оплату машинного часу, в залежності від обраного варіанта реалізації, складає:

$$С_М = С_{М-Г} \cdot Т,$$

I. $С_М = 60 \cdot 1145,76 = 68\,745,6$ грн.

II. $С_М = 60 \cdot 1097,12 = 65\,827,2$ грн.

Накладні витрати складають 67% від заробітної плати:

$$С_Н = С_{ЗП} \cdot 0,67,$$

I. $С_Н = 192\ 350,2 \cdot 0,67 = 128\ 874,6$ грн

II. $С_Н = 184184,5 \cdot 0,67 = 123\ 403,6$ грн.

Отже, вартість розробки ПП за варіантами становить:

$$С_{ПП} = С_{ЗП} + С_{ВІД} + С_М + С_Н,$$

I. $С_{ПП} = 192\ 350,2 + 42317 + 68\ 745,6 + 128\ 874,6 = 432\ 287,4$ грн.

II. $С_{ПП} = 184184,5 + 40520,6 + 65\ 827,2 + 123\ 403,6 = 413\ 935,9$ грн.

4.7 Вибір кращого варіанта програмного продукту техніко-економічного рівня

Розрахуємо коефіцієнт техніко-економічного рівня за формулою:

$$К_{ТЕР_j} = К_{Кj} / С_{Фj},$$

$$К_{ТЕР1} = 6,67/ 432\ 287,4 = 1,54 \cdot 10^{-5},$$

$$К_{ТЕР2} = 8,25/ 413\ 935,9 = 1,99 \cdot 10^{-5}.$$

Як бачимо, найбільш ефективним є другий варіант реалізації програми з коефіцієнтом техніко-економічного рівня $К_{ТЕР1} = 1,99 \cdot 10^{-5}$.

Після виконання функціонально-вартісного аналізу програмного комплексу що розроблюється, можна зробити висновок, що з альтернатив, що залишились після першого відбору двох варіантів виконання програмного комплексу оптимальним є другий варіант реалізації програмного продукту. У нього виявився найкращий показник техніко-економічного рівня якості $К_{ТЕР} = 1,99 \cdot 10^{-5}$.

Цей варіант реалізації програмного продукту має такі параметри:

- мова програмування – Python;
- використання готових бібліотек;
- вибір в якості середовища розробки Visual Code.

Даний варіант виконання програмного комплексу дає користувачу зручний інтерфейс, непоганий функціонал і швидкодію.

4.8 Висновки

В рамках функціонально-вартісного аналізу визначено та проведено оцінку основних функцій, визначено параметри, які характеризують програмний продукт, проведено експертне оцінювання параметрів та аналіз якості варіантів реалізації функцій, проведено економічний аналіз варіантів розробки.

В результаті проведено аналіз двох найбільш доцільних варіантів розробки програмного продукту з’ясовано, що найкращим є другий варіант як за коефіцієнтом технічного рівня так і за коефіцієнтом техніко-економічного рівня.

ВИСНОВКИ

В рамках даної роботи проведено огляд підходів, що використовуються для побудови прогнозів. Досліджено властивості багатовимірних процесів, що повинні враховуватись при вирішенні задачі прогнозування.
Досліджено методи прогнозування різних класів, зокрема моделі ВАР, дерев рішень та алгоритмів, які базуються на них, мережі LSTM, критерії якості моделей, що описують динаміку розвитку процесу, критерії якості прогнозів.

Проведено порівняльний аналіз прогнозів, отриманих за допомогою моделей ВАР, бустінгу над деревами рішень, мереж LSTM на тестовому датасеті, який містить дані змін кількості хімічних часток в повітрі.

Розроблено програмний продукт для побудови прогнозів за допомогою розглянутих вище методів, що надає інструменти для попередньої обробки даних, побудови графіків, корелограм, виведення статистичних критеріїв, вибору параметрів моделей, параметрів процедур навчання.

Модель ВАР погано апроксимує динаміку змін процесів різної природи, що зумовлено наявністю нелінійних залежностей та високими впливом невимірюваних впливів.

Модель бустінгу над деревами рішень при прогнозуванні всіх розглянутих рядів є однією з найефективніших, проте якість прогнозів залежить від розміру навчальної вибірки. Та тренування моделі потребує небагато часу.

Модель LSTM не поступається в ефективності моделі MLP на розглянутих даних за умови, що навчальна вибірка має велику кількість вимірів, інакше якість прогнозів суттєво погіршується, але потребує великої кількості ресурсів та часу для тренування моделі.

LGBM та LSTM можуть ефективно використовуватись для побудови як короткострокових так і довгострокових прогнозів, однак також потребують попередньої обробки даних та значно більшої кількості як обчислювальних так і часових ресурів.

В рамках подальших досліджень доцільно розглянути наступні напрямки:

- модифікація моделі ВАР, що дозволить враховувати сезонність;
- використання інших методів для побудови прогнозів, наприклад моделі нелінійної регресії, інші види моделей машинного навчання;
- додання до навчальної вибірки фіч різної природи, наприклад дані часу проведення спостережень.
- поліпшення користувацького інтерфейсу.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Бідюк П.І., Романенко В.Д., Тимощук О.Л. Аналіз часових рядів (навчальний посібник). Київ: Політехніка, 2010. 317 с.
2. Robert M. Kunst Nonlinear time series URL: https://homepage.univie.ac.at/robert.kunst/nonlin1.pdf (дата звернення: 10.04.2021).
3. Detecting stationarity in time series data URL: https://towardsdatascience.com/detecting-stationarity-in-time-series-data-d29e0a21e638 (дата звернення: 13.05.2021).
4. Dickey D. A. and Fuller W. A. Distribution of the Estimators for Autoregressive Time Series with a Unit Root / Journal of the American Statistical Association. 74. 1979. p. 427-431.
5. Wei W.S. Multivariate Time Series Analysis and Applications: 2019 URL: https://www.wiley.com/en-us/Multivariate+Time+Series+Analysis+and+Applications-p-9781119502852 (дата звернення: 22.04.2021).
6. Wayne A. Fuller Introduction to Statistical Time Series Second Edition First published:15 December 1995 URL: https://onlinelibrary.wiley.com/doi/book/10.1002/9780470316917 (дата звернення: 23.04.2021).
7. Villez K. Multivariate and qualitative data-analysis for monitoring, diagnosis and control of sequencing batch reactors for wastewater treatment: 2007 URL: https://www.researchgate.net/publication/231169980_Multivariate_and_qualitative_data-analysis_for_monitoring_diagnosis_and_control_of_sequencing_batch_reactors_for_wastewater_treatment (дата звернення: 5.05.2021

8. Jansen S. Machine Learning for Algorithmic Trading Second Edition: 2020 URL: https://www.amazon.com/Machine-Learning-Algorithmic-Trading-alternative/dp/1839217715 (дата звернення: 28.04.2021).
9. Бідюк П.І. моделювання і прогнозування гетероскедастичних процесів / Системні дослідження та інформаційні технології, 2004, № 1 URL: http://dspace.nbuv.gov.ua/bitstream/handle/123456789/50334/10-Bidyuk.pdf?sequence=1 (дата звернення: 05.05.2021).
10. Halls- Moore M.L. Advanced algorithmic trading. 2016 URL: https://www.twirpx.com/file/2191921/ (дата звернення: 10.05.2021).
11. Montgomery. Douglas C. Introduction to time series analysis and forecasting I Douglas C. Montgomery. Cheryl L. Jennings, Murat Kulahci. - (Wiley series in probability and statistics) Includes bibliographical references and index. ISBN 978-0-4 71-65397-4 (cloth) Published by John Wiley & Sons. Inc .. Hoboken. New Jersey 2008 455p.
12. Hesham K. Alfares and Mohammad Nazeeruddin Electric load forecasting: literature survey and classification of methods International / Journal of Systems Science, 2002, vol. 33, № 1, P. 23-34
13. Tsay, Ruey S.Analysis of financial time series / Ruey S. Tsay. 3rd ed. p. cm. (Wiley series in probability and statistics) ISBN 978-0-470-41435-4 (cloth)., 1951. 671p.
14. Norman R. Draper, Harry Smith Applied Regression Analysis, 3rd Edition ISBN: 978-0-471-17082-2. April 1998. 736p.
15. Прогнозирование: Принципы и практика / Forecasting: Principles and Practice (2nd ed) URL: https://otexts.com/fpp2/ (дата звернення: 02.05.2021).
16. E. Fiesler Neural Network Classication and Formalization URL: http://citeseerx.ist.psu.edu/viewdoc/download;jsessionid=058F51732669552BB0362DCDF25AFDFE?doi=10.1.1.30.9696&rep=rep1&type=pdf (дата звернення: 08.05.2021).

17. Haykin, Simon Neural networks and learning machines / Simon Haykin.—3rd ed. Rev. ed of: Neural networks. 2nd ed., 1999.ISBN-13: 978-0-13-147139-9 ISBN-10: 0-13-147139-2 906 URL: http://dai.fmph.uniba.sk/courses/NN/haykin.neural-networks.3ed.2009.pdf (дата звернення: 08.05.2021).
18. Xuan-Hien Le, Hung Viet Ho , Giha Lee, Sungho Jung Application of Long Short-Term Memory (LSTM) Neural Network for Flood Forecasting URL: https://www.researchgate.net/publication/334268507_Application_of_Long_Short-Term_Memory_LSTM_Neural_Network_for_Flood_Forecasting#pf13 (дата звернення: 12.05.2021).

ДОДАТОК А ЛІСТИНГ ПРОГРАМИ

Файл config.py

```
from hyperopt import hp, fmin, tpe, Trials, STATUS_OK
hyper_space_lgbm = {
    'learning_rate':    hp.uniform('learning_rate',0.001,0.1),
    'num_leaves':       hp.quniform('num_leaves', 31, 100, q=1),
    'min_split_gain':   hp.uniform('min_split_gain', 0, 1),
    'reg_alpha':        hp.uniform('reg_alpha',0,1),
    'reg_lambda':       hp.uniform('reg_lambda',0,1),
    'n_estimators':     hp.quniform('n_estimators', 100, 250, q=1) }
trials_num = 20
```

Файл validation.py

```
from sklearn.metrics import mean_absolute_error, mean_squared_error
import pandas as pd
import warnings
from lightgbm import LGBMRegressor
import lightgbm
from sklearn.model_selection import TimeSeriesSplit
from sklearn.model_selection import cross_val_score
from hyperopt import hp, fmin, tpe, Trials, STATUS_OK
import numpy as np
```

```
from functools import partial
from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score
warnings.simplefilter('ignore')
def train_valid_split(df,size=0.8):
    train = df[:int(size*(len(df)))]
    valid = df[int(size*(len(df))):]
    return train,valid

def train_test_split_func(X,y,size=0.25):
    train_x, val_x, train_y, val_y = train_test_split(X,y,test_size=size)
    return train_x, val_x, train_y, val_y

def metrics(df_true,df_pred):
    val_metrics = pd.DataFrame(columns=['column','mae','mse'])
    for i,column in enumerate(df_true):
        mae=mean_absolute_error(df_true[column],df_pred[column])
        mse=mean_squared_error(df_true[column],df_pred[column])
        r2 = r2_score(df_true[column],df_pred[column])
            val_metrics =
      val_metrics.append({'column':column,'mae':mae,'mse':mse,'R^2':r2},ignore_
      index=True)
    return val_metrics
```

```
def evaluate_metric(params,X_train,Y_train,max_evals,model):
    if params is not None:
        if type(model) is lightgbm.sklearn.LGBMRegressor:
            params['n_estimators'] = np.int(params['n_estimators'])
            params['num_leaves'] = np.int(params['num_leaves'])
        model.set_params(**params)
    model.set_params(**params)
    num_splits = 5
    rmse_score = 0
    kf = TimeSeriesSplit(n_splits=num_splits)
    for i, (train_index, test_index) in enumerate(kf.split(X_train)):
        y_train, y_valid = pd.DataFrame(Y_train.iloc[train_index].copy()), pd.DataFrame(Y_train.iloc[test_index])
        x_train, x_valid = pd.DataFrame(X_train.iloc[train_index, :].copy()), pd.DataFrame(X_train.iloc[test_index, :].copy())
        fit_model = model.fit(x_train, y_train)
        pred = model.predict(x_valid)
        rmse_score += mean_squared_error(y_valid, pred)
        del y_valid, x_train, x_valid, y_train
    rmse_score/=num_splits
    return  {'loss': rmse_score, 'params': params,'status': STATUS_OK}
```

```
def get_best_params(X_train,Y_train,max_evals,model,hyper_space):
    trials = Trials()
    best_params = fmin(
            fn=partial(evaluate_metric, X_train=X_train,
Y_train=Y_train,max_evals=max_evals,model = model),
            space=hyper_space,
            algo=tpe.suggest,
            max_evals=max_evals,
            trials=trials,
            rstate=np.random.RandomState(1),
            show_progressbar=True
        )
    return best_params


def series_to_supervised(data, n_in=30,n_out= 2, predict=True):
    c_names = data.columns
    n_vars = 1 if type(data) is list else data.shape[1]
    df = pd.DataFrame(data)
    cols, names = list(), list()
    for i in range(n_in, 0, -1):
        cols.append(df.shift(i))
        names += ['%s(t-%d)' % (n, i) for n in c_names]
    for i in range(0, n_out):
```

```
        cols.append(df.shift(-i))
        if i == 0:
            names += [('%s(t)' % n) for n in c_names]
        else:
            names += [('%s(t+%d)' % (n, i)) for n in c_names]
    agg = pd.concat(cols, axis=1)
    agg.columns = names
    test = agg.tail(1)
    agg.dropna(inplace=True)
    if predict:
        agg = agg.append(test,ignore_index=True)
    return agg


def cross_val_prediction(model,params,X_train,Y_train,X_test):
    if params is not None:
        if type(model) is lightgbm.sklearn.LGBMRegressor:
            params['n_estimators'] = np.int(params['n_estimators'])
            params['num_leaves'] = np.int(params['num_leaves'])
        model.set_params(**params)
    num_splits = 5
    kf = TimeSeriesSplit(n_splits=num_splits)
    y_test_pred = pd.Series([0 for i in range(len(X_test))])
```

```
    for i, (train_index, test_index) in enumerate(kf.split(X_train)):
        y_train, y_valid = pd.DataFrame(Y_train.iloc[train_index].copy()), pd.DataFrame(Y_train.iloc[test_index])
        x_train, x_valid = pd.DataFrame(X_train.iloc[train_index, :].copy()), pd.DataFrame(X_train.iloc[test_index, :].copy())
        fit_model = model.fit(x_train, y_train)
        pred = model.predict(X_test)
        y_test_pred += pred.squeeze()
        del y_valid, x_train, x_valid, y_train
    y_test_pred = y_test_pred / num_splits
    return y_test_pred
```

Файл prediction_methods.py

```
from statsmodels.tsa.vector_ar.var_model import VAR
from statsmodels.tsa.statespace.varmax import VARMAX
from statsmodels.tsa.arima_model import ARMA
from collections import Counter
from itertools import chain
from lightgbm import LGBMRegressor
from sklearn.linear_model import LinearRegression
from data import find_cointegrated_pairs,pq_calc,common_counter
from statsmodels.tsa.holtwinters import ExponentialSmoothing
import pandas as pd
```

```
from sklearn.metrics import mean_absolute_error, mean_squared_error
from sklearn.model_selection import TimeSeriesSplit
from statsmodels.tsa.arima.model import ARIMA
import warnings
import numpy as np
from validation import evaluate_metric,get_best_params,cross_val_prediction
from config import hyper_space_lgbm
from validation import series_to_supervised, train_valid_split
from config import trials_num
warnings.simplefilter('ignore')
def varma_prediction(train,test,steps):
    p,q = get_var_pq_params(train)
    model = VARMAX(train,order=(p, q))
    model_fit = model.fit(disp=False)
    if not steps:
        prediction = model_fit.forecast(steps=len(test))
    else:
        prediction = model_fit.forecast(steps=steps)
    multi_predicts_df = pd.DataFrame(prediction, columns = train.columns)
    return multi_predicts_df


def arma_prediction(train,test,steps=None):
```

```
    prediction_df = pd.DataFrame(columns = train.columns)
    for col in train.columns:
        p,q = get_arma_pq_params(train[col])
        model = ARMA(train[col], order= (p,q))
        model_fit = model.fit(disp=False)
        if not steps:
            prediction = model_fit.forecast(steps=len(test[col]))
        else:
            prediction = model_fit.forecast(steps=steps)
        prediction_df[col] = prediction
    return prediction_df


def boosting_validation(df,train_columns,valid_size,inverting_times):
    ma = get_best_ma(df,train_columns,valid_size)
    print("BEST_MA",ma)
    t1_columns = [column+str("(t+1)") for column in train_columns]
    t_columns = [column+str("(t)") for column in train_columns]
    train = residuals_add(df,t_columns,ma,inverting_times,predict= False)
    train, test = train_valid_split(train,valid_size)
    prediction_df = pd.DataFrame()
    true_y = pd.DataFrame()
    for column in t1_columns:
```

```
        train_y = train[column]
        train_x = train.drop(t1_columns,axis = 1)
        test_y = test[column]
        test_x = test.drop(t1_columns,axis = 1)
        lgbm_model = LGBMRegressor()
        lgbm_best_params = get_best_params(train_x,train_y,trials_num,lgbm_model,hyper_space_lgbm)
        lgbm_predicts = cross_val_prediction(lgbm_model,lgbm_best_params,train_x,train_y,test_x)
        column = column[:-5]
        prediction_df[column] = lgbm_predicts
        true_y[column] = test_y
    return prediction_df,true_y,ma,lgbm_best_params


def boosting_prediction(df,train_columns,steps,ma,best_params,inverting_times):
    t1_columns = [column+str("(t+1)") for column in train_columns]
    t_columns = [column+str("(t)") for column in train_columns]
    train = residuals_add(df,t_columns,ma,inverting_times,predict= True)
    test = train.tail(1)
    train = train.dropna(axis = 0)
    prediction_df = pd.DataFrame()
    all_predicts = pd.DataFrame(columns = train_columns)
    for column in t1_columns:
```

```
        train_y = train[column]
        train_x = train.drop(t1_columns,axis = 1)
        test_y = test[column]
        test_x = test.drop(t1_columns,axis = 1)
        lgbm_model = LGBMRegressor()
        lgbm_predicts = cross_val_prediction(lgbm_model,best_params,train_x,train_y,test_x)
        column = column[:-5]
        prediction_df[column] = lgbm_predicts
    df = df.append(prediction_df,ignore_index=True)
    all_predicts = all_predicts.append(prediction_df,ignore_index=True)
    for i in range(steps-1):
        train = residuals_add(df,t_columns,ma,inverting_times,predict= True)
        test = train.tail(1)
        train = train.dropna(axis = 0)
        prediction_df = pd.DataFrame()
        for column in t1_columns:
            train_y = train[column]
            train_x = train.drop(t1_columns,axis = 1)
            test_y = test[column]
            test_x = test.drop(t1_columns,axis = 1)
            lgbm_model = LGBMRegressor()
```

```
            lgbm_predicts =
cross_val_prediction(lgbm_model,best_params,train_x,train_y,test_x)
            column = column[:-5]
            prediction_df[column] = lgbm_predicts
        df = df.append(prediction_df,ignore_index=True)
        all_predicts = all_predicts.append(prediction_df,ignore_index=True)
    return all_predicts


def get_best_ma(df,train_columns,valid_size):
    t1_columns = [column+str("(t+1)") for column in train_columns]
    t_columns = [column+str("(t)") for column in train_columns]
    best_rmse = np.inf
    best_ma = 0
    break_count = 0
    mas = np.arange(10,df.shape[0]//2,10)
    for ma in mas:
        train = series_to_supervised(df,ma,predict= False)
        train, test = train_valid_split(train,valid_size)
        prediction_df = pd.DataFrame()
        kf = TimeSeriesSplit(n_splits=10)
        true_y = pd.DataFrame()
        for column in t1_columns[:1]:
            train_y = train[column]
```

```
            train_x = train.drop(t1_columns,axis = 1)
            test_y = test[column]
            test_x = test.drop(t1_columns,axis = 1)
            model = LGBMRegressor()
            pred = cross_val_prediction(model,None,train_x,train_y,test_x)
            mse = mean_squared_error(test_y, pred)
            if mse < best_rmse:
                best_rmse = mse
                best_ma = ma
                break_count = 0
            else:
                break_count+=1
            if break_count==3:
                break
    return best_ma


def get_var_pq_params(integrated_ts):
    p_coeffs = []
    q_coeffs = []
    for col in integrated_ts.columns:
        p,q = pq_calc(integrated_ts[col])
        p_coeffs.append(p)
```

```
        q_coeffs.append(q)
    p = common_counter(p_coeffs)
    q = common_counter(q_coeffs)
    if p == 0 and q == 0:
        p = 1
    return p,q


def residuals_add(train,train_columns,ma,inverting_times,predict):
    train = series_to_supervised(train,n_in = ma,predict = predict)
    resid_columns = ['ARIMA_'+str(column) for column in train_columns] + ['HWES_'+str(column) for column in train_columns]
    resid_df = pd.DataFrame(columns = resid_columns)
    for column in train_columns:
        p,q = pq_calc(train[column])
        i = 1
        if column in inverting_times.keys():
            d = 1
        else:
            d = 0
        model = ARIMA(train[column], order= (p,d,q))
        while True:
            try:
                model_fit = model.fit()
```

```
            except:
                i+=1
                p,q = p-1, q-1
                model = ARIMA(train[column], order= (p,d,q))
            else:
                break
        resid_df['ARIMA_'+str(column)] = model_fit.resid
        if resid_df['ARIMA_'+str(column)].isna().any():
            resid_df = resid_df.drop('ARIMA_'+str(column), axis = 1)
        model = ExponentialSmoothing(train[column])
        model_fit = model.fit()
        resid_df['HWES_'+str(column)] = model_fit.resid
        if resid_df['HWES_'+str(column)].isna().any():
            resid_df = resid_df.drop('HWES_'+str(column), axis = 1)
    train = pd.concat([train,resid_df],axis = 1)
    return train
```

Файл data.py

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

```
import pandas as pd
import statsmodels.api as sm
from statsmodels.tsa.api import VAR
from statsmodels.tsa.stattools import adfuller
from statsmodels.tsa.stattools import acf, pacf,q_stat, adfuller
from numpy import cumsum, log, polyfit, sqrt, std, subtract
from scipy.stats import spearmanr, pearsonr,probplot, moment
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
from collections import Counter
from itertools import chain
import scipy.stats as stats
from fracdiff import Fracdiff, FracdiffStat, fdiff
import warnings
warnings.simplefilter('ignore')


def read_df(filename):
    try:
        if filename[-4:]=='xlsx':
            df = pd.read_excel(filename, index_col=None)
        elif filename[-3:]=='csv':
            df = pd.read_csv(filename, index_col=None)
        else:
```

```
            print("You can pass only csv or xlsx")
        df = df.dropna(axis='columns', how ='all')
        df = df.dropna(axis = 0, how = 'all')
        df = df.head(500)
        df = df.iloc[:,3:7]
        return df
    except:
        print("Bad format")
        return None

def plot_timeseries(df):
    df.plot(subplots=True,figsize=(400,400))
    plt.show()

def stationarity_check(df,signif=0.05):
    non_stat_cols=[]
    for column in df.columns:
        ts = df[column]
        dftest = adfuller(ts, autolag='AIC')
        adf = pd.Series(dftest[0:4], index=['Test Statistic','p-value','# Lags','# Observations'])
        for key,value in dftest[4].items():
            adf['Critical Value (%s)'%key] = value
```

```
        p = adf['p-value']
        if p <= signif:
            stationarity = 1
        else:
            stationarity = 0
            non_stat_cols.append(column)
    return non_stat_cols


def make_stationary(df):
    df_copy = df.copy()
    non_stat_cols= stationarity_check(df_copy)
    column_diff_dict = dict.fromkeys(non_stat_cols, 0)
    if len(non_stat_cols) != 0:
        for column in non_stat_cols:
            f = FracdiffStat()
            df_copy[column]=
pd.Series(f.fit_transform(df_copy[column].values.reshape(-1,1)).squeeze())
            column_diff_dict[column] = (f.d_[0],f.window)
    else:
        return df_copy,column_diff_dict
    return df_copy,column_diff_dict


def plot_correlogram(x, fig, axes,lags=40, title=None):
```

```
    lags = min(10, int(len(x)/5)) if lags is None else lags
    x.plot(ax=axes[0][0])
    q_p = np.max(q_stat(acf(x, nlags=lags), len(x))[1])
    stats = f'Q-Stat: {np.max(q_p):>8.2f}\nADF: {adfuller(x)[1]:>11.2f} \nHurst: {round(hurst(x.values),2)}'
    axes[0][0].text(x=.02, y=.85, s=stats, transform=axes[0][0].transAxes)
    probplot(x, plot=axes[0][1])
    mean, var, skew, kurtosis = moment(x, moment=[1, 2, 3, 4])
    s = f'Mean: {mean:>12.2f}\nSD: {np.sqrt(var):>16.2f}\nSkew: {skew:12.2f}\nKurtosis:{kurtosis:9.2f}'
    axes[0][1].text(x=.02, y=.75, s=s, transform=axes[0][1].transAxes)
    plot_acf(x=x, lags=lags, zero=False, ax=axes[1][0])
    plot_pacf(x=x, lags=lags, zero=False, ax=axes[1][1])
    axes[1][0].set_xlabel('Lag')
    axes[1][1].set_xlabel('Lag')
    fig.suptitle(title, fontsize=20)
    fig.tight_layout()
    fig.subplots_adjust(top=.9)
def plot_pred_graphs(ts,predict_ts,val_ts,widget_method,title,fig, ax1, ax2):
    if widget_method == "BoDT" or widget_method == "LSTM":
        ts_copy = ts[:-len(val_ts)]
        ts_copy = ts_copy.append(val_ts,ignore_index=True)
        val_ts = ts_copy.tail(len(val_ts))
```

```
        ts_copy = ts.append(predict_ts,ignore_index=True)
        predict_ts = ts_copy.tail(len(predict_ts))
        del ts_copy
    ax1.plot(ts,'blue',label='Actual')
    ax1.plot(val_ts,'green',label='Validation')
    ax2.plot(ts,'blue',label='Actual')
    ax2.plot(predict_ts,'green',label='Predicted')
    ax1.set_title("Validation")
    ax2.set_title("Nstep prediction")
    ax1.legend()
    ax2.legend()
    fig.suptitle(title, fontsize=20)
    fig.tight_layout()
    fig.subplots_adjust(top=.9)


def hurst(ts):
    lags = range(2, 100)
    tau = [sqrt(std(subtract(ts[lag:], ts[:-lag]))) for lag in lags]
    poly = np.polyfit(log(lags), log(tau), 1)
    return poly[0]*2.0
```

```
def grangers_causation_matrix(data, variables, maxlag=40,test='ssr_chi2test', verbose=False):
    df = pd.DataFrame(np.zeros((len(variables), len(variables))), columns=variables, index=variables)
    for c in df.columns:
        for r in df.index:
            test_result = grangercausalitytests(data[[r, c]], maxlag=maxlag, verbose=False)
            p_values = [round(test_result[i+1][0][test][1],4) for i in range(maxlag)]
            if verbose: print(f'Y = {r}, X = {c}, P Values = {p_values}')
            min_p_value = np.min(p_values)
            df.loc[r, c] = min_p_value
    df.columns = [var + '_x' for var in variables]
    df.index = [var + '_y' for var in variables]
    return df


def find_cointegrated_pairs(dataframe, critial_level = 0.05):
    n = dataframe.shape[1]
    pvalue_matrix = np.ones((n, n))
    keys = dataframe.columns
    pairs = []
    for i in range(n):
        for j in range(i+1, n):
```

```
            series1 = dataframe[keys[i]]
            series2 = dataframe[keys[j]]
            result = sm.tsa.stattools.coint(series1, series2)
            pvalue = result[1]
            pvalue_matrix[i, j] = pvalue
            if pvalue < critial_level:
                pairs.append((keys[i], keys[j], pvalue))
    integrated_columns = []
    for pair in pairs:
        integrated_columns.append(pair[0])
        integrated_columns.append(pair[1])
    integrated_columns=list(set(integrated_columns))
    return integrated_columns, pairs


def param_calc(arr,alpha,nlags,nobs):
    var = np.ones(nlags + 1) / nobs
    var[0] = 0
    var[1] = 1. / nobs
    var[2:] *= 1 + 2 * np.cumsum(arr[1:-1]**2)
    interval = stats.norm.ppf(1 - alpha / 2.) * np.sqrt(var)
    print(interval)
    count = 0
```

```
    max_count = 0
    for i in range(1,len(interval)):
        if arr[i]>interval[i] or arr[i]< - interval[i]:
            count+=1
        else:
            count = 0
        if count>max_count:
            max_count = count
    return max_count


def pq_calc(ts):
    alpha = 0.05
    nobs  = len(ts)
    nlags = min(40,len(ts)//2 - 1)
    acf, _ = sm.tsa.acf(ts,nlags = nlags, alpha=alpha)
    pacf, _ = sm.tsa.pacf(ts,nlags = nlags, alpha=alpha)
    p = param_calc(pacf,alpha,nlags,nobs)
    q = param_calc(acf,alpha,nlags,nobs)
    return p,q


def get_single_columns(df,integrated_columns):
    single_columns = list(set(df.columns) - set(integrated_columns))
```

```
    return single_columns


def common_counter(coeff_list):
    most_common_list =Counter(coeff_list).most_common()
    high = most_common_list[0][1]
    h_index= most_common_list[0][0]
    for pair in most_common_list:
        if pair[1] == high:
            h_index = pair[0]
        else:
            continue
    return h_index


def multi_single_ts(df):
    integrated_columns, pairs = find_cointegrated_pairs(df)
    single_columns = list(set(df.columns) - set(integrated_columns))
    return df[integrated_columns], df[single_columns]


def invert_diff(df_forecast, columns_diff_dict):
    df_fc = df_forecast.copy()
    columns = columns_diff_dict.keys()
    for col in columns:
```

```
        f = Fracdiff(d = - columns_diff_dict[col][0],window = columns_diff_dict[col][1])
        diff = f.fit_transform(df_forecast[col].values.reshape(-1,1))
        df_fc[col] = pd.Series(diff.squeeze())
    return df_fc


def series_to_supervised(data, n_in=30, dropnan=True, predict = True):
    c_names = data.columns
    n_out=2
    n_vars = 1 if type(data) is list else data.shape[1]
    df = pd.DataFrame(data)
    cols, names = list(), list()
    for i in range(n_in, 0, -1):
        cols.append(df.shift(i))
        names += ['%s(t-%d)' % (n, i) for n in c_names]
    for i in range(0, n_out):
        cols.append(df.shift(-i))
        if i == 0:
            names += [('%s(t)' % n) for n in c_names]
        else:
            names += [('%s(t+%d)' % (n, i)) for n in c_names]
    agg = pd.concat(cols, axis=1)
    agg.columns = names
```

```
    test = agg.tail(1)
    if dropnan:
        agg.dropna(inplace=True)
    if predict:
        agg = agg.append(test,ignore_index=True)
    return agg
```

Файл windows.py

```
from PyQt5 import QtWidgets,QtWebEngineWidgets,QtCore
from PyQt5.QtWidgets import *
import PyQt5.QtGui as QtGui
from PyQt5.QtCore import QAbstractTableModel, Qt
from data import *
import pandas as pd
import numpy as np
import time
from validation import train_valid_split,metrics
from prediction_methods import varma_prediction,arma_prediction,boosting_validation,boosting_prediction
import matplotlib.pyplot as plt
from matplotlib.backends.backend_qt5agg import FigureCanvasQTAgg as FigureCanvas
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
```

```
from qasync import asyncSlot
from PyQt5.QtCore import QObject, QThread, pyqtSignal
import warnings
warnings.simplefilter('ignore')


class MainWindow(QWidget):
    def __init__(self,file_path):
        super().__init__()
        self.df= read_df(file_path)
        self.stationary_df, self.inverting_times = make_stationary(self.df)
        self.w_prediction =
Window_predictions(self.df,self.stationary_df,self.inverting_times)
        self.w_graphs = Window_graphs(self.df)
        self.model_df = pandasModel(self.df)
        self.view = QTableView()
        self.setWindowTitle("TimeSeriesPredictionSystem")
        self.Button1 = QPushButton("Датасет")
        self.Button2 = QPushButton("Графики")
        self.Button3 = QPushButton("Предсказание")
        self.Button1.setFixedSize(300, 80)
        self.Button2.setFixedSize(300, 80)
        self.Button3.setFixedSize(300, 80)
        self.layout = QGridLayout()
```

```
        self.layout.addWidget(self.Button1, 0, 0)
        self.layout.addWidget(self.Button2, 0, 1)
        self.layout.addWidget(self.Button3, 0, 2)
        self.layout.setHorizontalSpacing(5)
        self.layout.setVerticalSpacing(5)
        self.Button1.clicked.connect(self.dataset_func)
        self.Button2.clicked.connect(self.show_graphs)
        self.Button3.clicked.connect(self.show_predict)
        self.Button1.setFont(QtGui.QFont("Serif", 10, QtGui.QFont.Bold))
        self.Button2.setFont(QtGui.QFont("Serif", 10, QtGui.QFont.Bold))
        self.Button3.setFont(QtGui.QFont("Serif", 10, QtGui.QFont.Bold))
        self.setLayout(self.layout)

    def show_graphs(self):
        self.w_graphs.show()

    def show_predict(self):
        self.w_prediction.show()

    def dataset_func(self):
        self.show_tables(self.view,self.df)
```

```
    def plot_func(self):
        plot_timeseries(self.df)


    def show_tables(self,view,df):
        df = pandasModel(df)
        view.setModel(df)
        view.move(600, 500)
        view.resize(800,600)
        view.resizeColumnsToContents()
        view.show()


class Window_graphs(QWidget):
    def __init__(self,df):
        super().__init__()
        self.df = df
        self.setWindowTitle("Графики")
        self.Button1 = QPushButton("Plot")
        self.label = QLabel('Выберите временной ряд', self)
        self.Button1.setFixedSize(200, 50)
        self.label.setFixedSize(300,50)
        self.fig, self.axes = plt.subplots(nrows=2, ncols=2, figsize=(12, 16))
        self.canvas = FigureCanvas(self.fig)
```

```
        self.listwidget = QListWidget()
        self.listwidget.setFixedSize(300,100)
        for i,col in enumerate(self.df.columns):
            self.listwidget.insertItem(i,col)
        layout = QGridLayout()
        layout.addWidget(self.Button1, 2, 0)
        layout.addWidget(self.label, 0, 0)
        layout.addWidget(self.listwidget, 1, 0)
        layout.addWidget(self.canvas, 3, 0)
        layout.setHorizontalSpacing(10)
        layout.setVerticalSpacing(10)
        self.Button1.clicked.connect(self.plot_graphs)
        self.listwidget.clicked.connect(self.clicked)
        self.Button1.setFont(QtGui.QFont("Serif", 10, QtGui.QFont.Bold))
        self.label.setFont(QtGui.QFont("Serif", 10, QtGui.QFont.Bold))
        self.listwidget.setFont(QtGui.QFont("Serif", 10, QtGui.QFont.Bold))
        self.widget_item = None
        self.setLayout(layout)

    def clicked(self):
        item = self.listwidget.currentItem()
        self.widget_item = item.text()
```

```
    def plot_graphs(self):
        self.axes[0][0].clear()
        self.axes[1][0].clear()
        self.axes[1][1].clear()
        self.axes[0][1].clear()
        plot_correlogram(self.df[self.widget_item],title = self.widget_item,fig = self.fig, axes = self.axes)
        self.canvas.draw()


class Prediction_graphs(QWidget):
    def __init__(self,df,val_df,predict_df):
        super().__init__()
        self.df = df
        self.stationary_df, self.inverting_times = make_stationary(self.df)
        self.val_df = val_df
        self.predict_df = predict_df
        self.methods = ["VARMA","BoDT","LSTM"]
        self.setWindowTitle("Графики")
        self.Button1 = QPushButton("Plot")
        self.label = QLabel('Выберите временной ряд', self)
        self.method = QLabel('Выберите метод', self)
```

```
self.Button1.setFixedSize(200, 50)
self.label.setFixedSize(300,50)
self.fig, (self.ax1,self.ax2) = plt.subplots(nrows=1, ncols=2, figsize=(12, 16))
self.canvas = FigureCanvas(self.fig)
self.listwidget = QListWidget()
self.listwidget.setFixedSize(300,100)
self.methodwidget = QListWidget()
self.methodwidget.setFixedSize(300,100)
for i,col in enumerate(self.df.columns):
    self.listwidget.insertItem(i,col)
for i,method in enumerate(self.methods):
    self.methodwidget.insertItem(i,method)
layout = QGridLayout()
layout.addWidget(self.label, 0, 0)
layout.addWidget(self.method, 0, 1)
layout.addWidget(self.listwidget, 1, 0)
layout.addWidget(self.methodwidget, 1, 1)
layout.addWidget(self.Button1, 2, 0)
layout.addWidget(self.canvas, 3, 0)
layout.setHorizontalSpacing(10)
layout.setVerticalSpacing(10)
self.Button1.clicked.connect(self.plot_graphs)
```

```
        self.listwidget.clicked.connect(self.clicked)
        self.methodwidget.clicked.connect(self.clicked_method)
        self.Button1.setFont(QtGui.QFont("Serif", 9, QtGui.QFont.Bold))
        self.label.setFont(QtGui.QFont("Serif", 9, QtGui.QFont.Bold))
        self.method.setFont(QtGui.QFont("Serif", 9, QtGui.QFont.Bold))
        self.listwidget.setFont(QtGui.QFont("Serif", 9, QtGui.QFont.Bold))
        self.methodwidget.setFont(QtGui.QFont("Serif", 9, QtGui.QFont.Bold))
        self.widget_item = None
        self.methodwidget_item=None
        self.setLayout(layout)

    def clicked(self):
        item = self.listwidget.currentItem()
        self.widget_item = item.text()

    def clicked_method(self):
        item = self.methodwidget.currentItem()
        self.methodwidget_item = item.text()

    def plot_graphs(self):
            if self.predict_df[self.methodwidget_item] is not None and
self.val_df[self.methodwidget_item] is not None:
                self.ax1.clear()
```

```
            self.ax2.clear()

plot_pred_graphs(self.df[self.widget_item],self.predict_df[self.methodwidget_item
][self.widget_item],self.val_df[self.methodwidget_item][self.widget_item],self.met
hodwidget_item,title = self.widget_item,fig = self.fig, ax1 = self.ax1,ax2=self.ax2)
            self.canvas.draw()
        else:
            self.warning_box("Сначала сделайте предсказания!")



    def warning_box(self,text):
        msgBox = QMessageBox()
        msgBox.setIcon(QMessageBox.Information)
        msgBox.setText(text)
        msgBox.setWindowTitle("Внимание!")
        msgBox.setFixedSize(300,150)
        msgBox.setStandardButtons(QMessageBox.Ok)
        msgBox.exec()


class Response():
    def __init__(self):
        self.method = None
        self.predicts_df = None
```

```
        self.val_predicts_df = None
        self.best_ma = None
        self.best_params = None
        self.test_y = None


class PredictionMethods(QtCore.QObject):
    finished = pyqtSignal()
    result = pyqtSignal(Response)
    def
__init__(self,stationary_df,multiseries_df,singleseries_df,step_size,method,invertin
g_times,best_ma,best_params):
        super().__init__()
        self.df = stationary_df
        self.methods = ["VARMA","Boosting on decision trees","LSTM"]
        self.widget_method = method
        self.multiseries_df=multiseries_df
        self.singleseries_df=singleseries_df
        self.step_size = step_size
        self.inverting_times = inverting_times
        self.ma = best_ma
        self.params = best_params
        self.response = Response()
```

```
    def nstep_prediction(self):
        if self.widget_method == 'VARMA':
            multi_predicts_df = varma_prediction(self.multiseries_df,None,steps = self.step_size)
            single_predicts_df = arma_prediction(self.singleseries_df,None,steps = self.step_size)
            self.response.predicts_df = pd.concat([multi_predicts_df, single_predicts_df], axis=1)
            self.response.method = self.widget_method
            self.finished.emit()
            self.result.emit(self.response)
        elif self.widget_method == "BoDT":
            self.response.predicts_df = boosting_prediction(self.df,self.df.columns,self.step_size,self.ma,self.params,self.inverting_times)
            self.response.method = self.widget_method
            self.finished.emit()
            self.result.emit(self.response)
        elif self.widget_method == "LSTM":
            pass


class ValidPredictionMethods(QtCore.QObject):
    finished = pyqtSignal()
```

```
    result = pyqtSignal(Response)
    def
__init__(self,stationary_df,multi_train_df,multi_valid_df,single_train_df,single_va
lid_df,method,inverting_times):
        super().__init__()
        self.df = stationary_df
        self.multi_train_df = multi_train_df
        self.multi_valid_df = multi_valid_df
        self.single_train_df = single_train_df
        self.single_valid_df = single_valid_df
        self.widget_method = method
        self.inverting_times = inverting_times
        self.response = Response()

    def validation_prediction(self):
        if self.widget_method == 'VARMA':
            multi_predicts_df                                                =
varma_prediction(self.multi_train_df,self.multi_valid_df,steps = None)
            single_predicts_df                                               =
arma_prediction(self.single_train_df,self.single_valid_df,steps = None)
            self.response.val_predicts_df          =          pd.concat([multi_predicts_df,
single_predicts_df], axis=1)
            self.response.method = self.widget_method
            self.finished.emit()
```

```
            self.result.emit(self.response)
        elif self.widget_method == "BoDT":
            self.response.val_predicts_df, test_y, best_ma, best_params =
boosting_validation(self.df,self.df.columns,0.75,self.inverting_times)
            self.response.method = self.widget_method
            self.response.test_y = test_y
            self.response.best_ma = best_ma
            self.response.best_params = best_params
            self.finished.emit()
            self.result.emit(self.response)
        elif self.widget_method == "LSTM":
            pass


class Window_predictions(QWidget):
    def __init__(self,df,stationary_df,inverting_times):
        super().__init__()
        self.df = df
        self.stationary_df = stationary_df
        self.inverting_times = inverting_times
        self.multiseries_df,self.singleseries_df = multi_single_ts(self.df)
        self.methods = ["VARMA","BoDT","LSTM"]
        self.setWindowTitle("VarPrediction")
        self.Button1 = QPushButton("Валидировать")
```

```
        self.Button2 = QPushButton("Предсказать на n шагов")
        self.val_textbox = QLineEdit(self)
        self.nstep_textbox = QLineEdit(self)
        self.listwidget = QListWidget()
        self.listwidget.setFixedSize(300,50)
        for i,method in enumerate(self.methods):
            self.listwidget.insertItem(i,method)
        self.val_label = QLabel('Введите размер валидационной выборки', self)
        self.nstep_label = QLabel('Введите кол-во предсказанных шагов', self)
        self.Button3=QPushButton("Результаты тестового прогноза")
        self.Button4=QPushButton("Результаты прогноза на n шагов")
        self.Button5=QPushButton("Сохранить результаты \nтестового прогноза")
        self.Button6=QPushButton("Сохранить результаты тестового \nпрогноза
на n шагов")
        self.Button7=QPushButton("Графики прогнозов")
        self.method_label = QLabel('Выберите метод прогноза', self)
        self.Button1.setFixedSize(300, 50)
        self.Button2.setFixedSize(300, 50)
        self.Button3.setFixedSize(300, 50)
        self.Button4.setFixedSize(300, 50)
        self.Button5.setFixedSize(300, 50)
        self.Button6.setFixedSize(300, 50)
        self.Button7.setFixedSize(300, 70)
```

```
self.val_textbox.setFixedSize(300, 30)
self.nstep_textbox.setFixedSize(300, 30)
self.val_label.setFixedSize(300,50)
self.nstep_label.setFixedSize(300,50)
self.listwidget.setFixedSize(300,70)
layout = QGridLayout()
layout.addWidget(self.method_label, 0, 0)
layout.addWidget(self.listwidget, 1, 0)
layout.addWidget(self.Button1, 4, 0)
layout.addWidget(self.Button2, 4, 1)
layout.addWidget(self.val_label, 2, 0)
layout.addWidget(self.nstep_label, 2, 1)
layout.addWidget(self.val_textbox, 3, 0)
layout.addWidget(self.nstep_textbox, 3, 1)
layout.addWidget(self.Button3, 5, 0)
layout.addWidget(self.Button4, 5, 1)
layout.addWidget(self.Button5, 6, 0)
layout.addWidget(self.Button6, 6, 1)
layout.addWidget(self.Button7, 1, 1)
layout.setHorizontalSpacing(5)
layout.setVerticalSpacing(5)
self.Button1.clicked.connect(self.validation_prediction)
```

```
self.Button2.clicked.connect(self.nstep_prediction)
self.Button3.clicked.connect(self.table_validation)
self.Button4.clicked.connect(self.table_nstep)
self.Button5.clicked.connect(self.save_valid)
self.Button6.clicked.connect(self.save_prediction)
self.Button7.clicked.connect(self.plot_prediction)
self.listwidget.clicked.connect(self.clicked_method)
self.Button1.setFont(QtGui.QFont("Serif", 8, QtGui.QFont.Bold))
self.Button2.setFont(QtGui.QFont("Serif", 8, QtGui.QFont.Bold))
self.Button3.setFont(QtGui.QFont("Serif", 8, QtGui.QFont.Bold))
self.Button4.setFont(QtGui.QFont("Serif", 8, QtGui.QFont.Bold))
self.Button5.setFont(QtGui.QFont("Serif", 8, QtGui.QFont.Bold))
self.Button6.setFont(QtGui.QFont("Serif", 8, QtGui.QFont.Bold))
self.Button7.setFont(QtGui.QFont("Serif", 8, QtGui.QFont.Bold))
self.listwidget.setFont(QtGui.QFont("Serif", 8, QtGui.QFont.Bold))
self.val_label.setFont(QtGui.QFont("Serif", 8, QtGui.QFont.Bold))
self.nstep_label.setFont(QtGui.QFont("Serif", 8, QtGui.QFont.Bold))
self.method_label.setFont(QtGui.QFont("Serif", 8, QtGui.QFont.Bold))
self.val_predicts_df = dict.fromkeys(self.methods, None)
self.predicts_df = dict.fromkeys(self.methods, None)
self.val_metrics_df = dict.fromkeys(self.methods, None)
self.view1 = QTableView()
```

```
        self.view2 = QTableView()
        self.view3 = QTableView()
        self.view4 = QTableView()
        self.train_df = dict.fromkeys(self.methods, None)
        self.valid_df = dict.fromkeys(self.methods, None)
        self.widget_item = None
        self.BoDT_best_ma = None
        self.LSTM_best_ma = None
        self.BoDT_params = None
        self.w_graphs =
Prediction_graphs(self.df,self.val_predicts_df,self.predicts_df)
        self.setLayout(layout)

    def plot_prediction(self):
        self.w_graphs.show()

    def validation_prediction(self):
        try:
            val_size = 1 - float(self.val_textbox.text())
        except:
            self.warning_box("Введите число от 0.05 до 0.5")
        try:
            if (val_size>=0.5 and val_size<=0.95):
```

```
            self.train_df['VARMA'], self.valid_df['VARMA'] = train_valid_split(self.stationary_df,val_size)
            multi_train_df, multi_valid_df = self.train_df['VARMA'][self.multiseries_df.columns],self.valid_df['VARMA'][self.multiseries_df.columns]
            single_train_df, single_valid_df = self.train_df['VARMA'][self.singleseries_df.columns],self.valid_df['VARMA'][self.singleseries_df.columns]
            if self.widget_item == 'VARMA':
                self.thread = QtCore.QThread()
                self.ValidPred = ValidPredictionMethods(self.stationary_df,multi_train_df,multi_valid_df,single_train_df,single_valid_df,self.widget_item,self.inverting_times)
                self.ValidPred.moveToThread(self.thread)
                self.thread.started.connect(self.ValidPred.validation_prediction)
                self.ValidPred.finished.connect(self.thread.quit)
                self.ValidPred.finished.connect(self.ValidPred.deleteLater)
                self.thread.finished.connect(self.thread.deleteLater)
                self.thread.start()
                self.ValidPred.result.connect(self.get_valid_method_output)
                self.Button1.setEnabled(False)
                self.Button2.setEnabled(False)
                self.thread.finished.connect(lambda: self.enableButton())
            elif self.widget_item == 'BoDT':
```

```
                self.thread = QtCore.QThread()
                self.ValidPred = ValidPredictionMethods(self.stationary_df,multi_train_df,multi_valid_df,single_train_df,single_valid_df,self.widget_item,self.inverting_times)
                self.ValidPred.moveToThread(self.thread)
                self.thread.started.connect(self.ValidPred.validation_prediction)
                self.ValidPred.finished.connect(self.thread.quit)
                self.ValidPred.finished.connect(self.ValidPred.deleteLater)
                self.thread.finished.connect(self.thread.deleteLater)
                self.thread.start()
                self.ValidPred.result.connect(self.get_valid_method_output)
                self.Button1.setEnabled(False)
                self.Button2.setEnabled(False)
                self.thread.finished.connect(lambda: self.enableButton())
            elif self.widget_item == 'LSTM':
                pass
        else:
            self.warning_box("Введите число от 0.05 до 0.5")
    except Exception as e:
        self.warning_box(f"Ошибка {e}")

def nstep_prediction(self):
    try:
```

```
        try:
            step_size = int(self.nstep_textbox.text())
        except:
            self.warning_box("Введите число от 1 до 20")
        if (step_size>0 and step_size<=20):
            if self.widget_item == 'VARMA':
                self.thread = QtCore.QThread()
                self.Pred =
PredictionMethods(self.stationary_df,self.multiseries_df,self.singleseries_df,step_s
ize,self.widget_item,self.inverting_times,self.BoDT_best_ma,self.BoDT_params)
                self.Pred.moveToThread(self.thread)
                self.thread.started.connect(self.Pred.nstep_prediction)
                self.Pred.finished.connect(self.thread.quit)
                self.Pred.finished.connect(self.Pred.deleteLater)
                self.thread.finished.connect(self.thread.deleteLater)
                self.thread.start()
                self.Pred.result.connect(self.get_predict_method_output)
                self.Button1.setEnabled(False)
                self.Button2.setEnabled(False)
                self.thread.finished.connect(lambda: self.enableButton())
            elif self.widget_item == 'BoDT':
                if self.BoDT_best_ma is None and self.BoDT_params is None:
                    self.warning_box("Сперва сделайте валидацию!")
```

```
                else:
                    self.thread = QtCore.QThread()
                    self.Pred = PredictionMethods(self.stationary_df,self.multiseries_df,self.singleseries_df,step_size,self.widget_item,self.inverting_times,self.BoDT_best_ma,self.BoDT_params)
                    self.Pred.moveToThread(self.thread)
                    self.thread.started.connect(self.Pred.nstep_prediction)
                    self.Pred.finished.connect(self.thread.quit)
                    self.Pred.finished.connect(self.Pred.deleteLater)
                    self.thread.finished.connect(self.thread.deleteLater)
                    self.thread.start()
                    self.Pred.result.connect(self.get_predict_method_output)
                    self.Button1.setEnabled(False)
                    self.Button2.setEnabled(False)
                    self.thread.finished.connect(lambda: self.enableButton())
            elif self.widget_item == 'LSTM':
                pass
            else:
                self.warning_box("Введите число от 1 до 20")
        except Exception as e:
            print(e)

    def enableButton(self,mode = True):
```

```
        self.Button1.setEnabled(mode)
        self.Button2.setEnabled(mode)

    def get_valid_method_output(self,s):
        method = s.method
        self.BoDT_best_ma = s.best_ma
        self.BoDT_params = s.best_params
        self.val_predicts_df[method] = s.val_predicts_df
        if method=='BoDT' or method=='LSTM':
            self.valid_df[method] = s.test_y
        self.val_metrics_df[method] =
metrics(self.valid_df[method],self.val_predicts_df[method])
        self.val_predicts_df[method] =
invert_diff(self.val_predicts_df[method],self.inverting_times)

    def get_predict_method_output(self,s):
        method = s.method
        self.predicts_df[method] = s.predicts_df
        self.predicts_df[method] =
invert_diff(self.predicts_df[method],self.inverting_times)

    def save_valid(self):
        for key in self.val_predicts_df.keys():
```

```
            if self.val_predicts_df[key] is not None:

self.val_predicts_df[key].to_csv(f'output/val_{key}_predicts.csv',index=False)
        for key in self.val_metrics_df.keys():
            if self.val_metrics_df[key] is not None:

self.val_metrics_df[key].to_csv(f'output/{key}_metrics.csv',index=False)


    def save_prediction(self):
        for key in self.predicts_df.keys():
            if self.predicts_df[key] is not None:
                self.predicts_df[key].to_csv(f'output/{key}_predicts.csv',index=False)


    def plot_valid(self):
        pass


    def plot_predictions(self):
        pass


    def clicked_method(self):
        item = self.listwidget.currentItem()
        self.widget_item = item.text()
```

```
    def table_validation(self):
        if self.val_metrics_df[self.widget_item] is not None and self.val_predicts_df[self.widget_item] is not None :
            self.show_tables(self.view1,self.val_predicts_df[self.widget_item])
            self.show_tables(self.view2,self.val_metrics_df[self.widget_item])
        else:
            self.warning_box("Сперва нажмите на кнопку\n'Валидировать'")

    def table_nstep(self):
        if self.predicts_df[self.widget_item] is not None:
            self.show_tables(self.view3,self.predicts_df[self.widget_item])
        else:
            self.warning_box("Сперва нажмите на кнопку\n'Предсказать на n шагов'")

    def warning_box(self,text):
        msgBox = QMessageBox()
        msgBox.setIcon(QMessageBox.Information)
        msgBox.setText(text)
        msgBox.setWindowTitle("Внимание!")
        msgBox.setFixedSize(300,150)
        msgBox.setStandardButtons(QMessageBox.Ok)
        msgBox.exec()
```

```
    def show_tables(self,view,df):
        df = pandasModel(df)
        view.setModel(df)
        view.move(600, 500)
        view.resize(800,600)
        view.resizeColumnsToContents()
        view.show()

class pandasModel(QAbstractTableModel):
    def __init__(self, data):
        QAbstractTableModel.__init__(self)
        self._data = data

    def rowCount(self, parent=None):
        return self._data.shape[0]

    def columnCount(self, parnet=None):
        return self._data.shape[1]

    def data(self, index, role=Qt.DisplayRole):
        if index.isValid():
```

```
            if role == Qt.DisplayRole:
                tmp = self._data.iloc[index.row(), index.column()]
                if isinstance(tmp, float):
                    return "%.2f" % tmp
                return str(tmp)
        return None

    def headerData(self, col, orientation, role):
        if orientation == Qt.Horizontal and role == Qt.DisplayRole:
            return self._data.columns[col]
        elif orientation == Qt.Vertical and role == Qt.DisplayRole:
            return self._data.index[col]
        return None
```

Файл main.py

```
import sys
from PyQt5.QtWidgets import QApplication
from windows import MainWindow
import argparse
import sys
import os
import warnings
```

```
warnings.simplefilter('ignore')


def create_arg_parser():
    parser = argparse.ArgumentParser(description='####')
    parser.add_argument('inputFile',
                        help='Path to the input file with time series')
    return parser


if __name__ == '__main__':
    try:
        arg_parser = create_arg_parser()
        parsed_args = arg_parser.parse_args(sys.argv[1:])
        if os.path.exists(parsed_args.inputFile):
            filename = parsed_args.inputFile
        else:
            print("Файл несуществует")
        app = QApplication(sys.argv)
        window = MainWindow(filename)
        window.show()
        sys.exit(app.exec_())
    except:
        print("Правильный ввод: <py main.py dataset.csv>")
```

ДОДАТОК Б ІЛЮСТРАТИВНИЙ МАТЕРІАЛ

Система для аналізу багатовимірних часових рядів

ВИКОНАВ: СТУДЕНТ ГРУПИ КА-77 БУХАНЕВИЧ РОДІОН
НАУКОВИЙ КЕРІВНИК: Д.Т.Н. ПРОФ. БІДЮК ПЕТРО ІВАНОВИЧ

Актуальність дослідження

Приведемо декілька прикладів задач, які потребують застосування прогнозу:

- рух цін акцій на фондових ринках;
- попит товарів;
- врожайність;
- кількість несправностей обладнання;

Дослідження цих задач може включати аналіз десятків та сотень показників. Для цього потрібні методи аналізу багатовимірних часових рядів.

Мета, предмет, об’єкт дослідження

Об’єкт дослідження:	Коінтегровані процеси, представленні багатовимірними рядами даних.
Предмет дослідження:	Алгоритми VAR, методи машинного навчання, такі як ансамблі дерев рішень та рекурентні нейронні мережі.
Мета дослідження:	Аналіз сильних та слабких сторін методів, порівняльний аналіз результатів прогнозу.

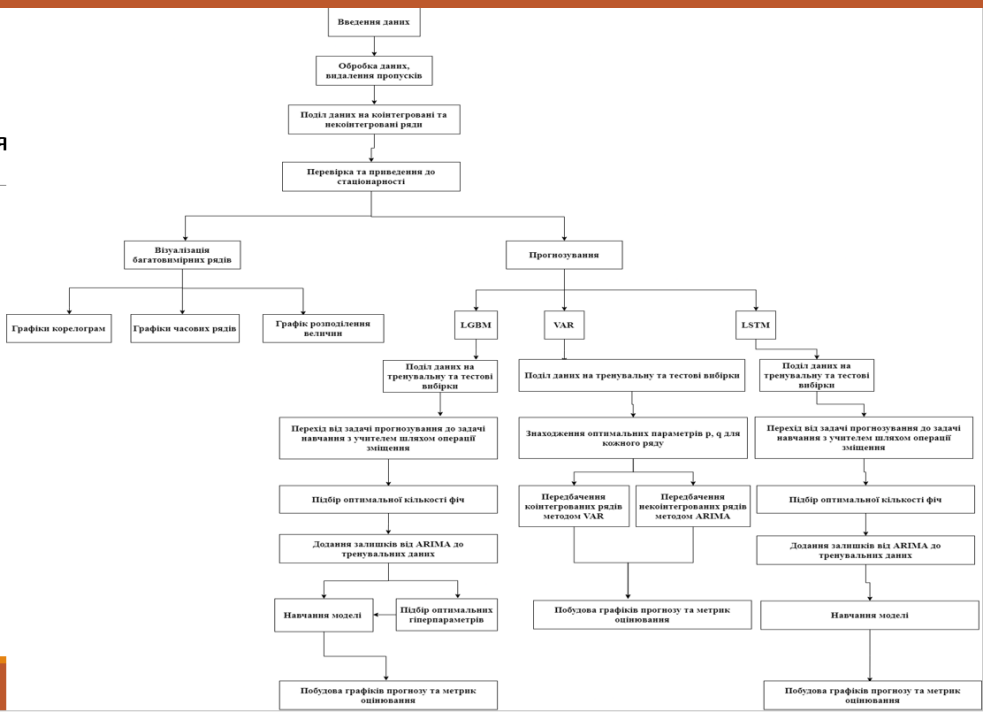
Постановка задачі

- Провести аналіз існуючих методів та підходів до вирішення задачі прогнозування багатовимірних процесів, що представлені у вигляді багатовимірних часових рядів.
- Обрати статистичні дані, що описують динаміку коінтегрованних процесів різної природи для оцінювання короткострокових прогнозів.
- Обрати декілька моделей різних класів для оцінювання прогнозів.
- Розробити програмний продукт для побудови досліджуваних моделей та прогнозів.
- Проаналізувати та порівняти результати, отримані за допомогою різних методів.
- Визначення можливих напрямів подальших досліджень.

Досліджувані моделі

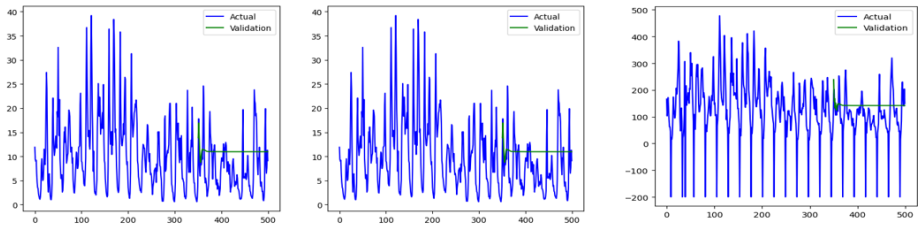
Модель ВАР	Ансамбль дерев рішень	Мережа LSTM
Узагальнена модель VAR(p, q) має наступний вигляд: $X_t = c + \Phi_1 X_{t-1} + \dots + \Phi_p X_{t-p} + \varepsilon_t - \Theta_1 \varepsilon_{t-1} - \dots - \Theta_q \varepsilon_{t-q}$ де ε_t - n-вимірний векторний процес, який є білим шумом Φ_i – матриця коефіцієнтів n x n. Θ_i – матриця коефіцієнтів n x n.	Ансамблювання моделей передбачає поєднання декількох моделей машинного навчання в єдину нова модель, яка повинна краще робити передбачення, ніж будь-яка самостійна модель дерева рішень.	Основною перевагою моделі рекурентної нейронної мережі є те, що кожен вихід моделі є функцією, яка залежить як від попереднього виходу так і від поточного входу. Рекурентна нейронна мережа є узагальненням нейронної мережі з внутрішньою пам'яттю. За допомогою цих внутрішніх станів мережа може обробляти послідовності даних.

Розроблена система для оцінювання прогнозів



Оцінювання прогнозів методом VARMA

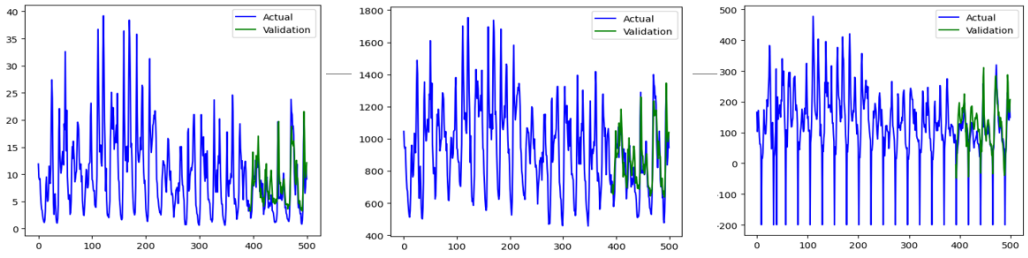
В якості вхідних даних був обраний датасет, який містить кількісний зміст різних хімічних часток в повітрі, таких як: PT08, NO, C6H6 та інших.



C6H6 PT08 NOx

	MAE	MSE	R^2
C6H6	2.54	10.54	0.58
PT08	97.90	14793.79	0.66
NOx	42.82	3304.65	0.56

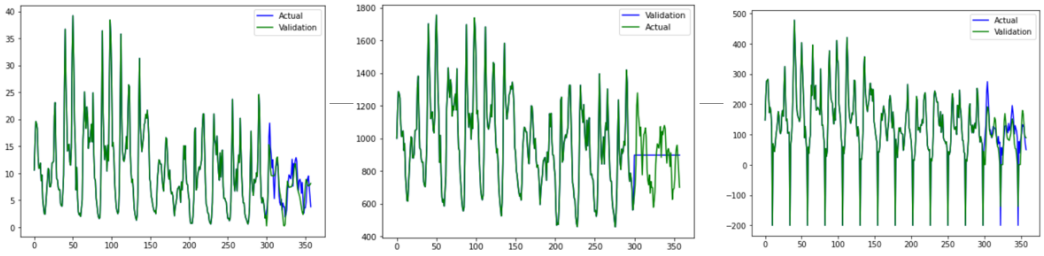
Оцінювання прогнозів методом LightGBM



C6H6 PT08 NOx

	MAE	MSE	R^2
C6H6	2.54	10.54	0.58
PT08	97.90	14793.79	0.66
NOx	42.82	3304.65	0.56

Оцінювання прогнозів з допомогою мережі LSTM



	MAE	MSE	R^2
C6H6	2.13	6.47	0.56
PT08	166.64	41007.32	-0.62
NOx	36.83	2251.15	0.56

Порівняємо час тренування моделей на трьох коінтегрованих часових рядах, кожен з яких має 500 спостережень.

	VAR	LGBM	LSTM
Час в секундах	230	55	360

Висновки

- Розроблено оригінальний програмний продукт для побудови прогнозів динаміки розвитку коінтегрованих процесів, представлених за допомогою багатовимірних часових рядів
- Виконано порівняльний аналіз прогнозів, отриманих за допомогою моделей VARMA, ансамблей дерев рішень, мережі LSTM
- Модель VARMA погано апроксимує багатовимірні процеси, що зумовлені наявністю нелінійних залежностей та високим впливом невимірюваних факторів.
- Модель ансамблю дерев рішень добре апроксимує процеси, якість прогнозу залежить від навчальної вибірки. Проте тренування моделі та підбір оптимальних гіперпараметрів потребують небагато часу. Модель є стійкою до викидів в даних.
- Модель LSTM не поступається в ефективності моделі ансамблю дерев рішень. Проте є випадки, коли модель не навчається через наявність викидів в даних. До того ж мережа потребує багато часу та потужностей для обчислень.

Напрями подальших досліджень

- Модифікація моделі
- Використання інших методів для побудови прогнозів, наприклад інших видів нейронних мереж, GRU, LSTM згорткові мережі.
- Поліпшення інтерфейсу, додання зручних функцій для користування.

Список наукових досягнень

Буханевич Р.М. Система для аналізу багатовимірних часових рядів. Кібернетика та Системний Аналіз. 2021. № 1. С. 26-54.

Дякую за увагу!

ДОДАТОК В СПИСОК НАУКОВИХ ДОСЯГНЕНЬ

1. Буханевич Р.М. Система для аналізу багатовимірних часових рядів. *Кібернетика та Системний Аналіз*. 2021. № 1. С. 26-54.