МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Методи штучного інтелекту в кібербезпеці

*Рекомендовано Методичною радою КПІ ім. Ігоря Сікорського
як навчальний посібник для здобувачів ступеня доктор філософії за освітньою програмою «Кібербезпека»
спеціальності 125 «Кібербезпека»*

Київ
КПІ ім. Ігоря Сікорського
2022

Назва: Методи штучного інтелекту в кібербезпеці [Електронний ресурс] : навч. посіб. для здобувачів спец. 125 «Кібербезпека» / КПІ ім. Ігоря Сікорського ; уклад.: І.В. Стьопочкіна, О.М. Новіков. – Електронні текстові дані (1 файл: 19,9 Мбайт). – Київ : КПІ ім. Ігоря Сікорського, 2022 – 82 с.

Гриф надано Методичною радою КПІ ім. Ігоря Сікорського (протокол № 4 від 07.04.2022 р.)
за поданням Вченої ради Фізико-технічного інституту (протокол № 21 від 20.12.2021 р.)

Електронне мережне навчальне видання

Методи штучного інтелекту в кібербезпеці

Укладачі: *Новіков Олексій Миколайович,* доктор техн. наук, професор
Стьопочкіна Ірина Валеріївна, канд. техн. наук

Відповідальний редактор *Смирнов Сергій Анатолійович*, к.ф.-м.н., с.н.с.

Рецензент *Родіонов Андрій Миколайович, к.т.н., доцент кафедри інформаційної безпеки*

Навчальний посібник укладений для опанування тем дисципліни “Методи штучного інтелекту в кібербезпеці”. Посібник укладено згідно змістовного наповнення дисципліни. У складі посібника наведено огляд методів штучного інтелекту та сферу їх застосувань в кібербезпеці. Наведено теоретичні відомості щодо основних методів, основні особливості їх застосування, також надо контрольні запитання до теми для закріплення матеріалу. Матеріал посібника може бути використаний для одержання науково-практичних орієнтирів при здійсненні дисертаційного дослідження.

ЗМІСТ

ВСТУП

Методи штучного інтелекту - це методи, які дають можливість імітувати людську поведінку для виконання певних завдань і можуть поступово «навчатися», використовуючи отриману інформацію.

Наприклад, чат-боти використовують штучний інтелект, щоби видаляти шкідливих користувачів; «розумні помічники» використовують штучний інтелект для формування висновку по небезпечності ситуації на основі аналізу ряду показників; механізми рекомендацій автоматично підбирають користувачам дані, на основі переглянутих раніше (на основі створення віртуального профілю користувача та поведінкового профілю).

Очевидно, що в складі методів штучного інтелекту для забезпечення здатності до самонавчання є методи машинного навчання, а для забезпечення визначених алгоритмів дій, які функціонують за визначеними правилами, широко використовуються також елементи методів математичної логіки.

Наразі в кібернетичній безпеці широко використовуються методи штучного інтелекту в складі SIEM систем, фреймворків безпеки, які дозволяють аналізувати ситуацію та видавати користувачеві поради, як діяти далі. Елементарним прикладом можуть бути онлайнові продукти, які за посиланням надають користувачу висновок, чи є посилання шкідливим, чи безпечним. Штучний інтелект може використовуватись і в методах offensive security для того, щоби імітувати голос цільового користувача або написання листів відповідно до його стилю висловлювань, зокрема для здійснення автоматизованого тестування на проникнення.

Метою даного посібника є надання огляду найпопулярніших застосувань методів штучного інтелекту в задачах кібербезпеки та уявлення про сферу використання відповідних моделей та методів, їхні особливості, переваги та недоліки.

Матеріал посібника ілюстрований результатами, які були одержані у дисертаційних та інших наукових дослідженнях, що виконувались під керівництвом укладачів посібника у 2015-2020 р. в Навчально-науковому Фізико-технічному інституті. Разом із теоретичним матеріалом надаються контрольні запитання для кращого опанування матеріалу.

1 КЛАСИ ЗАДАЧ, ЯКІ РОЗВ'ЯЗУЮТЬСЯ МЕТОДАМИ ШТУЧНОГО ІНТЕЛЕКТУ

1.1 Застосування методів в кібербезпеці

Розглянемо види задач, які можна розв'язувати методами штучного інтелекту [1,2]:

- Регресія (або прогноз) - завдання прогнозування наступного значення на основі попередніх знань.
- Класифікація - завдання розділення явищ на різні категорії. Еталони що відповідають класам, відомі.
- Кластеризація - класи невідомі, групування за схожістю.
- Правила пошуку асоціацій (або рекомендацій) - завдання рекомендувати щось на основі попереднього досвіду.
- Зменшення розмірності - узагальнення, завдання пошуку інших і найбільш важливих ознак у ряді зразків.
- Генеративні моделі - завдання створення чогось на основі попередніх знань про розподіл.

Класи машинного навчання:

- Supervised learning (навчання з учителем) – умова – розмічені дані, із вказівкою наприклад, шкідливий файл чи ні. правильна конструкція чи ні. На основі розмічених даних приймається рішення про нові дані.
- Ensemble learning - розширення supervised learning, змішення простих моделей для одержання більш ефективної складної.
- Навчання без учителя – підхід, який використовується коли нема розмічених даних, і модель повинна розмітити їх самостійно, засновуючись на певних властивостях. Призначається для пошуку аномалій, менш точний ніж контрольовані підходи.
- Навчання з підкріпленням (reinforced learning) – підхід, орієнтований на зворотний зв'язок від середовища, коли поведінка моделі має реагувати на зміни середовища.
- Active learning – підклас навчання із підкріпленням, коли «учитель» допомагає виправляти поведінку на додачу до зміни середовища.

Методи, які використовуються при розв'язанні задач класифікації:

- LogisticRegression (LR);
- K-найближчих сусідів (K-NN);
- Метод опорних векторів (SVM);
- NaiveBayes;
- Decision Tree Classification;

- Random Forest;
- Глибинне навчання (нейронні мережі).

Типові задачі класифікації в контексті кібербезпеки:

- Робота спам-фільтрів;
- Робота засобів антивірусного захисту;
- Визначення джерел трафіку, типу пристроїв для задання політик;
- Класифікація на «аномальні» та «звичайні» події при виявленні атак.

Методи, які використовуються при розв'язання задач кластеризації:

- K-середніх (K-means);
- Latent Dirichlet allocation (LDA) – natural language processing в задачах кібербезпеки;
- Density Based Clustering.

Типові задачі кластеризації в контексті кібербезпеки:

- Захист від шкідливого програмного забезпечення;
- Безпечна електронна пошта;
- Розділення файлів за контентом;
- Кластеризація користувачів;
- Кластеризація текстів для безпеки соціальних мереж.

Методи, які використовуються для навчання правилам асоціації:

- Apriori;
- Eclat;
- Frequent Pattern-Growth;
- Машина Больцмана з глибинними обмеженнями (Restricted Boltzmann Machine;
- Мережа глибинної довіри (DBN);
- Автоенкодери.

Застосунки:

- Управління ризиками.

Якщо компанія стикається з хвилею інцидентів і пропонує різні типи відповідей, система дізнається тип відповіді на конкретний інцидент (наприклад, позначити його як помилкове спрацьовування, змінити значення ризику, запустити розслідування) .

В задачах штучного інтелекту часто треба виконувати зменшення розмірності моделі машинного навчання. Це виконується за допомогою методів:

- Метод головних компонентів (PCA) (метод факторного аналізу).
- Сингулярне розкладення (SVD) (використовується в LSA і як складова PCA)

- Т-розподілене стохастичне вкладення сусідів (T-SNE) (для візуалізації даних великої розмірності в 2D, 3D просторі).
- Лінійний дискримінантний аналіз (LDA) (пошук лінійної комбінації ознак, яка розділяє різні класи. Має зв'язок із регресійним аналізом та PCA, працює за умови, коли вимірювання, виконані на незалежних змінних для кожного спостереження є неперервними. Групи відомі) .
- Прихований семантичний аналіз (LSA) (аналіз спрямованості та змісту текстів).
- Факторний аналіз (ФА) (методи: метод головних компонент, кореляційний аналіз, метод максимальної правдоподібності. Нечіткий факторний аналіз). Дослідження факторів, які впливають на зміну захищеності системи тощо.
- Незалежний компонентний аналіз (ICA) (в задачах перехоплення витоку технічними каналами, виділення корисних сигналів визначеного виду серед інших, напр. прослуховування в шумній кімнаті).
- Невід'ємна матрична факторизація (NMF) (обробка аудио сигналів, кластеризація, комп'ютерний зір).

Генеративні моделі представляють свобою окремий клас моделей штучного інтелекту, який призначений для імітації фактичних даних на основі попередніх рішень та даних. До подібних моделей можна віднести:

- Марківські ланцюги.
- Генетичні алгоритми.
- Моделі глибинного навчання
  - GAN (імітація відео (DeepFake), прикладів фаззінга),
  - LSTM, GRU.
- Варіаційні автоенкодери (генерація зображень).
- Машини Больцмана (розв'язання оптимізаційних задач, аналіз відеозображень, розпізнавання людської поведінки).

Такі моделі використовуються для генерації текстів (наприклад, текстів листів соціальної інженерії для автоматизованого тестування на проникнення, фото-відео зображень, голосу для тестування систем біометричної автентифікації; списків вхідних параметрів для тестування застосунку на наявність вразливостей).

1.2 Регресійні моделі

Метою регресії є прогнозування значення однієї або декількох безперервних цільових змінних t, враховуючи значення D-вимірного вектора x вхідних змінних (лінійна регресія) [3-5].

Поліном є конкретним прикладом широкого класу функцій, званих моделями лінійної регресії, які поділяють властивість бути лінійними функціями регульованих параметрів

Найпростішою формою моделей лінійної регресії є також лінійні функції вхідних змінних.

Постановка задачі регресії

Дано навчальний набір даних, що включає N спостережень {xn}, де n = 1,. . . , N, разом із відповідними значеннями {tn}, метою є передбачення значення t для нового значення x.

У найпростішому підході це можна зробити безпосередньо побудувавши відповідну функцію y (x), значення якої для нових входів x становлять прогнози для відповідних значень t.

Більш загально, з імовірнісної точки зору, ми прагнемо змоделювати прогнозний розподіл p (t | x), оскільки це виражає нашу невизначеність щодо значення t для кожного значення x. З цього умовного розподілу ми можемо робити прогнози t для будь-якого нового значення x таким чином, щоб мінімізувати очікуване значення відповідно обраної функції втрат.

Модель регресії можна представити як

$$y(x, w) = w_o + w_1 x_1 + \cdots + w_D x_D, \qquad (1.1)$$
$$x = (x_1, \ldots, x_D)^T.$$

Також можна представити клас моделей виду:

$$y(x, w) = w_o + \sum_{j=1}^{M-1} w_j \, \phi_j(x), \qquad (1.2)$$

де $\phi_j(x)$ – це базисні функції. Параметр w_0 дозволяє виконати «зсув» у даних, і інколи називається «bias offset». Для знаходження параметрів w_0... w_D використовується принцип максимальної правдоподібності або метод наменших квадратів.

Приклади базисних функцій показані на рис. 1.1.

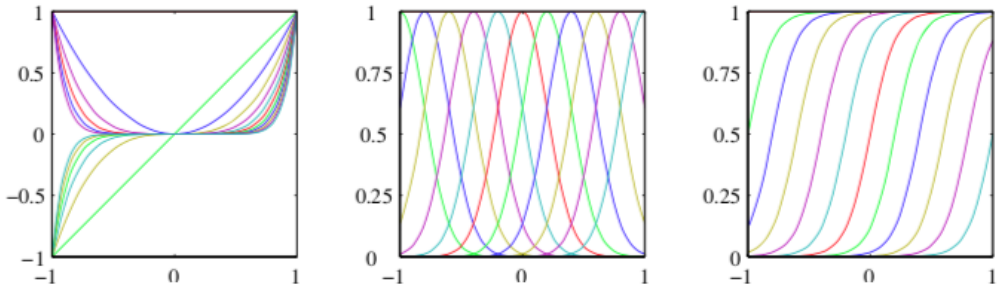


Рис. 1.1. Поліноміальні, гауссівські та сигмоїдальні функції.

Приклади використання

- Прогнози;
- Аналітика дієвості політики безпеки;
- Аналітика ризиків;
- Задачі класифікації.

Приклад1 . Застосунок для прогнозу частоти атак певного виду (DDoS) (рис.1.2, табл.1.1).

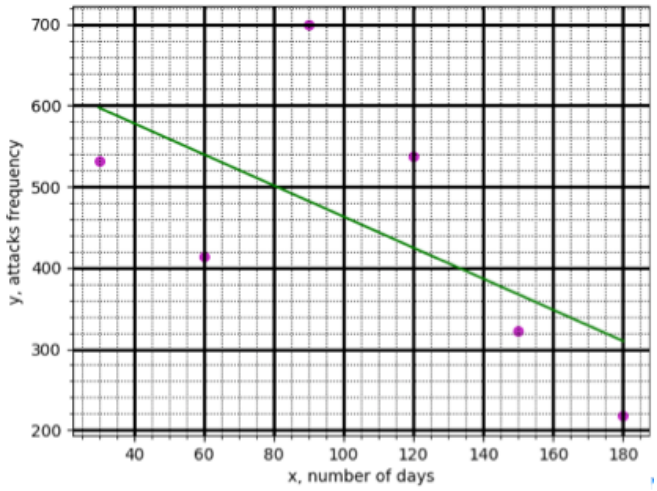


Рис. 1.2. Регресійна пряма і вихідні дані [6]

Таблиця 1.1. Вихідні дані для побудови регресійної моделі

Час(дні)	30	60	90	120	150	180
К-сть атак DDoS	532	414	699	538	322	217

Приклад 2. Застосунок множинної регресії для оцінки CVSS.

Нехай Y - оцінка CVSS, залежна змінна. CVSS прогнозується, виходячи з середовища та характеристик IIoT-пристрою.

Факторами є X_1 – кількість уразливостей, а саме загальна кількість уразливостей, виявлених статичними та динамічними інструментами виявлення вразливостей (напр. для IIoT-пристрою), X_2 – середній мережевий трафік вхідного сигналу, записаний протягом тижня атаки в MBPS(мегабайт/секунду).

Коефіцієнти регресійної моделі одержано методом градієнтного спуску.

$$CVSS = -0.20121785 + 0.10074424 \cdot X_1 + 0.23957254 \cdot X_2.$$

Тут X_1- кількість вразливостей на IIoT-пристрої ; X_2- пропонований середній вхідний мережевий трафік IIoT-пристрою / тиждень (в MBPS) .

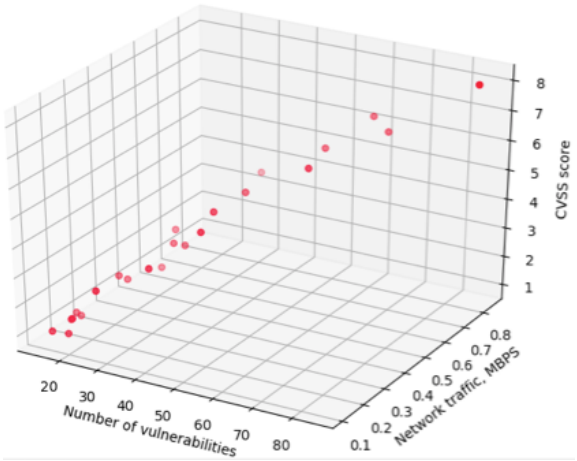


Рис. 1.3. Оцінка CVSS в залежності від факторів [5]

1.3 Використання лінійних моделей для класифікації

Мета класифікації: віднести вхідний вектор x до одного з K дискретних класів C_k, де $k = 1, \ldots, K$.

Передумова: класи приймаються як несумісні, так що кожен вхід призначається одному і тільки одному класу. Таким чином, вхідний простір поділяється на області прийняття рішень, межі яких називаються межами рішень або поверхнями прийняття рішень.

Розглядаємо лінійні моделі для класифікації- поверхні рішення є лінійними функціями вхідного вектора x і, отже, визначаються $(D - 1)$ -

вимірними гіперплощинами в D-мірному вхідному просторі. Набори даних, класи яких можна точно розділити лінійними поверхнями рішення, називаються лінійно відокремлюваними.

Для проблем регресії цільовою змінною t був просто вектор реальних чисел, значення яких ми хочемо передбачити.

1.4 Запитання для самоперевірки

1. В чому полягає відмінність між класифікацією та кластеризацією?
2. Які алгоритми використовують для класифікації?
3. Які методи використовують для кластеризації?
4. В чому полягають особливості генеративних моделей та сфера їх застосування?
5. Для якої мети використовують методи зменшення розмірності, назвіть приклади?
6. Які типові задачі кібербезпеки можна розв’язувати із використанням методів машинного навчання?
7. Якими є переваги та недоліки регресійних моделей?
8. В чому полягає відмінність логістичної регресії від лінійної? Які задачі можна розв'язувати із використанням регресії?
9. Що таке дискримінантні функції? Яким чином проводиться їх вибір?

2 ПРИНЦИПИ ДІЇ ГЛИБИННОГО НАВЧАННЯ. ТЕОРЕТИЧНІ ЗАСАДИ КОНСТРУЮВАННЯ МЕРЕЖ

2.1 Базові можливості машинного навчання

Асоціативна пам’ять

Будемо казати, що система має асоціативну пам’ять, якщо при подачі на її вхід деякої картини вона автоматично відбирає та подає на вихід найбільш близьку до неї картину, що знаходилась в пам’яті. Іншими словами, за достатньо великим фрагментом чи зіпсованим зображенням така система може відновити повне зображення. Дія асоціативної пам’яті є частковим випадком розпізнавання образів.

Один із можливих шляхів аналогової реалізації асоціативної пам’яті полягає у тому, щоб побудувати розподілену динамічну систему, або мережу дискретних елементів, атракторами якої у її фазовому просторі будуть прообрази, які потрібно запам’ятати.

Будемо називати інформацію, яку необхідно запам’ятати, прообразом; а інформацію, що зберігається у системі з асоціативною пам’яттю, картиною або образом для даного прообраза.

Кожна така картина, якій відповідає деякий атрактор системи, має свою область притягування, і будь-яка початкова умова, яка представляє собою якусь допустиму картину, подану на вхід системи, з еволюцією системи повинна попасти в одну з її областей притягування. Таким чином, з ходом часу, в процесі еволюції системи ця початкова структура трансформується у найбільш близький атрактор, який зберігається в пам’яті, а точніше, у атрактор, до області протягування якого належить така структура.

Особливості такого запам’ятовування:

- можливість розпізнавання прообразів (чи то класифікація прообразів);
- паралельність.

Основна задача, яка постає при створенні асоціативної пам’яті – задання такої системи, яка б мала атрактори, які відповідали би прообразам, які ми хочемо запам’ятати. Один із розв’язків цієї задачі було запропоновано Дж. Хопфілдом. Ним була розроблена модель динамічної системи, яка забезпечує всі умови для асоціативного запам’ятовування. Така модель базується на особливій фізичній системі – так званому «спіновому склі».

Модель Хопфілда.

Загальні правила функціонування системи

Розглянемо систему, яка складається з N елементів. Кожен із цих елементів називається «ізингівським спіном», який може знаходитись в одному із двох станів S_i:

$$S_i = \pm 1,\ i = 1..N. \tag{2.1}$$

Енергія E взаємодіючих спінів визначається як

$$E = -\frac{1}{2}\sum_{i,j} J_{ij} S_i S_j,\ t \to \infty, \tag{2.2}$$

де J_{ij} - елементи декої матриці взаємодії (вагові коефіцієнти зв'язків).

Нехай система змінюється таким чином, що при зміні будь-якого $S_i, i = 1..N$ загальна енергія системи зменшується.

Тобто, в процесі еволюції система намагається понизити свою повну енергію, а в граничному випадку при $t \to \infty$ система приходить до стану із мінімумом повної енергії:

$$E \to \min.$$

Якщо $\forall i, j\ J_{ij} > 0$, то в стані з мінімумом повної енергії всі пари спінів мають однаковий напрямок $S_i = 1$ або $S_i = -1$. При цьому енергія взаємодії будь-якої пари спінів визначається як

$$\delta E_{ij} = -J_{ij} S_i S_j \tag{2.3}$$

і досягає свого мінімального значення $\delta E_{ij} = -J_{ij}$.

Стан системи, в якому всі спіни спрямовані за одним напрямком (тобто стани елементів мають один знак), називається *феромагнітним*.

У спінових стеклах матриця $J = \{J_{ij}\}$ складається з випадкових елементів, які приймають як додатні, так і від'ємні значення. В стані із мінімумом повної енергії E для спінового скла енергії всіх пар спінів, як правило, неможуть бути одночасно мінімальними.

Розглянемо такі трійки елементів S_i, S_j, S_k, що добуток

$$J_{ij} J_{jk} J_{ki} < 0.$$

Але тоді як би ми не змінювали значення елементів S_i, S_j, S_k, нам не вдасться мінімізувати одночасно енергії взаємодії пар $\delta E_{ij}, \delta E_{jk}, \delta E_{ki}$ всі одразу.

Система, в якій умови мінімальності енергії взаємодії для різних пар спінів несумісні , називається *фрустрованою*.

Внаслідок фрустрації спінове скло має багато станів із мінімумами повної енергії, які відповідають різним спіновим конфігураціям. Ця властивість дає можливість співставляти кожному з таких мінімумів деякий прообраз.

Для великих N всі мінімуми системи характеризуються приблизно однаковою енергією E. Тобто при заданих вагових коефіцієнтах J_{ij} спінове скло може зберігати у пам'яті велику кількість просторових картин. При випадковому виборі J_{ij} характер цих картин також випадковий.

Формування матриці зв'язків.

Наступна задача полягає у цілеспрямованому виборі коефіцієнтів J_{ij} таким чином, щоб стійкими виявилися не випадкові, а задані картини. Ці картини будуть зберігатися у пам'яті і відігравати роль «еталонних» образів.

Нехай потрібно записати образ, який задається визначеною сукупністю станів

$$\{S_i = \xi_i\},\ \xi_i = \pm 1.$$

Для цього нам потрібно обрати вагові коефіцієнти J_{ij} так, щоб $E \to \min$. Це відбувається при $J_{ij} = \xi_i \xi_j$. При цьому для будь-якої пари спінів їхня енергія взаємодії визначається як

$$\delta E_{ij} = -J_{ij}\xi_i\xi_j = -\xi_i^2\xi_j^2 = -1$$

тобто досягає найменшого можливого значення. Відповідно, загальна енергія при цьому буде також мінімальною.

Недолік такої системи полягає в тому, що вона може зберігати лише один образ.

Для того, щоб зберігати M образів, коефіцієнти J_{ij} повинні задаватися правилом Хебба.

Правило Хебба

Для того, щоб записати в асоціативну пам'ять на основі спінової системи M прообразів, кожен з яких задається набором значень $\{\xi_j^m\}, m = 1,...,M$, потрібно обирати вагові коефіцієнти J_{ij} у вигляді

$$J_{ij} = \sum_{m=1}^{M} \xi_i^m \xi_j^m \ . \tag{2.4}$$

Причому різні набори повинні бути ортогональними:

$$\frac{\sum_{j=1}^{N} \xi_j^{m_1} \xi_j^{m_2}}{N} = \delta_{m_1 m_2} = \begin{cases} 1, m_1 = m_2; \\ 0, m_1 \neq m_2. \end{cases} \tag{2.5}$$

Тоді навіть при достатньо великій кількості M прообразів, які потрібно запам'ятати, можна гарантувати, що вони будуть стійкими конфігураціями спінового скла, які відповідають мінімумам енергії. Дійсно, повна енергія системи

$$E = -\frac{1}{2}\sum_{i=1}^{N} f_i S_i ,$$

де $f_i = \sum_j J_{ij} S_j$ - поле, яке діє на спін S_j. В стані з мінімумом енергії всі спіни повнні бути спрямовані за полем, тобто повинно виконуватись

$$sign S_i = sign f_i . \qquad (2.6)$$

Нехай коефіцієнти зв'язку задані правилом Хеба (2.4), а напрямки спінів відповідають одній із записаних картин $\{\xi_i^m\}$. Перевіримо виконання умови (2.6) із урахуванням умови ортогональності (2.5):

$$f_i S_i = \sum_j (\sum_{m=1}^{M} \xi_i^{m_1} \xi_j^{m_1}) \xi_i^{m_2} \xi_j^{m_2} = \sum_j (\xi_i^1 \xi_j^1 + \xi_i^2 \xi_j^2 + ... + \xi_i^{m_2} \xi_j^{m_2} + ...) \xi_i^{m_2} \xi_j^{m_2} =$$

$$= N \xi_i^{m_2} \xi_i^{m_2} = N > 0,$$

умову виконано.

Вимога (2.5) є досить жорсткою. Покажемо, що при великій кількості станів N випадково взяті образи зазвичай будуть майже ортогональними, і цієї наближеної ортогональності достатньо для успішної роботи асоціативної пам'яті. Нехай ми взяли дві випадкові послідовності $\{\xi_i^1\}$ та $\{\xi_i^2\}$, які складаються з ± 1. Кожен елемент у цих послідовностях обирається незалежно від інших, а значення ± 1 є рівноімовірними. Для таких образів кожна із величин $n_i = \xi_i^1 \xi_i^2$ є випадковою величиною із двома рівно імовірними значеннями $n_i = \pm 1$, а вираз

$$\sum_{j=1}^{N} \xi_j^{m_1} \xi_j^{m_2} ,$$

який входить до складу умови ортогональності, є сумою N незалежних випадкових величин n_i. Тому дисперсія випадкової величини

$$\sum_{j=1}^{N} \xi_j^{m_1} \xi_j^{m_2} = N_{12}$$

зростає як

$$< N_{12}^2 >= N .$$

Тоді вираз, що знаходиться у лівій частині умови ортогональності (2.5) запишеться як

$$\frac{\sqrt{< N_{12}^2 >}}{N} \sim \frac{1}{\sqrt{N}} \xrightarrow[N\to\infty]{} 0.$$

Тобто при кількості спінів у системі $N \to \infty$ ортогональність виконується автоматично. Але експериментально встановлено, що для того, щоб похибки, визвані неточним виконанням ортогональності, не заважали суттєво розпізнаванню образів, повинні виконуватись певні співвідношення між N та кількістю прообразів, які потрібно зберегти у пам'яті:

1. Приблизно при $M/N \sim 10^{-2}$ з'являється небезпека неправильного спрацювання при розпізнаванні образу;
2. Приблизно при $M/N \sim 10^{-1}$ записані картини перестають відповідати мінімумам енергії системи.

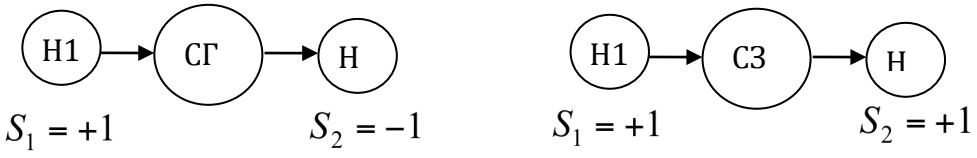
Застосування моделі Хопфілда до нейронних мереж

Нейронна мережа складається із нейронів, які пов'язані між собою чутливими каналами – аксонами та дендритами, причому дендрити кожного нейрона лише приймають імпульси, а аксони лише передають. Взаємодія даного нейрона за допомогою аксонів та дендритів відбувається не прямим під'єднанням до інших нейронів, а через спеціальні області взаємодії – синапси. Синапси бувають збуджуючі та гальмуючі. Збуджуючий синапс передає сигнал таким, як він є, а гальмуючий – змінює його.

Механізм дії нейрона є наступним: коли сумарний потенціал, який створюється на нейроні імпульсами збудження та гальмування, що приходять по дендритам від синапсів, перевищує деякий пороговий рівень, нейрон дає імпульс збудження, який розповсюджується по аксону до інших нервових клітин. Якщо підтримувати цей потенціал постійним, нейрон буде генерувати періодичну послідовність імпульсів збудження, яка відповідає стану активності нейрона, розділених короткими проміжками рефрактерності. При потенціалі, який не перевищує порогів рівень, клітина зберігає стан спокою.

Застосуємо модель Хопфілда до опису нейронної мережі. Стан $j-$го нейрона будемо характеризувати змінною S_j, яка приймає значення 1, якщо нейрон активний, і значення -1, якщо нейрон знаходиться в стані спокою. При проходженні гальмуючого синапсу активний імпульс перетворюється на

сигнал гальмування, а при проходженні збуджуючого синапсу він не змінюється (рис.2.1.)



а) б)

Рис. 2.1.Проходження активного імпульсу: а)гальмуючим;
б)збуджуючим синапсом. Н1,Н2 – нейрони, СГ- гальмуючий синапс,
СЗ-збуджуючий синапс.

Синаптичний зв'язок є односпрямованим. Нехай зв'язок між i-м та j-м нейронами характеризується коефіцієнтом зв'язку J_{ij} та J_{ji}, причому в загальному випадку $J_{ij} \neq J_{ji}$. Тоді формально можна побудувати енергію взаємодії:

$$\delta E_{ij} = -\frac{1}{2}(J_{ij}S_iS_j + J_{ji}S_iS_j).$$

Можливі стійкі картини активності системи з двох нейронів відповідають мінімуму енергії.

У природніх нейронних мережах людського мозку механізм довгого запам'ятовування може суттєво відрізнятись. Проведені експерименти свідчать, що процес навчання характеризується встановленням нових синаптичних контактів між нейронами і модифікацією існуючих зв'язків.

2.2 Системи із можливістю навчання

Простим прикладом системи із можливістю навчання є модель Хопфілда, розглянута раніше, у якій використано правило Хебба (2.4). Сам процес навчання для такої системи (і для більшості систем) являє собою налаштування коефіцієнтів матриці зв'язку $J = \{J_{ij}\}$. Така система складається з N бістабільних елементів, між ними встановлено відповідно $N(N-1)/2$ зв'язків. Кожен із цих зв'язків характеризується коефіцієнтом J_{ij} який в процесі навчання може змінюватися.

Процес навчання для системи, представленої моделлю Хопфілда

До кожного з елементів S_i підводиться відповідне значення ξ_i^m зі складу m- го прообраза. Зв'язок між i та j-м елементами здійснюється відповідно коефіцієнту J_{ij} , причому коефіцієнт зв'язку після підведення m -го прообразу змінюється за правилом:

$$J_{ij} = J_{ij} + \xi_i^m \xi_j^m . \qquad (2.6)$$

Після цього до елементів мережі підводяться значення наступного прообразу ξ_j^{m+1} і коефіцієнти знову змінюються за правилом Хебба (2.5).

Процес розпізнавання для системи, представленої моделлю Хопфілда

Після того, як навчання завершено, мережу можна використовувати для розпізнавання образів. В цьому режимі коефіцієнти зв'язку J_{ij} фіксуються. До кожного із елементів мережі S_i підводяться значення ξ_i для розпізнавання образа. Ці значення формують початкові стани $S_i = \xi_i$ при $t = 0$. Після присвоєння початкового стану всім елементам зовнішній вплив відключається і елементи починають функціонувати за правилом переходів:

$$S_i(t+1) = sign(\sum_j J_{ij} S_j(t)),$$

час вважається дискретним.

Після проходження деякого інтервалу часу переходи зупиняються і мережа переходить до стійкого стану , що відповідає найближчому із образів, які зберігаються у мережі. Зчитуючи значення елементів, які встановились, можна безпосередньо оперувати цим образом.

Всі картини, які надходять на вхід, система розділяє за принципом близькості їх до класів, що відповідають образам, які зберігаються в пам'яті системи. Таким чином, система, що функціонує за моделлю Хопфілда, здатна розв'язувати деякі задачі класифікації і розпізнавання образів.

Переваги і недоліки систем, що функціонують за моделлю Хопфілда

Основна перевага – простий алгоритм навчання, який не є дуже трудомістким.

Основний недолік - система завжди у якості найближчого до образу $\{\xi_i\}$, що подається на вхід, подає той, який максимально із ним співпадає ($\{\xi_i^m\}$), тобто той, для якого величина

$$\sum_i \xi_i \xi_i^m \to \max.$$

Таким чином, подібна система не здатна розпізнавати якісь більш складні суттєві признаки або закономірності у відповідності між вхідними даними та тими, що зберігаються у її пам'яті.

Недоліки систем, що функціонують за моделлю Хопфілда, у деякій мірі ліквідовані у системах іншого виду – персептронах.

Процес навчання у персептронах

Персептрон загального виду представляє собою систему, яка складається із декількох шарів елементів. Але спочатку розглянемо персептрон, в якому є лише два шара – вхідних та вихідних елементів.

Стани вхідних елементів позначимо $\{I_j\}$, стані вихідних елементів позначимо $\{O_j\}$. Стани елементів можуть приймати значення {0,1}. Кожен I_j з'єднується з O_i зв'язком із вагою J_{ij}. При заданих станах вхідних елементів та відомих коефіцієнтах вагових зв'язків стани вихідних елементів визначаються за правилом:

$$O_i = \gamma(\sum_j J_{ij} I_j), \tag{2.7}$$

де функція σ визначається як

$$\gamma(z) = \begin{cases} 0, z = 0; \\ 1, z > 0. \end{cases}$$

Вираз (2.7) встановлює відповідність між значеннями, які спостерігаються на виході, значенням які спостерігаються на вході. Існують персептрони, для яких правило (2.7) має імовірнісний характер: вважають, що при заданій величині $\sum_j J_{ij} I_j$ вихідний елемент O_i знаходиться у стані $O_i = 1$ з імовірністю:

$$p(O_i = 1) = \frac{1}{1 + e^{\frac{-\sum_j J_{ij} I_j}{\Theta}}},$$

де Θ - керуючий параметр. Вважається, що при $\Theta = 0$ імовірнісний алгоритм співпадає із детермінованим.

Безпосередньо *процес навчання* полягає у визначенні вагових коефіцієнтів J_{ij}. Настройка цих коефіцієнтів відбувається наступним чином:

1.Задається вхідна картина $\{I_j\}$.

2.Для неї вказується бажана картина на виході $\{D_j\}$.

3.Персептрон оперує вхідною картиною $\{I_j\}$ на виходах з'являється результат - $\{O_j\}$.

4. Результат порівнюється із бажаною картиною, формується похибка:

$$\sigma_j = D_j - O_j.$$

5.Для зменшення похибки коригують вагові коефіцієнти зв'язку J_{ij} за формулою:

$$J_{ij} = J_{ij} + \Delta J_{ij}, \tag{2.8}$$

де $\Delta J_{ij} = \varepsilon \sigma_i I_j$, ε - малий параметр.

Вираз (2.8) передбачає, що коригування вагових коефіцієнтів відбувається лише для зв'язків, які приходять від активних входів ($I_j = 1$), і тільки якщо відповідний вихід O_j є помилковим ($\sigma_i \neq 0$).

Недоліки персептронів

Розглянутий персептрон із двома шарами не здатний виконувати деякі функції, наприклад, функцію «виключаюче АБО» (табл. 2.1).

Таблиця 2.1. Таблиця істинності елемента «Виключаюче АБО»

y / x	0	1
0	0	1
1	1	0

Запишемо бажану залежність виходу персептрона від входів x, y, яка відповідає табл.2.1:

$$D = \begin{cases} 0, I_x = I_y = 0; \\ 1, I_x = 1, I_y = 0; \\ 1, I_x = 0, I_y = 1; \\ 0, I_x = I_y = 1. \end{cases} \tag{2.9}$$

Реальна залежність виходу від входів, задається правилом:

$$O = \gamma(J_x I_x + J_y I_y).$$

Тоді

$$O = \begin{cases} 0, I_x = I_y = 0; \\ \gamma(J_x), I_x = 1, I_y = 0; \\ \gamma(J_y), I_x = 0, I_y = 1; \\ \gamma(J_x + J_y), I_x = I_y = 1, \end{cases}$$

що протиречить (2.9), оскільки щоб $\gamma(J_x) = \gamma(J_x) = 1$, потрібно, щоб $J_x > 0, J_y > 0$, але тоді виходить $\gamma(J_x + J_y) = 1$.

2.3 Узагальнені персептрони

Узагальнені персептрони складаються з n шарів (рис.2.2), елементи k-го шару зв'язані з елементами $k-1$ та $k+1$ шарів. Зв'язки між виходами елементів k-го шару та входами елементів $k+1$ шару характеризуються коефіцієнтами J_{ji}^k, які можуть бути як додатніми, так і від'ємними. Перший шар складається із вхідних елементів, останній шар складається з вихідних елементів. Таким чином, між вхідними та вихідними шарами є $n-2$ шари елементів, які називаються прихованими. Кількість елементів у різних шарах може бути різною.

Функціонування персептрону

Позначимо стани елементів вхідного шару $\{I_j\}$, а вихідного шару $\{O_j\}$. Входи прихованого k-го шару будемо позначати x_i^k, виходи y_i^k, причому

$$y_i^k = \frac{1}{1 + e^{-x_i^k}}; \tag{2.10}$$

$$x_j^{k+1} = \sum_i J_{ji}^k y_i^k. \tag{2.11}$$

Вираз (2.10) обрано з урахуванням того, що при зміні x від $-\infty$ до ∞ значення y повинно монотонно змінюватись від 0 до 1.

Навчання персептрону

Навчання полягає в тому, щоб добрати зв'язки J_{ji}^k таким чином, щоб між входами персептрону та його виходами існувала така залежність, яка гарантує, що при подачі на його вхід якогось еталонного прообразу, на його виході з'являлась відповідна цьому прообразу задана нами картина.

Алгоритм навчання складається з наступних етапів:

1. $m = 1$
2. На перший шар подається еталонний m-й прообраз $\{I_{jm}\}$, для якого відома бажана реакція на виході $\{D_{jm}\}$.
3. Отримується реальна реакція персептрона на введений прообраз - $\{O_{jm}\}$.
4. $m = m + 1$, перехід на пункт 1.
5. Формується похибка

$$\sigma = \frac{1}{2}\sum_{m=1}^{M}\sum_{j}(D_{jm} - O_{jm})^2,$$

яка є мінімальною лише якщо для всіх M еталонних прообразів результат їхньої обробки співпав із бажаним.

6. Обчислюються коригуючі поправки для коефіцієнтів зв'язку:

$$\Delta J_{ji} = -\varepsilon \frac{\partial \sigma}{\partial J_{ji}}, \qquad (2.12)$$

де величина $\frac{\partial \sigma}{\partial J_{ji}}$ обчислюється за методом зворотного розповсюдження похибок.

Примітка: замість виразу (2.12) для коригування коефіцієнтів зв'язку можна використовувати формулу

$$\Delta J_{ji}^{k}(t) = -\varepsilon \frac{\partial \sigma}{\partial J_{ji}^{k}(t)} + \alpha J_{ji}^{k}(t-1),$$

де час t є дискретним і змінюється після закінчення кожного циклу навчання, коефіцієнт $0 < \alpha < 1$. При $t = 0$ коефіцієнтам α, ε присвоюють випадкові значення з інтервалу $(0,1)$.

2.4 Метод зворотного розповсюдження похибок

Метод зворотного розповсюдження похибок полягає у розв'язанні задачі оптимізації, яка полягає у знаходженні таких значень коефіцієнтів зв'язку J_{ji}^{k}, що

$$\sigma = \frac{1}{2}\sum_{m=1}^{M}\sum_{j}(D_{jm} - O_{jm})^2 \to \min.$$

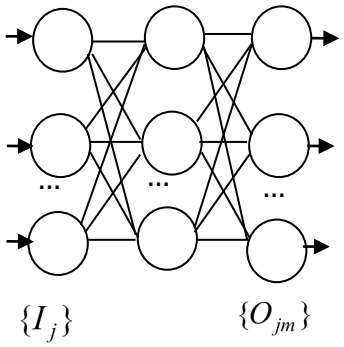


Рис.2.2.Схема узагальненого персептрона, $n = 3$

Ця задача фактично зводиться до задачі знаходження $\frac{\partial \sigma}{\partial J_{ji}}$, а далі – до виконання корекції коефіцієнтів згідно методу градієнтного спуску

$$J_{ji} = J_{ji} + \Delta J_{ji}$$

де ΔJ_{ji} визначається виразом (2.12).

Будемо вважати, що значення входів x_j^k та виходів y_j^k розглядаються у випадку подання на вхід персептрону якогось еталонного прообразу.

При виводі подальших формул будемо використовувати наступну тотожність

$$\frac{\partial \sigma}{\partial y_i^{k-1}} = \sum_j \frac{\partial \sigma}{\partial y_j^k} \frac{\partial y_j^k}{\partial x_j^k} \frac{\partial x_j^k}{\partial y_i^{k-1}}. \qquad (2.13)$$

Враховуючи (2.11), (2.12) та (2.13), можна записати рекурентне співвідношення

$$\frac{\partial \sigma}{\partial y_i^{k-1}} = \sum_j \frac{\partial \sigma}{\partial y_j^k} y_j^k (1 - y_j^k) J_{ji}^{k-1}. \qquad (2.14)$$

Вираз (2.14) можна обчислити для $(k-1)$-го шару, якщо відомі значення похідної $\frac{\partial \sigma}{\partial y_j^k}$ та виходів k-го шару. Для останнього шару ці значення відомі зі співвідношення

$$\frac{\partial \sigma}{\partial y_i^n} = y_i^n - D_i. \qquad (2.15)$$

Тоді починаючи з передостаннього шару і рухаючись в напрямку першого, можна послідовно застосовувати формулу (2.14) і обчислити $\frac{\partial \sigma}{\partial y_i^{k-1}}$, $k = n, ..., 2$ для кожного i-го елементу.

Перейдемо до одержання безпосередньо величини $\frac{\partial \sigma}{\partial J_{ji}}$:

$$\frac{\partial \sigma}{\partial J_{ji}^{k-1}} = \frac{\partial \sigma}{\partial y_j^k} \frac{\partial y_j^k}{\partial x_j^k} \frac{\partial x_j^k}{\partial J_{ji}^{k-1}}. \tag{2.16}$$

Враховуючи (2.10), (2.11) вираз (2.16) набуває виду

$$\frac{\partial \sigma}{\partial J_{ji}^{k-1}} = \frac{\partial \sigma}{\partial y_j^k} y_j^k (1 - y_j^k) y_j^{k-1}. \tag{2.17}$$

Алгоритм знаходження $\frac{\partial \sigma}{\partial J_{ji}}$ на основі методу зворотного розповсюдження похибок приймає вид:

1. Обчислити значення виходів y_j^k, $k = 1, ..., n$, за формулою (2.10) при подачі на вхід еталонного прообразу, задатись значеннями J_{ji}.
2. Обчислити значення похідної (2.15) для останнього шару.
3. З урахуванням результатів п.2 обчислити (2.14) для $k = n, ..., 2$.
4. Обчислити (2.17) на основі результатів п.1, 3.

Приклади використання персептронів:

- Розпізнавання дзеркальної симетрії (узагальнений персептрон);
- Робота із зображеннями.

В кібербезпеці такі задачі зустрічаються при розпізнаванні паттернів шкідливого ПЗ, представленого у графічному виді [7,8], при біометричній автентифікації за зображенням.

2.5 Машина Больцмана

Основні принципи функціонування

Машиною Больцмана будемо називати мережу, яка складається з бістабільних елементів, стани яких описуються двійковими змінними $S_i = \{0,1\}$.

Елементи мережі поділяються на:

1.Видимі - це вхідні та вихідні елементи, на які підводиться вхідна картина, або, відповідно, з яких знімається реакція мережі.
2.Приховані – всі елементи, які не є видимими.

Енергія мережі визначається виразом:

$$E = -\frac{1}{2}\sum_{i,j} J_{ij} S_i S_j .$$

Алгоритм переходів

Елементи, які складають машинуБольцмана, змінюють свої стани за імовірнісним правилом:

i-й елемент на наступному кроці за часом приймає активний стан $S_i = 1$ з імовірністю

$$p_i = \frac{1}{1 + \exp(-\frac{\Delta E_i}{\theta})}, \qquad (2.18)$$

при фіксованих станах інших елементів. У формулі (2.18)

$$\Delta E_i = \sum_j J_{ij} S_j$$

-це величина зменшення енергії при переході i-го елементу у активний стан, θ - параметр, який слугує аналогом температури.

В результаті переходів за правилом (2.18) система приходить до стану «теплової» рівноваги, який характеризується тим, що імовірність знаходження картини активності мережі у стані $\{S_i\}$ задається розподілом Больцмана

$$p(\{S_j\}) = \frac{\exp(-\frac{E(\{S_i\})}{\theta})}{\Sigma},$$

де Σ - нормувальний коефіцієнт.

Навчання машини Больцмана

Мета навчання – побудова внутрішньої моделі машини Больцмана, яка відтворює із достатньо високим ступенем точності закономірності зв’язків між різними структурами, притаманні реальному світові, який оточує машину. Різним внутрішнім моделям будуть відповідати різні внутрішні зв’язки J_{ij} між елементами машини.

Нехай ми розглядаємо структури оточуючого середовища двох класів - A, B. Тоді входи та виходи моделі, яку ми будуємо, будуть характеризуються структурами класів A, B, а саме: входам притаманні структури класу A, а виходам притаманні структури класу B. Структури, які відносяться до класу

А, занумеруємо індексом $\alpha = 1,2,...k$, а структури, які відносяться до класу В, занумеруємо індексом $\beta = 1,2,...,l$.

Тоді входи машини Больцмана будемо позначати

$$I^{\alpha} = (I_1^{\alpha}, I_2^{\alpha}, ..., I_n^{\alpha}),$$

а виходи

$$O^{\alpha} = (O_1^{\alpha}, O_2^{\alpha}, ..., O_n^{\alpha}).$$

Нехай зв'язок між структурами класів А та В в оточуючому машину світі має статистичний характер і характеризується сукупними ймовірностями $p(\beta,\alpha)$ спостереження різноманітних структур β та α.

Введемо умовні ймовірності:

$$\pi(\beta \mid \alpha) = \frac{p(\beta,\alpha)}{p(\alpha)},$$

яка відповідає спостереженню структури β в оточуючому середовищі, якщо в ньому присутня структура α; та

$$\hat{\pi}(\beta \mid \alpha),$$

яка відповідає спостереженню структури β на виходах машини, якщо у на входах присутня структура α.

Тоді процес навчання машини Больцмана зведеться до зміни вагових коефіцієнтів зв'язку J_{ij} таким чином, щоб розходження між умовними ймовірностями реального світу $\pi(\beta \mid \alpha)$ і умовними ймовірностями $\hat{\pi}(\beta \mid \alpha)$, притаманними машині Больцмана, було мінімальним.

У якості критерію близькості двох розподілів будемо використовувати міру

$$G = \sum_{\alpha,\beta} p(\beta,\alpha) \ln \left| \frac{\pi(\beta \mid \alpha)}{\hat{\pi}(\beta \mid \alpha)} \right|,$$

яка досягає мінімуму у нулі при співпаданні ймовірностей π та $\hat{\pi}$ $\forall \alpha, \beta$.

Обчислимо похідні $\partial G / \partial J_{ij}$. Оскільки від вагових коефіцієнтів зв'язку залежать лише імовірності $\hat{\pi}$, то

$$\frac{\partial G}{\partial J_{ij}} = -\sum_{\alpha,\beta} p(\beta,\alpha) \frac{\partial}{\partial J_{ij}} \ln \hat{\pi}(\beta \mid \alpha). \qquad (2.19)$$

Знайдемо подхідну (2.19) при довільно обраних картинах активності вхідних I та вихідних O елементів . Картину активності прихованих елементів позначимо S.

Тоді при подачі на вхід заданої постійної картини I ймовірність знайти приходні і вихідні елементи у станах S та O відповідно визначається як

$$p_I(S,O) = \frac{1}{\Sigma_I}\exp(-\frac{E_I(S,O)}{\theta}),$$

де

$$\Sigma_I = \sum_{S,O}\exp(-\frac{E_I(S,O)}{\theta}).$$

Оскільки

$$\hat{\pi}(O \mid I) = \sum_S p_I(S,O),$$

то

$$\hat{\pi}(O \mid I) = \frac{1}{\Sigma_I}\sum_S \exp(-\frac{E_I(S,O)}{\theta}). \qquad (2.20)$$

Введемо суму

$$\Sigma_{I,O} = \sum_S \exp(-\frac{E_I(S,O)}{\theta}), \qquad (2.21)$$

яка обчислюється при фіксованих станах вхідних та вихідних елементів. За допомогою (2.21) рівність (2.20) набуває вигляду:

$$\hat{\pi}(O \mid I) = \frac{\Sigma_{I,O}}{\Sigma_I}.$$

Звідси,

$$\frac{\partial}{\partial J_{ij}}\ln\hat{\pi}(O \mid I) = \Sigma_{I,O}^{-1}\frac{\partial\Sigma_{I,O}}{\partial J_{ij}} - \Sigma^{-1}\frac{\partial\Sigma_I}{\partial J_{ij}}.$$

Враховуючи визначення Z_I та $Z_{I,O}$, а також залежність енергії від коефіцієнтів J_{ij}, отримуємо

$$\frac{\partial}{\partial J_{ij}}\ln\hat{\pi}(O \mid I) = \frac{(< S_iS_j >_{I,O} - < S_iS_j >_I)}{\theta}, \qquad (2.22)$$

де $< S_iS_j >_I$ та $< S_iS_j >_{I,O}$ - середні за часом від добутку активностей i-го та j-го елементів, взяті при фіксованому стані лише вхідних або вхідних та вихідних елементів відповідно.

Підставимо (2.22) до (2.19):

$$\frac{\partial G}{\partial J_{ij}} = \frac{\sum_{\alpha,\beta} p(\beta,\alpha)(< S_i S_j >_\alpha - < S_i S_j >_{\alpha,\beta})}{\theta}. \qquad (2.23)$$

Після знаходження (2.23) виконується корекція вагових коефіцієнтів J_{ij} на величину

$$\Delta J_{ij} = -\varepsilon \frac{\partial G}{\partial J_{ij}} \qquad (2.24)$$

У частковому випадку, коли поява різних картин α є рівноімовірною, вираз (2.23) набуває вигляду

$$\frac{\partial G}{\partial J_{ij}} = \frac{\sum_{\alpha} (< S_i S_j >_\alpha - < S_i S_j >_{\alpha,\beta(\alpha)})}{k\theta}, \qquad (2.25)$$

де k -кількість різних картин α.

Тоді один цикл процедури навчання складається з етапів:

1. $\alpha = 1$.
2. На вхід машини подається картина α; вихідні елементи фіксуються у станах $\beta(\alpha)$, які повинні спостерігатися. Усереднюючи по великих інтервалах часу, обчислюють значення
   $$< S_i S_j >_{\alpha,\beta(\alpha)}$$
   для всіх пар прихованих елементів.
3. На вхід машини подається картина α, але вихідні елементи не фіксуються. Усереднюючи по великих інтервалах часу, обчислюють значення $< S_i S_j >_\alpha$.
4. Обчислити вираз (2.25), знайти
   $J_{ij} = J_{ij} + \Delta J_{ij}$,
   де ΔJ_{ij} визначається за формулою (2.24).
5. $\alpha = \alpha + 1$. Якщо $\alpha \le k$, перехід на п.2, інакше процедура закінчена.

Такі цикли потрібно повторювати до тих пір, доки не буде досягнуто потрібну точність розпізнавання. Якщо помилки не зменшуються, можна ввести додаткові приховані елементи і провести навчання заново.

Приклади використання машин Больцмана:

- розпізнавання голосних звуків, що вимовляються різними людьми (задачі біометричної автентифікації);
- поділ речень на слова (задачі аналізу текстів природної мови).

Недоліки машини Больцмана

Основний недолік полягає в тому, що навчання є досить тривалим процесом (особливо у випадку реалізації машини Больцмана на ЕОМ), для якого потрібно багаторазове накопичення статистики.

2.6 Складнощі розв'язання задач на основі систем із можливістю навчання

Використовувати чи не використовувати нейро-подібні моделі?

Процес навчання більшості із розглянутих систем зводиться до розв'язання задач оптимізації. Розв'язання подібних задач є досить трудомістким процесом, який у випадку його реалізації на ЕОМ вимагає великої кількості машинних ресурсів. Іншим варіантом є аналогова реалізація мереж систем із можливістю навчання, яка припускає можливість паралельного здійснення обчислень, накопичення статистик тощо. Такий варіант дає можливість розв'язувати складні задачі оптимізації за більш прийнятні терміни.

Якщо ми використовуємо аналогову модель, то на основі принципів функціонування розглянутих систем (систем на основі моделі Хопфілда, машини Больцмана) можна швидко і ефективно розв'язувати деякі складні задачі оптимізації. Розглянемо приклад.

Приклад. Розв'яжемо оптимізаційну задачу розбиття графа на дві рівні частини.

Нехай граф складається з парної кількості елементів, зв'язаних між собою. Нам потрібно так розбити граф на дві рівні за кількістю елементів групи, щоб повна кількість зв'язків між елементами, які належать цим двом групам, була мінімальною.

Нехай кожній вершині графа відповідає елемент, стан якого характеризується змінною $S_i = \{1, -1\}$. Ребра графа будуть характеризуватися матрицею зв'язків $J_{ij} = \{1,0\}$, причому $J_{ij} = 1$, якщо існує ребро-зв'язок між i та j-м елементом, і відповідно, $J_{ij} = 0$, якщо зв'язок відсутній. Крім того, $J_{ii} = 0 \ \forall i$.

Будемо ділити граф на дві рівні групи, причому елементи першої групи мають стани $S_i = 1$, а елементи другої групи - $S_i = -1$.

Кількість зв'язків між елементами цих двох груп становить

$$N_{зв} = \frac{1}{8}\sum_{i,j} J_{ij}(S_i - S_j)^2 . \qquad (2.26)$$

Повна кількість зв’язків у графі становить

$$N_{повн} = \sum_{i,j} J_{ij} \,. \tag{2.27}$$

Тоді вираз (2.26) можна представити наступним чином:

$$N_{зв} = \frac{1}{8}\sum_{i,j} J_{ij}(S_i - S_j)^2 = \frac{1}{8}\sum_{i,j} J_{ij} S_i^2 - \frac{2}{8}\sum_{i,j} J_{ij} S_i S_j + \frac{1}{8}\sum_{i,j} J_{ij} S_j^2 = \frac{1}{4}\sum_{i,j} J_{ij} - \frac{1}{4}\sum_{i,j} J_{ij} S_i S_j =$$

$$= \frac{1}{2} N_{повн} - \frac{1}{4}\sum_{i,j} J_{ij} S_i S_j . \tag{2.28}$$

Величина

$$L = \sum_i S_i = n_1 - n_2 , \tag{2.29}$$

де n_1, n_2 - кількість елементів у першій та другій групах відповідно. За умовою задачі повинно бути

$$L = 0 .$$

Тоді оптимальне розбиття графа на дві рівні групи відповідає абсолютному мінімуму функції

$$E = N_{зв} + \varepsilon L^2 \to \min , \tag{2.30}$$

причому $\varepsilon > 0$, ε -велике.

В результаті підстановки (2.28) та (2.29) у (2.30) отримаємо:

$$E = \frac{1}{2} N_{повн} + \sum_{i,j} (\varepsilon - \frac{1}{4} J_{ij}) S_i S_j . \tag{2.31}$$

Вираз (2.31) можна інтерпретувати як енергію системи, заданої даним графом. Для знаходження мінімуму функції (2.31) можна побудувати аналогову систему, яка змінює стани за правилом переходів моделі Хопфілда:

$$S_i = sign \sum_j J_{ij} S_j \,. \tag{2.32}$$

Це правило гарантує, що на кожному кроці еволюції системи функція (2.31) буде строго зменшуватися. В ідеалі ці переходи повинні привести до стану із глобальним мінімумом енергії. Але тут існує складність, що

пов’язана із можливістю існування багатьох локальних мінімумів, яку ми розглянемо нижче.

Примітка. Задачі розбиття графів з вищевказаним принципом зустрічаються при розв’язанні багатьох практичних задач кібернетичної безпеки. Робота з графами досить часто зустрічається у кібербезпеці, оскільки у вигляді графів можна представити мережі різної природи, і в тому числі мережі об’єктів критичної інфраструктури. Окремо варто згадати графи атак. Графи використовуються при проектуванні великих інтегральних схем в задачах технічного захисту інформації.

Основна проблема, яка виникає при розв’язанні подібних задач – те, що вони є експоненційно складними ($O \sim \exp(N)$, де O- кількість операцій при точному розв’язанні, N – кількість елементів (вершин) графа).

Проблема локальних мінімумів

Використання детермінованих алгоритмів функціонування систем із можливістю навчання, наприклад подібних правилу переходів Хопфілда (2.32), часто приводить до пастки, яка полягає в тому, що стани системи «застрягають» у одному з локальних мінімумів, не потрапивши до глобального.

Це пов’язано із тим, що рельєф функції виду (2.31) може мати складний характер із багатьма локальними мінімумами (рис. 2.3).

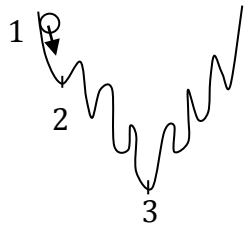


Рис.2.3. Рельєф з багатьма локальними мінімумами

В цьому разі, якщо система починає еволюціонувати зі стану 1, використання правила (2.32), яке вимагає строгого зменшення енергії системи на кожному кроці, призведе до того, що система прийде до стану 2, і ніколи не потрапить у потрібний нам стан 3, який є глобальним мінімумом.

Для того, щоб уникнути такої ситуації, використовують метод відпалу.

Метод імітації відпалу

Цей метод полягає в тому, щоб дозволити (із деякою малою імовірністю) системі здійснювати переходи у стани, які приводять до підвищення повної енергії системи. Це дасть системі змогу «вискочити» із пастки локального мінімуму і продовжити шлях до глобального.

Імовірність здійснення вказаних переходів визначається формулою:

$$p \sim \exp(-\frac{\Delta E}{\theta}),$$

де ΔE - різниця енергій системи у станах до і після переходу, θ - параметр, який відіграє роль температури. Керування параметром θ дозволяє не затримуватись у локальних мінімумах за рахунок можливості підвищення енергії системи, і навпаки, заборонити переходи у стани, що підвищують енергію, при досягненні глобального мінімума.

Примітка. Принцип відпалу прийшов із фізики твердого тіла. При швидкому охолодженні з розплаву тіло утворює аморфну невпорядковану структуру, яка відповідає одному із великої кількості локально стабільних станів енергії. Кристал відповідає абсолютному мінімуму потенціальної енергії. Для того, щоб досягти кристалічного стану з розплаву, потрібно застосовувати метод відпалу, який полягає у повільному зниженні температури θ.

2.7 Питання для самоперевірки

1. В чому полягає принцип асоціативного запам'ятовування у розподілених системах типу Хопфілда-Хебба?
2. Назвіть недоліки двошарового персептрона та системи Хопфілда-Хебба.
3. Які співвідношення для корекції вагових коефіцієнтів використовують для двошарового персептрона?
4. Які співвідношення для корекції вагових коефіцієнтів використовують для багатошарового (узагальненого) персептрона?
5. В яких задачах використовують моделі Больцмана? В чому їх відмінність від персептронів та систем Хопфілда-Хебба?

3 ОЦІНКИ ПОХИБОК ПРИ ЗАСТОСУВАННІ МЕТОДІВ МАШИННОГО НАВЧАННЯ

3.1 ROC – криві

Для того, щоби мати уявлення про якість одержаних результатів в результаті застосування, зокрема, методів машинного навчання, треба мати певні метрики для оцінки точності результатів застосування певних методів та моделей. При цьому можна керуватись графічними та числовими підходами. Графічні характеристики забезпечують більшу наочність. Серед таких характеристик можна зазначити ROC- криві.

В якості графічної ілюстрації точності застосування методу машинного навчанння можна використати ROC – криві (рис.3.1).

Нехай розв’язується задача класифікації з двома класами {0, 1}. Алгоритм видає деяку оцінку (може, але не обов'язково, ймовірність) приналежності об'єкта до класу 1, оцінка приналежить відрізку [0, 1].

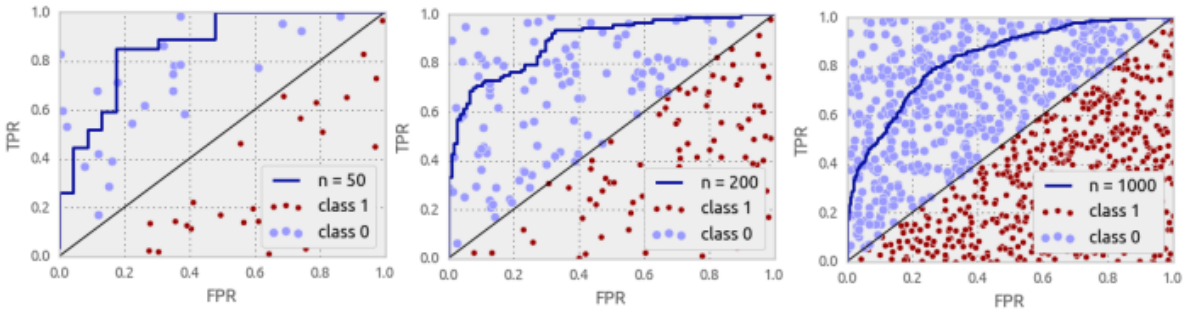


Рис.3.1. Задача бінарної класифікації та відповідні ROC-криві

Результат роботи алгоритмів на фіксованій тестовій вибірці візуалізується за допомогою ROC-кривої (ROC = receiver operating characteristic, яка називається «крива помилок»), а якість оцінюється як площа під цією кривою - AUC (AUC = “area under the curve”, площа під кривою). Покажемо на конкретному прикладі, як будується крива.

Нехай алгоритм видав оцінки, як показано в табл. 3.1.

Таблиця 3.1. Оцінки

Ідентифікатор	Оцінка	Клас
1	0,5	0
2	0,1	0
3	0,2	0

4	0,6	1
5	0,2	1
6	0,3	1
7	0,0	0

Впорядкуємо рядки табл. 3.1 за зменшенням відповідей алгоритма – отримаємо табл. 3.2. В ідеалі стовпець «клас» теж стане впорядкованим (спочатку йдуть 1, потім 0); а в найгіршому випадку - порядок буде оберненим (спочатку 0, потом 1); у випадку «сліпого угадування» буде випадкове розподілення 0 і 1.

Таблиця 3.2. Впорядковані відповіді алгоритма

Ідентифікатор	Оцінка	Клас
4	0,6	1
1	0,5	0
6	0,3	1
3	0,2	0
5	0,2	1
2	0,1	0
7	0,0	0

Таблиця 3.3. Визначення правильних міток теста

Ідентифікатор	>0,25	Клас
4	1	1
1	1	0
6	1	1
3	0	0
5	0	1
2	0	0
7	0	0

Щоб побудувати ROC-криву, слід взяти одиничний квадрат на координатній площині, розбити його на m рівних проміжків горизонтальними лініями і на n - вертикальними, де m - число 1 серед правильних міток теста (в прикладі m=3), n - число нулів (n=4). У результаті квадрат розбивається на сітку на m × n блоків.

Тепер будемо проглядати рядки таблиці 3.2 згори униз і прорисовувати на сітці лінії, переходячи із одного вузла до іншого. Стартуємо з точки (0, 0).

Якщо значення мітки класу в поточному рядку 1, то робимо крок вгору; якщо 0, то робимо крок вправо. В результаті ми потрапимо в точку (1, 1), оскільки зробили в сумі m кроків вгору і n кроків вправо (Рис.3.2).

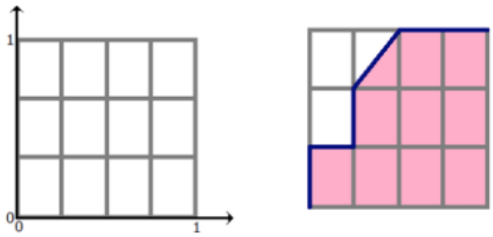


Рис. 3.2. Приклад побудованої кривої

Рис.3.3. Приклад ідеальної кривої (AUC=1), кривої що відповідає випадковому блуканню, та найгіршого випадку – AUC=0

Примітка 1: Якщо у декількох об'єктів значення оцінок рівні, то ми робимо крок в точку, яка на a блоків вище і b блоків правіше, де a - число одиниць в групі об'єктів з одним значенням мітки, b - число нулів в ній. Зокрема, якщо всі об'єкти мають однакову мітку, то ми відразу крокуємо з точки (0, 0) в точку (1, 1).

Примітка 2: для тестової виборки, що містить об’єкти тільки одного класу крива не визначена.

Кожен зафарбований блок кривої відповідає парі об’єктів {об’єкт класа 1, об’єкт класа 0} для якої алгоритм правильно передбачив порядок (об’єкт класа 1 одержав оцінку вище, ніж об’єкт класа 0), а незафарбований – парі на якій він помилився (рис.3.3, 3.4).

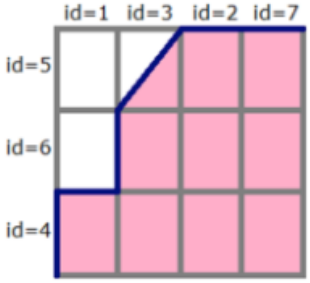


Рис. 3.4. Зв’язок кривої із парами об’єктів

Таким чином, AUC ROC – це доля пар об’єктів виду (об’єкт класа 1, об’єкт класу 0) які алгоритм впорядкував вірно.

Чисельно формула виглядає так:

$$\frac{\sum_{i=1}^{q}\sum_{j=1}^{1} I(y_i<y_j)I`(a_i<a_j)}{\sum_{i=1}^{q}\sum_{j=1}^{1} I(y_i<y_j)}, \tag{3.1}$$

де

$$I`(a_i < a_j) = \begin{cases} 0, a_i > a_j \\ 0{,}5, a_i = a_j \\ 1, a_i < a_j. \end{cases}$$

$$I`(y_i < y_j) = \begin{cases} 0, y_i \geq y_j \\ 1, y_i < y_j. \end{cases}$$

де a_i – відповідь алгоритма на об'єкті i, y_i – його мітка (клас), q – кількість об'єктів у класі. В формулі враховано випадок рівності відповідей алгоритму на декількох об'єктах.

Приклади вигляду кривих для методів машинного навчання, які набули популярність у вирішенні задач кібербезпеки, наведено на рис. 3.5.

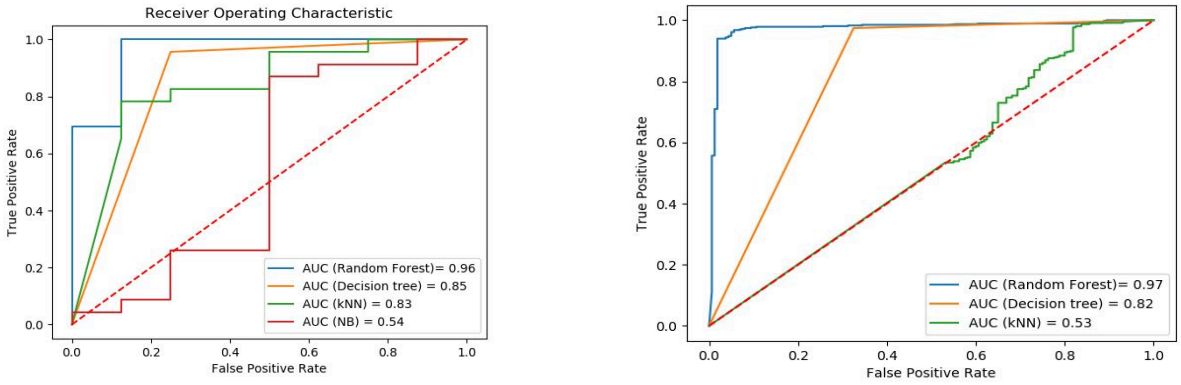


Рис. 3.5. Приклади вигляду кривих

Зауважимо, що в залежності від виду задачі різні методи можуть показувати кращий, або гірший результат.

3.2 Оцінка класифікатора - точність, повнота, F-міра

Чисельна оцінка якості алгоритму

Метрика 1. Точність.

У найпростішому випадку такий метрикою може бути частка зразків, наприклад, документів за якими класифікатор прийняв правильне рішення.

$$A = P/N$$

де A - точність, P – кількість зразків по яких класифікатор прийняв правильне рішення, N – розмір навчальної виборки.

Недолік метрики: присвоює всім зразкам однакову вагу, що може бути некоректно для випадку, коли розподіл зразків у складі навчальної виборки переважає в сторону одного чи декількох класів. Відповідно, в цьому випадку класифікатор має більше інформації по цих класах, і в рамках цих класів прийматиме більш адекватні рішення. В такому випадку зразки цих класів будуть розпізнаватись краще, оцінка буде висока, однак на інших класах класифікатор може працювати досить погано.

Вихід – робити навчальну виборку збалансованою, або використовувати іншу метрику.

Метрика 2. Точність та повнота

Точність (precision) та повнота (recall) є метриками, які використовуються при оцінці більшої частини алгоритмів видобуття інформації.

Точність в межах класу – це частка зразків, які дійсно належать цьому класу відносно всіх зразків, які система віднесла до цього класу.

Для розрахунку цих показників використовують таблицю (табл. 3.4), яка створюється для кожного класу окремо.

Таблиця 3.4. Таблиця контингентності класу

Категорія		Експертна оцінка	
		Позитивна	Негативна
Оцінка системи	Позитивна	TP	FP
	Негативна	FN	FN

В табл. 3.4 TP – істинно-позитивне рішення, TN- істинно негативне рішення, FP – хибно-позитивне рішення, FN – хибно-негативне рішення.

Тоді точність та повнота визначаються наступними співвідношеннями:

$$Prec = \frac{TP}{TP + FP},$$
$$Rec = \frac{TP}{TP + FN}.$$

Для мультикласової класифікації використовують *матрицю неточностей, або промахів (confusion matrix).*

Це матриця розміру $N \times N$, де N – кількість класів. Стовпці матриці відносяться до експертних рішень, а рядки – до рішень класифікатора. Коли

ми класифікуємо зразок із тестової виборки, то додаємо одиницю до числа, що стоїть на перетині рядку класа, який повернув класифікатор, та стовпця класу, до якого дійсно відноситься зразок (рис.3.6).

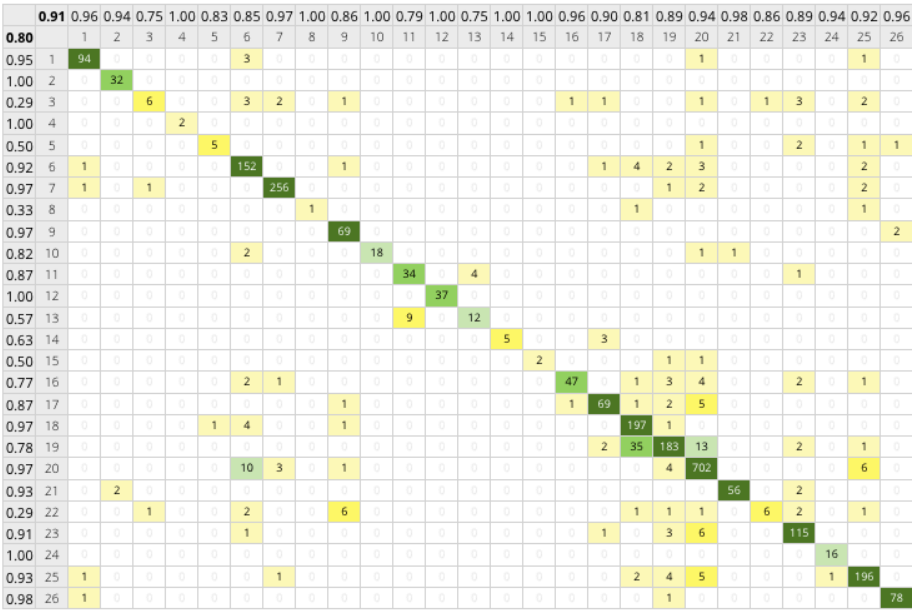


Рис. 3.6. Приклад матриці неточностей, 26 класів.

Діагональна перевага у такій матриці свідчить про те, що більшість документів класифікатор визначає правильно.

Тоді точність та повнота в рамках класу визначаються наступним чином:

$$Prec_c = \frac{a_{cc}}{\sum_{i=1}^{n} a_{ci}}$$

(відношення діагонального елемента матриці до суми рядка класу),

$$Rec_c = \frac{a_{cc}}{\sum_{i=1}^{n} a_{ic}}$$

(відношення діагонального елемента матриці до суми стовпця класу).

Результуюча точність класифікатора розраховується як арифметичне середнє по його точності за усіма класами, так само як і повнота.

Цей підхід називається *macro-averaging*.

В мові Python є необхідні засоби для обчислення відповідних величин, зокрема:

```
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
```

Відповідний рядок використання цих функцій може виглядати так:

```
file_for_precision.write(str(str(day) + ":" + str(precision_score(y_pred, y_actu, average='macro')) + ":" + str(recall_score(y_pred, y_actu, average='macro'))))
```

Метрика 3. F- міра

Чим вищими є точність та повнота, тим краще, однак, вони не можуть бути досягнуті одночасно в реальності. Тому дяким компромісом, який поєднує інформацію про точність та повноту, є *F*-міра.

Це є середнє гармонічне між точністю та повнотою:

$$F = 2\frac{Prec \cdot Rec}{Prec + Rec}$$

Ця формула надає однакову вагу точності та повноті, тому *F*-міра буде зменшуватись однаково при зменшенні точності та повноти. Однак, можливо надати вагу одному з показників (якщо таке вимагається постановкою задачі):

$$F = (\alpha^2 + 1)\frac{Prec \cdot Rec}{\alpha^2 Prec + Rec},$$

де $0 < \alpha < 1$ при приоритетності точності, $\alpha > 1$ при приоритетності повноти. При $\alpha = 1$ формула переходить у попередню (яку ще називають F_1-мірою).

3.3 Перенавчання та метод DropOut

Інколи методи машинного навчання можуть показувати різні значення метрик на різних виборках.

Перенавчанням (*overfitting*) називають одну із проблем при навчанні глибинних нейронних мереж, яка діагностується за наступними ознаками:

- модель добре пояснює тільки приклади із навчальної виборки;
- приклади, які не брали участь в навчанні, діагностуються погано.

Отже, явище перенавчання негативно позначається на точності діагностування.

Вирішується проблема перенавчання за допомогою методу проріджування (чи то методу виключення)[9] – англ. *Dropout*.

Вид мережі до проріджування та після показаний на рис. 3.7.

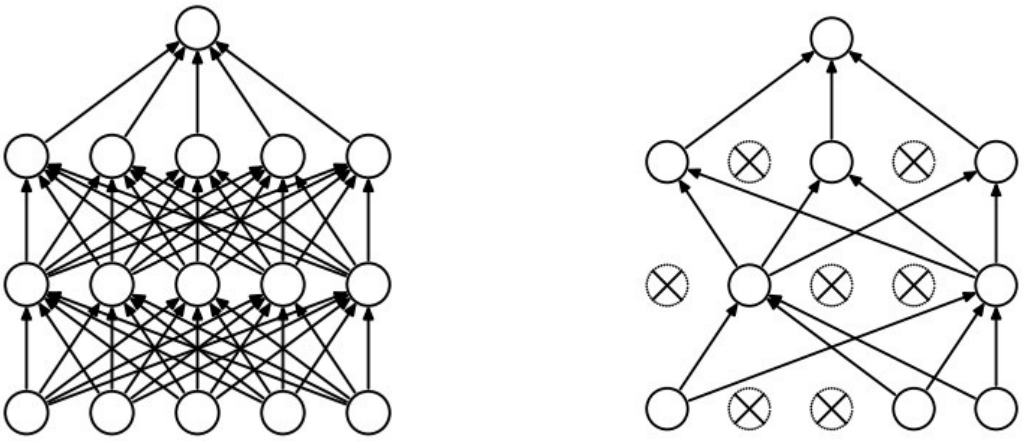


Рис. 3.7. Мережа до та після проріджування

Основна ідея методу Dropout – замість навчання однієї Deep Neural Network (DNN) навчити ансамбль декількох DNN а потім усереднити одержані результати. Мережі для навчання одержуються шляхом виключення із мережі нейронів із імовірністю p, відповідно імовірність того що нейрон залишається у мережі є q=1-p.

“Виключення” нейрона означає, що при будь-яких вхідних та вихідних даних та параметрах він повертає нуль. Виключені нейрони не вносять внесок в процес навчання на жодному із етапів алгоритма зворотного розповсюдження помилок, тому, виключення хоча б одного із нейронів відповідає навчанню нової нейронної мережі.

3.4 Розмір виборки для машинного навчання

Ще один вид графічних характеристик - Learning Curve (рис. 3.10), або навчальна крива [10], представляє собою графік, що складається із залежності середньої помилки моделі на даних використаних для навчання і залежності середньої помилки на тестових даних. У теорії існують два основні варіанти, які вийдуть при побудові даного графіка.

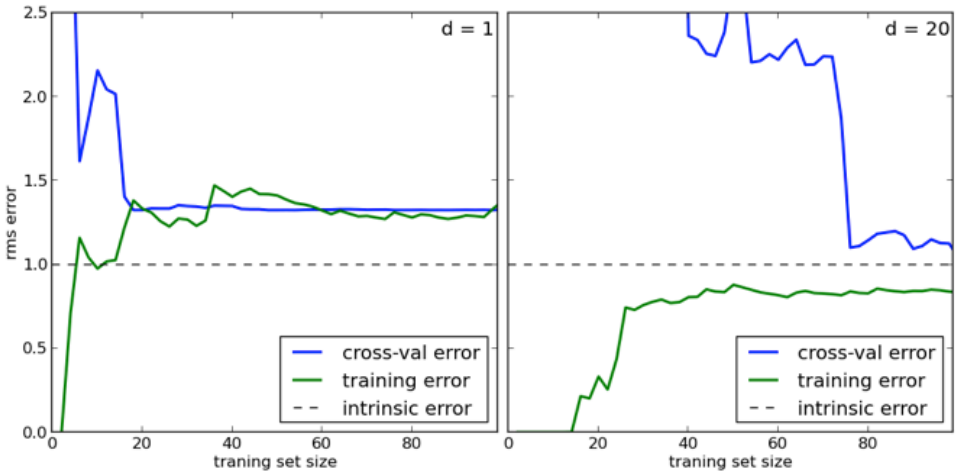


Рис. 10. Варіанти навчальної кривої

Перший варіант – відповідає випадку, коли модель недонавчена або має високе зміщення (High bias). Основна ознака такої ситуації - це висока середня помилка як для тренувальних даних так і для тестових. В цьому випадку залучення додаткових даних не поліпшить якість моделі.

Другий варіант - коли модель перенавчилась або має велику варіативність (High variance). Візуально можна визначити за наявністю великого розриву між тестовою і тренувальною кривими і низькою тренувальною помилкою. Тут навпаки більше даних може привести до поліпшення у тестовій помилці і, відповідно, до поліпшення моделі.

Вихідну виборку зазвичай розбивають на тренувальну та тестову у співвідношенні 60/40. Однак, коли одержання додаткових даних ускладнене, а модель потребує великої кількості даних для навчання, припустимим є збільшення частки тренувальної виборки по відношенню до тестової, наприклад у співвідношенні 90/10.

Детальніше про методи конструювання датасету – в [11].

3.5 Питання для самоперевірки

1. Як використовується ROC-крива для одержання уявлення про успішність застосування методу? Як вона будується?
2. Якими є недоліки критеріїв оцінки класифікатора – точності та повноти? Чому F- міра є більш повним критерієм?
3. Коли варто використовувати confusion matrix для оцінки точності та повноти класифікатора? Як вона будується?
4. В чому полягає явище перенавчання та як його визначити за Learning Curve?
5. Чим ми керуємось, коли визначаємо розмір навчальної виборки?

4 ВИЯВЛЕННЯ ВРАЗЛИВОСТЕЙ КОДУ НА ОСНОВІ МЕТОДІВ ГЛИБИННОГО НАВЧАННЯ

4.1 Постановка задачі та основні методи

Проблема статичного аналізу коду на притаманні йому слабкості зараз уже не розв'язується за допомогою наявних аналізаторів коду у повній мірі. Серед таких аналізаторів можна вказати RATS та інші вбудовані у середовища розробки засоби, які дозволяють, маючи базу типових уразливих конструкцій та функцій, будувати зауваження щодо необхідності внесення виправлень.

Однак, із зростанням варіативності написання вихідного коду та великої кількості мов програмування цей підхід не достатньо ефективний.

Натомість можна використовувати методи глибинного навчання, які можуть видобувати узагальнені структури з великого обсягу даних та класифікувати їх як безпечні чи небезпечні, що якраз є вдалим при великих обсягах кодів. Моделі на основі глибинного навчання можуть перенавчатись для різних мов, для різних типів проектів, адаптуючись під особливості вихідних кодів, що складно зробити, маючи статичну базу накопичених слабкостей. Подібний підхід використано в роботі [12].

Іншою перевагою методів глибинного навчання є можливість бачити приховані ознаки слабкостей, які є неочевидними з точки зору статичного аналізатора.

При використанні методів глибинного навчання слід попередньо вирішити супутні задачі, які дадуть змогу використати відповідну модель у виді нейронної мережі:

1. Представлення тестованої програми у виді, придатному для подачі на вхід моделі у виді нейронної мережі.

2. Виявлення слабкості, та відповідні ознаки.

3. Вибір адекватної моделі нейронної мережі, оскільки відомо, що різні типи мереж з різною успішністю можуть бути застосовані до певного класу задач.

Нейронна мережа, як правило, приймає дані у виді числових векторів, тому слід використати спосіб перетворення програми на вектор числових даних без втрати необхідної семантики. Таким способом можуть бути а) робота з бінарними даними програми б) робота з абстрактним синтаксичним деревом програми (АСД), в)кодування вихідного тексту програми.

Визначення. Код гаджетом називається логічно завершена конструкція коду, що посилається на виклик певної функції або аргументи виклику функції.

Визначення. Абстрактне синтаксичне дерево (АСД) - в інформатиці кінцеве позначене орієнтоване дерево, в якому внутрішні вершини зіставлені (позначені) з операторами мови програмування, а листя - з відповідними операндами. Таким чином, листя є порожніми операторами і представляють тільки змінні і константи. АСД відрізняється від дерева розбору тим, що в ньому відсутні вузли і ребра, для тих синтаксичних правил, що не впливають на семантику програми.

Приклад АСД для алгоритму Евкліда

while b ≠ 0

if a > b

a := a − b

else

b := b − a

return a

наведено на рис. 4.1.

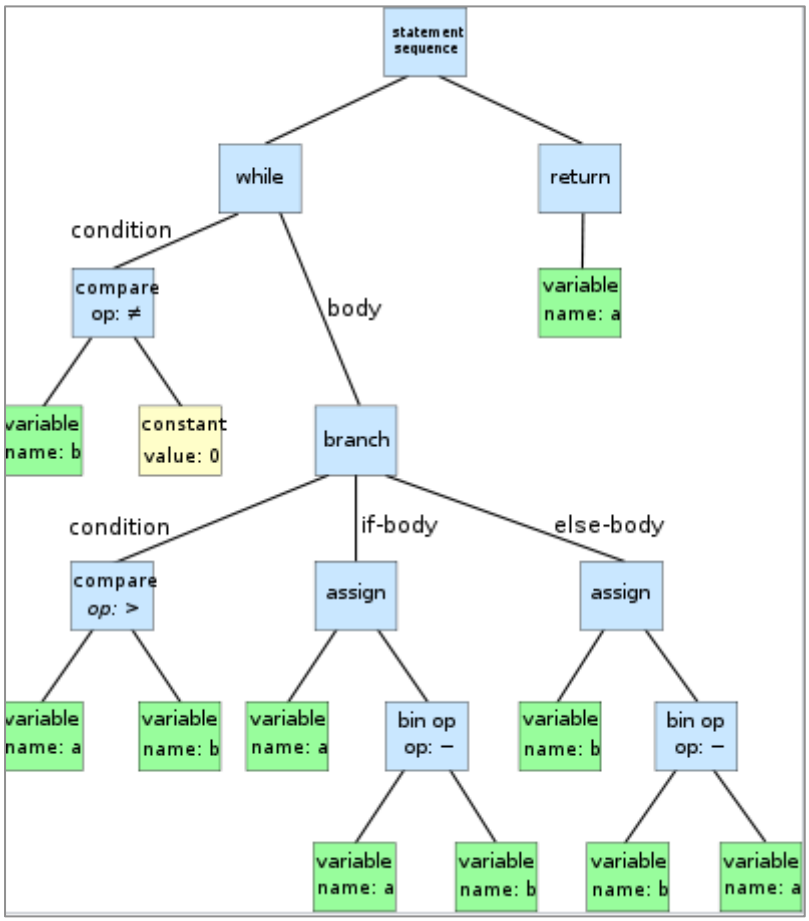


Рис. 4.1. АСД для алгоритму

Розглянемо далі представлення коду програми у виді АСД.

Попередня обробка вихідного коду складається із наступних етапів:

1. Представлення програми в виді АСД.
2. Виділення код-гаджетів.
3. Представлення код-гаджетів у виді векторів. Цей етап виконується із використанням допоміжної функції, яка представляє текстові дані у виді числового вектору (*word2vec*). Після одержання числового представлення дані уже можна подавати на вхід нейронної мережі.

Спосіб одержання АСД. Для вихідних кодів C/C++ можна використовувати clang compiler toolkit [13]. Для того, щоб отримати АСД потрібно ввімкнути дамп-режим(-ast-dump). Для цього потрібно ввести наступну команду:

```
Clang –ast –dump test.cc
```

Працюючи з вихідними кодами треба обрати, що буде елементарними досліджуваними частинами - функції, рядки коду чи увесь файл. З початкового коду треба виокремити фрагмент, який буде пов'язаний з потенційною вразливістю. Із цим фрагментом (код-гаджетом) потім буде проводитись робота. Тобто код-гаджет представляє собою набір операторів, які мають зв’язок із з потоком даних. Код-гаджет краще відокремлювати із програми, перетвореної до АСД.

Алгоритм виділення код-гаджетів є наступним:

1. Попередня обробка. На основі початкового коду, використовуючи вбудовану попередню обробку clang, створюється новий файл без директив препроцесора, які були визначені користувачем.
2. Побудова АСД із використанням clang compiler toolkit.
3. Пошук початкових точок. Під початковою точкою розуміється місце в початковому коді, з якого починається аналіз. Зазвичай у якості початкових точок використовують виклики функцій зі стандартної бібліотеки C/C++ (наприклад, malloc, memcpy, fopen і т.д.)
4. Створення графу залежностей – це граф залежності аргументів (або повернених значень) від початкової точки.
5. Перейменування змінних або функцій. На цьому кроці треба вдатись до абстрагування від користувацьких імен. Ми замінюємо всі імена функцій і змінних символічними іменами, такими як: <<VAR_1>>, <<VAR_2>>, і т.д та <<METHOD_1>>, <<METHOD_2>>. Це приводить усі код-гаджети до уніфікованого виду.

6. Створення код-гаджету. На основі графу залежностей, код-гаджет формується з токенів, взятих з вузлів графу.

Демонстрація алгоритму вилучення код гаджетів зображена на рис. 4.2, 4.3.

```
public Integer getMinElement(List myList) {
   if(myList.size() >= 0) {
      return ListManager.getFirst(myList);
   }
   return 0;
}
```

```
public TYPE_1 METHOD_1 ( TYPE_2 VAR_1 )
{ if ( VAR_1 . METHOD_2 ( ) >= INT_1 )
{ return TYPE_3 . METHOD_3 ( VAR_1 ) ; }
return INT_1 ; }
```

Рис. 4.2. Заміни користувацьких змінних на універсальні назви [12]

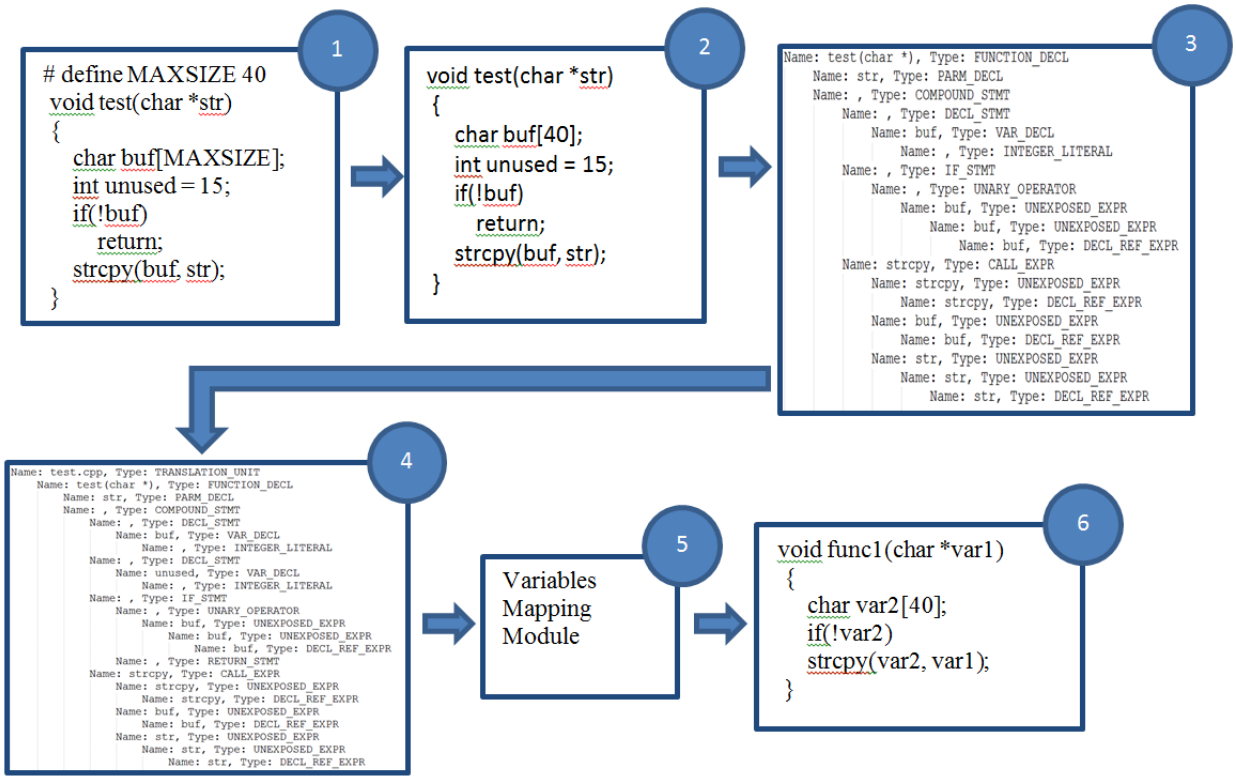


Рисунок 4.3. Побудова код-гаджету [12]

4.2 Локалізація вразливості та обробка вхідних параметрів

Важливим етапом є пошук місця вразливості. Для того, щоб визначити більш точне місце вразливості, програма повинна бути представлена з більш високим ступенем деталізації, ніж програма або функція в цілому. На рівні

код-гаджету можна досить точно локалізувати вразливість, оскільки в більшості випадків код-гаджет складається лише з декількох рядків.

Перетворення коду у вектор вхідних параметрів нейронної мережі

Після виділення код-гаджетів з початкового коду, їх треба перетворити до числового векторного виду. Для цього можна використати алгоритм word2vec.

Алгоритм word2vec приймає набір вихідних текстів (в нашому випадку це вихідні коди) в якості вхідних даних і зіставляє кожному слову вектор, видаючи координати слів на виході. Спочатку він створює словник, «навчаючись» на вхідних текстових даних, а потім обчислює векторне подання слів. Векторне подання ґрунтується на контекстної близькості: слова, що зустрічаються в тексті поруч з однаковими словами (а отже, мають схожий зміст), у векторному поданні матимуть близькі координати векторів-слів. Отримані вектори-слова можуть бути використані для обробки природної мови та машинного навчання.

Приклад роботи word2vec подано на рис 4.4.

Рис. 4.4. Приклад числового представлення слів за допомогою word2vec

В даному випадку схожі слова мають близьке числове представлення.

Це дає змогу відрізняти вразливий код від невразливого, при поданні коду в числовому виді.

4.3 Вибір типу нейронної мережі

Для задач подібного виду можна обрати нейронну мережу з довгою короткочасною пам'яттю (LSTM), оскільки треба мати можливість враховувати впливи одних операторів програми на інші. Однак нейронна мережа LSTM у стандартній формі не враховує ефектів впливу аргументів функції на більш ранні та на більш пізні операції в програмі. Це відбувається тому, що мережа односпрямована. Тому більш доречним є використання двоспрямованих мереж BLSTM.

На рисунку 4.5 показано схему будови нейронної мережі BLSTM з декількома шарами BLSTM, “щільним” шаром і шаром softmax. Шари BLSTM мають напрямки вперед і назад і містять кілька клітин LSTM. “Щільний” шар зменшує кількість векторів, отриманих в результаті рівня BLSTM, а шар softmax приймає вектори з “щільного” шару в якості вхідних даних і відповідає за представлення результату класифікації, який забезпечує зворотний зв'язок для оновлення нейронної мережі. Результатом фази навчання є нейронна мережа BLSTM з налаштованими коефіцієнтами моделі. На фазі класифікації на вхід подається вектор, що відповідає досліджуваному “зразку”, а на виході формується рішення класифікації.

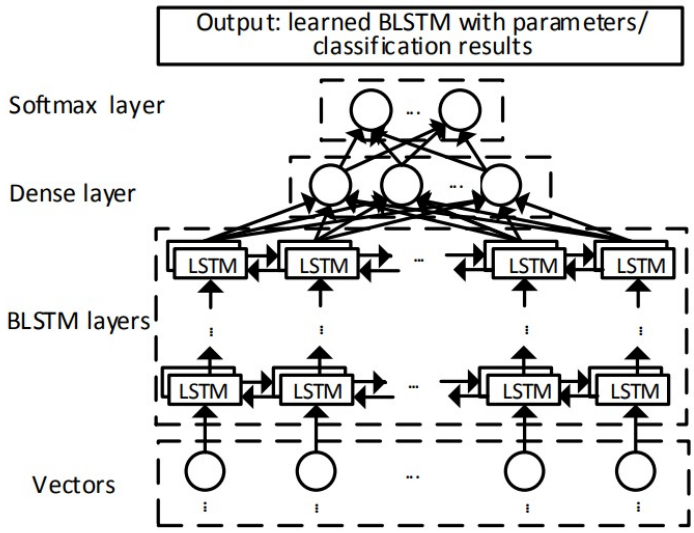


Рис. 4.5. Структурна схема нейронної мережі BLSTM

4.4 Визначення помилок безпеки у вихідному коді

В результаті, при аналізі вразливостей вихідного коду проекту виділяються:

1. Фаза виявлення та представлення у виді придатному для вводу в нейронну мережу;

2. Фаза навчання нейронної мережі;
3. Фаза класифікації (на іншій частині зразків, ніж ті, на яких було проведено навчання) (рис. 4.6).

1. Фаза виявлення та представлення

Під час фази виконуються наступні дії:

1) Попередня обробка початкового коду, яка включає використання бібліотек для отримання усіх залежностей в початковому коді та пошук початкових точок виклику можливо вразливих функцій.

2) Виділення код-гаджетів: виділення усіх початкових точок вразливих функцій в код-гаджеті, mapping коду та розмітка код гаджетів як вразливі або невразливі.

3)Представлення код-гаджетів у векторному виді за допомогою алгоритму word2vec.

2. Після проходження фази навчання нейронної мережі на виході є навчена (із встановленими коефіцієнтами згідно тренувального датасету із розмічених код-гаджетів, представлених у векторному виді) модель нейронної мережі.

3. Фаза виявлення. На цій фазі використовується вже навчена модель нейронної мережі, яка була отримана в результаті попередньої фази та вихідний код програми, представлений у виді, сприйнятному для моделі, який потрібно перевірити на наявність в ній помилок безпеки. В ході фази виконуються наступні дії:

1. Попередня обробка початкового коду;
2. Виділення код-гаджетів;
3. Векторне представлення код-гаджетів;
4. Використання готової моделі нейронної мережі для отримання результатів класифікації.

На виході моделі буде число в межах від 0 до 1, яке показує степінь уразливості відповідного код-гаджету, тобто 0 – не вразливий, 1 – вразливий. Варіант реалізації такого алгоритму розроблено в роботі [12].

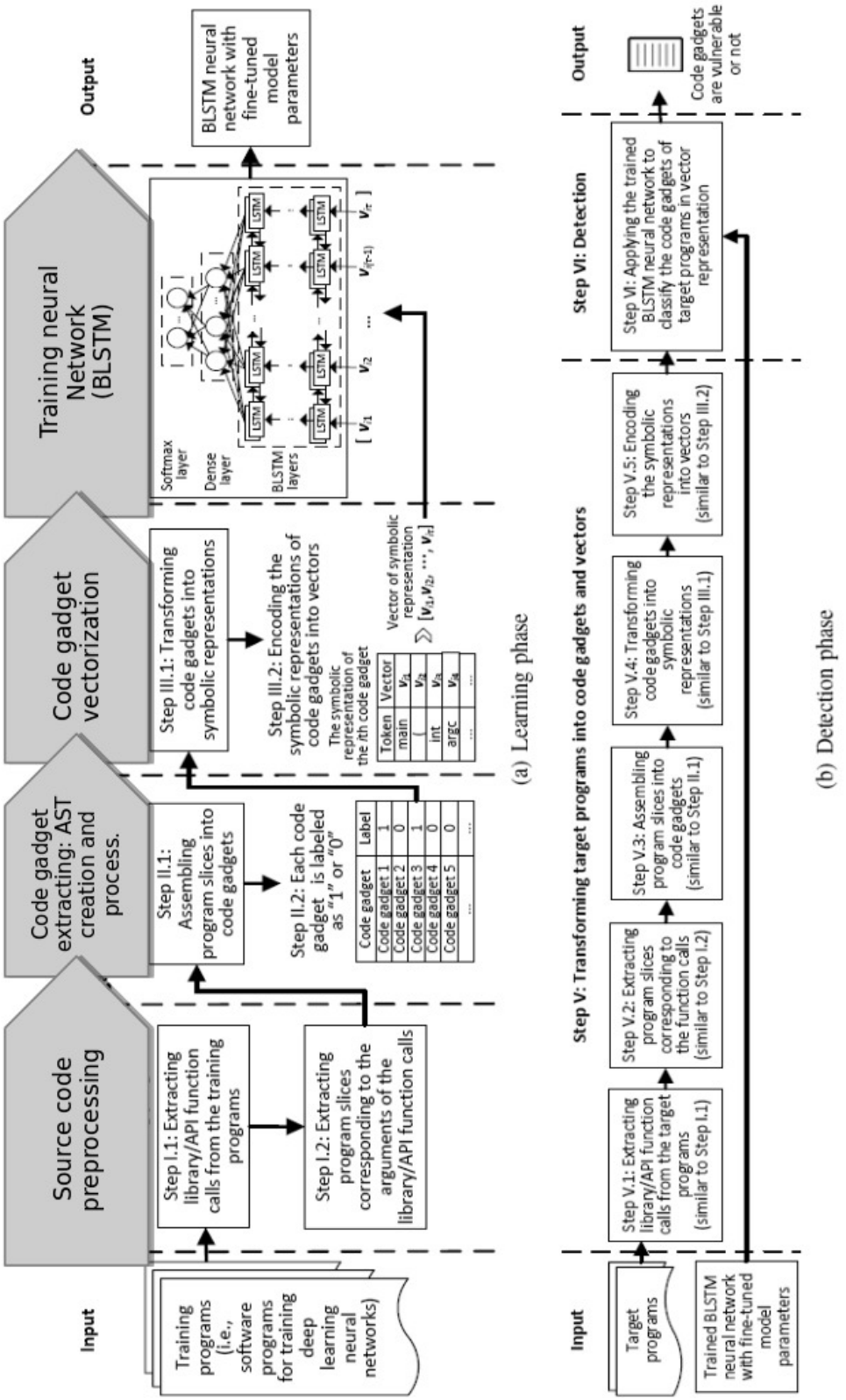


Рис. 4.6. Загальна архітектура системи: (а) Фаза навчання (б) Фаза виявлення [12].

4.5 Запитання до самоперевірки

1. Які основні етапи треба виконати, щоби представити вихідні текстові дані, у виді, придатному для обробки математичною моделлю у виді

нейронної мережі? Вважаємо, що в якості вихідних даних можуть бути: а) код; б) текстові повідомлення.

2. Для чого використовують структури типу абстрактного синтаксичного дерева при аналізі вихідних даних?
3. Яким чином необхідно підготувати датасет для навчання, щоби якість навчання була якомога вищою?
4. Які вимоги задачі треба врахувати при підборі типу моделі (зокрема, типу нейронної мережі)?
5. В чому полягає основна відмінність LSTM мереж від BLSTM? Наведіть приклади задач для яких застосовуються перші та другі мережі.

5 ЗАДАЧІ КЛАСИФІКАЦІЇ ТРАФІКУ ПРИСТРОЇВ ІНТЕРНЕТУ РЕЧЕЙ

5.1 Постановка задачі

У мережах типу мереж розумного міста може бути присутнім трафік від різних пристроїв інтернету речей. Частина з цих пристроїв характеризується тим, що протоколи мережного спілкування із ними не мають засобів автентифікації, і, навіть, явного визначення типу пристрою. Однак, з точки зору політики безпеки необхідно мати змогу відокремлювати пристрої різних типів один від одного (зокрема, по трафіку, який від них походить) для того щоби далі обробляти дані від них відповідно до окремих правил політики безпеки, і відповідно налаштовувати спілкування із ними згідно цих правил. Причому такий поділ має відбуватись у режимі реального часу. Тобто, на міжмережному екрані має діяти попередньо навчена модель, яка дасть змогу розпізнавати пристрої відповідного класу, а результати зможуть використовуватись при заданні відповідних правил безпеки для роботи із трафіком цього пристрою.

Основні передумови

Розглянемо випадок мережі (наприклад, локальної), в якій пристрої мають підключення до глобальної мережі. Для того щоби розвантажити центр обробки запитів від таких пристроїв, можна запропонувати метод класифікації “на краю” мережі (що відповідає парадигмі Edge Computing). Зокрема, розглянемо метод класифікації пристроїв, який може бути реалізований на мережевому шлюзі.

Види зв’язку пристроїв Інтернету речей з іншими пристроями (віддаленими серверами для аналізу даних, локальних пристроїв для передачі інформації) можна розділити на такі види:

1) Зв'язок між пристроями (device-to-device, D2D). Сюди входить прямий зв’язок між фізичними об’єктами локальної мережі. Цей вид зв’язку енергоекономічний та швидкодіючий.
2) Зв'язок між пристроями і інфраструктурою (device-to-infrastructure, D2I). Сюди входить зв'язок між пристроями однієї мережі (локальна мережа) і пристроями або службами іншої мережі (віддалена мережа). Даний тип зв'язку використовується для передачі великої кількості даних для подальшого поглибленого аналізу.

Пристрої IoT можуть мати можливості мережного зв’язку, сенсори та датчики для обміну інформацією із оточуючим середовищем – а отже,

можуть представляти собою достатньо різноманітну множину об’єктів – від планшетів до кондиціонерів. Класифікація пристроїв на два глобальні класи дасть уявлення про функціональні можливості пристроїв: а) нескладні пристрої; б) складні пристрої .

Із цих двох класів більш вразливими є пристрої, що мають нескладну функціональність. А пристрої, що мають більш складну функціональність, навпаки, можуть містити у собі загрозу для більш слабких пристроїв, оскільки вони можуть здійснювати користувацьке налаштування базового функціоналу вузькоцільових пристроїв. Зазвичай, таке налаштування виконується безпосередньо адміністраторами мереж.

В цілому можна виокремити наступні класи пристроїв:

1. Вузькоцільові пристрої. Як правило, це ресурсо- обмежені пристрої (передавачі, камери з обмеженою та/або нескладною функціональністю, розумні лампи, лічильники тощо). Зазвичай, пристрої даного типу не надають користувацький інтерфейс, тому будь-яке початкове налаштування можливе за допомогою програмного забезпечення пристроїв з більш складною функціональністю за допомогою дротового або бездротового з'єднання.
2. Багатоцільові пристрої. Сюди будемо відносити пристрої з достатньо потужними програмно-апаратними ресурсами (смартфони, планшети, телевізори тощо). Вони надають користувачу необхідний інтерфейс комунікації (прикладні програмні забезпечення, операційні системи тощо).

Існує значна кількість робіт на тему класифікації пристроїв на відповідні класи. Значна їх частина стосується локального розділення пристроїв, що підключенню до мережі. Але існують роботи, присвячені задачі класифікації пристроїв на універсальні класи, що відокремлюють пристрої не за функціональним застосуванням, а за принципом роботи.

Одна із робіт пропонує використовувати наступні класи пристроїв Інтернету речей:

1. Контролери (сюди можна віднести: високотехнологічні маршрутизатори, контролери-посередники між пристроями Інтернету речей та виконують управління ними). Наприклад: *Samsung SmartThings Hub, Philips Hue Hub, Wink hub, Logitech Harmony Hub*.
2. Камери (пристрої, головним функціоналом яких є фото-відео зйомка та передача отриманих даних провідним або безпровідним способом). Наприклад: *Nest Camera, Belkin Netcam, Netgear Arlo Camera, D-Link DSC-5009L Camera, Logitech Logi Circle, Nest Cam IQ*.
3. Командні пристрої (нескладні електронні пристрої, призначені для

виконання людських команд відповідно до функціональних можливостей). Наприклад: *TP-ling WiFi Plug, Belkin WeMo Switch, TP-Link Smart WiFi LED Bulb, WeMo Crockpot, Roomba*.

4. Сенсори/датчики (пристрої, основним функціоналом яких є збір даних про стан навколишнього середовища та реагування на нестандартні або аномальні явища. Маються на увазі не складові інших пристроїв IoT, а самостійні пристрої). Наприклад: *Belkin WeMo motion Sensor, Nest Protect smoke alarm, Netatmo weather station*.
5. Гаджети (пристрої з різноманітними функціональними можливостями, призначені для роботи з користувачем). Наприклад: *Android Tablet, iPhone, iPad, Samsung SmartTV*.

5.2 Виділення рис для класифікації на основі потоку пакетів пристроїв Інтернету речей

Пристрої Інтернету речей генерують трафік (вхідний та вихідний) в залежності від певних функцій конфігурації та служб застосунків. Характер генерації трафіку кожного пристрою є унікальним. Це дає змогу виділити із трафіку поведінкові профілі, які складаються із характерних особливостей, чому присвячено роботу. На основі цих особливостей можна провести класифікацію, як запропоновано в [14].

Структура потоку мережевих пакетів

Потік мережевих пакетів може бути відображено у виді послідовності:

$$S = \{P^1, P^2, P^3, P^4, P^5, ..., P^j, ...\}$$

де P^j – відображає інформацію, отриману з j-того пакету. Кожен запис пакету P^j включає в себе інформацію, що відноситься до даного пакету та включає:

$$P^j = \{t^j, length^j, protocol^j, eth.src^j, eth.dst^j, others^j\} \quad (5.1)$$

де, t^j - час фіксації пакету; $length^j$ - довжина пакету; $protocol^j$ - тип протоколу; $eth.src^j$, $eth.dst^j$ – MAC-адреса джерела пакету та MAC-адреса кінцевого отримувача відповідно; $others^j$ – це інша інформація, в тому числі IP адреси, встановлені прапорці і т.і.

Пакети впорядковані за часом

$$t^1 < t^2 < \cdots < t^j$$

Розглядаючи мережу, що складається з N пристроїв, потік трафіку відображається як:

$$S = \{P^1, P^1, P^1, P^2, P^2, ..., P^j, ...\},$$

де P^j позначає i-й пакет пристрою d_n.

5.3 Аналіз потоку пакетів

Можна виділити такі характеристики потоку пакетів:

1. Час сну.
2. Середнє значення обсягів трафіку за активний час роботи.
3. Середній розмір пакету.
4. Пікова / Середня швидкість потоку пакетів.
5. Час активної роботи, тобто час активної генерації та приймання мережевих пакетів.
6. Кількість IP-адрес, з якими пристрій встановлює з'єднання.
7. Кількість типів протоколів, що використовується при комунікації.
8. Кількість DNS-запитів.
9. Інтервал DNS пакетів.
10. Інтервал NTP пакетів.

Однак, не всі вищевказані особливості можуть бути обрані у якості ідентифікатора класу. На рисунку 5.1 показано, що інтервали NTP та DNS кожного пристрою не можуть слугувати для точного встановлення типу пристрою.

З іншого боку, середній показник швидкості передачі пакетів, який може бути описаний наступним співвідношенням:

$$Rate = Active\ volume \div Active\ time,$$

де *Active volume* (значення кількості трафіку за активний час) - загальна сума завантажених і переданих байт; *Active time* (час активної роботи) - час неперервної передачі між першим і останнім надісланим пакетом. З цього співвідношення можна побачити, що швидкість буде корелювати із цими показниками. Тому, цей показник має бути відхилений, як надлишковий. Також на цей показник впливає конфігурація самої мережі.

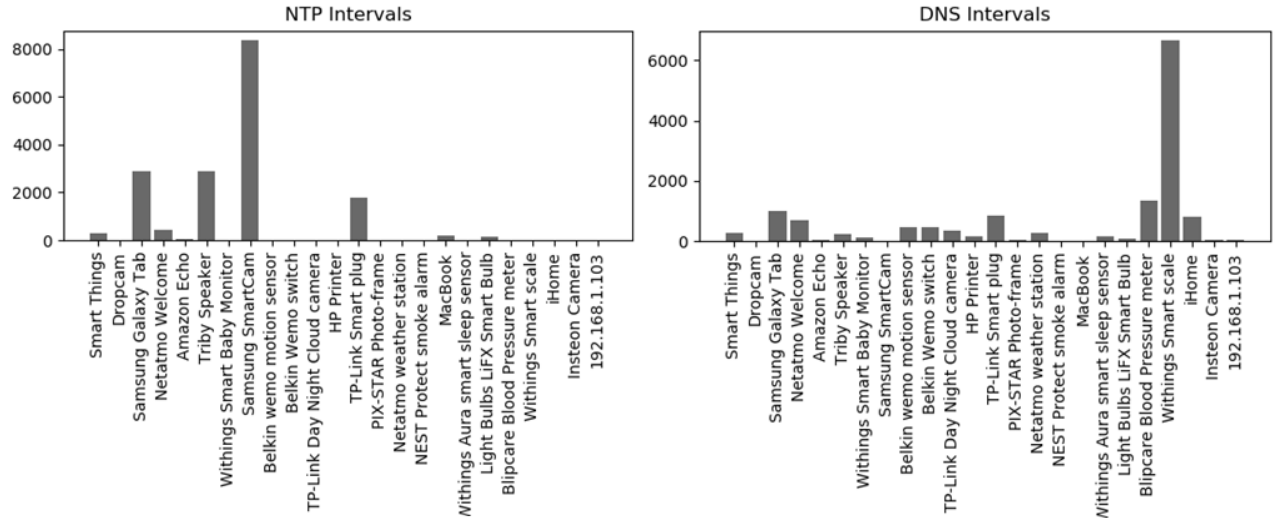


Рис. 5.1. NTP та DNS інтервали різних пристроїв в мережі [14].

Щодо часу сну кожного пристрою, можна зауважити, що велика кількість багатоцільових пристроїв за певних умов можуть вести себе аналогічно вузькоцільовим пристроям (рис. 5.2).

Отже, час сну теж не може бути однозначною ознакою.

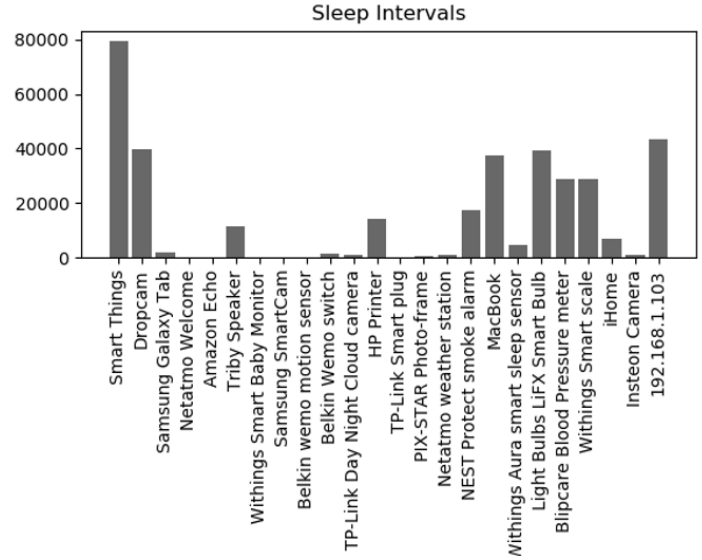


Рис. 5.2. Час сну IoT пристроїв [14]

Одними із надійних показників вважаються:

1. кількість DNS запитів, оскільки за спостереженнями кількість DNS-запитів від багатоцільових пристроїв Інтернету речей суттєво перевищує кількість DNS-запитів від вузькоцільових пристроїв Інтернету речей (рис. 5.3).
2. кількість унікальних імен доменів, з якими встановлюють з’єднання

пристрої інтернету речей.

3. Сила зв’язності багатоцільових пристроїв перевищує силу зв’язності вузькоцільових пристроїв. Сила зв’язності - це індекс зв’язків між даним пристроєм та іншими у мережі. (Це слідує із того, що багатоцільовий пристрій здатен підтримувати більшу кількість зв’язків із різноманітними іншими пристроями, ніж вузькоцільовий).
4. Поле User-Agent в HTTP заголовку присутнє в більшості випадків для багатоцільових пристроїв. Особливість може бути використана лише у випадку незашифрованого трафіку. По замовчуванню ця характеристика може бути встановлена в нуль, але якщо дане поле вдалось виявити в трафіку – то 1.

З урахуванням цього, поведінковий профіль пристрою може містити наступні показники:

1. Кількість унікальних DNS запитів.
2. Кількість типів протоколів, що використовуються в процесі комунікації.
3. Тип пристрою за допомогою поля user-agent.
4. Кількість унікальних з’єднань в локальній мережі.
5. Сила зв’язності пристроїв в локальній мережі.
6. Кількість доменних імен, з котрими пристрій має TCP з’єднання.
7. Кількість невідомих з’єднань.
8. Середній час TCP з’єднання.
9. Середній трафік за одну TCP сесію.
10. Кількість всіх з’єднань.

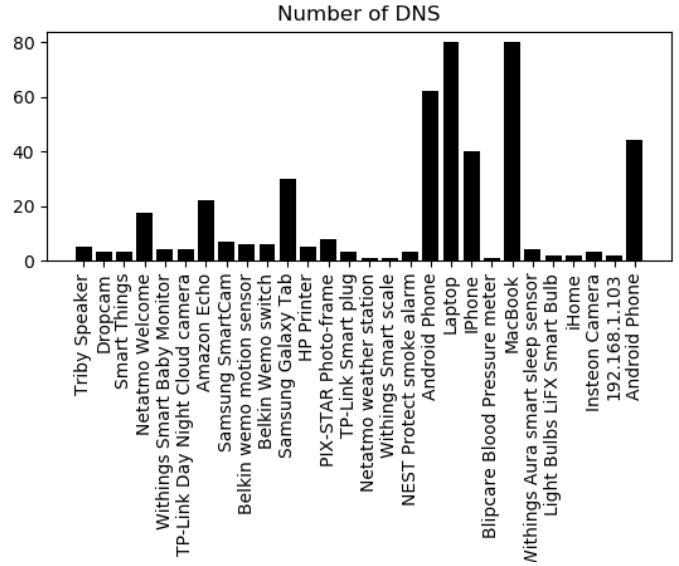


Рис. 5.3 – Кількість DNS запитів пристроїв в мережі [14]

Для обчислення сили зв’язності кожного пристрою, наприклад, можна

використати технологію Google‘s PageRank, що аналізує орграф зв’язності в мережі, кожне ребро якого має вагу, що дорівнює кількості пакетів згенерованих кожним пристроєм.

При обчисленні кожного з запропонованих показників необхідно враховувати пропускну здатність мережевого пристрою, в даному випадку мережевого шлюзу.

5.4 Вибір методу машинного навчання

Для бінарної класифікації пристроїв Інтернету речей за допомогою отриманого поведінкового профілю можна застосувати алгоритм навчання з учителем Random Forest, який дає стабільно високу точність. Він часто використовується через простоту та різноманітні області застосування (класифікація, регресія). Для дії алгоритма необхідно побудувати множину дерев прийняття рішень. Random Forest заснований майже на тих самих параметрах, що і класичне дерево рішень, але додає додаткову випадковість при конструюванні дерев. Замість пошуку найбільш важливої функції при розбитті вузла, він шукає кращу ознаку серед випадкової підмножини ознак. Це призводить до великої різноманітності, яке зазвичай призводить до кращої моделі.

Отже, в Random Forest алгоритм розщеплення вузла враховує випадкову підмножину ознак. Є можливість збільшувати випадковість побудови дерева, шляхом використання випадкових порогів для кожної ознаки. Натомість класичний алгоритм дерева рішень шукає найкращі значення порогів.

Алгоритм Random Forest дозволяє виміряти відносну важливість кожної ознаки в прогнозі. Вимірювання важливості ознак виконується шляхом спостереження за тим, наскільки вузли дерева, які використовують цю особливість, зменшують “забруднення” всіх дерев в лісі. Відповідний бал обчислюється автоматично для кожної ознаки після навчання, результати є зваженими і їх сума дорівнює одиниці.

Важливість ознак дає змогу вирішити, які з них можна чи необхідно відкинути. Це ті, що не вносять достатнього впливу в процес прогнозування. Велика кількість зайвих ознак - це не дуже добре для моделі, оскільки вона виявляється перенавантаженою.

Принцип роботи алгоритму показано на рис. 5.4.

Нехай вхідний набір даних складається з N векторів розмірністю M, де M – це кількість ознак, які подаються на вхід алгоритму. Також задається параметр m ($0 < m \leq M$) – кількість випадкових ознак, які обираються для

побудови дерев рішень, стандартно приймають $m = \sqrt{M}$ для задач класифікації та $M/3$ в задачах регресії. Цей параметр налаштовується у першу чергу, при достатньому числі дерев у лісі. Також задається загальна кількість дерев, які треба побудувати.

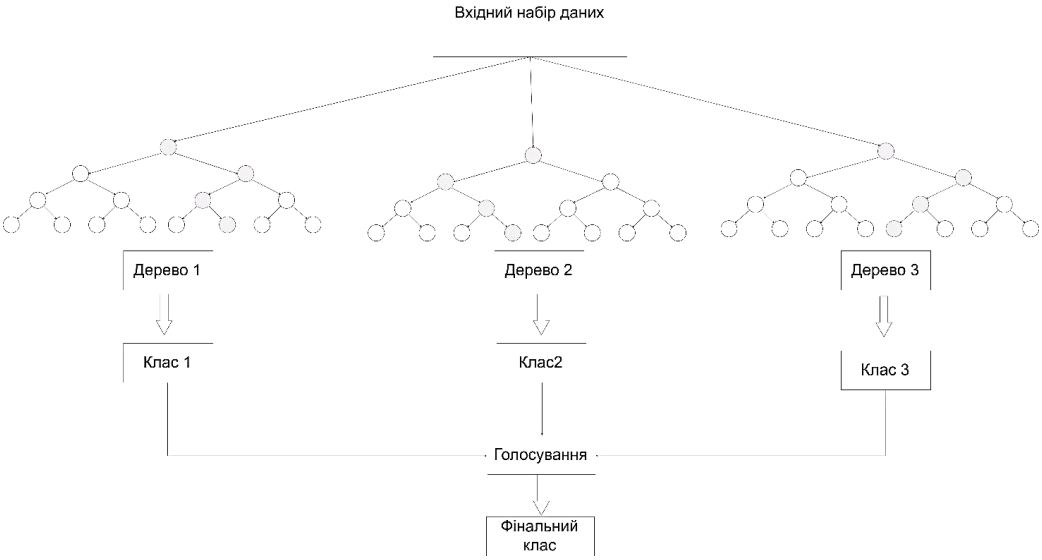


Рис. 5.4 . Принцип роботи алгоритму Random Forest

Дерева рішень будуються випадковим чином, незалежно. Процедура побудови дерева рішень наступна:

1. Побудова випадкового набору даних, з векторів *m* випадково вибраних особливостей.
2. Генерація дерева рішень, що класифікує вектори побудованого набору даних. При створенні вузла дерева робиться вибір ознаки з *m* випадково обраних. Дерево будується до повного вичерпання набору даних та не піддається процедурі відсікання.
3. Проведення “голосування” за побудованими деревами. Обирається найбільш відповідний клас за заздалегідь заданим критерієм та генерується остаточний результат.

Алгоритм реалізований у бібліотеці scikit-learn (Рис. 5.5)

```
class
sklearn.ensemble.RandomForestClassifier(n_estimators=10,
criterion='gini', max_depth=None, min_samples_split=2,
min_samples_leaf=1, min_weight_fraction_leaf=0.0,
max_features='auto', max_leaf_nodes=None,
min_impurity_split=1e-07,
bootstrap=True, oob_score=False, n_jobs=1,
random_state=None, verbose=0, warm_start=False,
class_weight=None)
```

Рис. 5.5. Основні параметри алгоритму [15]

Далі реалізуються дії:

```
from sklearn.ensemble import RandomForestRegressor from sklearn.metrics import roc_auc_score
# далі - (X, y) - навчання, (X2, y2) - управління
# модель - розглядається регрессор
model = RandomForestRegressor(n_estimators=10, oob_score=True, random_state=1)
model.fit(X, y) # навчання
a = model.predict(X2) # прогноз
print ("AUC-ROC (oob) = ", roc_auc_score(y, model.oob_prediction_))
print ("AUC-ROC (test) = ", roc_auc_score(y2, a))
```

Тут $n\_estimators$ – кількість дерев (можна дослідити, як змінюється якість при збільшенні кількості, зазвичай збільшення цього параметру позитивно впливає на точність класифікації).

Параметр $max\_features$ –кількість ознак для вибору розщеплення. Збільшення параметру впливає на збільшення часу побудови лісу, дерева стають менш різноманітними.

Параметр $min\_samples\_split$ - як правило, не сильно впливає на результат, і можна залишити його заданим по замовчуванню. Зміна значень параметру не сильно впливає на якість, і необхідний оптимум знайти часто не вдається. Збільшення параметру впливає незначно на спад якості навчання, а час побудови випадкового лісу дещо скорочується.

Параметр $min\_samples\_leaf$ – можна залишити заданим по замовчуванню. В регресійних задачах рекомендовано використовувати значення рівним 5.

Максимальна глибина дерев задається параметром *max_depth*. Параметр впливає на швидкість побудови лісу, а також на якість навчання та управління. Рекомендовано використовувати максимально можливу глибину, крім випадків коли кількість об'єктів надто велика, і при побудові виходять надто глибокі дерева, що вимагає обчислювальних витрат. Неглибокі дерева рекомендовано використовувати у задачах із великою кількістю випадкових "викидів", які зашумлюють основну масу даних.

Критерій розщеплення задається параметром $criterion$. В бібліотеці sklearn для регресійних задач реалізовано критерії mse та mae, які відповідають функціям помилки, які вони мінімізують. У більшості задач використовують mse. Для класифікації реалізовані критерії gini та entropy, що відповідають критеріям розщеплення Джині та ентропійному критерію.

Також у різних реалізаціях бібліотек може бути заданий параметр samplesize, який регламентує, зі скількох дерев робити підвиборку для побудови

кожного дерева. Зазвичай рекомендовано обирати підвиборку із “поверненням”: bootstrap=True (це задає “bagging” – bootstrap aggregating).

В бібліотеці sklearn є можливість задати кількість процесорів, на яких відбувається побудова лісу: n_jobs=1 – використовувати один процесор, n_jobs=-1 – використовувати максимально можливу кількість процесорів. Налаштування гіперапараметрів для scikit learn описано в [15].

5.5 Багатокласова класифікація на основі нейронних мереж

В якості моделі нейронної мережі для мультигрупової класифікації будемо використовувати RNN-CNN конструкцію.

LSTM - це різновид рекурентних нейронних мереж (RNN), спеціально розроблена для обробки послідовних даних (рис. 5.5). Вона є дуже ефективною для багатьох застосунків, починаючи від розпізнавання мови, розпізнавання рукописного введення і виявлення аномалій.

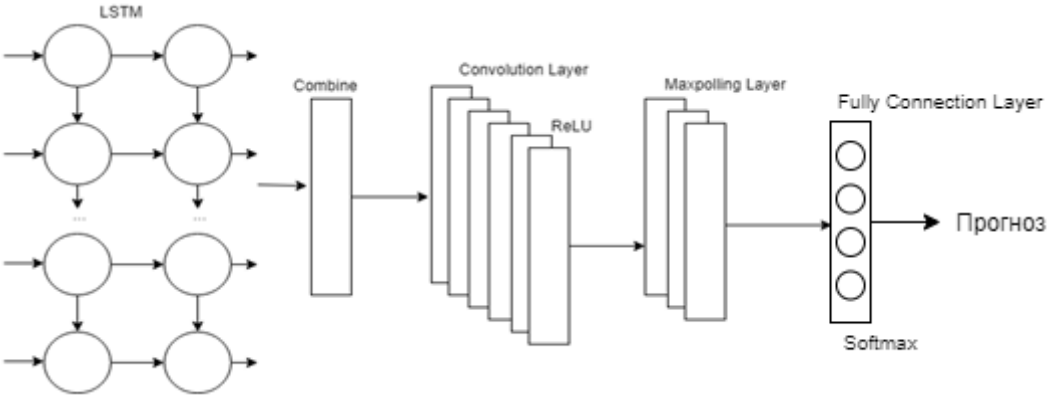


Рис. 5.5. Схема нейронної LSTM-мережі

Для базової LSTM клітинки, вхід включає в себе дві частини x_k та h_{k-1}, де h_{k-1} − вихід останньої клітинки на тому ж рівні. Обчислення LSTM клітинки задається наступними рівняннями:

$$g^{(k)} = tan\mathrm{h}(W^{gx}x_k + W^{g\mathrm{h}}\mathrm{h}_{(k-1)} + b^g)\ i^{(k)} = \sigma(W^{ix}x_k + W^{i\mathrm{h}}\mathrm{h}_{(k-1)} + b^i)$$

$$f^{(k)} = \sigma(W^{fx}x_k + W^{f\mathrm{h}}\mathrm{h}_{(k-1)} + b^f)$$

$$o^{(k)} = \sigma(W^{ox}x_k + W^{o\mathrm{h}}\mathrm{h}_{(k-1)} + b^o)\ s^{(k)} = g^{(k)} \odot i^{(k)} + s^{(k-1)} \odot f^{(k)}$$

$$\mathrm{h}^{(k)} = \tanh(s^{(k)}) \odot o^{(k)}\ \textit{Convolution Layer}.$$

Виходом LSTM є t векторів. Вектори об’єднуються в t стовпців, формуючи двовимірний вектор. Конволюційний шар використовує кілька

фільтрів. Потім вихідний результат цих фільтрів передаються в функції лінійної або нелінійної активації, такі як ReLU, для формування вихідних даних.

Maxpooling Layer. Вихідний сигнал потім безпосередньо подається в Maxpooling Layer. Maxpooling Layer зменшує розмірність входу, вибравши тільки максимальне значення з $n \times n$ об'єктів, де $n \times n$ - розмір фільтру.

Fully Connection Layer. Після Maxpooling Layer дані знову перетворюються в вектор і передаються в Fully Connection Layer з операцією відсіву перед подачею на останній рівень моделі. Операція відсіву допомагає запобігти перенавчанню. В останньому рівні функція softmax вибирається в якості активної функції для обчислення ймовірностей різних класів. Клас з найбільшою ймовірністю буде остаточним прогнозом для вхідних даних.

Оскільки кожен пристрій генерує великий обсяг трафіку, необхідно виконати сегментацію необробленого трафіку перед видобутком ознак та навчанням моделі. Інтенсивність трафіку є різною для різних пристроїв, та змінюється для різних проміжків часу (рис. 5.6). До того ж, однотипові пакети можути бути надмірними, а час витрачений на їх обробку, призведе до додаткових обчислювальних витрат, це і є поясненням необхідності сегментації шляхом розбиття загального потоку пакетів пристроїв Інтернету речей на складові фіксованої довжини часу *T*.

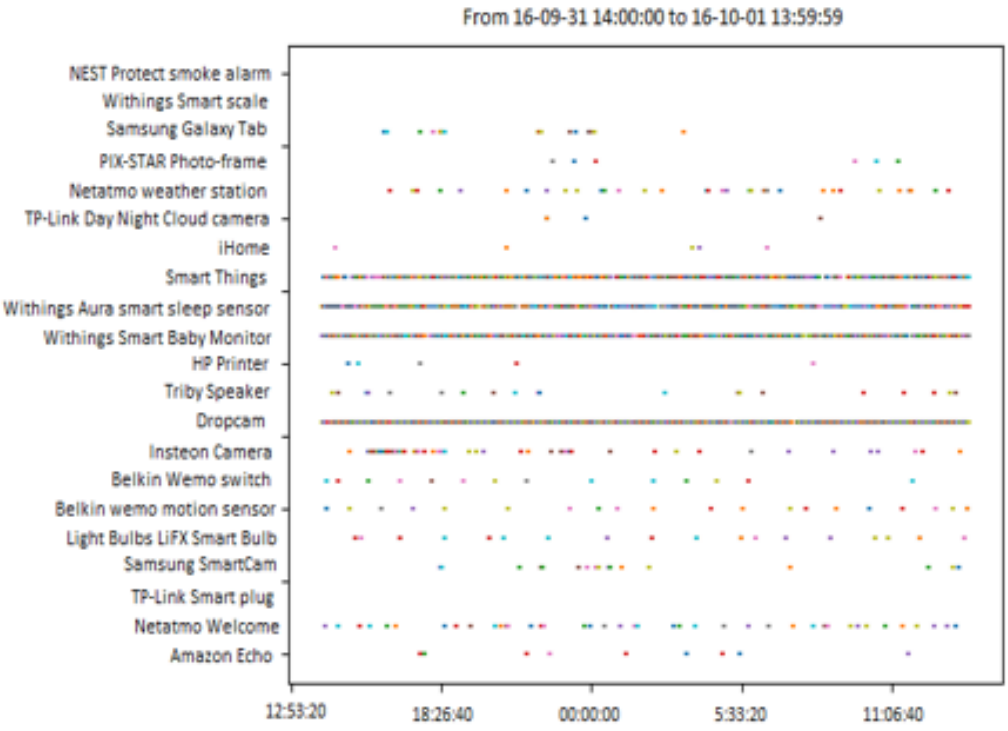


Рис. 5.6. Навантаження трафіку різних пристроїв Інтернету речей [14]

Особливості кожного сегменту можуть включати наступні показники:

1. Кількість пакетів користувача.
2. Середня довжина пакетів користувача.
3. Максимальне значення довжини пакетів користувача.
4. Кількість службових пакетів.
5. Середня довжина службових пакетів.
6. Максимальна довжина службових пакетів.
7. Кількість унікальних MAC.
8. Кількість DNS пакетів.
9. Кількість NTP пакетів.
10. Кількість унікальних IP адрес.
11. Кількість ICMP пакетів.
12. Кількість ARP пакетів.
13. Кількість HTTP пакетів.
14. Кількість HTTPS пакетів.
15. Кількість одержаних пакетів.
16. Кількість надісланих пакетів.
17. Кількість mDNS пакетів.

В результаті, кожен сегмент може бути представлений як d_1 вектор особливостей x^i .

5.6 Використання результатів класифікації з точки зору кібербезпеки

Одержані результати можуть бути використані для:
1)налаштування правил міжмережного екрану згідно класів пристроїв.
2)задання окремих політик безпеки для різних пристроїв інтернету речей.
3) налаштування віртуальних мереж в “розумних” середовищах для відокремлення різних класів IoT-пристроїв. Кількість необхідних VLAN визначається кількістю класів IoT-пристроїв. Для кожної VLAN має бути визначений окремий пул IP-адрес, який дозволить використовувати в мережі набагато більшу кількість пристроїв. Це спрощує обробку трафіку, оскільки з’являється можливість застосовувати правила міжмережного екрана до конкретних підмереж.

5.7 Запитання до самоперевірки

1. Які задачі аналізу безпеки комп’ютерних систем та мереж можна розв’язати методами штучного інтелекту? Наведіть приклади.
2. Які методи доречно застосовувати для: а) двокласової; б)

мультикласової класифікації трафіку пристроїв IoT? Яка мета такої класифікації в контексті задач кібербезпеки?

3. Які характеристики можна використовувати для класифікації трафіку пристроїв IoT?

4. Якими є переваги алгоритму Random Forest, в яких задачах кібербезпеки доречне його використання?

6 SMT/SAT РОЗВ'ЯЗНИКИ В ЗАДАЧАХ КІБЕРБЕЗПЕКИ

6.1 Основні визначення

Задача виконуваності формул у теоріях (SMT - Satisfiability modulo theories) представляє собою узагальнений варіант задачі визначення виконуваності булевих формул.

SMT-формула - це формула в деякій формальній логіці, яка включає функції і предикатні символи. Ці функції та символи можуть мати певну інтерпретацію. В багатьох задачах необхідно визначити, чи є формула виконуваною чи ні. На відміну від задачі про виконуваність булевих формул, замість булевих змінних SMT-формула може містити довільні змінні, а предикати - це булеві функції від цих змінних. Стандартна задача SAT – знаходити множину змінних, які, підставлені у булеві вирази, дадуть в результаті "true".

За рядом досліджень показано, що тестування програмного забезпечення може зводитися до завдання виконуваності булевих формул (SAT) і виконуваності формул в теоріях (SMT) [17].

Алгоритми можуть бути представлені у вигляді певних булевих формул, при обмеженнях на можливі комбінації вхідних параметрів.

Сучасні SAT-розв'язники здатні обробляти формули з сотнями тисяч змінних, що показує їх перспективність для використання у великій кількості задач. Розв'язники SMT теж є дуже потужними, зокрема Barcelogic, CVC, MathSAT, Yices і Z3, створені для найбільш поширених логік (теорій).

В сучасних умовах, при наявності потужного програмно-апаратного забезпечення для обчислень SAT-розв'язники можуть використовуватись для аналізу програмного забезпечення.

Основні застосування SAT/SMT в сфері безпеки програмного забезпечення:

- Статичний аналіз на вразливості програмного коду;
- Створення експлойтів (для задач тестування);
- Вивчення захисту від копіювання, дослідження та модифікації;
- Аналіз непротиречивості та повноти політики безпеки, формальна верифікація відсутності вразливостей протоколів, програмного забезпечення тощо [18].

Однією із проблем, яка виникає при застосуванні розв'язника, є генерація обмежень. На вхід розв'язника необхідно подавати обмеження на стани та вхідні параметри, одначе, задати такі обмеження для реальних систем є досить складною задачею, яка інколи не має точного розв'язку. Це накладає

обмеження на істинність відповідей, які може надати розв’язник.

SMT дозволяє більш природно моделювати властивості коду, ніж SAT при розв’язанні задач аналізу програмних застосунків.

Розв'язники SMT можуть використовуватись в якості «чорного ящика», на вхід якого подається задача, сформульована у вигляді булевого виразу, а на вихід видається відповідь на подану задачу.

6.2 Перевірка вразливостей програмного коду

Питання статичного аналізу програм пов’язані із існуючими досить давно питаннями оптимізації компіляторів. Статичний аналіз використовується для аналізу бінарних файлів, а також, з іншого боку, і вихідного коду на наявність вразливостей. Статичний аналіз може використовуватись як для пошуку механічних помилок, так і для пошуку вразливостей програмного коду.

Оптимізаційні компілятори повинні знати характеристики компільованої програми, щоби в результаті згенерувати ефективний код. Такими властивостями можуть бути [19]:

1. Наявність шматків “мертвого коду”, наприклад, функцій, які не викликаються з main. Якщо так, код піддається оптимізації за рахунок винищення таких шматків.
2. Наявність виразів всередині циклів, значення яких залишається сталим. Якщо так, код піддається оптимізації за рахунок виносу подібних виразів за межі циклу.
3. Залежність змінної від вхідних даних програми. За умови відсутності залежності, значення змінної можна попередньо обчислити під час компіляції.
4. Діапазони змін значень змінних. При цьому можна керувати вибором представлення змінної під час виконання.
5. Наявність покажчиків на несуміжні структури даних у пам'яті. Це може бути підставою для паралельної обробки відповідних фрагментів програми.

Найуспішніші інструменти аналізу, розроблені для виявлення помилок (або перевірки відсутності помилок), спрямовані на загальні властивості коректності, які застосовуються до більшості або всіх програм, написаних на певних мовах програмування. В деяких мовах програмування, зокрема C, існує ряд типових слабкостей, які можуть призводити до критичних уразливостей безпеки. У більш безпечних мовах, таких як Java, такі помилки, як правило, менш серйозні, але вони все одно можуть спричинити збої

програм.

Прикладами таких властивостей є:

1. Чи існує вхід, який веде до нульового покажчика розмежування, ділення на нуль або переповнення арифметики?
2. Чи всі змінні ініціалізовані перед їх читанням?
3. Чи завжди доступ до масивів здійснюється в їх межах?
4. Чи можуть існувати покажчики на звільнену пам'ять?
5. Чи завершується програма на кожному введенні?

Інші властивості правильності залежать від специфікацій, наданих програмістом для окремих програм (або бібліотек), наприклад, чи всі твердження гарантовано матимуть успіх? Твердження виражають специфічні властивості коректності програми, які передбачається мати у всіх виконаннях.

В програмному забезпеченні на основі мобільних платформ та веб-застосунків дуже важливі властивості перевірок потоків інформації:

1. Чи сприймає ПЗ запити користувачів до операцій файлової системи без належної фільтрації та перевірки, що призводить до порушень конфіденційності та цілісності.
2. Чи надається доступ невповноваженим користувачам до інформації з обмеженим доступом, внаслідок помилок проектування ПЗ. Це є порушеннями конфіденційності.

В ПЗ, яке забезпечує розпаралелювання, розподілення обчислень та ПЗ на основі моделей виконання, керованих подіями, необхідно знати поведінкові особливості ПЗ:

1. Чи присутня обробка даних у швидкісному режимі, режимі реального часу? Чи різні потоки можуть використовувати спільні ресурси без належної синхронізації?
2. Чи може програма (або частини програми) зайти в глухий кут? Це питання виникає для багатопотокових програм, які використовують блокування для синхронізації.
3. Сучасні інтегровані середовища розробки проводять різні види аналізу програм для підтримки налагодження, рефакторингу та розуміння програм. Сюди входять такі питання, як: Які функції можливо викликати в певному рядку, або навпаки, звідки можна викликати певну функцію; на яких кроках програми змінній може бути присвоєне поточне значення; чи може значення однієї змінної впливати на значення іншої змінної? Під час налагодження такі питання виникають в процесі пошуку помилок.
4. Які типи значень може мати змінна x? Такі питання характерні для

нетипізованих мов програмування, наприклад OCaml, JavaScript або Python.

Доведення безпечності певного протоколу, перевірка безпечності політики безпеки, безпечності певного алгоритму функціонування програми виконується методами формальної верифікації. Основними методами верифікації програми є доведення теорем , абстрактна інтерпретація та перевірка моделі. Ці задачі теж можуть виконуватись із використанням SAT/SMT розв’язників.

Задача тестування програмного забезпечення може зводитися до завдання виконуваності булевих формул (SAT) і виконуваності формул в теоріях (SMT).

SMT-розв'язники можуть використовуватися для формальної та неформальної перевірки коду. Різниця між неформальною та формальною перевірками полягає в наступному:

- мета неформальної перевірки – це доведення відсутності вразливостей коду, доступних зловмиснику.
- мета формальної перевірки, для чого використовують в основному SMT- розв'язники- це перевірка функціональної коректності програмного забезпечення.

Застосування SAT/SMT - розв’язників для статичної перевірки коду здійснюється у фреймворку Triton. Triton призначений для реалізації підходу конколічного виконання (з англ. concolic execution framework), реалізований як Pintool - засіб аналізу програмного забезпечення для платформ Windows та Linux. Цей фреймворк надає можливість використовувати розширені можливості динамічного бінарного аналізу:

1) Механізм символьного виконання;
2) Представлення семантики SMT;
3) Інтерфейс з SMT-розв’язником ;
4) Механізм taint -аналізу;

та інші.

Taint-аналіз дозволяє поширити по програмі помічені дані. Дане завдання є ключовим для інформаційної безпеки, оскільки саме за допомогою розповсюдження помічених даних виявляються уразливості, пов'язані з ін’єкціями даних (SQL- ін’єкції, міжсайтовий скриптинг, підробка файлового шляху і так далі), а також з витоком конфіденційних даних (небезпечні дії з паролем, небезпечна передача даних).

Приклади задач кібербезпеки, які можна реалізувати за допомогою механізмів використаних в цьому фреймворку:

1. Аналіз слідів конкретних даних – вміст регістрів та значення пам'яті в кожній точці програми;
2. Символьне виконання - символьне вираження регістрів та пам'яті в кожній точці програми;
3. Виконання символьного фаззингу;
4. Генерація та вирішення обмежень;
5. Регістри виконання та модифікація пам’яті ;
6. Повторне відтворення слідів дій безпосередньо в пам’яті.

Частково в цих задачах можуть бути задіяні і SAT/SMT –солвери.

Цікавим є механізм символьного виконання, який дозволяє виконувати програми із використанням символьних змінних замість конкретних значень. Саме символьне виконання дозволяє перетворювати семантику програми в логічну формулу. Символьне виконання може побудувати і зберегти формулу шляху програми. Шлях у даному випадку - це послідовність інструкцій ($T = (Ins_1, Ins_2, ..., Ins_n)$). Інструкції представляються символьними виразами. Обчислюючи формулу та її заперечення, ми можемо пройти всі шляхи та «покрити» весь код, на відміну від конкретного виконання, яке покриває тільки певний шлях (для всіх шляхів це може виявитись обчислювально складною задачею). Символьний вираз передається розв’язнику SMT для генерування конкретного значення.

Вся семантика інструкцій представлена через SMT-LIB. SMT-LIB [20] - це міжнародна ініціатива, спрямована на сприяння дослідженню та розробці в рамках SMT.

Taint-аналіз може надавати інформацію про те, якими регістрами та адресами пам'яті користувач оперує в кожній точці програми. Цей вид аналізу допомагає налаштувати символьні змінні (символьна змінна - це по суті область пам'яті, якою користувач може керувати), обмежує механізм символьного виконання відповідною частиною коду. У кожній інструкції гілки ми безпосередньо знаємо, чи може користувач пройти обидві гілки (це в основному використовується для покриття коду). Мета цього аналізу - визначити чи є регістр або пам’ять поміченими. Тоді як метою символьного механізму є побудова символьних виразів на основі семантики інструкцій. Ці підходи допомагають здійснити мету механізму розв’язника - створити модель виразу (умову шляху).

Зокрема, алгоритм дій при створенні моделі може бути наступний:

1) Якщо ціль не помічена - модель не формується.
2) Якщо механізм розв’язника повертає «unsat» - помічені входи не можуть впливати на керуючий потік, щоб пройти цей шлях.

3) Якщо механізм розв’язника повертає «sat» - шлях може бути пройдений, використовуються фактично помічені входи.
4) Модель на виході дає набір конкретних входів для цього шляху.

Прикладом подібного розв’язника є Z3, всередині якого зберігається стек формул та декларацій, наданих користувачем. Від наданих формул та декларацій залежить ефективність SMT- розв’язника. На даний момент SMT-оцінювач працює досить повільно, і відповідно втрачається багато часу для оцінки виразів. Одним із інструментів, який повинен покращити швидкість оцінювача, є спрощувач SMT. В подальшому дослідниками планується розробити алгоритми для спрощення логічних структур, що складають основу SMT. Мета полягає в тому, щоб зареєструвати деякі правила перетворення виразів перед тим, як відправляти вирази розв’язнику. Це дозволить скоротити час, необхідний на обробку програмного коду.

6.3 Створення експлойтів

Розробка експлойтів – відносно нова галузь дослідження. Вона може використовуватись як частина offensive security, так і для задач тестування ПЗ. Відповідні властивості та методи також використовуються і зловмисниками.

Задачі кібербезпеки, які розв’язуються тут, можна віднести до двох категорій:

- автоматизована генерація вхідних даних, спрямованих на захоплення потоку управління системою,
- автоматизована генерація шкідливих корисних навантажень.

Робота в першій категорії спиралася на системи виконання для генерації обмежень. Такі системи відстежують семантичні взаємозв'язки та обмеження між символічними вхідними даними та всіма іншими байтами в пам'яті програм та регістрах центрального процесора. Використовуючи цю інформацію, можна генерувати формули SMT, які задають питання щодо потенційних значень таких байтів.

Системи генерації експлойтів використовують цю можливість, щоб перевірити, чи може зловмисник контролювати дані, що відповідають потенційно чутливим покажчикам.

Гаджет - це послідовність інструкцій у межах спільної бібліотеки або виконуваного файлу в цільовій програмі, яка виконує деякі корисні обчислення і закінчується передачею потоку управління наступному гаджету в послідовності.

Для реалізації експлойта використовується підхід Return-Oriented

Programming, за якого гаджет завершується RET (це інструкція повернення). У процесі кібератаки зловмисники роблять так, щоб усі гаджети виконувались через інструкції повернення, а для цього вони певним чином формують їх ланцюжки (послідовності) – їх називають ROP-ланцюжками. Пошук необхідних гаджетів у коді програми чи бібліотеки може виконуватись автоматизовано. Колекція таких гаджетів, як правило, поєднується в ланцюги для виконання певного завдання, наприклад, зміни дозволів доступа до сегмента пам'яті, копіювання чи ін., за задумом зловмисника.

Наразі, у фреймворках з автоматизованої побудови ROP-ланцюжків використовуються можливості SMT- солверів. Наприклад, інструменти фреймворку знаходять доступні гаджети як вхідні змінні, аналізують семантику введення (доступний набір гаджетів), для забезпечення непротиречивості роботи ланцюжка будується граф виконання програми, по якому знаходяться всі доступні доступні шляхи, а потім використовується розв’язник, для зворотного обчислення цих шляхів; ланцюги ROP генеруються автоматично.

Зокрема, при побудові логічної моделі визначається функція f(x) як значення регістру, де x означає позицію. Програма може бути перетворена на логічну математичну модель, тобто модель, доступну для аналізу засобами SMT, з якої ми можемо перевірити задовільність SMT. Відповідно до базової теорії, якщо є присвоєння, яке робить формулу істинною, то формула виконується. В іншому випадку формула є незадовільною. За допомогою моделі можна встановити зв’язок вмісту позиції, тобто адреси пам’яті, де дані повинні бути записані; із відповідними вхідними даними. Розв’язник повертає істинне значення, що означає що вхідне значення робить формулу істинною, і хибне значення, що означає незадовільність вхідного значення. Тобто, модель набору рішень дає відповідність між набором позицій та набором вхідних даних.

6.4 Перевірка безпеки мережі

За допомогою розв’язників можна вирішувати ряд задач щодо політики безпеки комп’ютерних мереж [19].

Якщо у нас є досить складна мережа, завжди актуальною задачею залишається, чи є ця мережа захищеною проти певного вида атак чи ні. Для розв’язання цієї задачі можна використовувати логічний граф атак, вершини якого представляють собою такі види: a) вершина, що відповідає привілеям (права користувача, ftp, rsh і т.д.); b) вершина, що відповідає потенційній

можливості використанню експлойта на хості, оскільки хост містить вразливості; с) вершина, що відповідає налаштуванням конфігурації, що протидіють атаці (правила файрволу, IDS-конфігурація, тощо).

За допомогою інструментів типа MulVAL є можливим побудувати відповідний граф для мережі. MulVAL – це інструмент аналізу безпеки, який, враховуючи початкові конфігурації мережі (машини, активні служби, доступність між хостами тощо) та базу даних відомих вразливостей, може визначити всі потенційні шляхи атак, за допомогою яких зловмисник може використовувати систему. Вхідними даними для аналізу є конфігурація хоста, конфігурація мережі, дані про користувачів мережі, модель взаємодії всіх компонентів та політика захисту мережі.

Граф атаки в мережі може бути представлений у вигляді булевої формули. Ця формула може бути подана на вхід SMT- розв'язника, при відповідних обмеженнях, які характеризують особливості функціонування мережі. Розв'язник може надати відповіді на питання реалізованості атаки в мережі із заданою конфігурацією, питання непротиречивості та достатності існуючих правил, засобів політики безпеки. Однак, врахування всіх можливих факторів успішності атаки при створенні математичної логічної моделі є складною задачею, яка потребує високої кваліфікації.

6.5 Запитання до самоперевірки

1. В яких задачах кібербезпеки мереж можливе застосування SMT – розв`язників?
2. Проаналізуйте можливості розв'язника Z3 як інструменту підтримки експертних рішень.

7 ШТУЧНИЙ ІНТЕЛЕКТ У ЗАДАЧАХ ЗАБЕЗПЕЧЕННЯ ДОСТУПНОСТІ ТА САМОВІДНОВЛЕННЯ КРИТИЧНИХ КОМПОНЕНТІВ СИСТЕМИ

7.1 Методи моніторингу та відновлення компонентів ОС

Методи моніторингу та відновлення наразі досягли суттєвої розвиненності. В складі операційних систем ці методи обов'язково повинні використовуватись для аналізу працездатності складових ОС та прийнятті рішень при відмовах різних типів. Відповідні методи є запорукою доступності застосунків та операційної системи.

Пов'язані із методами моніторингу методи самовідновлення можуть бути реалізовані на рівні прикладних застосунків, системного ПЗ, на рівні апаратного забезпечення.

В практиці моніторингу стану компонентів ОС та доступності мережних компонентів використовуються підходи на основі: часу існування (TTL); тестувань, перевірок доступності; перевірок відповідності подій та/або файлів певному шаблону чи сигнатурі.

Відновлення компонентів ОС може відбуватись на основі таких підходів реактивного відновлення; профілактичного відновлення (які, відповідно, відбуваються як реакція на певне порушення функціонування, або із профілактичною метою).

Моніторинг на основі TTL

Перевірки на основі часу TTL (time-to-live) засновані на припущенні, що припускають, що апаратне забезпечення або програмне забезпечення буде через встановлені проміжки часу підтверджувати своє нормальне функціонування. Система моніторингу приймає відповідні сигнали і аналізує останній стан. Якщо цей стан не був оновлений протягом певного часу, система запускає механізм відновлення до попереднього вдалого стану. При наявності збоїв процес може завершатись не вдало, не відправляти сигнали про успішне функціонування, тоді після завершення TTL система моніторингу та поновлень відреагує на це.

Моніторинг на основі перевірки доступності

В даному випадку використовується можливість виконувати запит ping або його аналог для перевірки стану програмного та/або апаратного забезпечення. Якщо відповідь не отримана, або одержана некоректна відповідь, запускається механізм відновлення чи попередження. Цей підхід менш ресурсоємний, ніж підхід на основі TTL [3].

7.2 Життєвий цикл систем самовідновлення та методи штучного інтелекту

Основні етапи ЖЦ для системи самовідновлення це:

1. Моніторинг;
2. Аналіз стану об’єкту під наглядом;
3. Діагностика несправностей, змін та помилок;
4. Відновлення після збоїв;
5. Збереження подій для накопичення досвіду.

Згідно зазначених етапів можна зауважити, що використання методів штучного інтелекту необхідне на етапах 2, 3, та, інколи, 4.

Системи самовідновлення допомагають забезпечити критерії доступності згідно НД ТЗІ 2.5-004-99, а саме, критерію доступності “стійкість до відмов”. Згідно вимог цього критерію повинна бути запроваджена політика безпеки щодо стійкості до відмов та відновлення стану та визначені компоненти, до який ця політика відноситься і типи відмов після яких система в змозі продовжувати функціонування. Системи самовідновлення можуть захищати компоненти системи від відмов, які можуть призводити до недоступності послуг, а також спроможні повідомити адміністратора (розробника), про відмову будь-якого захищеного компонента.

Системи з властивістю самовідновлення забезпечують (ДС-2 або ДС-3), що відповідає стійкості з погіршенням характеристик обслуговування або без погіршення. Система самовідновлення може відновити модуль, або якщо це тимчасово неможливо, відключити його.

Також системи самовідновлення забезпечують послугу “відновлення після збоїв”. Використовуючи технології самовідновлення можна повернути комп’ютерну систему у відомий захищений стан після відмови. За НД ТЗІ 2.5-004-99 технології самовідновлення включають автоматизоване відновлення (ДВ-2). Система після відмови має здатність визначити можливість повернення системи до нормального стану в автоматичному режимі (без втручання користувача) і якщо це можливо, повернути КС до нормального функціонування – що, по суті, визначає необхідність застосування методів машинного навчання та штучного інтелекту для здійснення цих операцій.

Використання методів штучного інтелекту для моніторингу та відновлення системи широко використовується в подібних задачах. Різноманітність сервісів, які потребують вирішення задач моніторингу та відновлення, та їх архітектури, висуває різні потреби щодо способів

застосування методів, одержання вихідних даних, типів методів. Система моніторингу повинна забезпечувати гнучкість та підбір оптимального набору параметрів для роботи об’єкта, який вона забезпечує. Методи штучного інтелекту надають можливість побудови паттернів нормальної роботи сервісів на основі даних з модулів моніторингу, класифікації несправностей та прийняття рішень щодо типів відновлення.

7.3 Використання методів штучного інтелекту у поєднанні з традиційними підходами

В роботі [21] запропоновано приклад використання AI у поєднанні із моніторингом TTL та пінгуванням. Навчання відбувається із одним, визначеним класом даних. Система самовідновлення навчається на цих даних у тренувальному режимі, коли ОС знаходиться в нормальному стані, в цьому стані перевірки проводяться традиційними методами TTL та пінгування. Після навчання моделі штучного інтелекту, модулі, які працюють за традиційним підходом, можна відключити для підвищення продуктивності. При виявленні аномальної поведінки модулем штучного інтелекту, модулі TTL та пінгування можуть використовуватися як додаткові, для перевірки стану компонентів системи.

Для задач виявлення аномальних станів пропонується використоувати нейронну мережу, яка тренована на навчальному наборі даних, серед інших – обрано згортковий вид мереж, U-Net.

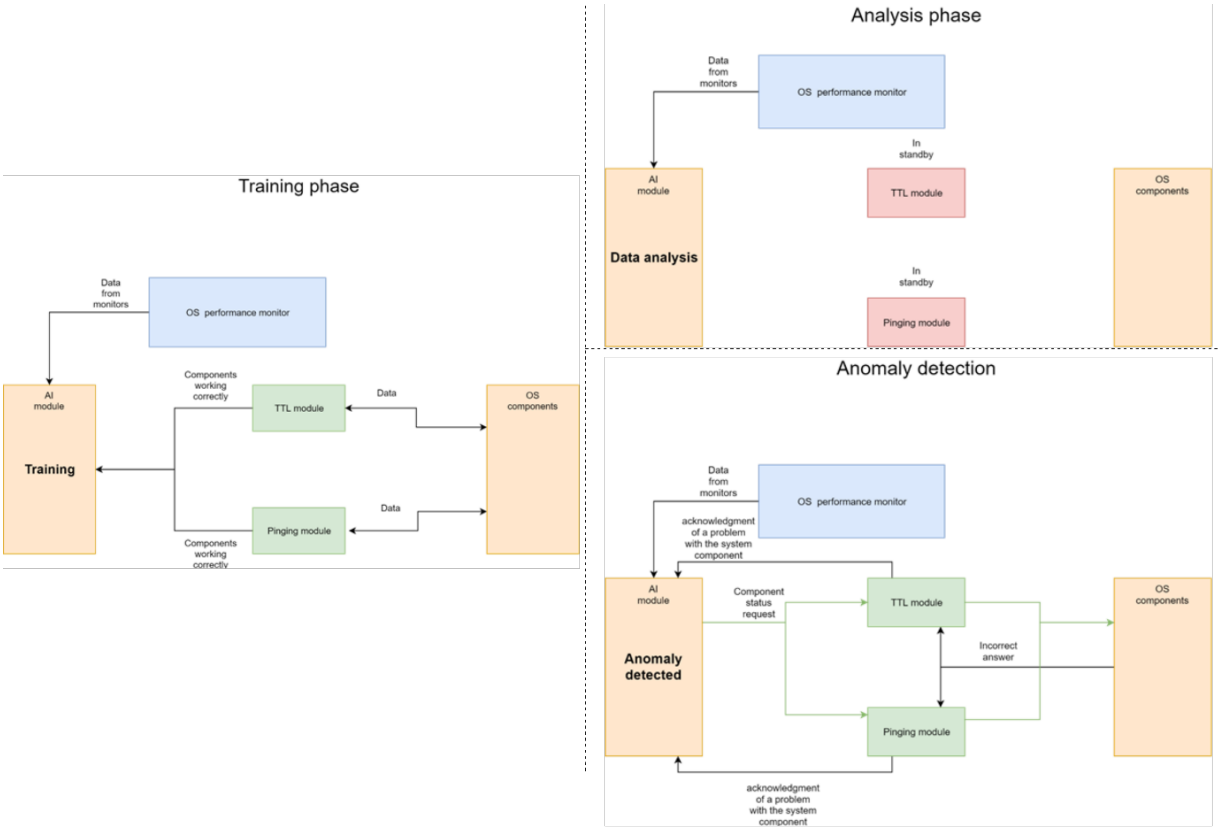


Рис. 7.1. Схема моделі на етапах навчання, аналізу та виявлення аномалій [21]

Для виявлення аномалій, коли відомі лише дані при нормальній роботі системи, можна використовувати алгоритми на базі однокласового методу опорних векторів (SVM): Protractor, EDR-, LCS-, DTW-ядро на базі SVM; Метод опорних векторів (SVM) дає гарні результати при виявленні аномалій через свою здатність забезпечувати нелінійну класифікацію за допомогою функції ядра[21]. Однокласові SVM використовуються для відокремлення даних одного конкретного цільового класу, від інших даних. Вони навчаються лише з позитивних прикладів, тобто точок даних з цільового класу.

Вищезазначені алгоритми машинного навчання мають недолік, який полягає у складності визначення причин прийняття тих чи інших рішень. Проблемою є також час навчання та кількість необхідних ресурсів для їх роботи. Отже, краще використовувати компроміс між детермінованими правилами та машинним навчанням. Варіантом такого методу може бути “дерево рішень”. Decision Tree досить швидко навчається, причини прийняття рішень зрозумілі для експерта [21].

7.4 Алгоритм прогнозування

В роботі [21] було запропоновано алгоритм прогнозування стану системи на базі правил.

На основі набору правил створюються два списки:

1. Список помилок.
2. Список запущених подій.

$$TE - List = \{f_i \rightarrow \{e_{i1}, e_{i2}, \dots, e_{ik}\}: 1 \leq i \leq N_f\}$$

$$TE - List = \{e_m \rightarrow \{f_{m1}, f_{m2}, \dots, f_{mn}\}: 1 \leq m \leq N_e\}$$

Тут f_i – це критичні події, а e_i – не критичні. Під час прогнозування, коли відбувається подія e:

1. Додати e в набір подій прогнозування

$$E = \{e_1, e_2, \dots, e_n\},$$

де події сортуються у порядку зростання та видалення e_i, коли

$$|T_E - T_i| > T_{predict\_window}.$$

2. Отримати потенційні збої, які можуть бути викликані e згідно

$$TF - List: e \rightarrow \{f_1, f_2, \dots, f_k\}.$$

3. За кожну невдачу у наборі $\{f_1, f_2, \dots, f_k\}$, обробити список подій згідно з $TF - List: f_i \rightarrow \{e_{i1}, e_{i2}, \dots, e_{ik}\}$

4. Якщо $\{e_{i1}, e_{i2}, \dots, e_{ik}\} \subseteq E$, тоді провести попередження про відмову f_i, яке може статися на протязі $f_{prediction\_window}$

7.5 Побудова детермінованих правил на основі Decision Tree

Дерево прийняття рішень – це інструмент підтримки прийняття рішень, який використовує деревовидний граф або модель рішень і їх можливі наслідки, включаючи випадкові події, витрати ресурсів і корисність. Це один із способів відображення алгоритму, який містить тільки умовні оператори управління. Найчастіше після навчання з учителям результатом є дерево з правилами у вузлах і прогнозом. Вирішальне правило – це деяка функція від об’єкта, що дозволяє визначити, в яку з дочірніх вершин потрібно помістити даний об’єкт.

Алгоритм побудови:

1. Перевірка критерію. Якщо він виконується, обрати для вузла прогноз, що можна зробити декількома способами.

2. Інакше треба розбити множину на декілька, які не перетинаються. У загальному випадку в вершині t задається правило прийняття рішень $Q_t(x)$,

яке приймає деякий діапазон значень. Цей діапазон розбивається на R_t непересічних множин об'єктів, $S_1, S_2, \dots, S_{R_t}$, де R_t - кількість нащадків у вершини, а кожне S_i - це безліч об'єктів, які потрапили в i-го потомка.

3. Множина в вузлі розбивається відповідно до встановленого правила, для кожного вузла алгоритм запускається рекурсивно.

Правила. Найчастіше в якості $Q_t(x)$ беруть одну з ознак $x^{i(t)}$:

$S_t(j) = \{x \in X : h_j \leq x^{i(t)} \leq h_{j+1}\}$ для обраних $h_1, \dots, h_{j+1}$,

$S_t(1) = \{x \in X : \langle x, v \rangle \leq 0\}$; $S_t(2) = \{x \in X : \langle x,v \rangle > 0\}$ перевірка вузла

$S_t(1) = \{x \in X : p(x, x_0) \leq h\}$; $S^t(2) = \{x \in X : p(x, x_0) > h\}$, де відстань p визначено в деякому метричному просторі (наприклад,

$p(x, y) = |x - y|$).

$S_t(1) = \{x \in X : x^{i(t)} \leq h\}$; $S_t(2) = \{x \in X : x^{i(t)} > h\}$

- предикати, $\langle x,v \rangle$ – скалярний добуток векторів.

В цілому, взяти можна будь-які правила прийняття рішень. На основі правил, сформованих за допомогою предикатів, можна отримати дерево з високою точністю на навчальній виборці.

7.6 Виявлення аномалій у часових рядах за допомогою згорткових нейронних мереж

Згорткові нейронні мережі використовуються для аналізу зображень та інших задач. Для аналізу аномалій у часових рядах варто розглядати ряд як одновимірне зображення, вимір - це час. Багатовимірний часовий ряд може мати довільну кількість вимірів. Підхід пропонує наступну архітектуру для сегментації часових рядів, як показано на рисунку 2.4.

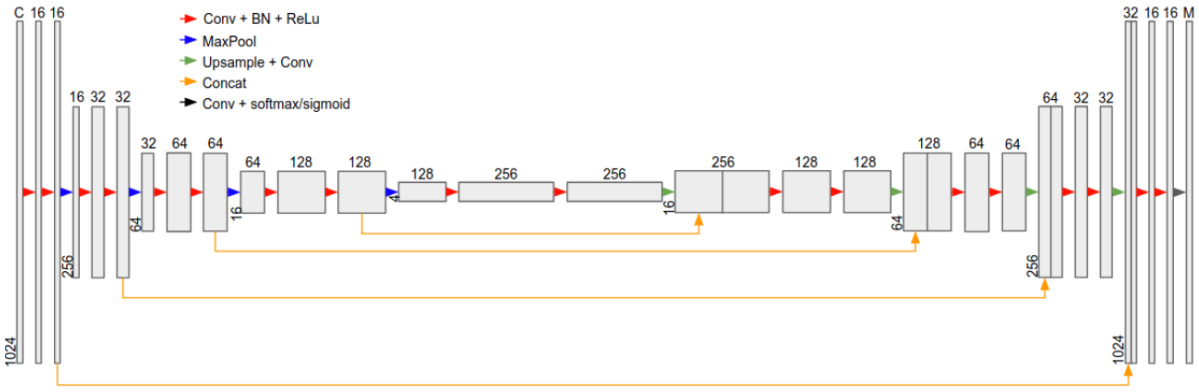


Рис.7.2. Архітектура U-Net для сегментації часових рядів

Детальніше принцип роботи U-Net розглянуто в [22].

Модель спочатку має бути натренована, а потім розгорнута для роботи в потоковому режимі. Дані одержуються, наприклад, із датчиків, сенсорів чи іншого обладнання. Кожен момент часу на вхід моделі повинен надходити відповідний сигнал. Може бути так, що за один проміжок часу може бути кілька сигналів, які сприятимуть більш надійному спрацюванню моделі. В залежності від особливостей задачі, архітектура нейронної мережі може бути змінена [21].

Часові ряди можуть містити як великі, так і малі значення. Їх рекомендовано нормалізувати. Мала кількість подій, пов’язаних із аномаліями в даних, може погіршити якість моделі.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. S. Varshney et al, Review of various Artificial Intelligence Techniques and its applications, 2019. IOP Conf. Ser.: Mater. Sci. Eng. 594 012023, DOI: 10.1088/1757-899X/594/1/012023.
2. G. Belani, The Use of Artificial Intelligence in Cybersecurity: A Review. URL: https://www.computer.org/publications/tech-news/trends/the-use-of-artificial-intelligence-in-cybersecurity.
3. S. Ray, 7 Regression Techniques you should know! URL: https://www.analyticsvidhya.com/blog/2015/08/comprehensive-guide-regression/
4. C.M.Bishop, Pattern recognition and Machine learning, 2006. Springer.
5. K.P.Murphy, Machine Learning: A Probabilistic Perspective, 2012. MIT Press.
6. В. Кисіль, Політика безпеки для промислового Інтернету речей та оцінка її дієвості, 2019. Магістерська дисертація. URL: https://ela.kpi.ua/bitstream/123456789/31393/1/Kysil_magistr.pdf
7. В. Кожокар, І.Стьопочкіна, Структурні закономірності еволюціонування метаморфного шкідливого ПЗ// Правове, нормативне та метрологічне забезпечення системи захисту інформації в Україні.- 2017, №1(33).-С.52-57.
8. Кожокар В.Ю., Стьопочкіна І.В. Виявлення спільних закономірностей у зразках метаморфного шкідливого ПЗ на основі статичних характеристик. XV Всеукраїнська науково-практична конференція студентів, аспірантів та молодих вчених “Теоретичні і прикладні проблеми фізики, математики та інформатики” 25-27 травня 2017, т.2.
9. J.Brownlee, A Gentle Introduction to Dropout for Regularizing Deep Neural Networks, 2018. URL: https://machinelearningmastery.com/dropout-for-regularizing-deep-neural-networks/
10. Learning curve. URL: https://www.ritchieng.com/machinelearning-learning-curve/
11. A. Gonfalonieri, How to Build A Data Set For Your Machine Learning Project, 2019. URL: https://towardsdatascience.com/how-to-build-a-data-set-for-your-machine-learning-project-5b3b871881ac
12. А. Черноусов, Метод автоматичного виявлення помилок безпеки в програмному забезпеченні на основі глибинного навчання, 2019. Магістерська дисертація. URL:https://ela.kpi.ua/handle/123456789/31602
13. Clang 15.0.0git documentation. URL: https://clang.llvm.org/docs/LibASTMatchersTutorial.html

14. В. Мельник, Класифікація пристроїв Інтернету речей з використанням методів машинного навчання, 2019. Магістерська дисертація. URL: https://ela.kpi.ua/handle/123456789/31387
15. Sklearn.ensemble.RandomForestClassifier. URL: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html
16. Tuning the hyper-parameters of an estimator. URL: https://scikit-learn.org/stable/modules/grid_search.html
17. SAT/SMT by example URL: https://sat-smt.codes/SAT_SMT_by_example.pdf
18. S. Mitra, Tutorial 1: Modern SMT Solvers and Verification. URL: https://www.iitg.ac.in/pbhaduri/GIAN-CPS/Doc/Tut-1.pdf
19. К. Рішко, Застосування SAT/SMT- розв'язників в задачах кібербезпеки, 2020. Магістерська дисертація.
20. SMT-LIB. The Satisfability Modulo Theories Library. https://smtlib.cs.uiowa.edu
21. О. Фокін, Технології самовідновлення для розподілених систем на базі штучного інтелекту, 2019. Магістерська дисертація. URL: https://ela.kpi.ua/bitstream/123456789/31604/1/Fokin_magistr.pdf
22. J.Zhang. U-Net. Line-by-line explaination, 2019. URL: https://towardsdatascience.com/unet-line-by-line-explanation-9b191c76baf5