ЗМІСТ

АНОТАЦІЯ

Магістерська робота «Розробка програмного забезпечення для аналізу та прогнозування фондового та криптовалютного ринку на мові Python» Кос Андрій Миколайович, Тернопільський національний технічний університет імені І. Пулюя, факультет комп'ютерно–інформаційних систем і програмної інженерії, кафедра програмної інженерії, група СПм–61, Тернопіль, 2021.

Пояснювальна записка містить: 60 с. 7 рис., 1 табл., презентація.

Метою роботи є розробка програмної системи для прогнозування сегментів ринку на стратегічному періоді з використанням нейронних мереж для аналізу і прогнозування фондового та криптовалютного ринку.

Ключові слова: ФРЕЙМВОРК, PYTHON, ДАТАСЕТ, НЕЙРОННА МЕРЕЖА

SUMMARY

Master's thesis "Development of software for analysis and forecasting of stock and cryptocurrency market in Python" Kos Andriy Mykolayovych, Ternopil National Technical University named after I. Pulyuy, Faculty of Computer Information Systems and Software Engineering, Department of Software Engineering, SPm-61 Group , Ternopil, 2021.

The explanatory note contains: 60 p. Fig. 7, 1 table, presentation.

The aim of the work is to develop a software system for forecasting market segments for the strategic period using neural networks for analysis and forecasting of the stock and cryptocurrency market.

Keywords: FRAMEWORK, PYTHON, DATASET, NEURAL NETWORK

ВСТУП

В сучасному світі, де панують тенденції на діджиталізацію та автоматизацію усіх процесів, явище нейронних мереж, які замінюють людину в деяких аспектах при прийнятті тих чи інших рішень — не новина.

Кожного дня створюються сотні нових нейронних мереж, кожна з яких виконує свою задачу, розпізнавання мови і жестів, розпізнавання образів, класифікація предметів, тощо.

Подібне явище не оминуло і одну з найважливіших галузей людського життя, економіку. Вже достатньо довгий час в статистиці і економіці використовують різного роду додатки з використання нейронних мереж і штучного інтелекту, які допомагають спеціалістам виконувати ті чи інші дії, виключаючи так званий «людський фактор», тобто фактори, які можуть спричини помилку, притаманні виключно людині, такі як стрес, втома, різного роду недуги, емоційний стан, які впливають на концентрацію, увагу та можливість адекватно оцінювати ситуацію і приймати рішення.

Виходячи з описаної вище актуальності явища нейронних мереж в економіці і статистиці, об’єктом дослідження було обрано використання нейронних мереж в сфері економіки і статистики.

В свою чергу, предметом дослідження стала розробка та дослідження програмної системи для прогнозування сегментів ринку на стратегічному періоді з використанням нейронних мереж.

Мета роботи полягає в розробленні нейронної мережі для аналізу і прогнозування фондового та криптовалютного ринку.

Методи дослідження — аналіз літературних та інших джерел, комп’ютерне моделювання та проектування.

Для досягнення поставленої мети слід виконати наступні завдання:

1) Провести аналіз предметної області
    a. Проаналізувати поняття нейронної мережі

b. Проаналізувати процес прогнозуванні даних з використанням нейронних мереж
c. Проаналізувати проблематику прогнозування даних з використанням нейронних мереж
d. Розглянути існуючі рішення

2) Провести аналіз методів вирішення задачі
   a. Розглянути парадигми навчання нейронних мереж
   b. Розглянути алгоритми роботи нейронних мереж
   c. Обрати алгоритм аналізу даних
3) Провести огляд інформаційного забезпечення
   a. Аналіз мови програмування
   b. Аналіз середовища розробки
   c. Аналіз додаткового інструментарію
4) Реалізувати програмний засіб
   a. Створити діаграму варіантів використання
   b. Створити діаграму діяльності
   c. Створити діаграму IDEF0
   d. Створити діаграму компонентів
   e. Розробити основні механізми
   f. Описати створення навчальних датасетів
   g. Провести тестування

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Технічний аналіз

У фінансах технічний аналіз — це методологія аналізу для прогнозування напрямку цін шляхом вивчення минулих ринкових даних, насамперед ціни та обсягу [1]. Поведінкова економіка та кількісний аналіз використовують багато з тих самих інструментів технічного аналізу, який, будучи аспектом активного управління, суперечить більшій частині сучасної теорії портфеля. Ефективність як технічного, так і фундаментального аналізу заперечується гіпотезою ефективного ринку, яка стверджує, що ціни на фондовому ринку є по суті непередбачуваними, а дослідження щодо того, чи дає технічний аналіз будь-які переваги, дало неоднозначні результати [5].

Принципи технічного аналізу випливають із сотень років даних фінансового ринку.Деякі аспекти технічного аналізу почали з’являтися в звітах амстердамського торговця Джозефа де ла Веги про фінансові ринки Нідерландів у 17 столітті. В Азії технічний аналіз вважається методом, розробленим Хоммою Мунехісою на початку 18 століття, який еволюціонував у використання техніки свічників і сьогодні є інструментом технічного аналізу для створення діаграм [10].

Журналіст Чарльз Доу опублікував деякі зі своїх висновків у редакційних статтях для The Wall Street Journal. Він вважав, що в цих даних можна знайти закономірності та бізнес-цикли, концепцію, згодом відому як «теорія Доу». Однак сам Доу ніколи не виступав за використання своїх ідей як стратегії біржової торгівлі.

У 1920-х і 1930-х роках Річард В. Шабакер опублікував кілька книг, які продовжили роботу Чарльза Доу та Вільяма Пітера Гамільтона в їхніх книгах

«Теорія і практика фондового ринку» та «Технічний аналіз ринку». У 1948 році Роберт Д. Едвардс і Джон Магі опублікували «Технічний аналіз фондових тенденцій», який широко вважається однією з основоположних робіт цієї дисципліни. Він займається виключно аналізом тенденцій та моделями діаграм і залишається у використанні дотепер. Ранній технічний аналіз був майже виключно аналізом діаграм, оскільки обчислювальна потужність комп'ютерів була недоступна для сучасного рівня статистичного аналізу. Як повідомляється, Чарльз Доу створив форму аналізу точкових і фігурних діаграм. З появою поведінкових фінансів як окремої дисципліни в економіці Пол В. Аззопарді об'єднав технічний аналіз з поведінковими фінансами і ввів термін «поведінковий технічний аналіз».

Інші піонери методів аналізу включають Ральфа Нельсона Елліота, Вільяма Делберта Ганна та Річарда Вайкоффа, які розробили свої відповідні методики на початку 20 століття. За останні десятиліття було розроблено та вдосконалено більше технічних засобів і теорій, при цьому все більше уваги приділяється комп'ютерним методам із використанням спеціально розробленого комп'ютерного програмного забезпечення.

Фундаментальні аналітики вивчають прибутки, дивіденди, активи, якість, співвідношення, нові продукти, дослідження тощо. Техніки також використовують багато методів, інструментів і прийомів, одним з яких є використання діаграм. Використовуючи діаграми, технічні аналітики намагаються визначити цінові моделі та ринкові тенденції на фінансових ринках і намагаються використати ці моделі [13].

Техніки, які використовують діаграми, шукають архетипні моделі цінових графіків, такі як добре відомі моделі «голова і плечі» або моделі подвійного розвороту зверху/внизу, вивчають технічні індикатори, ковзні середні та шукають форми, такі як лінії підтримки, опору, канали та більш незрозумілі утворення, такі як прапори, вимпели, дні балансу та візерунки чашок і ручок [15].

Технічні аналітики також широко використовують різного роду ринкові індикатори, деякі з яких є математичними перетвореннями ціни, часто

включаючи збільшення і зниження обсягів, дані про зростання/зниження та інші вхідні дані. Ці показники використовуються, щоб допомогти оцінити, чи є актив тенденційним, і, якщо є, то ймовірність його напряму та продовження. Технічні спеціалісти також шукають зв'язки між індексами ціни/обсягу та ринковими показниками. Приклади включають ковзну середню, індекс відносної сили та MACD. Інші напрямки дослідження включають кореляції між змінами в опціонах (неявна волатильність) і співвідношеннями пут/колл з ціною. Також важливими є такі показники настроїв, як співвідношення пут/колл, співвідношення бика/ведмедя, короткі відсотки, неявна волатильність тощо.

У технічному аналізі існує багато прийомів. Прихильники різних технік (наприклад: свічковий аналіз, найстаріша форма технічного аналізу, розроблена японським зерновим трейдером; гармоніка; теорія Доу; і хвильова теорія Елліота) можуть ігнорувати інші підходи, але багато трейдерів поєднують елементи більш ніж однієї техніки. . Деякі технічні аналітики використовують суб'єктивне судження, щоб вирішити, який(і) шаблон(и) певний інструмент відображає в даний момент і якою має бути інтерпретація цієї моделі. Інші використовують суто механічний або систематичний підхід до ідентифікації та інтерпретації шаблонів.

На відміну від технічного аналізу фундаментальний аналіз, вивчення економічних факторів, які впливають на те, як інвестори оцінюють фінансові ринки. Технічний аналіз стверджує, що ціни вже відображають усі основні фундаментальні фактори. Технічні індикатори призначені для виявлення тенденцій, хоча ні технічні, ні фундаментальні показники не є ідеальними. Деякі трейдери використовують виключно технічний або фундаментальний аналіз, а інші використовують обидва типи для прийняття торгових рішень [16].

Технічний аналіз використовує моделі та правила торгівлі на основі перетворень ціни та обсягу, таких як індекс відносної сили, ковзні середні, регресії.

Технічний аналіз відрізняється від підходу фундаментального аналізу до аналізу безпеки та акцій. У фундаментальному рівнянні M = P/E технічний аналіз

є дослідженням М (множинного). Множина охоплюють психологію, яка зазвичай багата, тобто ступінь бажання купувати/продавати. Також у М є здатність платити, оскільки, наприклад, витрачений бик не може змусити ринок піднятися вище, а ведмідь з хорошим підбором — ні. Технічний аналіз аналізує ціну, обсяг, психологію, грошові потоки та іншу ринкову інформацію, тоді як фундаментальний аналіз розглядає факти компанії, ринку, валюти або товару. Більшість великих брокерських компаній, торгових груп або фінансових установ, як правило, мають команду технічного та фундаментального аналізу.

У 1960-х і 1970-х роках вона була широко відкинута вченими. У огляді 2007 року Ірвін і Парк повідомили, що 56 із 95 сучасних досліджень показали, що він дає позитивні результати, але зазначили, що багато позитивних результатів викликали сумніви через такі проблеми, як підгляд даних, тому докази на підтримку технічних аналіз був непереконливим; багато вчених досі вважають його невідрізним від псевдонауки.Вчені, такі як Юджин Фама, кажуть, що докази технічного аналізу мізерні та несумісні зі слабкою формою гіпотези ефективного ринку.Користувачі вважають, що навіть якщо технічний аналіз не може передбачити майбутнє, він допомагає визначити тенденції, тенденції та торгові можливості [20].

Хоча деякі поодинокі дослідження вказують на те, що технічні правила торгівлі можуть призвести до постійної прибутковості в період до 1987 року, [21][7][22][23] більшість наукових робіт зосереджена на природі аномального положення іноземної валюти. ринок.Є припущення, що ця аномалія пов’язана з втручанням центрального банку, яке, очевидно, не призначено для передбачення технічного аналізу [24].

Основний принцип технічного аналізу полягає в тому, що ціна ринку відображає всю відповідну інформацію, яка впливає на цей ринок. Тому технічний аналітик розглядає історію торгової моделі цінних паперів або товарів, а не зовнішні чинники, такі як економічні, фундаментальні та новинні події. Вважається, що цінова дія має тенденцію повторюватися через колективну,

шаблонну поведінку інвесторів. Тому технічний аналіз зосереджується на визначених цінових тенденціях і умовах [26].

Виходячи з того, що вся релевантна інформація вже відображається в цінах, технічні аналітики вважають, що важливо розуміти, що інвестори думають про цю інформацію, відомої та сприйнятої.

Технічні аналітики вважають, що ціни мають напрямок тенденції, тобто вгору, вниз, убік (плоскій) або в певну комбінацію. Основне визначення цінової тенденції спочатку було запропоновано теорією Доу [13].

Прикладом цінного паперу, який мав очевидну тенденцію, є AOL з листопада 2001 року по серпень 2002 року. Технічний аналітик або послідовник тенденції, який усвідомлює цю тенденцію, шукатиме можливості продати цей цінний папір. AOL постійно знижується в ціні. Щоразу, коли акція зростала, продавці входили на ринок і продавали акції; звідси «зигзагоподібний» рух ціни. Серія «нижчих максимумів» і «нижчих мінімумів» є ознакою падіння акцій.Іншими словами, щоразу, коли акція рухалася нижче, вона опускалася нижче своєї попередньої відносно низької ціни. Щоразу, коли акція піднімалася вище, вона не могла досягти рівня попередньої відносно високої ціни.

Зауважте, що послідовність низьких мінімумів і нижчих максимумів почалася не раніше серпня. Потім AOL робить низьку ціну, яка не перетинає відносно низьку ціну, встановлену раніше в цьому місяці. Пізніше в тому ж місяці акція досягає відносного максимуму, рівним останньому відносному максимуму. У цьому технік бачить серйозні ознаки того, що спадний тренд принаймні призупиняється і, можливо, закінчується, і, ймовірно, припинить активний продаж акцій на цьому етапі.

Для техніка емоції на ринку можуть бути ірраціональними, але вони існують. Оскільки поведінка інвестора повторюється дуже часто, фахівці вважають, що на графіку з’являться впізнавані (і передбачувані) моделі цін.Розпізнавання цих закономірностей може дозволити техніку вибирати операції, які мають більшу ймовірність успіху [29].

Технічний аналіз не обмежується створенням графіків, але він завжди враховує цінові тенденції.Наприклад, багато фахівців стежать за дослідженнями настроїв інвесторів. Ці опитування оцінюють ставлення учасників ринку, зокрема, ведмежі вони чи бичачі. Технічні спеціалісти використовують ці опитування, щоб визначити, чи триватиме тенденція, чи може розвинутися розворот; вони, швидше за все, передбачають зміни, коли опитування свідчать про екстремальні настрої інвесторів.Наприклад, опитування, які демонструють переважну бичачу позицію, є доказом того, що висхідний тренд може змінитися; передумова полягає в тому, що якщо більшість інвесторів налаштовані на бичачих, вони вже купили ринок (очікуючи підвищення цін). І оскільки більшість інвесторів налаштовані на бичачих і інвестують, можна припустити, що покупців залишилося небагато. Це залишає більше потенційних продавців, ніж покупців, незважаючи на бичачі настрої. Це свідчить про те, що ціни матимуть тенденцію до зниження, і є прикладом протилежної торгівлі [31].

промисловість

Галузь у всьому світі представлена Міжнародною федерацією технічних аналітиків (IFTA), яка є федерацією регіональних і національних організацій. У Великобританії галузь представлена Товариством технічних аналітиків (STA). STA була членом-засновником IFTA, нещодавно відсвяткувала своє 50-річчя і сертифікувала аналітиків дипломом з технічного аналізу. У Канаді галузь представлена Канадським товариством технічних аналітиків.В Австралії галузь представлена Австралійською асоціацією технічних аналітиків (ATAA)[33] (яка є членом IFTA) та Australian Professional Technical Analyts (APTA) Inc [34].

Товариства професійного технічного аналізу працювали над створенням сукупності знань, що описують область технічного аналізу. Сукупність знань є центральною в цій галузі як спосіб визначення того, як і чому може працювати технічний аналіз. Потім його можуть використовувати наукові кола, а також регулюючі органи для розробки відповідних досліджень і стандартів для цієї галузі.

Програмне забезпечення

Програмне забезпечення технічного аналізу автоматизує функції складання діаграм, аналізу та звітності, які допомагають технічним аналітикам переглядати та прогнозувати фінансові ринки (наприклад, фондовий ринок).

На додаток до встановлюваних програмних пакетів на базі настільних комп'ютерів у традиційному розумінні, в галузі з'явилися хмарні програми та інтерфейси програмного забезпечення (API), які забезпечують технічні показники (наприклад, MACD, Bollinger Bands) через протоколи RESTful HTTP або інтранет. .

Сучасне програмне забезпечення технічного аналізу часто доступне у вигляді веб-додатків або додатків для смартфонів, без необхідності завантажувати та встановлювати пакет програм.

Систематична торгівля

Нейронні мережі

З початку 1990-х років, коли з'явилися перші практично придатні для використання типи, популярність штучних нейронних мереж (ШНМ) швидко зросла. Це адаптивні програмні системи зі штучним інтелектом, які були натхненні тим, як працюють біологічні нейронні мережі. Вони використовуються, тому що вони можуть навчитися виявляти складні закономірності в даних. З точки зору математики, вони є універсальними апроксиматорами функцій, що означає, що, за умови правильних даних і правильно налаштованих, вони можуть фіксувати та моделювати будь-які відносини введення-виведення. Це не тільки усуває потребу в людській інтерпретації діаграм або ряду правил для генерування сигналів входу/виходу, але також забезпечує міст до фундаментального аналізу, оскільки змінні, що використовуються у фундаментальному аналізі, можуть використовуватися як вхідні дані.

Оскільки ШНМ по суті є нелінійними статистичними моделями, їх точність і можливості прогнозування можна перевірити як математично, так і емпірично. У різних дослідженнях автори стверджували, що нейронні мережі, які використовуються для генерації торгових сигналів із різними технічними та фундаментальними вхідними параметрами, значно перевершують стратегії

купівлі-утримання, а також традиційні методи лінійного технічного аналізу в поєднанні з експертними системами, заснованими на правилах [39].

У той час як просунута математична природа таких адаптивних систем тримала нейронні мережі для фінансового аналізу переважно в академічних дослідницьких колах, останніми роками більш зручне програмне забезпечення для нейронних мереж зробило цю технологію більш доступною для трейдерів.

Backtesting

Систематична торгівля найчастіше використовується після тестування інвестиційної стратегії на історичних даних. Це відомо як бектестування. Тестування назад найчастіше виконується для технічних індикаторів, але його можна застосувати до більшості інвестиційних стратегій (наприклад, фундаментальний аналіз). У той час як традиційне тестування проводилося вручну, воно зазвичай проводилося лише на відібраних людиною акціях, і тому було схильне до попередніх знань щодо відбору запасів. З появою комп’ютерів тестування можна виконувати на цілі біржі протягом десятиліть історичних даних за дуже короткий проміжок часу.

Використання комп’ютерів має свої недоліки, оскільки воно обмежується алгоритмами, які може виконувати комп’ютер. Деякі торгові стратегії покладаються на людську інтерпретацію і непридатні для комп'ютерної обробки.Лише технічні показники, які є повністю алгоритмічними, можна запрограмувати для комп’ютеризованого автоматизованого тестування.

Поєднання з іншими методами прогнозування ринку

Джон Мерфі стверджує, що основними джерелами інформації, доступною для техніків, є ціна, обсяг і відкритий інтерес.Інші дані, такі як показники та аналіз настроїв, вважаються другорядними.

Однак багато технічних аналітиків виходять за межі чистого технічного аналізу, поєднуючи інші методи прогнозування ринку зі своєю технічною роботою. Інший такий підхід, аналіз злиття, накладає фундаментальний аналіз на технічний, намагаючись підвищити продуктивність портфельного менеджера.

Технічний аналіз також часто поєднується з кількісним аналізом та економікою. Наприклад, нейронні мережі можуть використовуватися для визначення міжринкових відносин [44].

Технічні аналітики також використовують опитування інвесторів та інформаційні бюлетені, а також індикатори настроїв на обкладинці журналів.

Емпіричні докази

Чи справді працює технічний аналіз, є предметом суперечок. Методи дуже відрізняються, і різні технічні аналітики іноді можуть робити суперечливі прогнози на основі одних і тих же даних. Багато інвесторів стверджують, що вони отримують позитивні прибутки, але академічні оцінки часто виявляють, що вони мають малу передбачувану силу.З 95 сучасних досліджень 56 дійшли висновку, що технічний аналіз дав позитивні результати, хоча упередженість відстеження даних та інші проблеми ускладнюють аналіз.Нелінійне передбачення з використанням нейронних мереж іноді дає статистично значущі результати прогнозування.

Нещодавнє дослідження показало, що технічні торгові стратегії є ефективними на китайському ринку: «Нарешті, ми знаходимо значну позитивну прибутковість від торгівлі на покупку, створену протилежною версією правила перехрещення ковзного середнього, правила прориву каналу та Правило торгівлі смугою Болліндже́ра після врахування трансакційних витрат 0,50%»

Впливове дослідження 1992 року Brock et al. які, здавалося, знайшли підтримку для технічних правил торгівлі, було перевірено на предмет перевірки даних та інших проблем у 1999 році; вибірка, охоплена Brock et al. був стійкий до відстеження даних.

Згодом всебічне дослідження питання, проведене амстердамським економістом Гервіном Гріффіоеном, приходить до висновку, що: «для індексів фондового ринку США, Японії та більшості західноєвропейських індексів рекурсивна процедура прогнозування поза вибіркою не виявляється прибутковою після невеликих трансакційних витрат. Більше того, для достатньо високих трансакційних витрат шляхом оцінки CAPM було виявлено, що технічна торгівля

не показує статистично значущої потужності прогнозування поза вибіркою, скоригованої на ризик, майже для всіх індексів фондового ринку». застосовні до «стратегій імпульсу»; Всебічний огляд даних і досліджень 1996 року прийшов до висновку, що навіть невеликі трансакційні витрати призведуть до неможливості вловити будь-який надлишок від таких стратегій [40].

Гіпотеза ефективного ринку

Гіпотеза ефективного ринку (EMH) суперечить основним положенням технічного аналізу, стверджуючи, що минулі ціни не можуть бути використані для прибуткового прогнозування майбутніх цін. Таким чином, він вважає, що технічний аналіз не може бути ефективним. Економіст Юджин Фама опублікував основоположну статтю про EMH в Journal of Finance у 1970 році і сказав: «Коротко кажучи, докази на підтримку моделі ефективних ринків є великими, а (дещо унікально в економіці) суперечливих доказів небагато».

Однак, оскільки майбутні ціни на акції можуть сильно вплинути на очікування інвесторів, фахівці стверджують, що з цього випливає лише те, що минулі ціни впливають на майбутні ціни. Вони також вказують на дослідження в галузі поведінкових фінансів, зокрема на те, що люди не є раціональними учасниками, якими їх видає EMH. Техніки вже давно стверджують, що ірраціональна поведінка людини впливає на ціни акцій, і що така поведінка призводить до передбачуваних результатів [35].

Прихильники EMH відповідають, що, хоча окремі учасники ринку не завжди діють раціонально (або мають повну інформацію), їхні сукупні рішення врівноважують одне одного, що призводить до раціонального результату (оптимістам, які купують акції та пропонують більш високі ціни, протистоять песимісти, які продають свої акції , що утримує ціну в рівновазі).Так само повна інформація відображається в ціні, оскільки всі учасники ринку об’єднують на ринку свої індивідуальні, але неповні знання.

Гіпотеза випадкового блукання

Гіпотеза випадкового блукання може бути отримана з гіпотези слабкої форми ефективних ринків, яка базується на припущенні, що учасники ринку

повністю враховують будь-яку інформацію, що міститься в минулих змінах цін (але не обов’язково іншу публічну інформацію). У своїй книзі «Випадкова прогулянка по Уолл-стріт» економіст з Прінстона Бертон Малкіл сказав, що технічні інструменти прогнозування, такі як аналіз моделей, в кінцевому підсумку мають бути безпрограшними: «Проблема полягає в тому, що як тільки така закономірність стане відома учасникам ринку, люди будуть діяти відповідно до спосіб, який запобігає цьому в майбутньому.»[57] Малкіель заявив, що хоча імпульс може пояснити деякі зміни курсу акцій, імпульсу недостатньо для отримання надлишкового прибутку. Малкіель порівнював технічний аналіз з «астрологією».

Наприкінці 1980-х років професори Ендрю Ло і Крейг Мак-Кінлей опублікували роботу, яка поставила під сумнів гіпотезу випадкового блукання. У 1999 році у відповіді на Malkiel, Ло і МакКінлей зібрали емпіричні роботи, які поставили під сумнів застосовність гіпотези, які припускали невипадковий і, можливо, прогнозний компонент руху цін на акції, хоча вони обережно вказували, що відкидання випадкового блукання не означає обов'язково визнати недійсним EMH, який є цілком окремою концепцією від RWH. У статті 2000 року Ендрю Ло проаналізував дані зі США з 1962 по 1996 рік і виявив, що «кілька технічних індикаторів надають додаткову інформацію і можуть мати деяку практичну цінність».Бертон Малкіл відкинув порушення, згадані Ло та МакКінлі, як занадто малі, щоб отримати від них прибуток.

Техніки кажуть, що теорія EMH і випадкових блукань ігнорують реалії ринків, оскільки учасники не є повністю раціональними і що поточні зміни ціни не залежать від попередніх рухів.Деякі дослідники з обробки сигналів спростовують гіпотезу випадкового блукання, згідно з якою ціни на фондовому ринку нагадують процеси Вінера, оскільки статистичні моменти таких процесів і реальні дані про біржі значно відрізняються щодо розміру вікна та міри подібності.Вони стверджують, що перетворення ознак, які використовуються для опису аудіо та біосигналів, також можна використовувати для успішного

прогнозування цін на фондовому ринку, що суперечить гіпотезі випадкового блукання.

Індекс випадкового блукання (RWI) — це технічний індикатор, який намагається визначити, чи є рух ціни акції випадковим за своєю природою чи результатом статистично значущої тенденції. Індекс випадкового блукання намагається визначити, коли ринок перебуває в сильному тенденції до зростання чи спаду, вимірюючи діапазони цін понад N, і як він відрізняється від того, що можна було б очікувати від випадкового блукання (випадкове зростання або зниження). Більший діапазон свідчить про сильнішу тенденцію.

Застосовуючи теорію перспектив Канемана і Тверського до руху цін, Пол В. Аззопарді надав можливе пояснення, чому страх змушує ціни різко падати, а жадібність поступово підвищує ціни.Така поведінка цін на цінні папери, що зазвичай спостерігається, різко суперечить випадковому блуканню. Вимірюючи жадібність і страх на ринку, інвестори можуть краще формулювати довгі та короткі позиції портфеля.

Деякі моделі, такі як трикутник продовження або модель розвороту, можуть бути створені з припущенням двох окремих груп інвесторів з різними оцінками оцінки. Основні припущення моделей полягають у тому, що скінченність активів і використання тенденцій, а також оцінка при прийнятті рішень. Багато закономірностей є математично логічними наслідками цих припущень.

Однією з проблем традиційного технічного аналізу була складність визначення закономірностей у спосіб, який дозволяє об'єктивне тестування.

Японські моделі свічників включають моделі на кілька днів, які знаходяться в межах висхідного або спадного тренду. Caginalp і Laurent були першими, хто провів успішний широкомасштабний тест шаблонів. Математично точний набір критеріїв було перевірено, спочатку використавши визначення короткострокового тренда шляхом згладжування даних і врахування одного відхилення в згладженому тренді. Потім вони розглянули вісім основних триденних моделей розвороту свічок у непараметричний спосіб і визначили моделі як набір нерівностей. Результати були позитивними з переважною

статистичною впевненістю для кожної з моделей, які використовували набір даних про всі акції S&P 500 щодня за п’ятирічний період 1992–1996 років.

Серед найосновніших ідей традиційного технічного аналізу є те, що тенденція, колись встановлена, має тенденцію продовжувати. Однак перевірка цієї тенденції часто змушує дослідників дійти висновку, що акції є випадковим блуканням. Одне дослідження, проведене Потерба і Саммерс, виявило невеликий ефект тренду, який був занадто малим, щоб мати цінність для торгівлі. Як зазначив Фішер Блек, «шум» у даних про ціну торгівлі ускладнює перевірку гіпотез.

Один із методів уникнення цього шуму був виявлений у 1995 році Кагінальпом і Константином, які використовували співвідношення двох по суті ідентичних закритих фондів, щоб усунути будь-які зміни в оцінці. Закритий фонд (на відміну від відкритих) торгується незалежно від вартості чистих активів, і його акції не можуть бути викуплені, а торгуються лише серед інвесторів, як і будь-які інші акції на біржах. У цьому дослідженні автори виявили, що найкраща оцінка завтрашньої ціни — це не вчорашня ціна (як це вказує на гіпотезу ефективного ринку), а також не чиста ціна імпульсу (а саме, така ж відносна зміна ціни від вчорашнього до сьогодні триває з сьогодні на завтра). Але скоріше це майже половина шляху між ними.

Починаючи з характеристики минулого часу еволюції ринкових цін з точки зору швидкості ціни та прискорення ціни, була розроблена спроба загальних рамок технічного аналізу з метою встановлення принципової класифікації можливих закономірностей, що характеризують відхилення або дефекти від стану ринку випадкового блукання та його трансляційних інваріантних властивостей у часі. Класифікація ґрунтується на двох безрозмірних параметрах, число Фруда, що характеризує відносну силу прискорення по відношенню до швидкості, і прогноз часового горизонту, розміреного до періоду навчання. Виявлено, що тенденційні та протилежні моделі співіснують і залежать від безрозмірного часового горизонту. Використовуючи підхід групи ренормалізації, підхід

імовірнісного сценарію демонструє статистично значущу прогностичну силу практично на всіх перевірених ринкових фазах.

Опитування сучасних досліджень Парка та Ірвіна показало, що більшість знайшли позитивний результат технічного аналізу.

Використовуючи набори даних із понад 100 000 пунктів, вони демонструють, що тенденція має вплив, який принаймні наполовину менший, ніж оцінка. Ефекти обсягу та волатильності, які є меншими, також очевидні та статистично значущі. Важливим аспектом їх роботи є нелінійний ефект тренду. Позитивні тенденції, які відбуваються в межах приблизно 3,7 стандартних відхилень, мають позитивний ефект. Для сильніших тенденцій зростання негативно впливає на прибуток, що свідчить про те, що отримання прибутку відбувається у міру збільшення величини тенденції до зростання. Ці методи можна використовувати для вивчення поведінки інвестора та порівняння базових стратегій різних класів активів.

У 2013 році Кім Ман Луї і Т Чонг зазначили, що попередні результати технічного аналізу в основному повідомляли про прибутковість конкретних правил торгівлі для певного набору історичних даних. Ці минулі дослідження не брали до уваги торговця людини, оскільки жоден трейдер у реальному світі не буде механічно приймати сигнали від будь-якого методу технічного аналізу. Тому, щоб розкрити правду технічного аналізу, ми повинні повернутися до розуміння ефективності досвідчених і початківців трейдерів.

1.2 Проблематика прогнозування даних з використанням нейронних мереж

Основними проблемами прогнозування з використанням нейронних мереж можна вважати:

- вибір вірного алгоритму обробки даних;
- побудова коректних датасетів для навчання та тестування.

Проблема вибору вірного і підходящого алгоритму обробки даних полягає у наявності великої кількості різноманітних алгоритмів штучного інтелекту, які виконують аналіз по-своєму. Для вибору вірного алгоритму необхідно провести глибокий аналіз інформації стосовно процесу роботи алгоритмів штучного інтелекту для обробки даних.

Побудова коректних датасетів представляє собою просту проблему збору даних, так як для навчання нейронної мережі потрібна велика кількість даних, так як навчання займає велику кількість часу і вимагає великих об’ємів навчальної вибірки.

1.3 Огляд існуючих рішень

R — це мова програмування та безкоштовне програмне середовище для статистичних обчислень і графіки, яке підтримується основною командою R і R Foundation for Statistical Computing.Він широко використовується серед статистиків і майнерів даних для розробки статистичного програмного забезпечення та аналізу даних. Опитування, дослідження аналізу даних та дослідження баз даних наукової літератури показують значне зростання популярності R; з серпня 2021 року R займає 14-е місце в індексі TIOBE, що є показником популярності мов програмування [9].

Офіційне програмне середовище R є пакетом GNU. Він написаний переважно на C, Fortran і R (тому він частково розміщений самостійно) і є вільно доступним під загальною суспільною ліцензією GNU. Для різних операційних систем надаються попередньо скомпільовані виконувані файли. Він має інтерфейс командного рядка, але доступні кілька графічних інтерфейсів сторонніх виробників, таких як RStudio, інтегроване середовище розробки; і Jupyter, інтерфейс ноутбука.

R та його бібліотеки реалізують різноманітні статистичні та графічні методи, включаючи лінійне та нелінійне моделювання, класичні статистичні тести, просторовий та часовий аналіз, класифікацію, кластеризацію та інші. R легко розширюється за допомогою функцій і розширень, а його спільнота відома своїм активним внеском у пакети. Багато стандартних функцій R написані на самому R, що дозволяє користувачам легко слідувати зробленим алгоритмічним виборам. Для інтенсивних обчислень код C, C++ і Fortran можна зв’язувати та викликати під час виконання. Досвідчені користувачі можуть писати код C, C++, Java, .NET або Python, щоб безпосередньо маніпулювати об'єктами R. R є дуже розширюваним завдяки використанню пакетів, поданих користувачами, для конкретних функцій і конкретних областей дослідження. Завдяки своїй спадщині

S, R має потужніші засоби об'єктно-орієнтованого програмування, ніж більшість мов статистичних обчислень. Розширенню його також сприяють його правила лексичного визначення області.

Ще одна сильна сторона R — статична графіка; він може створювати графіки якості публікації, включаючи математичні символи. Динамічна та інтерактивна графіка доступна через додаткові пакети.

Statistica

Statistica — це пакет програмного забезпечення для передової аналітики, спочатку розроблений компанією StatSoft і в даний час підтримується TIBCO Software Inc.[1] Statistica забезпечує аналіз даних, керування даними, статистику, інтелектуальний аналіз даних, машинне навчання, аналіз тексту та процедури візуалізації даних.

Statistica — це набір аналітичних програмних продуктів і рішень, спочатку розроблених компанією StatSoft і придбаних компанією Dell у березні 2014 року. Програмне забезпечення включає в себе набір даних для аналізу, керування даними, візуалізації даних та процедур аналізу даних; а також різноманітні методи прогнозного моделювання, кластеризації, класифікації та дослідження [2]. Додаткові методи доступні завдяки інтеграції з безкоштовним середовищем програмування R з відкритим кодом.Різні пакети аналітичних методів доступні в шести лінійках продуктів.

Statistica включає аналітичні та дослідницькі графіки на додаток до стандартних 2- і 3-вимірних графіків. Дії кисті (інтерактивне маркування, маркування та виключення даних) дозволяють досліджувати викиди та досліджувати дані.

Робота програмного забезпечення зазвичай включає завантаження таблиці даних і застосування статистичних функцій зі спадних меню або (у версіях, починаючи з 9.0) із стрічки. У меню потім буде запропоновано вказати змінні та тип необхідного аналізу. Немає необхідності вводити командні рядки. Кожен аналіз може включати графічний або табличний вихід і зберігається в окремій робочій книзі.

1.4 Персептрон

У машинному навчанні персептрон — це алгоритм контрольованого навчання бінарних класифікаторів. Двійковий класифікатор — це функція, яка може вирішити, чи належить вхід, представлений вектором чисел, до певного класу.[1] Це тип лінійного класифікатора, тобто алгоритм класифікації, який робить свої прогнози на основі функції лінійного предиктора, що поєднує набір ваг з вектором ознак.

Алгоритм персептрона був винайдений у 1958 році в Корнельській авіаційній лабораторії Френком Розенблатом [3] за фінансування Управління військово-морських досліджень США[4].

Персептрон мав бути машиною, а не програмою, і хоча його перша реалізація була в програмному забезпеченні для IBM 704, згодом вона була реалізована в спеціальному обладнанні як «персептрон Mark 1». Ця машина була розроблена для розпізнавання зображень: вона мала масив із 400 фотоелементів, випадковим чином підключених до «нейронів». Вага кодувалася в потенціометрах, а оновлення ваги під час навчання здійснювалися електродвигунами.[2]

Хоча персептрон спочатку здавався багатообіцяючим, швидко було доведено, що персептрони неможливо навчити розпізнавати багато класів шаблонів. Це призвело до того, що область досліджень нейронних мереж застоюється протягом багатьох років, перш ніж було визнано, що нейронна мережа з прямим зв'язком з двома або більше шарами (також звана багатошаровим персептроном) мала більшу обробну потужність, ніж персептрони з одним шаром (також називають одношаровим). шар персептрон).

Одношарові персептрони здатні вивчати лише лінійно роздільні шаблони. Для завдання класифікації з деякою функцією поетапної активації один вузол матиме одну лінію, що розділяє точки даних, що утворюють шаблони. Більша кількість вузлів може створити більше розділових ліній, але ці лінії потрібно якимось чином об’єднати, щоб утворити складніші класифікації. Другого шару

персептронів або навіть лінійних вузлів достатньо, щоб вирішити багато проблем, які інакше не можна розділити.

У 1969 році відома книга Марвіна Мінського та Сеймура Пеперта під назвою «Персептрони» показала, що для цих класів мережі неможливо вивчити функцію XOR. Часто вважають (помилково), що вони також припустили, що подібний результат буде мати місце для багатошарової персептронної мережі. Однак це неправда, оскільки і Мінський, і Пеперт вже знали, що багатошарові персептрони здатні створювати функцію XOR. (Див. сторінку Perceptrons (книга) для отримання додаткової інформації.) Тим не менш, часто змішуваний текст Мінського/Паперта спричинив значне зниження інтересу та фінансування досліджень нейронних мереж. Минуло ще десять років, поки дослідження нейронних мереж відродилися в 1980-х роках. Цей текст був передрукований у 1987 році як «Perceptrons – Expanded Edition», де показано та виправлено деякі помилки в оригінальному тексті.

Алгоритм персептрона ядра вже був представлений в 1964 році Айзерманом та співавторами[5]. Гарантії граничних меж були надані для алгоритму Персептрона в загальному нероздільному випадку спочатку Фройндом і Шапіром (1998),[1] а нещодавно Мохрі та Ростамізаде (2013), які розширили попередні результати та дають нові межі L1.[6]]

Персептрон — це спрощена модель біологічного нейрона. Незважаючи на те, що для повного розуміння нейронної поведінки часто потрібна складність біологічних моделей нейронів, дослідження показують, що лінійна модель, подібна до персептрону, може викликати деяку поведінку, яка спостерігається у реальних нейронів.[7]

Алгоритм навчання

Нижче наведено приклад алгоритму навчання для одношарового персептрона. Для багатошарових персептронів, де існує прихований шар, необхідно використовувати більш складні алгоритми, такі як зворотне поширення. Якщо функція активації або основний процес, що моделюється персептроном, є нелінійними, можна використовувати альтернативні алгоритми

навчання, такі як правило дельта, якщо функція активації є диференційованою. Тим не менш, алгоритм навчання, описаний у наведених нижче кроках, часто буде працювати навіть для багатошарових персептронів з нелінійними функціями активації.

Коли кілька персептронів об’єднані в штучну нейронну мережу, кожен вихідний нейрон працює незалежно від усіх інших; таким чином, вивчення кожного результату можна розглядати ізольовано.

Варіанти

Кишеньковий алгоритм з трещоткою (Gallant, 1990) вирішує проблему стабільності навчання персептрону, зберігаючи найкраще рішення, яке було досі, «у своїй кишені». Потім кишеньковий алгоритм повертає рішення в кишені, а не останнє рішення. Його також можна використовувати для нероздільних наборів даних, де метою є знайти персептрон з невеликою кількістю помилок. Однак ці рішення виглядають суто стохастично, і, отже, кишеньковий алгоритм не наближається до них поступово в ході навчання, і не гарантується, що вони з’являться в межах заданої кількості кроків навчання.

Алгоритм Максовера (Wendemuth, 1995) є «надійним» у тому сенсі, що він буде сходитися незалежно від (попередніх) знань про лінійну роздільність набору даних.[12] У лінійно роздільному випадку це вирішить проблему навчання – при бажанні навіть з оптимальною стабільністю (максимальний запас між класами). Для нероздільних наборів даних він поверне рішення з невеликою кількістю помилок. У всіх випадках алгоритм поступово наближається до рішення в процесі навчання, не запам’ятовуючи попередні стани і без стохастичних стрибків. Конвергенція полягає в глобальній оптимальності для роздільних наборів даних і до локальної оптимальності для нероздільних наборів даних.

Проголосований персептрон (Freund and Schapire, 1999) є варіантом, що використовує кілька зважених персептронів. Алгоритм запускає новий персептрон щоразу, коли приклад неправильно класифікується, ініціалізуючи вектор ваг остаточними вагами останнього персептрона. Кожному персептрону також буде присвоєно іншу вагу, що відповідає тому, скільки прикладів вони

правильно класифікують, перш ніж неправильно класифікувати один, і в кінці результатом буде зважене голосування за всі персептрони.

У роздільних задачах навчання персептрону також може бути спрямоване на знаходження найбільшого розділювача між класами. Так званий персептрон оптимальної стабільності можна визначити за допомогою ітераційних схем навчання та оптимізації, таких як алгоритм Min-Over (Krauth and Mezard, 1987) [11] або AdaTron (Anlauf and Biehl, 1989)). 13] AdaTron використовує той факт, що відповідна задача квадратичної оптимізації є опуклою. Персептрон оптимальної стабільності разом із трюком ядра є концептуальними основами машини опорного вектора.

a-perceptron також використовував шар попередньої обробки фіксованих випадкових ваг із пороговими вихідними одиницями. Це дозволило персептрону класифікувати аналогові моделі, проектуючи їх у двійковий простір. Фактично, для проекційного простору достатньо великого розміру візерунки можуть стати лінійно роздільними.

Інший спосіб розв’язувати нелінійні задачі без використання кількох шарів – використовувати мережі вищого порядку (одиниця сигма-пі). У цьому типі мережі кожен елемент у вхідному векторі розширюється кожною попарною комбінацією помножених входів (другого порядку). Це можна розширити на мережу n-порядку.

Однак слід пам’ятати, що найкращий класифікатор – це не обов’язково той, який ідеально класифікує всі навчальні дані. Дійсно, якби у нас було попереднє обмеження, що дані надходять з рівноваріантних гауссових розподілів, лінійне розділення у вхідному просторі є оптимальним, а нелінійне рішення переобладнане.

Інші алгоритми лінійної класифікації включають Winnow, машину опорного вектора та логістичну регресію.

РОЗДІЛ 2
ПРОЕКТУВАННЯ

2.1 Опис мови програмування

Python — це інтерпретована, об’єктно-орієнтована мова програмування високого рівня з динамічною семантикою. Його високорівневі вбудовані структури даних у поєднанні з динамічним типізацією та динамічним зв’язуванням роблять його дуже привабливим для швидкої розробки додатків, а також для використання в якості мови сценаріїв або склеювання для з’єднання існуючих компонентів. Простий, легкий у засвоєнні синтаксис Python підкреслює читабельність і, отже, знижує витрати на обслуговування програми. Python підтримує модулі та пакунки, що сприяє модульності програм і повторному використанню коду. Інтерпретатор Python і велика стандартна бібліотека доступні у вихідній або двійковій формі безкоштовно для всіх основних платформ і можуть вільно поширюватися.

Часто програмісти використовують Python через підвищену продуктивність, яку він забезпечує. Оскільки етапу компіляції немає, цикл редагування-тестування-налагодження неймовірно швидкий. Налагоджувати програми Python легко: помилка або неправильне введення ніколи не призведе до помилки сегментації. Натомість, коли інтерпретатор виявляє помилку, він створює виняток. Якщо програма не вловлює виняток, інтерпретатор друкує трасування стека. Налагоджувач рівня вихідного коду дозволяє перевіряти локальні та глобальні змінні, оцінювати довільні вирази, встановлювати точки зупину, переходити через код по рядку і так далі. Налагоджувач написаний на самому Python, що свідчить про інтроспективну силу Python. З іншого боку, часто найшвидший спосіб налагодити програму – це додати кілька операторів print до джерела: швидкий цикл редагування-тестування-налагодження робить цей простий підхід дуже ефективним.

Python часто порівнюють з іншими інтерпретованими мовами, такими як Java, JavaScript, Perl, Tcl або Smalltalk. Порівняння з C++, Common Lisp і Scheme також можуть бути просвітливими. На практиці вибір мови програмування часто диктується іншими реальними обмеженнями, такими як вартість, доступність, навчання, попередні інвестиції або навіть емоційна прихильність.

Java

Зазвичай очікується, що програми на Python працюватимуть повільніше, ніж програми на Java, але вони також займають набагато менше часу на розробку. Програми на Python зазвичай в 3-5 разів коротші, ніж еквівалентні програми на Java. Цю різницю можна пояснити вбудованими високорівневими типами даних Python та його динамічним типом. Наприклад, програміст Python не витрачає часу на оголошення типів аргументів або змінних, а потужний поліморфний список і типи словників Python, для яких багата синтаксична підтримка вбудована прямо в мову, знаходять застосування майже в кожній програмі Python. Через введення під час виконання час виконання Python повинен працювати інтенсивніше, ніж Java. Наприклад, оцінюючи вираз a+b, він повинен спочатку перевірити об’єкти a і b, щоб з’ясувати їх тип, який невідомий під час компіляції. Потім він викликає відповідну операцію додавання, яка може бути перевантаженим методом, визначеним користувачем. Java, з іншого боку, може виконувати ефективне додавання цілого або з плаваючою комою, але вимагає оголошення змінних для a і b і не допускає перевантаження оператора + для екземплярів визначених користувачем класів.

З цих причин Python набагато краще підходить як мова «клей», тоді як Java краще характеризувати як мову реалізації низького рівня. Насправді, ці два разом створюють чудову комбінацію. Компоненти можна розробляти на Java та об’єднувати для формування додатків на Python; Python також можна використовувати для створення прототипів компонентів, поки їхній дизайн не буде «загартований» у реалізації Java. Для підтримки такого типу розробки розробляється реалізація Python, написана на Java, яка дозволяє викликати код Python з Java і навпаки. У цій реалізації вихідний код Python перекладається в

байт-код Java (за допомогою бібліотеки часу виконання для підтримки динамічної семантики Python).

Javascript

«Об’єктна» підмножина Python приблизно еквівалентна JavaScript. Як і JavaScript (і на відміну від Java), Python підтримує стиль програмування, який використовує прості функції та змінні без участі в визначеннях класів. Однак для JavaScript це все, що є. Python, з іншого боку, підтримує написання набагато більших програм і кращого повторного використання коду завдяки справжньому об’єктно-орієнтованому стилю програмування, де класи та спадкування відіграють важливу роль.

Perl

Python і Perl походять із схожого досвіду (скрипти Unix, які давно переросли) і мають багато схожих функцій, але мають іншу філософію. Perl наголошує на підтримці типових завдань, орієнтованих на програми, наприклад, вбудовані регулярні вирази, функції сканування файлів і створення звітів. Python наголошує на підтримці загальних методологій програмування, таких як розробка структури даних та об’єктно-орієнтоване програмування, і заохочує програмістів писати читабельний (і, отже, підтримуваний) код, надаючи елегантну, але не надто загадкову нотацію. Як наслідок, Python наближається до Perl, але рідко перевершує його у своїй початковій області застосування; однак Python може застосовуватися далеко за межами ніші Perl.

Tcl

Як і Python, Tcl можна використовувати як мову розширення програми, а також як окрему мову програмування. Однак Tcl, який традиційно зберігає всі дані у вигляді рядків, слабкий щодо структур даних і виконує типовий код набагато повільніше, ніж Python. Tcl також не має функцій, необхідних для написання великих програм, таких як модульні простори імен. Таким чином, хоча «типова» велика програма, що використовує Tcl, зазвичай містить розширення Tcl, написані на C або C++, які є специфічними для цієї програми, еквівалентну програму Python часто можна написати на «чистому Python». Звичайно, розробка

на чистому Python набагато швидша, ніж писати й налагоджувати компонент C або C++. Було сказано, що єдиною корисною якістю Tcl є набір інструментів Tk. Python прийняв інтерфейс до Tk як стандартну бібліотеку компонентів графічного інтерфейсу.

Tcl 8.0 вирішує проблему швидкості, надаючи компілятор байт-коду з обмеженою підтримкою типів даних і додає простори імен. Однак це все ще набагато більш громіздка мова програмування.

Smalltalk

Можливо, найбільша різниця між Python і Smalltalk полягає в більш «основному» синтаксисі Python, який дає йому перевагу в навчанні програміста. Як і Smalltalk, Python має динамічний введення та зв'язування, і все в Python є об'єктом. Однак Python відрізняє вбудовані типи об'єктів від визначених користувачем класів і наразі не дозволяє успадковувати вбудовані типи. Стандартна бібліотека типів даних колекції Smalltalk є більш досконалою, тоді як бібліотека Python має більше можливостей для роботи з Інтернетом та реальністю WWW, такими як електронна пошта, HTML і FTP.

Python має іншу філософію щодо середовища розробки та поширення коду. Якщо Smalltalk традиційно має монолітний «образ системи», який містить як середовище, так і програму користувача, Python зберігає як стандартні модулі, так і користувацькі модулі в окремих файлах, які можна легко перегрупувати або розповсюдити за межами системи. Одним із наслідків є те, що існує більше одного варіанту підключення графічного інтерфейсу користувача (GUI) до програми Python, оскільки графічний інтерфейс не вбудований в систему.

C++

Майже все, що сказано для Java, також стосується C++, тільки більше: якщо код Python зазвичай в 3-5 разів коротший за еквівалентний код Java, він часто в 5-10 разів коротший за еквівалентний код C++.

Common Lisp і Scheme

Ці мови близькі до Python за своєю динамічною семантикою, але дуже різні за своїм підходом до синтаксису. Слід зазначити, що Python має інтроспективні

можливості, подібні до можливостей Lisp, а програми Python можуть створювати та виконувати фрагменти програми на льоту. Зазвичай властивості реального світу мають вирішальне значення: Common Lisp великий (у всіх сенсах), а світ Scheme фрагментований між багатьма несумісними версіями, де Python має єдину, безкоштовну, компактну реалізацію.

Виходячи з порівняння Python з відомими на цей час мовами програмування, основною мовою розробки даного програмного забезпечення було обрано саме Python.

Python динамічно збирає сміття. Він підтримує кілька парадигм програмування, включаючи структуроване (включаючи процедурне), об'єктно-орієнтоване та функціональне програмування. Оскільки Python має повну стандартну бібліотеку, її зазвичай описують як мову, що включає батареї.

Як спадкоємець мови програмування ABC, Гідо ван Россум почав займатись дослідженнями Python наприкінці 1980-х і вперше випустив цю мову під назвою Python 0.9.0 в 1991 році. Python 2.0 був випущений у 2000 році, представивши нові функції, такі як розуміння списку та система збору сміття за допомогою підрахунку посилань, і був припинений у версії 2.7.18 у 2020 році. Python 3.0 був випущений в 2008 році і є основною редакцією мови. Мова не повністю сумісна із зворотним характером, і більшість кодів Python 2 не можуть нормально працювати на Python 3.

Python завжди був однією з найпопулярніших мов програмування.

Python був задуманий наприкінці 1980-х років Гвідо ван Россумом з Нідерландських Centrum Wiskunde & Informatica (CWI). Він є наступником мови програмування ABC, натхненної SETC, яка може обробляти винятки та взаємодіяти з операційною системою Amoeba. Його реалізація розпочалась у грудні 1989 року. Як головний розробник проекту, Ван Россум брав на себе повну відповідальність за проект до 12 липня 2018 року, коли оголосив про своє "вічне свято" титулом "дружнього диктатора життя" Пітона. Він знайшов своє відображення у своїй довгій історії спільнотою Python. Зобов'язання бути головним менеджером проекту. Зараз він займає керівну посаду в якості члена

ради наглядових органів із п'яти осіб. У січні 2019 року активні розробники ядра Python обрали Бретта Кеннона, Ніка Коглана, Баррі Варшаву, Керол Віллінг та Ван Россума членами комітету з п'яти осіб. З тих пір Гвідо ван Россум відкликав свою кандидатуру до ради 2020 року.

Python 2.0 був випущений 16 жовтня 2000 року з багатьма основними новими можливостями, включаючи збирач сміття для виявлення циклів та підтримку Unicode.

Python 3.0 був випущений 3 грудня 2008 року. Це серйозний перегляд мови, який не повністю сумісний з іншою стороною. Багато його основних функцій було перенесено до версій Python 2.6.x та 2.7.x. До розподілу Python 3 входить утиліта 2to3, яка автоматично (принаймні частково) перетворює код Python 2 у Python 3.

Термін дії Python 2.7 спочатку був встановлений у 2015 році, а потім перенесений на 2020 рік через побоювання, що велика кількість існуючого коду не може бути легко перенесена на Python 3. Більше виправлень безпеки та інших удосконалень не буде випущено. Наприкінці життєвого циклу Python 2 підтримує лише Python 3.6.x та новіші версії.

Оскільки всі версії Python мають проблеми з безпекою, швидкість Python 3.9.2 та 3.8.8 була прискорена, що призвело до можливого віддаленого виконання коду та отруєння веб-кешу.

Python - мова програмування з багатьма парадигмами. Він повністю підтримує об'єктно-орієнтоване програмування та структуроване програмування, а багато його функцій підтримують функціональне програмування та аспектно-орієнтоване програмування (включаючи дизайн метапрограми та мета-об'єкт (магічний метод)). Розширення підтримує багато інших парадигм, включаючи дизайн контрактів та логічне програмування.

Python використовує динамічне введення тексту та комбінацію підрахунку посилань та збирачів сміття.Сбирач сміття визначає цикл для управління пам'яттю. Він також має динамічну роздільну здатність імен (пізнє

прив'язування), щоб асоціювати імена методів та змінних під час виконання програми.

Python призначений для надання певної підтримки функціонального програмування на основі Lisp. За допомогою функцій фільтрування, відображення та скорочення; перелічено розуміння генераторів, словників, наборів та виразів. Стандартна бібліотека має два модулі (itertools та functools) для реалізації функціональних інструментів, запозичених у Haskell та Standard ML.

Основні принципи мови викладені в документі Zen of Python (PEP 20), який включає такі сентенції:

- Красиво-краще, ніж потворно.
- Явне краще, ніж неявне.
- Просте - краще, ніж складне.
- Комплекс краще, ніж комплекс.
- Розрахунок читабельності.

Python не має всіх своїх функцій, вбудованих у своє ядро, але розроблений таким чином, щоб бути дуже розширюваним (з модулями). Ця компактна модульність робить його особливо популярним як спосіб додавання програмованих інтерфейсів до існуючих програм. Бачення Ван Россума щодо малих основних мов, великих стандартних бібліотек та легко розширюваних перекладачів випливає з його невдоволення ABC, який підтримує протилежний підхід. Python прагне спростити та зменшити плутанину граматики та граматики, дозволяючи розробникам обирати методи кодування. На відміну від девізу Perl "Існує кілька способів зробити це", Python висвітлює філософію дизайну "Для цього повинен бути один - бажано, лише один очевидний спосіб". Алекс Мартеллі, співробітник Фонду програмного забезпечення Python і автор книги про Python, писав: "Опис чогось як" розумного "не вважається компліментом до культури Python".

Розробники Python прагнуть уникнути передчасної оптимізації та відмовляються виправляти некритичні частини посилальної реалізації CPython. Ці виправлення відбуваються за рахунок ясності, тим самим трохи покращуючи швидкість. Коли швидкість важлива, програмісти Python можуть переміщати критично важливі для часу функції до розширень, написаних мовами, такими як C, або використовувати компілятор PyPy, що встигає за часом. Також доступний Cython, який перетворює сценарії Python на C і здійснює виклики API рівня C безпосередньо до інтерпретатора Python.

Важливою метою розробників Python є збереження задоволення від використання. Це відображається в назві мови (данина поваги британській комедійній компанії Monty Python), а іноді і в цікавих підручниках та довідкових методах, таких як стандартний foo та bars.

Користувачів та шанувальників Python, особливо тих, хто вважається обізнаним або досвідченим, часто називають Pythonistas. Python має бути простою для читання мовою. Його формат візуально не бентежить, він часто використовує англійські ключові слова, тоді як інші мови використовують розділові знаки. На відміну від багатьох інших мов, він не використовує фігурні дужки для розділення блоків і допускає крапку з комою після операторів, але крапка з комою використовується рідко. Він має менше граматичних винятків та особливих випадків, ніж C або Pascal.

Python використовує пробіли замість фігурних дужок або ключових слів для розділення блоків. Збільшення відступу відбувається після певних операторів; зменшення відступу означає кінець поточного блоку. Тому візуальна структура програми точно відображає семантичну структуру програми. Цю особливість іноді називають `` зовнішнім " правилом, і деякі інші мови також мають це правило, але в більшості мов відступ не має семантичного значення. Рекомендований розмір відступу - чотири пробіли.

Оператори Python включають (серед іншого):

- Використовуйте знак рівності = оператор присвоєння.

- Оператор if, який використовується разом з else та elif (скорочення від else-if) для умовного виконання блоку коду.
- Оператор for, який фіксує кожен елемент у локальній змінній для використання вкладеним блоком для перегляду об'єкта, що ітерацію.
- Поки оператор, доки умова є істинним, виконується блок коду.
- Оператор try, який дозволяє захоплювати та обробляти витяги, що містяться у вкладених блоках коду (крім речень); він також гарантує, що незалежно від того, як блок виходить, код очищення в блоці завжди буде виконаний в кінцевому підсумку.
- Оператор підвищення застосовується для отримання зазначеного винятку або повторного підняття спійманого винятку.
- Оператори класів, які виконують блоки коду та приєднують свій локальний простір імен до класу для використання в об'єктно-орієнтованому програмуванні.
- Оператор Def, який визначає функцію або метод.
- Оператор with з Python 2.5, випущений у вересні 2006 р. [82], який охоплює блок коду в диспетчері контексту (наприклад, отримання блокування перед запуском кодового блоку, а потім звільнення блокування або відкриття файлу. Потім закрийте його), дозволяють поведінку подібних ресурсів ініціалізувати (RAII) та замінюють загальне використання try / final.
- Оператор Break, вихід із циклу.
- Оператор continue пропускає цю ітерацію та переходить до наступного пункту.
- Оператор del видалить змінну, що означає, що посилання на ім'я значення буде видалено, а спроби використання цієї змінної призведуть до помилки. Ви можете перепризначити видалені змінні.
- Виконувати функції NOP через операторів. Синтаксично це необхідно для створення порожнього блоку коду.

- Заява про затвердження, що використовується для перевірки умов, що застосовуються під час налагодження.
- Оператор Yield, який повертає значення з функції генератора. У Python 2.5 yield також є оператором. Ця форма використовується для реалізації спільного плану.
- Оператор return використовується для повернення значень із функцій.
- Оператор import використовується для імпорту модулів, функції чи змінні яких можна використовувати в поточній програмі. Існує три способи використання імпорту: імпорт <ім'я модуля> [як <псевдонім>] або <ім'я модуля> імпорт * або <ім'я модуля> для імпорту <визначення 1> [як <псевдонім 1>], <визначення 2 > [Як <псевдонім 2>], ...

Роль оператора присвоєння (=) полягає у прив'язуванні імені як посилання на окремий динамічно розподілений об'єкт. Потім ви можете будь-коли відхилити змінну до будь-якого об’єкта. У Python ім'я змінної є спільним власником посилання, і з ним не пов’язаний фіксований тип даних. Однак у цей час змінна посилається на об’єкт із типом. Це називається динамічним набором тексту і на відміну від статично набраних мов програмування, де кожна змінна може містити лише певний тип значення.

Python не підтримує оптимізацію хвостових викликів або першокласні розширення, і за словами Гвідо ван Россума, це ніколи не буде. Однак краща підтримка функцій, подібних до корутину, передбачена в 2.5, розширюючи тим самим генератор Python. Ледачий ітератор має максимум 2,5 генератора. Інформація передається в одному напрямку від генератора. За допомогою Python 2.5 ви можете передавати інформацію назад до функції генератора, тоді як за допомогою Python 3.3 ви можете передавати інформацію через кілька рівнів стека.

Деякі вирази Python подібні до таких, що зустрічаються в таких мовах, як C та Java, а інші - ні:

- Додавання, віднімання та множення однакові, але поведінка ділення інша. У Python є два типи розділів. Вони є базовим діленням (або цілочисельним поділом) // та плаваючою комою / діленням. Python також використовує оператори ** для просування.
- У Python 3.5 було введено нове твердження @infix. Він використовується бібліотеками, такими як NumPy, для множення матриць.
- У Python 3.8 введено синтаксис: =, який називається "моржовим оператором". Він присвоює значення змінним як частину більшого виразу. [91]
- У Python == порівнює з Java за значенням, Java порівнює числові значення за значенням, а об'єкти порівнює за посиланням. (Порівняння значень об'єктів у Java може бути здійснено за допомогою методу equals ().) Оператор is is is може бути використаний для порівняння ідентифікаторів об'єктів (порівняння посилань). У Python можна порівняти порівняння, наприклад <= b <= c.
- Python використовує слово і, або не для своїх булевих операторів, а не для символів &&, ||! Використовується в Java та C.Python має тип виразу, що називається розумінням списку, а також більш загальний вираз, який називається генераторським виразом.
- • Анонімні функції реалізовані з використанням лямбда-виразів, однак вони обмежені тим фактом, що тіло може бути лише виразом.
- • Якщо c інакше y, умовний вираз у Python буде записаний як x (відмінний від порядку операнда оператора c? X: y), що є однаковим у багатьох інших мовах.
- • Python розрізняє списки та кортежі. Список записаний як [1, 2, 3], він був змінений і не може використовуватися як ключ словника (ключ словника повинен бути постійним у Python). Кортеж записується як (1, 2, 3) і є незмінним, тому, поки всі елементи кортежу є незмінними, він може використовуватися як ключ словника. Оператор + може використовуватися для об'єднання двох кортежів, що безпосередньо не змінює їх вміст, але

створює новий кортеж, що містить елементи двох передбачених кортежів. Отже, враховуючи, що змінна t спочатку дорівнює (1,2,3), коли виконується t = t + (4, 5), спочатку обчисліть t + (4, 5), щоб отримати (1,2, 3) . 3, 4, 5), а потім призначити його назад до t, тим самим ефективно "змінюючи вміст t", одночасно відповідаючи інваріантній природі об'єкта кортежу. У чіткому контексті кортежі не потребують дужок.

- • Python має функцію декомпресії послідовності, де кілька виразів (кожен вираз обчислює весь вміст, який може бути призначений) (змінні, атрибути запису тощо) мають однакову форму з текстом, що становить кортеж, і, як правило, розміщуються зліва Одна сторона знака рівності у твердженні про присвоєння. Оператор очікує ітерабельний об’єкт праворуч від знака рівності. При сортуванні об’єкт видасть таку ж кількість значень, що й даний вираз, і відфільтрує їх, присвоюючи кожне створене значення лівій стороні відповідного виразу.
- • Python має оператор% "рядковий формат". Це схоже на рядок printf у C, наприклад, "спам =% s яйце =% d"% ("бла", 2) оцінюється як "спам = бла-яйця = 2". У Python 3 та 2.6+ метод str () формату () доповнив це, наприклад "спам = {0} яйце = {1}". Формат ("бла", 2). Python 3.6 додав "f-рядки": blah = "blah"; яйця = 2; f'spam = {blah} яйця = {яйця}.
- • Рядки в Python можна поєднувати шляхом "додавання" (те саме твердження, що і додавання цілих чи значень з плаваючою комою). Наприклад, "спам" + "яйце" повертає "спамера". Навіть якщо ваш рядок містить цифри, вони все одно будуть додані як рядки замість цілих чисел. Наприклад, "2" + "2" повертає "22".
- • Python має різні типи рядкових літералів:
  - Рядки, укладені в одинарні або подвійні лапки. На відміну від Unix Shell, мови Perl та Perl впливають на Perl, а одинарні та подвійні лапки працюють однаково. Обидва типи рядків використовують зворотні скісні риски (\) як вихідні символи. Інтерполяція рядків стала "відформатованим рядковим літералом" у Python 3.6.Рядки з

подвійними лапками, які починаються та закінчуються серією з трьох одинарних або подвійних лапок. Вони можуть охоплювати кілька рядків і функціонувати, як тут документи в оболонках, Perl і Ruby.

- Початковий рядок, представлений префіксом рядкового літералу перед r. Ескейп-послідовності не інтерпретуються; тому необроблені рядки корисні в місцях, де поширені зворотні скісні риски літер, наприклад, регулярні вирази та шляхи у стилі Windows. Порівняйте "@ -citation" у C #.

- Python має індекси масивів та вирази масивів у списках, позначених як [ключ], [старт: зупинка] або [старт: зупинка: крок]. Індекс базується на нулі та є від'ємним відносно кінця. Нарізання переміщує елементи від початкового індексу до (але не враховуючи) індексу зупинки. Третій варіант зрізу називається кроком або висотою, що дозволяє пропускати та обертати елементи. Індекс зрізу можна опустити, наприклад [:] повертає копію всього списку. Кожен елемент зрізу є неглибокою копією.

Python чітко розмежовує вирази та висловлювання, що відрізняється від таких мов, як Common Lisp, Scheme або Ruby. Це призводить до дублювання певних функцій.

Python зазвичай використовується в проектах зі штучного інтелекту та машинного навчання з такими бібліотеками, як TensorFlow, Keras, Pytorch та Scikit-learn. Як мова сценаріїв з модульною архітектурою, простим синтаксисом та розширеними засобами обробки текстів, Python зазвичай використовується для обробки природних мов. Python успішно впроваджений у багато програмних продуктів на мовах сценаріїв, включаючи програмне забезпечення з кінцевими елементами, таке як Abaqus, 3D-параметричні моделятори, такі як FreeCAD, 3ds Max, Blender, Cinema 4D, Lightwave, Houdini, Maya, modo, MotionBuilder, Softimage, Nuke visual ефекти Composer, програми для двовимірних зображень (такі як GIMP, Inkscape, Scribus та Paint Shop Pro) та програми для приміток (наприклад, автор та група). Налагоджувач GNU використовує Python як хороший принтер для відображення складних структур, таких як контейнери C ++.

Есрі виступає за те, що Python є найкращим вибором для сценаріїв в ArcGIS. Він також використовувався в декількох відеоіграх і був першою з трьох доступних мов програмування, що використовуються Google App Engine, а інші дві використовуються як Java та Go.

Багато операційних систем використовують Python як стандартний компонент. Він поставляється з більшістю дистрибутивів Linux, AmigaOS 4 (з використанням Python 2.7), FreeBSD (як пакет), NetBSD, OpenBSD (як пакет) та macOS, і може використовуватися з командного рядка (терміналу). Багато дистрибутивів Linux використовують інсталятори, написані на Python: Ubuntu використовує інсталятор Ubiquity, тоді як Red Hat Linux та Fedora використовують інсталятор Anaconda. Gentoo Linux використовує Python у своїй системі управління пакетами Portage.

Python широко використовується для захисту інформації, включаючи розробку експлойт-програм.

Більшість програм Sugar Laptop XO, які зараз розробляються Sugar Labs, написані на Python. Проект одноплатної комп'ютерної програми Raspberry Pi прийняв Python як основну мову програмування користувача.

LibreOffice включає Python і має намір замінити Java на Python. Його постачальник скриптів Python є основною функцією у версії 4.0 від 7 лютого 2013 року.

Виходячи з обсягу та потужних особливостей мови Python, описаних вище, можна зробити висновок, що вона універсальна. Ось чому для виконання цієї роботи було обрано саме цю мову програмування.

2.2 Опис середовища розробки

PyCharm – це інтегроване середовище розробки (IDE), що використовується в комп'ютерному програмуванні, зокрема для мови Python. Він розроблений чеською компанією JetBrains. Він забезпечує аналіз коду, графічний налагоджувач, інтегрований тестер одиниць, інтеграцію із системами контролю версій (VCSes), а також підтримує веб-розробку з Django, а також аналіз даних з Anaconda.

PyCharm є крос-платформним, з версіями для Windows, macOS та Linux. Видання Community випускається під ліцензією Apache, а також існує Professional Edition з додатковими функціями - випущено під власною ліцензією.

Особливості

- Допомога та аналіз коду із завершенням коду, виділенням синтаксису та помилок, інтеграцією лінтера та швидкими виправленнями
- Навігація по проекту та коду: спеціалізовані перегляди проекту, перегляди структури файлів і швидкий перехід між файлами, класами, методами та використанням
- Рефакторинг Python: включає перейменування, метод вилучення, введення змінної, введення константи, підтягування, натискання та інші
- Підтримка веб-фреймворків: Django, web2py і Flask [лише професійна версія][8]
- Інтегрований налагоджувач Python
- Інтегроване модульне тестування з порядковим покриттям коду
- Розробка Google App Engine Python [лише професійна версія]
- Інтеграція контролю версій: уніфікований інтерфейс користувача для Mercurial, Git, Subversion, Perforce і CVS зі списками змін і злиттям
- Підтримка наукових інструментів, таких як matplotlib, numpy і scipy [лише професійне видання][9]

Він конкурує в основному з низкою інших IDE, орієнтованих на Python, включаючи PyDev від Eclipse і більш широку IDE Komodo.

PyCharm надає API, щоб розробники могли писати власні плагіни для розширення можливостей PyCharm. Кілька плагінів з інших IDE JetBrains також працюють з PyCharm. Існує більше 1000 плагінів, які сумісні з PyCharm.[10]

Бета-версія була випущена в липні 2010 року, а версія 1.0 з’явилася через 3 місяці. Версія 2.0 була випущена 13 грудня 2011 року, версія 3.0 - 24 вересня 2013 року, а версія 4.0 - 19 листопада 2014 року

PyCharm Community Edition, версія з відкритим кодом PyCharm, стала доступною 22 жовтня 2013 р.

2.3 Опис додаткового інструментарію

scikit-image (раніше scikits.image) - це бібліотека зображень з відкритим кодом для мови програмування Python [2]. Він включає алгоритми сегментації, геометричного перетворення, маніпулювання кольоровим простором, аналізу, фільтрації, морфології, виявлення особливостей тощо. Він призначений для взаємодії з числовою та науковою бібліотеками Python NumPy та SciPy.

Проект scikit-image розпочався на scikits.image, а його автором був Стефан ван дер Вальт. Його назва походить від ідеї, що це "SciKit" (набір інструментів SciPy), розроблений та розповсюджений окремо стороннім розширенням SciPy [4]. Оригінальна база коду пізніше була широко переписана іншими розробниками. У листопаді 2012 року в різних наукових роботах наукові знання та наукові знання були описані як «ретельно модифіковані та популярні» [5]. Scikit-image також дуже активний у Google Summer of Code [6].

scikit-image в основному написаний на Python, а деякі основні алгоритми написані на Cython для підвищення продуктивності.

РОЗДІЛ 3
РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Діаграма варіантів використання

Діаграма використання найпростіша - це представлення взаємодії користувача із системою, яка показує взаємозв'язок між користувачем та різними випадками використання, в яких користувач бере участь. Діаграма випадків використання може ідентифікувати різні типи користувачів системи та різні випадки використання, і часто вона супроводжується також іншими типами діаграм. Варіанти використання представлені кругами або еліпсами.

Незважаючи на те, що сам випадок використання може детально вивчити кожну можливість, діаграма прикладів використання може допомогти забезпечити огляд системи на більш високому рівні. Раніше вже було сказано, що "схеми використання - це принципи вашої системи".

Через їх спрощений характер, схеми використання можуть бути хорошим інструментом комунікації для зацікавлених сторін. Креслення намагаються імітувати реальний світ і дають зацікавленій стороні уявлення про те, як буде розроблена система.

Метою діаграми використання є відображення динамічного аспекту системи. Додаткові схеми та документація можуть бути використані для забезпечення повного функціонального та технічного уявлення про систему. Вони забезпечують спрощене та графічне представлення того, що система насправді повинна робити.

Елементи:

- рамки системи (англ. system border) - прямокутник із назвою у верхніх частинах та еліпсами (прецедентами) всередині. Часто може бути опущено без корисної інформації,

- актор (англ. актор) - стилізований людський персонаж, обозначаючий набір ролей користувача (розуміється в широкому змісті: людина, внешняя сутність, клас, інша система), взаємодіючого з деякою сутністю (системною, підсистемою, класом). Актори не можуть бути пов'язані між собою з іншим (за вимкнення відносин щодо обробки / дослідження),
- прецедент - еліпс із надписом, що означає виконувану систематичну дію (може включати можливі варіанти), що призводить до спостерігаємих акторами результатів. Надпис може бути ім'ям або описом (з точки зору актора) того, "що" робить система (а не "як"). Ім’я прецедента зв’язано з неперервним (атомарним) сценарієм - конкретною послідовністю дій, ілюстріруючою повіддю. Під час сценарію актори обмінюються із систематичними повідомленнями. Сценарій може бути приведений на діаграмі прецедентів у відео UML-коментарі. З одним прецедентом може бути пов'язано кілька різних сценаріїв

На рисунку 3.1 зображено діаграму варіантів використання, яка описує можливі дії користувача в системі.

Рис. 3.1 — Діаграма варіантів використання

3.2 Діаграма діяльності

Діаграми діяльності — це графічне представлення робочих процесів поетапних дій і дій з підтримкою вибору, ітерації та паралельності. В уніфікованій мові моделювання діаграми діяльності призначені для моделювання як обчислювальних, так і організаційних процесів (тобто робочих процесів), а також потоків даних, що перетинаються з відповідними видами діяльності. Хоча діаграми діяльності в першу чергу показують загальний потік контролю, вони також можуть включати елементи, що показують потік даних між діями через одне або кілька сховищ даних.

Діаграми діяльності будуються з обмеженої кількості фігур, з'єднаних стрілками. Найважливіші типи фігур:

- крапки представляють дії;
- діаманти символізують рішення;
- смуги представляють початок (розщеплення) або кінець (об’єднання) одночасних дій;
- чорне коло представляє початок (початковий вузол) робочого процесу;
- обведене чорне коло представляє кінець (кінцевий вузол).

Стрілки йдуть від початку до кінця і представляють порядок, у якому відбуваються дії.

Діаграми діяльності можна розглядати як форму структурованої блок-схеми в поєднанні з традиційною схемою потоку даних. У типових методах блок-схем відсутні конструкції для вираження паралельності. Однак символи об’єднання та розбиття на діаграмах діяльності вирішують це лише для простих випадків; сенс моделі не зрозумілий, коли вони довільно поєднуються з рішеннями або циклами.

На рисунку 3.2 показана діаграма діяльності розробленої системи.

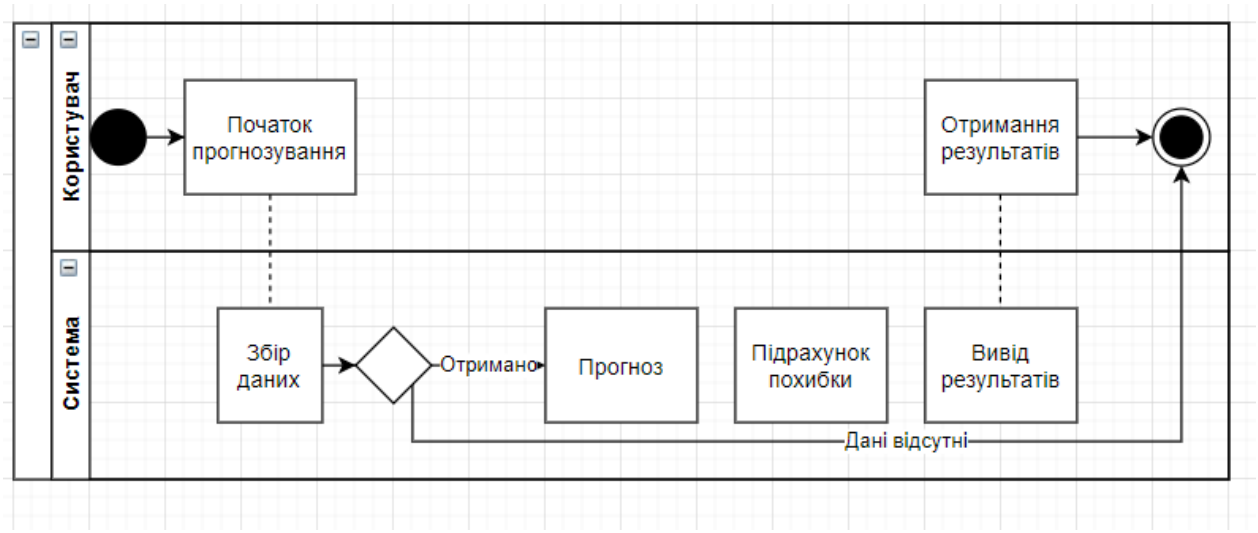


Рис. 3.2 — Діаграма діяльності інформаційної системи

3.3 Діаграма IDEF0

IDEF0, складена абревіатура ("Icam DEFinition for Function Modeling", де ICAM є абревіатурою від "Integrated Computer Aided Manufacturing"), є методологією моделювання функцій для опису виробничих функцій, яка пропонує мову функціонального моделювання для аналізу, розробки, реінжиніринг та інтеграція інформаційних систем, бізнес-процесів або аналіз програмної інженерії.

IDEF0 є частиною сімейства мов моделювання IDEF у сфері програмної інженерії та побудовано на мові функціонального моделювання Structured Analysis and Design Technique (SADT).

Метод функціонального моделювання IDEF0 призначений для моделювання рішень, дій та діяльності організації чи системи.[2] Він був похідним від усталеної мови графічного моделювання Structured Analysis and Design Technique (SADT), розробленої Дугласом Т. Россом і SofTech, Inc. У своїй оригінальній формі IDEF0 містить як визначення мови графічного моделювання (синтаксис і семантика), так і опис комплексної методології розробки моделей.[3]

ВВС США доручили розробникам SADT «розробити метод функціональної моделі для аналізу та передачі функціональної перспективи системи. IDEF0 має допомогти в організації системного аналізу та сприяти ефективному спілкуванню між аналітиком і замовником за допомогою спрощених графічних пристроїв».

На рисунку 3.3 зображено діаграму IDEF0, яка описує процес прогнозування ціни.

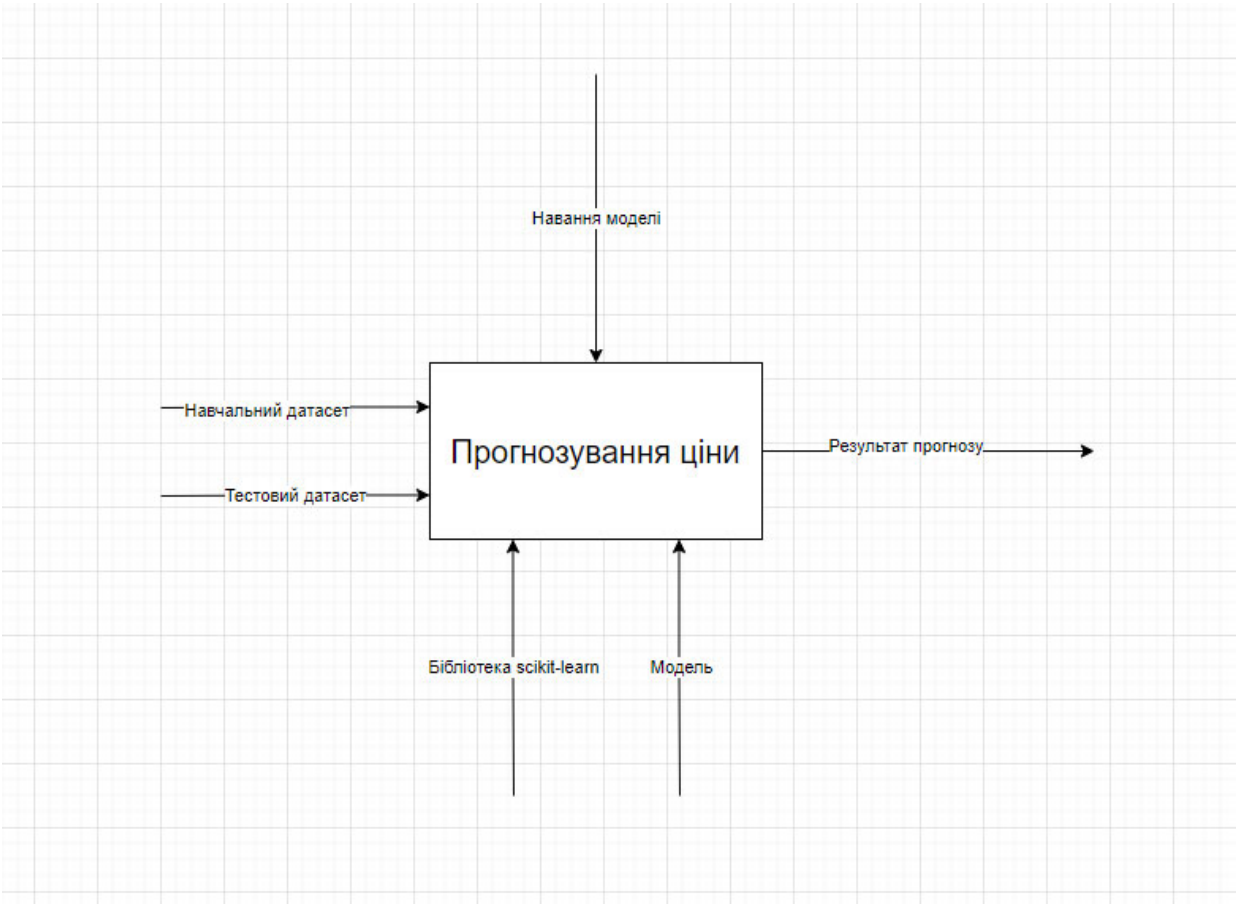


Рис. 3.3 — Діаграма IDEF0

3.4 Діаграма компонентів

В уніфікованій мові моделювання (UML) діаграма компонентів зображує, як компоненти з’єднуються разом, щоб утворити більші компоненти або програмні системи. Вони використовуються для ілюстрації структури як завгодно складних систем.

Діаграма компонентів дозволяє переконатися, що необхідна функціональність системи є прийнятною. Ці діаграми також використовуються як інструмент комунікації між розробником і зацікавленими сторонами системи. Програмісти та розробники використовують діаграми, щоб формалізувати дорожню карту впровадження, що дозволяє краще приймати рішення щодо призначення завдання або необхідних покращень. Системні адміністратори можуть використовувати діаграми компонентів для планування наперед, використовуючи уявлення про логічні програмні компоненти та їх взаємозв’язки в системі.

Діаграма компонентів розширює інформацію, наведену в елементі позначення компонента. Один із способів ілюстрації наданих і необхідних інтерфейсів зазначеним компонентом полягає у формі прямокутного відсіку, прикріпленого до компонентного елемента.[2] Іншим прийнятним способом представлення інтерфейсів є використання графічної конвенції «м'яч і гнізда». Надана залежність від компонента до інтерфейсу ілюструється суцільною лінією до компонента, що використовує інтерфейс із «льодяника» або кульки, позначеного назвою інтерфейсу. Необхідна залежність використання від компонента до інтерфейсу ілюструється півколом або сокетом, позначеним назвою інтерфейсу, прикріпленим суцільною лінією до компонента, якому потрібен цей інтерфейс. Успадковані інтерфейси можуть бути показані за допомогою льодяника, перед міткою імені символом каретки. Щоб

проілюструвати залежності між ними, використовуйте суцільну лінію із звичайним наконечником стрілки, що з’єднує розетку з льодяником.

На рисунку 3.4 зображена діаграма компонентів системи, яка показує взаємодію між компонентами системи та утворення з них цілісної системи.

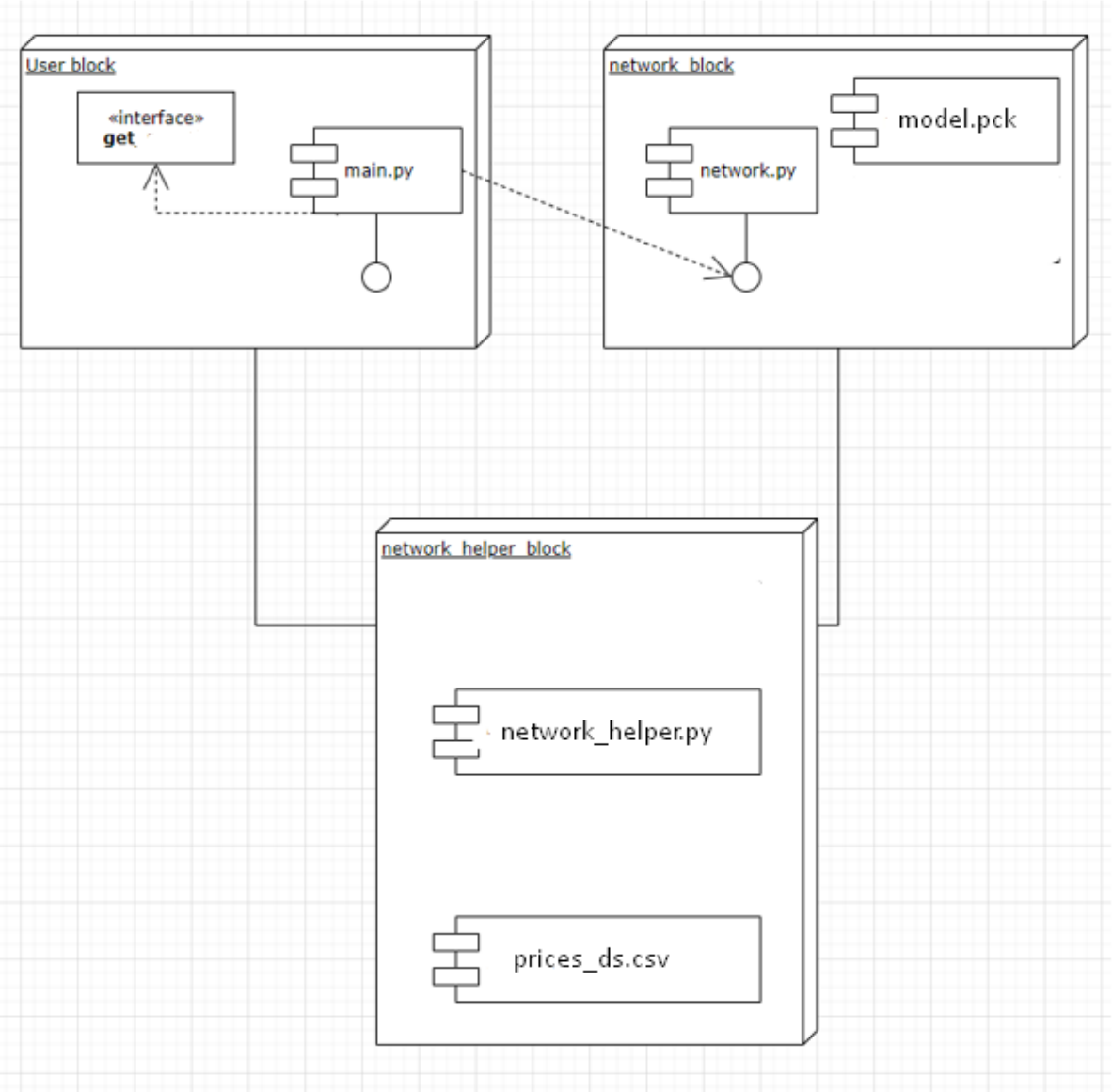


Рис. 3.4 — Діаграма компонентів системи

3.5 Розробка основних механізмів

Початок роботи над створенням нейронної мережі починається з розроблення і навчання моделі нейронної мережі. Кожна модель навчається на своєму окремому датасеті. Частина програмного коду даного етапу представлена нижче.

```
df = pd.read_csv("prices_month_ds.csv")
x = df.iloc[:, :4]
y = df.iloc[:, 4:5]
model_1 = LinearRegression()
model_1.fit(x.values, y.values)
pred(model_1, 'Non-weighted model')
sample_weight = np.ones(36) * 20
model_2 = LinearRegression()
model_2.fit(x.values, y.values, sample_weight)
pred(model_2, 'Weighted model')
model_3 = reduction(model_2, sample_weight)
pred(model_3, 'Reduced model')
do_charts_values(model_1, model_3)
do_charts_MSE(model_1, model_3)
```

Рис. 3.5 — Лістинг коду розробка моделі

Для створення і навчання моделей в даному випадку використовується бібліотека scikit-learn.

Після навчання і збереження моделі слід написати функціональну частину, яка буде посередником між інтерфейсом користувача і моделями.

Частина коду «посередника» представлена нижче.

```
Xp = [[26.6100, 26.5722, 26.4244, 26.3503],
      [27.9783, 28.0087, 28.0096, 28.0576],
      [26.6931, 26.6457, 26.6752, 26.6455],
      [26.9290, 26.8601, 26.8928, 27.0029],
      [26.9298, 26.8102, 26.7317, 26.7452]
      ]
y_pred = [26.3314, 28.0642, 26.6504, 27.0247, 26.7264]
predicts_unr = []
predicts_r = []
i = 0
for el in Xp:
    el = np.reshape(el, (1, -1))
    predicts_unr.append(model_unr.predict(el).tolist()[0][0])
```

Рис. 3.6 — Лістинг коду «посередника»

Посередник приймає в себе набори вхідних даних, розпаковує моделі і передає їм на обробку отримані вхідні дані, отримує від них результати і передає їх назад до інтерфейсу користувача.

Модуль інтерфейсу користувача за правилами проектування програмного забезпечення повинен відповідати лише за взаємодію між програмою і користувачем, без будь-яких функціональних навантажень. Через це модуль інтерфейсу спроектовано таким чином, щоб він збирав у користувача необхідні данні і передавав їх «посереднику», потім отримував результат і виводив його на екран. Частина коду блоку інтерфейсу представлена нижче.

```
def do_charts_MSE(model_unr, model_r):
    Xp = [[26.6100, 26.5722, 26.4244, 26.3503],
          [27.9783, 28.0087, 28.0096, 28.0576],
          [26.6931, 26.6457, 26.6752, 26.6455],
          [26.9290, 26.8601, 26.8928, 27.0029],
          [26.9298, 26.8102, 26.7317, 26.7452]
          ]
    y_pred = [26.3314, 28.0642, 26.6504, 27.0247, 26.7264]
    predicts_unr = []
    predicts_r = []
    i = 0
    j = 0

    for el in Xp:
        el = np.reshape(el, (1, -1))
        predicts_unr.append(mean_squared_error([y_pred[i]], model_unr.predict(el)))
        i += 1
    for elem in Xp:
```

Рис. 3.7 — Лістинг коду модуль інтерфейсу

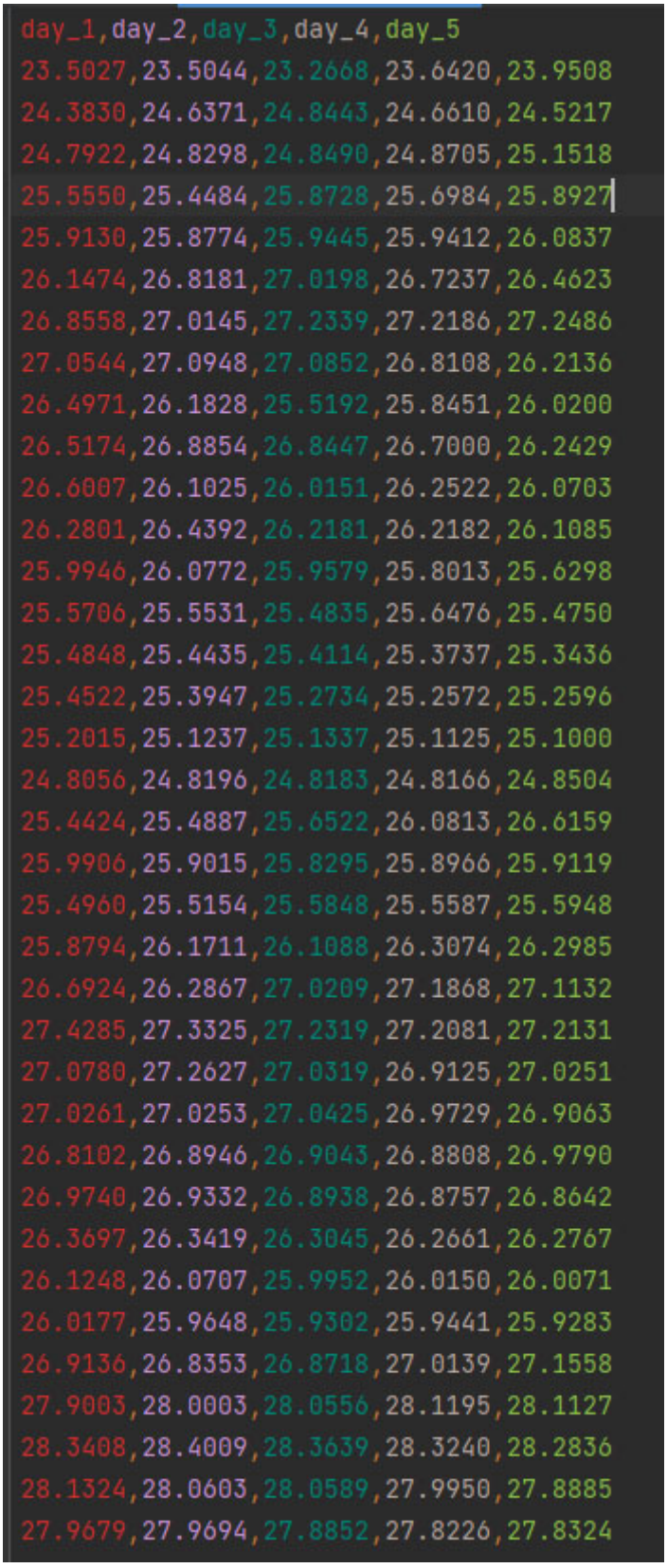
3.6 Створення навчальних вибірок

Набір даних (або датасет) — це набір даних. У разі табличних даних набір даних відповідає одній або кільком таблицям бази даних, де кожен стовпець таблиці представляє певну змінну, а кожен рядок відповідає заданому запису відповідного набору даних. Набір даних перераховує значення для кожної зі змінних, таких як висота і вага об’єкта, для кожного члена набору даних. Кожне значення відоме як дане. Набори даних також можуть складатися з колекції документів або файлів

У дисципліні відкритих даних набір даних — це одиниця вимірювання інформації, опублікованої в загальнодоступному сховищі відкритих даних. Європейський портал відкритих даних агрегує понад півмільйона наборів даних Деякі інші проблеми (джерела даних у реальному часі, нереляційні набори даних тощо) ускладнюють досягнення консенсусу щодо цього.

У випадку побудови нейронної мережі для навчання необхідно створювати великорозмірні датасети, на десятки тисяч прикладів даних. Такі обсяги датасетів пов’язані з прямою залежністю між розміром навчальної вибірки і точністю прогнозів навченої моделі.

Навчальний датасет зображено на рисунку 3.8



```
day_1,day_2,day_3,day_4,day_5
23.5027,23.5044,23.2668,23.6420,23.9508
24.3830,24.6371,24.8443,24.6610,24.5217
24.7922,24.8298,24.8490,24.8705,25.1518
25.5550,25.4484,25.8728,25.6984,25.8927
25.9130,25.8774,25.9445,25.9412,26.0837
26.1474,26.8181,27.0198,26.7237,26.4623
26.8558,27.0145,27.2339,27.2186,27.2486
27.0544,27.0948,27.0852,26.8108,26.2136
26.4971,26.1828,25.5192,25.8451,26.0200
26.5174,26.8854,26.8447,26.7000,26.2429
26.6007,26.1025,26.0151,26.2522,26.0703
26.2801,26.4392,26.2181,26.2182,26.1085
25.9946,26.0772,25.9579,25.8013,25.6298
25.5706,25.5531,25.4835,25.6476,25.4750
25.4848,25.4435,25.4114,25.3737,25.3436
25.4522,25.3947,25.2734,25.2572,25.2596
25.2015,25.1237,25.1337,25.1125,25.1000
24.8056,24.8196,24.8183,24.8166,24.8504
25.4424,25.4887,25.6522,26.0813,26.6159
25.9906,25.9015,25.8295,25.8966,25.9119
25.4960,25.5154,25.5848,25.5587,25.5948
25.8794,26.1711,26.1088,26.3074,26.2985
26.6924,26.2867,27.0209,27.1868,27.1132
27.4285,27.3325,27.2319,27.2081,27.2131
27.0780,27.2627,27.0319,26.9125,27.0251
27.0261,27.0253,27.0425,26.9729,26.9063
26.8102,26.8946,26.9043,26.8808,26.9790
26.9740,26.9332,26.8938,26.8757,26.8642
26.3697,26.3419,26.3045,26.2661,26.2767
26.1248,26.0707,25.9952,26.0150,26.0071
26.0177,25.9648,25.9302,25.9441,25.9283
26.9136,26.8353,26.8718,27.0139,27.1558
27.9003,28.0003,28.0556,28.1195,28.1127
28.3408,28.4009,28.3639,28.3240,28.2836
28.1324,28.0603,28.0589,27.9950,27.8885
27.9679,27.9694,27.8852,27.8226,27.8324
```

Рис. 3.8 — Приклад навчального датасету за прогнозом на місяць

3.7 Тестування

Для тестування розробленої системи було обрано метод чорного ящика. Тестування чорного ящика — це метод тестування програмного забезпечення, який перевіряє функціональність програми, не заглядаючи в її внутрішні структури чи роботи. Цей метод тестування можна застосувати практично на кожному рівні тестування програмного забезпечення: одиничному, інтеграційному, системному та приймальному. Його іноді називають тестуванням на основі специфікації.

Спеціальні знання коду програми, внутрішньої структури та знання програмування загалом не потрібні. Тестер знає, що програмне забезпечення має робити, але не знає, як воно це робить. Наприклад, тестувальник знає, що конкретні вхідні дані повертають певний незмінний вихід, але не знає, як програмне забезпечення виробляє вихідні дані в першу чергу.

Тестові випадки будуються на основі специфікацій та вимог, тобто того, що програма повинна робити. Тестові випадки, як правило, походять із зовнішніх описів програмного забезпечення, включаючи специфікації, вимоги та параметри конструкції. Хоча використовувані тести мають переважно функціональний характер, можуть використовуватися й нефункціональні тести. Конструктор тестів вибирає як дійсні, так і недійсні вхідні дані та визначає правильний вихід, часто за допомогою тестового оракула або попереднього результату, який, як відомо, є хорошим, без будь-якого знання внутрішньої структури тестового об’єкта.

Тестування починається з моделі прогнозування на рік, для даної моделі тест-кейси представлені в таблиці 3.1.

Таблиця 3.1 — Тест кейси для моделі прогнозування на рік

№ п.п.	Опис	Очікуваний результат	Отриманий результат
1	Введення даних, що вибиваються з ряду	Виведення результату, який не відповідає дійсності, проте є точним з огляду на введені данні	Виведення результату, який не відповідає дійсності, проте є точним з огляду на введені данні
2	Введення текстових даних замість числових	Відповідне повідомлення про помилку в роботі програми	Відповідне повідомлення про помилку в роботі програми
3	Введення занадто великого числа	Виведення результату, який не відповідає дійсності, проте є точним з огляду на введені данні	Виведення результату, який не відповідає дійсності, проте є точним з огляду на введені данні
4	Введення порожніх полів	Відповідне повідомлення про помилку в роботі програми	Відповідне повідомлення про помилку в роботі програми
5	Введення усіх полів великими числами	Виведення результату, який не відповідає дійсності, проте є точним з	Виведення результату, який не відповідає дійсності, проте є точним з

		огляду на введені данні	огляду на введені данні
6	Введення адекватних вхідних даних	Виведення прогнозу на найближчий рік	Виведення прогнозу на найближчий рік
7	Введення усіх полів порожніми	Відповідне повідомлення про помилку в роботі програми	Відповідне повідомлення про помилку в роботі програми

Тестування показує повну функціональну вірність, відсутність неточностей і багів в написаній системі.

РОЗДІЛ 4
ОХОРОНА ПРАЦІ ТА БЕЗПЕКА В НАДЗВИЧАЙНИХ СИТУАЦІЯХ

4.1 Охорона праці

Темою кваліфікаційної роботи є розробка програмного забезпечення для аналізу та прогнозування ринку валют. Дотримання основних вимог з охорони праці при роботі з ПК дозволяє безпечно використовувати програмні продукти. Тому необхідно розглянути санітарні норми та правила що стосуються приміщень та робочих місць де використовується ПК.

При розробці системи потрібно запобігти негативному впливу виробничих факторів, а саме дотримуватися правил та норм щодо приміщень де знаходитиметься робоче місце, відповідно до НПАОП 0.00-1.28-10 «Правила охорони праці під час експлуатації електронно-обчислювальних машин» та ДСанПіН 3.3.2.007-98 «Гігієнічні вимоги до організації роботи з візуальними дисплейними терміналами електронно-обчислювальних машин».

Відповідно до ДСанПіН 3.3.2.007-98 робоче місце працівника не можна розміщувати у підвалах та цокольних поверхах. Площа робочого місця для виконання розробки повинна становити не менше 6 м2, а об’єм 20 м3 . Приміщення, в якому працюють на ПК повинно бути забезпечене природнім та штучним освітлення згідно з нормами. Природне освітлення здійснюється крізь світлові отвори, направлені на північний схід і вони забезпечують коефіцієнт природньої освітленості (КПО) не менший ніж 1,5%. Розраховується КПО за методикою, що міститься у ДБН В.2.5-28-2006.

В приміщеннях де проводиться розробка та міститься ПК, штучне освітлення здійснюється системою загального рівномірного освітлення. Джерелом штучного освітлення слугують люмінесцентні лампи. Віконні отвори

приміщення обладнані вертикальними жалюзями та зовнішніми козирками. У даному приміщенні функціонує система опалення. Відповідно до санітарних норм та правил у даному приміщенні знаходиться аптечка першої медичної допомоги. Також, у даному приміщенні щоденно проводиться вологе прибирання.

Згідно правил НПАОП 0.00-1.28-10 обладнання та організація робочого місця працюючого з ПК має відповідати ергономічним вимогам ГОСТ 12.2.032-78, ГОСТ 22.269-76, ГОСТ 21.889-76, а саме:

оптимальна робоча поза користувача ПК забезпечується конструкцією робочого місця;

робоче місце розташоване так відносно світлових отворів, що природне світло падає збоку зліва;

відповідно вимогам, робочий стілець є підйомно-поворотним з регульованою висотою;

будова робочого стола відповідає вимогам ергономіки та дозволяє оптимально розміщувати на робочій поверхні ПК, допоміжне обладнання та документи.

Нормовані параметри мікроклімату, іонного складу повітря, вмісту шкідливих речовин мають відповідати вимогам СН 4088-86, СН 2152-80, ГОСТ 12.1.005-88, ГОСТ 12.1.007-76;

Мікрокліматичні умови приміщення повинні відповідати нормальним значенням таких показників:

температура повітря;

відносна вологість повітря;

швидкість руху повітря;

інтенсивність теплового (інфрачервоного) опромінення.За ступенем впливу на тепловий стан людини мікрокліматичної умови поділяють на оптимальні та допустимі.

Для робочої зони приміщення встановлюються оптимальні та допустимі мікрокліматичні умови з урахуванням складності виконуваної роботи та періоду

року. При виконанні роботи на ПК, що пов’язано з нервово-емоційним напруженням, у приміщенні потрібно підтримувати температура повітря +220С – +240С , відносну вологість 60-40%. Дані вимоги до параметрів мікроклімату містяться у санітарних нормах ДСН 3.3.6.042-99.

Оскільки розробка системи комплексної обробки подій здійснюються за ПК, необхідно проводити перерви по 15 хвилин через кожну годину. Якщо, виробничі обставини не дозволяють здійснювати часті перерви, тривалість роботи з ПК не повинна бути більшою чотирьох годин.

Відповідно до основних правил електробезпеки під час використання ПК потрібно дотримуватися таких вимог:

при використанні ліній електромережі необхідно запобігти виникненню електричного джерела займання внаслідок короткого замикання чи перевантаження мережі;

до електромережі ПК підключається тільки з а допомогою штепсельних з’єднань і електророзеток;

спеціальні контакти для підключення нульового захисного провідника повинні міститися у штепсельних з’єднаннях та електророзетках. При цьому, під’єднання нульового захисного провідника відбувається швидше, ніж під’єднання фазового та нульового робочого провідника;

підключення живлення ПК до двопровідної електромережі є недопустимим, навіть з використанням перехідних пристроїв. Тому здійснюється підключення по трьохпровідній мережі.

Згідно основних вимог до пожежної безпеки приміщення, у яких знаходиться ПК, мають бути оснащені переносними вуглекислотними або аерозольно-водопінними вогнегасниками. Підходи до засобів пожежогасіння повинні бути вільними. Також, згідно вимог НАПБ Б.06.004-2005 «Перелік однотипних за призначенням об'єктів, які підлягають обладнанню автоматичними установками пожежогасіння та пожежної сигналізації», у приміщенні де здійснюється робота з ПК, робочі місця повинні бути обладнанні системою автоматичної пожежної сигналізації з димовим пожежним сповіщувачем.

Таким чином, в даному підрозділі виконано огляд основних законодавчих актів та нормативів при роботі з ПК. Так як, дипломна робота магістра спрямована на розробку системи аналізу та прогнозування ринку валют, то було наведено основні стандарти і правила щодо влаштування робочих місць де використовують комп'ютерну техніку. Також було наведено вимоги, які стосуються приміщення де знаходиться робоче місце працівника. Крім того, розглянуті санітарні норми та правила щодо мікроклімату у даному приміщенні. Наведенні правила електробезпеки під час роботи з ПК. Також поданні основні вимоги до пожежної безпеки приміщення. При дотриманні даних правил гарантуються безпечні умови праці на ПК та запобігання шкідливих виробничих факторів.

4.2 Заходи щодо забезпечення безпеки

Ядерні вибухи, особливо в атмосфері й більш високих шарах, призводять до виникнення потужних електромагнітних полів з довжиною хвиль від 1 до 1000 м і більше. Ці поля через короткочасне існування називають електромагнітним імпульсом (EMI). EMI виникає при ядерному вибусі у воєнний час, у мирний час – при випробуванні ядерної зброї або ядерних аваріях і катастрофах в атмосфері й космосі.

Гамма-промені, які випускаються із зони вибуху в напрямі поверхні землі, поглинаються в більш щільних шарах атмосфери, вибиваючи з атомів повітря швидкі електрони, які летять у напрямку гамма-променів зі швидкістю світла, а позитивні іони (залишки атомів) залишаються на місці. У результаті поділу і переміщення позитивних і негативних зарядів у цій області й у зоні вибуху, а також при взаємодії зарядів з геомагнітним полем Землі утворюються елементарні й результуючі електричні та магнітні поля EMI, які досягають поверхні землі в зоні радіусом кількох сотень кілометрів. Виникають сильні

поперечні струми і утворюється подібність великої "плоскої антени", яка випромінює потужний ЕМІ з часом наростання порядна 10 нс і тривалістю більше 230 нс; зі смугою частот від 10 кГц до 100 МГц. Залежно від висоти ядерного вибуху за інших однакових умов змінюються характер, інтенсивність ЕМІ і дальність його поширення.

При наземному і низькому повітряному вибуху уражаюча дія ЕМІ спостерігається на відстані кількох кілометрів від центру вибуху. Під час ядерного вибуху на висотах від 3 до 25 км утворюється симетричне джерело генерації, але радіус поширення ЕМІ залишається обмеженим внаслідок сильного поглинання гамма-випромінювання в щільних шарах атмосфери.

Найбільшу уражаючу дію має ЕМІ, що виникає при екзоатмосферному вибуху (більше 40 км). Зі збільшенням висоти вибуху збільшується і район джерела генерації ЕМІ, досягаючи в діаметрі тисячі кілометрів і товщини 20 – 40 км. Так, під час вибуху на висоті 80 км ЕМІ буде поширюватися на площі радіусом 960 км, а під час вибуху на висоті 160 км – на площі радіусом 1400 км. Екзоатмосферний ЕМІ характеризується дуже малим часом наростання (декілька сот наносекунд), високою інтенсивністю електричного поля (більше 50 кВ/хв) і магнітного поля (близько 130 А/хв).

Розряд блискавки порівняно з ЕМІ має значно більшу тривалість зростання і спаду (5—300 мкс), створює дуже потужні поля (близько 100 кВ/хв), несе значно більшу енергію, але спектр частот становить близько 10 МГц, тоді як для ЕМІ він більше — 100 МГц. Пікове значення ЕМІ може досягти 50 000 Вт/хв, що дорівнює всій енергії яка випромінюється в радіочастотній частині спектра.

Ефективність і безпека роботи різноманітних науково-виробничих комплексів (НВК) багато в чому досягається завдяки застосуванню засобів інформатизації в різних сферах їх діяльності. Основну загрозу для інформаційних ресурсів до недавнього часу представляли хакерські атаки і впровадження комп'ютерних вірусів, які здійснюються програмним шляхом. Проте наявність генераторів потужних електромагнітних випромінювань, здатних негативно впливати на електронне устаткування, істотно змінило

пріоритети в області інформаційної безпеки. Сьогодні, в розряд першочергових, висувається проблема захисту від навмисного електромагнітного впливу.

Джерела електромагнітних імпульсів розробляються в ряді країн з метою досягнення якісно нового рівня радіолокації, радіозв'язку, технологій вирішення інших технічних завдань. Принцип їх роботи допускає генерацію і випромінювання в навколишній простір не лише одиничних електромагнітних сигналів, але і цілих пакетів. Параметри випромінювання таких пристроїв роблять їх дуже небезпечними при дії на мікроелектронні системи. Відносна простота виготовлення і доступність придбання таких генераторів, а також компактність цих приладів дозволяють розцінювати їх в якості потенційних засобів навмисного впливу, що дозволяють локально створювати ефекти, подібні до електромагнітних випромінювань ядерного вибуху.

Існує широка номенклатура генераторів, що формують електромагнітні імпульси, які призначені для перевірки стійкості електронного устаткування різних об'єктів до електромагнітного впливу. Характер порушень в роботі безпосередньо залежить від параметрів і рівня стійкості устаткування до цього впливу. Порушення в основному носять тимчасовий характер, проявляються під час впливу і зберігаються впродовж деякого періоду після цього впливу, причому виявити факт навмисного електромагнітного впливу, як в цей період, так і надалі є не можливим.

Електронна інфраструктура НВК, ставши об'єктом електромагнітної атаки, може зазнати ряд деструктивних змін, що, у свою чергу, приведе до збоїв в роботі електронного устаткування і далі – до функціональних порушень видів діяльності, що ним забезпечується.

Систематизований перелік можливих наслідків для засобів інформатизації, використовуваних в основних сферах діяльності, властивих об'єктам НВК, приведений в табл. 4.1.

Оскільки завади, що мають меншу енергію, виникають частіше ніж завади, що мають велику енергію, найбільш частою реакцією електронних систем на дію

електромагнітних завад буде не руйнування пристрою, а порушення його роботи або короткочасний збій в роботі з наступним відновленням порушеної функції.

Таблиця 4.1 – Деструктивні ефекти в електронному устаткуванні при навмисному електромагнітному впливі (НЕМВ)

Вид системи	**Вид ефекту в результаті дії**
Засоби телекомунікації	- зависання і перезавантаження комп'ютерів, - значне зниження інформаційного трафіку, - збільшення кількості помилок.
Засоби зв'язку і навігації	- зменшення ефективної дальності зв'язку (від 2 до 10 разів), - неправдиві свідчення, або збої в засобах навігації.
Засоби безпеки	- збої в системах контролю і управління доступом, - блокування охоронно-пожежної сигналізації, - мимовільне включення устаткування пожежогасіння, - спотворення зображень з камер відео нагляду.

Імпульсні перенапруження, що виникають при ядерних вибухах, розрядах блискавок і при комутації в силових електроустановках, здатні пошкодити, або зруйнувати як електронні прилади, так і цілі системи.

У нормах будівництва громовідводів приймають зазвичай струм блискавки до 200 тисяч ампер при тривалості близько 1 мс, хоча практично струм блискавки рідко перевищує 20-30 кА. Температура каналу при головному розряді може перевищувати 25000^{o} С. Довжина каналу блискавки може бути від 1 до 10 км, діаметр — декілька сантиметрів. При ударі блискавки в громовідвід електричний струм (імпульс у формі дзвона) поступає в землю і розтікається в ґрунті на всі боки до декількох десятків і навіть сотень метрів, причому через опір ґрунту цей струм створює на

ньому падіння напруги. Оскільки найбільший опір чинять шари ґрунту, що лежать поблизу місця входження струму в землю, то саме тут спостерігається найвища напруга. В міру віддалення від цієї точки опір проходженню струму зменшується, при цьому знижується і напруга.

Ще один шлях для проникнення завади від ЕМІ різної природи – протікання струмів по заземленому металевому корпусу і заземлених екранах кабелів, підключених до нього. Отже, забезпечити належний рівень захисту від електромагнітних завад електронної апаратури дуже непросто. Особливо складно це зробити на старих підстанціях, системи заземлення яких проектувалися для роботи з електромеханічним захистом, значно стійкішим до електромагнітних дій, ніж мікропроцесорні. А якщо врахувати, що небезпечні підйоми потенціалу в колах заземлення виникають також при аварійних коротких замиканнях в електричних мережах, то проблема стане ще складнішою.

В деяких випадках для запобігання такому підйому потенціалу в кола електронної апаратури контури заземлення силового устаткування і електронної апаратури роблять окремими. Проте на реально існуючих підстанціях виконати таке розділення нереально.

Таким чином тільки комплексне рішення проблеми дозволить уникнути впливу потужних електромагнітних завад. Це рішення повинне включати:

– використання мікропроцесорних реле захисту тільки на підстанціях, спроектованих і побудованих з урахуванням найсучасніших вимог до електромагнітної сумісності і розрахованих на експлуатацію високочутливої електронної апаратури;

– вдосконалення конструкції самих мікропроцесорних реле захисту;

– розміщення мікропроцесорних реле захисту в металевих шафах, спеціально призначених для захисту електронного устаткування і забезпечених фільтрами на усіх кабелях, що входять в шафу.

ВИСНОВКИ

Метою даної роботи була розробка та дослідження програмної системи для аналізу і прогнозування фондового та криптовалютного ринку.

Для виконання поставленої мети було виконано наступні завдання:

- Провести аналіз предметної області
  - Проаналізувати поняття нейронної мережі
  - Проаналізувати процес прогнозуванні даних з використанням нейронних мереж
  - Проаналізувати проблематику прогнозування даних з використанням нейронних мереж
  - Розглянути існуючі рішення
- Провести аналіз методів вирішення задачі
  - Розглянути парадигми навчання нейронних мереж
  - Розглянути алгоритми роботи нейронних мереж
  - Обрати алгоритм аналізу даних
- Провести огляд інформаційного забезпечення
  - Аналіз мови програмування
  - Аналіз середовища розробки
  - Аналіз додаткового інструментарію
- Реалізувати програмний засіб
  - Створити діаграму варіантів використання
  - Створити діаграму діяльності
  - Створити діаграму IDEF0
  - Створити діаграму компонентів
  - Розробити основні механізми
  - Описати створення навчальних датасетів
  - Провести тестування

Завдяки чіткому виконанню завдань, поставлених на початку роботи, в результаті виконання роботи було отримано повноцінну систему, що здатна виконувати закладений в неї функціонал і готова до використання в реальних умовах.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. McCulloch, Warren; Walter Pitts (1943). "A Logical Calculus of Ideas Immanent in Nervous Activity". Bulletin of Mathematical Biophysics. 5 (4): 115–133. doi:10.1007/BF02478259.
2. Kleene, S.C. (1956). "Representation of Events in Nerve Nets and Finite Automata". Annals of Mathematics Studies (34). Princeton University Press. pp. 3–41. Retrieved 17 June 2017.
3. Hebb, Donald (1949). The Organization of Behavior. New York: Wiley. ISBN 978-1-135-63190-1.
4. Farley, B.G.; W.A. Clark (1954). "Simulation of Self-Organizing Systems by Digital Computer". IRE Transactions on Information Theory. 4 (4): 76–84. doi:10.1109/TIT.1954.1057468.
5. Haykin (2008) Neural Networks and Learning Machines, 3rd edition
6. Rosenblatt, F. (1958). "The Perceptron: A Probabilistic Model For Information Storage And Organization in the Brain". Psychological Review. 65 (6): 386–408. CiteSeerX 10.1.1.588.3775. doi:10.1037/h0042519. PMID 13602029.
7. Werbos, P.J. (1975). Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences.
8. Rosenblatt, Frank (1957). "The Perceptron—a perceiving and recognizing automaton". Report 85-460-1. Cornell Aeronautical Laboratory.
9. Olazaran, Mikel (1996). "A Sociological Study of the Official History of the Perceptrons Controversy". Social Studies of Science. 26 (3): 611–659. doi:10.1177/030631296026003005. JSTOR 285702. S2CID 16786738.
10. Schmidhuber, J. (2015). "Deep Learning in Neural Networks: An Overview". Neural Networks. 61: 85–117. arXiv:1404.7828. doi:10.1016/j.neunet.2014.09.003. PMID 25462637. S2CID 11715509.

11. Ivakhnenko, A. G. (1973). Cybernetic Predicting Devices. CCM Information Corporation.
12. Ivakhnenko, A. G.; Grigor′evich Lapa, Valentin (1967). Cybernetics and forecasting techniques. American Elsevier Pub. Co.
13. Schmidhuber, Jürgen (2015). "Deep Learning". Scholarpedia. 10 (11): 85–117. Bibcode:2015SchpJ..1032832S. doi:10.4249/scholarpedia.32832.
14. Dreyfus, Stuart E. (1 September 1990). "Artificial neural networks, back propagation, and the Kelley-Bryson gradient procedure". Journal of Guidance, Control, and Dynamics. 13 (5): 926–928. Bibcode:1990JGCD...13..926D. doi:10.2514/3.25422. ISSN 0731-5090.
15. Mizutani, E.; Dreyfus, S.E.; Nishio, K. (2000). "On derivation of MLP backpropagation from the Kelley-Bryson optimal-control gradient formula and its application". Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks. IJCNN 2000. Neural Computing: New Challenges and Perspectives for the New Millennium. IEEE: 167–172 vol.2. doi:10.1109/ijcnn.2000.857892. ISBN 0-7695-0619-4. S2CID 351146.
16. Kelley, Henry J. (1960). "Gradient theory of optimal flight paths". ARS Journal. 30 (10): 947–954. doi:10.2514/8.5282.
17. "A gradient method for optimizing multi-stage allocation processes". Proceedings of the Harvard Univ. Symposium on digital computers and their applications. April 1961.
18. Minsky, Marvin; Papert, Seymour (1969). Perceptrons: An Introduction to Computational Geometry. MIT Press. ISBN 978-0-262-63022-1.
19. Linnainmaa, Seppo (1970). The representation of the cumulative rounding error of an algorithm as a Taylor expansion of the local rounding errors (Masters) (in Finnish). University of Helsinki. pp. 6–7.
20. Linnainmaa, Seppo (1976). "Taylor expansion of the accumulated rounding error". BIT Numerical Mathematics. 16 (2): 146–160. doi:10.1007/bf01931367. S2CID 122357351.

21. Dreyfus, Stuart (1973). "The computational solution of optimal control problems with time lag". IEEE Transactions on Automatic Control. 18 (4): 383–385. doi:10.1109/tac.1973.1100330.
22. Werbos, Paul (1982). "Applications of advances in nonlinear sensitivity analysis" (PDF). System modeling and optimization. Springer. pp. 762–770.
23. Mead, Carver A.; Ismail, Mohammed (8 May 1989). Analog VLSI Implementation of Neural Systems (PDF). The Kluwer International Series in Engineering and Computer Science. 80. Norwell, MA: Kluwer Academic Publishers. doi:10.1007/978-1-4613-1639-8. ISBN 978-1-4613-1639-8.
24. David E. Rumelhart, Geoffrey E. Hinton & Ronald J. Williams , "Learning representations by back-propagating errors ," Nature', 323, pages 533–536 1986.
25. J. Weng, N. Ahuja and T. S. Huang, "Cresceptron: a self-organizing neural network which grows adaptively," Proc. International Joint Conference on Neural Networks, Baltimore, Maryland, vol I, pp. 576–581, June 1992.
26. J. Weng, N. Ahuja and T. S. Huang, "Learning recognition and segmentation of 3-D objects from 2-D images," Proc. 4th International Conf. Computer Vision, Berlin, Germany, pp. 121–128, May 1993.
27. J. Weng, N. Ahuja and T. S. Huang, "Learning recognition and segmentation using the Cresceptron," International Journal of Computer Vision, vol. 25, no. 2, pp. 105–139, Nov. 1997.
28. J. Schmidhuber., "Learning complex, extended sequences using the principle of history compression," Neural Computation, 4, pp. 234–242, 1992.
29. Domingos, Pedro (22 September 2015). The Master Algorithm: How the Quest for the Ultimate Learning Machine Will Remake Our World. chapter 4: Basic Books. ISBN 978-0465065707.
30. Smolensky, P. (1986). "Information processing in dynamical systems: Foundations of harmony theory.". In D. E. Rumelhart; J. L. McClelland; PDP Research Group (eds.). Parallel Distributed Processing: Explorations in the Microstructure of Cognition. 1. pp. 194–281. ISBN 978-0-262-68053-0.

31. Ng, Andrew; Dean, Jeff (2012). "Building High-level Features Using Large Scale Unsupervised Learning". arXiv:1112.6209 [cs.LG].
32. Ian Goodfellow and Yoshua Bengio and Aaron Courville (2016). Deep Learning. MIT Press.
33. Cireşan, Dan Claudiu; Meier, Ueli; Gambardella, Luca Maria; Schmidhuber, Jürgen (21 September 2010). "Deep, Big, Simple Neural Nets for Handwritten Digit Recognition". Neural Computation. 22 (12): 3207–3220. arXiv:1003.0358. doi:10.1162/neco_a_00052. ISSN 0899-7667. PMID 20858131. S2CID 1918673.
34. Dominik Scherer, Andreas C. Müller, and Sven Behnke: "Evaluation of Pooling Operations in Convolutional Architectures for Object Recognition," In 20th International Conference Artificial Neural Networks (ICANN), pp. 92–101, 2010. doi:10.1007/978-3-642-15825-4_10.
35. 2012 Kurzweil AI Interview Archived 31 August 2018 at the Wayback Machine with Jürgen Schmidhuber on the eight competitions won by his Deep Learning team 2009–2012
36. "How bio-inspired deep learning keeps winning competitions | KurzweilAI". www.kurzweilai.net. Archived from the original on 31 August 2018. Retrieved 16 June 2017.
37. Graves, Alex; and Schmidhuber, Jürgen; Offline Handwriting Recognition with Multidimensional Recurrent Neural Networks, in Bengio, Yoshua; Schuurmans, Dale; Lafferty, John; Williams, Chris K. I.; and Culotta, Aron (eds.), Advances in Neural Information Processing Systems 22 (NIPS'22), 7–10 December 2009, Vancouver, BC, Neural Information Processing Systems (NIPS) Foundation, 2009, pp. 545–552.
38. Graves, A.; Liwicki, M.; Fernandez, S.; Bertolami, R.; Bunke, H.; Schmidhuber, J. (2009). "A Novel Connectionist System for Improved Unconstrained Handwriting Recognition" (PDF). IEEE Transactions on Pattern Analysis and Machine Intelligence. 31 (5): 855–868. CiteSeerX

10.1.1.139.4502. doi:10.1109/tpami.2008.137. PMID 19299860. S2CID 14635907.

39. Graves, Alex; Schmidhuber, Jürgen (2009). Bengio, Yoshua; Schuurmans, Dale; Lafferty, John; Williams, Chris editor-K. I.; Culotta, Aron (eds.). "Offline Handwriting Recognition with Multidimensional Recurrent Neural Networks". Neural Information Processing Systems (NIPS) Foundation. Curran Associates, Inc: 545–552.
40. Graves, A.; Liwicki, M.; Fernández, S.; Bertolami, R.; Bunke, H.; Schmidhuber, J. (May 2009). "A Novel Connectionist System for Unconstrained Handwriting Recognition". IEEE Transactions on Pattern Analysis and Machine Intelligence. 31 (5): 855–868. CiteSeerX 10.1.1.139.4502. doi:10.1109/tpami.2008.137. ISSN 0162-8828. PMID 19299860. S2CID 14635907.

ДОДАТКИ