МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Тернопільський національний економічний університет
Факультет комп'ютерних інформаційних технологій
Кафедра комп'ютерних наук

ІВАСЬКІВ Ігор Степанович

Методи машинного навчання в задачах виявлення нетипової поведінки складної системи/ Machine-learning methods in tasks of detection the atypical behavior of complex system

спеціальність: 8.05010302 - Інженерія програмного запезпечення
магістерська програма - Інженерія програмного забезпечення

Магістерська робота

Виконав студент групи ІПЗм-21
І. С. Іваськів

Науковий керівник:
к.т.н., доцент, О. Л. Козак

Магістерську роботу допущено
до захисту:

"___" _____ 20___ р.

Завідувач кафедри
_____ **А. В. Пукас**

ТЕРНОПІЛЬ - 2017

ЗМІСТ

АНОТАЦІЯ

Іваськів І.С. Методи машинного навчання в задачах виявлення нетипової поведінки складної системи.

Дипломна робота на здобуття освітньо-кваліфікаційного рівня магістр за спеціальністю 8.05010302 «Інженерія програмного забезпечення».

Дослідження присвячене проблемі використання методів машинного навчання в задачах виявлення нетипової поведінки складної системи. В роботі теоретично обґрунтована та експериментально перевірена методика використання моделі машини опорних векторів в якості засобу виявлення вторгнень в інформаційну систему. Запропоновано структуру та параметри штучної нейронної мережі для роботи в режимі детектора аномалій клавіатурного ритму. Запроектовано та реалізовано систему біометричної автентифікації на її основі. Розроблена система може бути використана в якості додаткового засобу перевірки користувачів при допуску до інформаційних ресурсів.

Ivaskiv I.S. Machine learning methods in tasks of detection the atypical behavior of complex system.

Master's degree work, specialty 8.05010302 “Software engineering”.

Graduate work devoted to the problem of using machine learning methods to detect atypical behavior of a complex system. The paper theoretically grounded and experimentally tested method of using support vector machines model as a means of detecting intrusions into the information system. It has been developed the structure and parameters of the artificial neural network for operation in detection of anomalies of keystroke dynamics. Designed and implemented biometric authentication system based on it. The system can be used as an additional tool in user-verification process.

ВСТУП

Виявлення нетипової поведінки системи є задачею знаходження патернів (шаблонів) в даних які не відповідають моделі «нормальної» поведінки. Знаходження таких відхилень від очікуваної поведінки в тимчасових даних є важливим для забезпечення нормального функціонування систем в багатьох галузях таких як економіка, біологія, інформаційні технології, фінанси, екологія та інших. Програмні додатки в таких галузях повинні мати можливість виявлення нетипової поведінки яка може бути сигналом системного збою або зловмисних дій, і повинні ініціювати відповідні кроки в напрямку прийняття коригувальних дій.

Нетипова поведінка системи в літературі з аналізу даних (датамайнінгу) та статистики також згадується як аномальність, відхилення, викид (outliers). Hawkins D. в [12] формально визначає концепцію викиду так:

«*Викид це спостереження яке відхиляється настільки від інших спостережень, що виникають підозри що він був породжений іншим механізмом*».

В більшості застосунків дані створюються одним або більше генеративним процесом, і можуть відображати активність в системі або спостереження за сутностями. Коли генеративний процес поводиться «незвичним способом» це є результатом створення викидів чи аномалій. Також часто викид містить важливу інформацію про анормальність характеристик системи і сутностей, що мала вплив на генеративний процес. Тож розпізнавання таких нетипових характеристик забезпечує здатність розпізнати суть проблеми.

Деякі приклади задач виявлення нетипової поведінки складних систем подано далі:

- **Системи виявлення вторгнень** (Intrusion Detection Systems): в багатьох локальних або об’єднаних в мережу комп’ютерних

системах збираються різного роду дані про системні виклики, мережевий трафік, завантаженість процесора, оперативної пам'яті, активність користувачів в системі. Ці дані можуть показувати нетипову поведінку викликану активністю зловмисника. Задача виявлення такої активності відома як виявлення вторгнень.

- **Шахрайство з кредитними картами**: доволі легка можливість отримання інформації про номер кредитної картки робить проблему шахрайства з банківськими картками досить розповсюдженою. В багатьох випадках, неавторизоване використання скомпрометованих карт має нетипові патерни, такі наприклад, як покупки з нетипових географічно місць. Такі патерни можуть бути використані для виявлення викидів, аномалій в даних за транзакціями.
- **Нетипові події в мережах датчиків (сенсорів):** сенсори часто використовуються для відстеження параметрів середовища, геоінформаційної локації в багатьох галузях. Раптові зміни типових моделей поведінки таких систем сенсорів можуть бути предметом інтересу.
- **Медична діагностика:** в багатьох задачах медичної діагностики дані збираються з різноманіття приладів таких як МРТ сканери, часові ряди ЕКГ тощо. Нетипові патерни на таких даних часто є наслідком хворобливих станів.
- **Природознавство:** значний обсяг просторово-часових даних про погодні умови, зміни клімату, або характеру рослинного покриву землі збирають за допомогою різних датчиків, супутників або дистанційних зондів. Аномалії в таких даних надають значущі ідеї про приховані впливи людства, екологічні тенденції, які можуть викликати такі аномалії.
- **Аномалії та нетипова поведінка у соціальних мережах:** сутності (ноди) в графі які нормально не поєднані можуть показати аномальні зв'язки один з одним.

У всіх вищенаведених застосунках, дані мають «нормальну» модель і аномалії розпізнаються як відхилення від цієї нормальної моделі. Це досить просто виконати для систем, в яких поведінка може бути визначена за допомогою простих математичних моделей – наприклад, систем які характеризуються гаусівським розподілом з відомим середнім і стандартним відхиленням. Проте, найбільш цікаві системи реального світу мають складну поведінку в часі. Для таких складних систем методи машинного навчання, датамайнінг та штучні нейронні мережі стають у нагоді для вивчення характеристик системи з спостережуваних даних.

Також важливою особливістю задач виявлення нетипової поведінки системи та виявлення аномалій є відсутність формального визначення аномальності. Зазвичай це поняття формалізуються в ході досліджень в залежності від обраного методу. В даній роботі ми поступимо подібним чином, вважаючи формалізацію поняття аномальності одним із завдань дослідження.

Тематиці виявлення аномалій присвячено ряд книг та досліджень. Класичні праці [28], [12], [20] в головному розглядають цю проблему з перспективи спільноти статистики. Більшість з них були написані до широкого розповсюдження технології баз даних та методів інтелектуального аналізу даних (datamining), і отже не досліджують методи і засоби виявлення нетипової поведінки в площині комп'ютерних наук. Більш сучасні дослідження проблеми вже проводяться фахівцями саме галузі комп'ютерних наук. Здобутки в напрямку баз даних, великих даних (BigData), інтелектуального аналізу (Datamining), машинного навчання (Machine Learning), нейронних мереж відкрили нові можливості в тематиці виявлення аномалій, відкрили нові аспекти проблеми такі як аномалії на дуже великих даних, або даних дуже великої розмірності.

Ряд досліджень розглядають концепцію виявлення аномалій з різних точок зору, методологій та різних типів даних [16], [24], [30], [29], [17], [18]. Серед них слід відзначити праці [29], [4] як найбільш сучасні та найповніші.

Тема дослідження “**Методи машинного навчання в задачах виявлення нетипової поведінки складної системи**” входить до плану науково-дослідних робіт кафедри комп’ютерних наук Тернопільського національного економічного університету.

Об’єктом дослідження є задачі виявлення нетипової поведінки складних систем.

Предметом дослідження є методика використання методів машинного навчання в задачах виявлення нетипової поведінки складних систем.

Метою дослідження є розробка методу та експериментальної системи виявлення вторгнень в інформаційну мережу а також системи біометричної автентифікації користувачів на основі методів машинного навчання.

Згідно з метою дослідження розв’язувались такі **завдання:**

1. Виявити стан проблеми виявлення нетипової поведінки з використанням методів машинного навчання та ступінь її розробки в інформаційних джерелах.
2. Вивчити основні типи задач виявлення аномалій.
3. Дослідити основні моделі даних методів машинного навчання які можуть бути використані в задачах виявлення аномалій.
4. Вивчити методику оцінки якості роботи алгоритмів машинного навчання в задачах вивявлення нетипової поведінки.
5. Детальне дослідження методу однокласової класифікації засобом машини опорних векторів та його використання в задачі виявлення вторгнень.
6. Дослідження методик використання штучних нейронних мереж в якості засобу виявлення аномалій.
7. Розробити архітектуру та програмну реалізацію системи біометричної автентифікації на базі штучної нейронної мережі що працює в режимі детектора аномалій.

Методи дослідження

- вивчення і теоретичний аналіз науково-технічної літератури при обгрунтуванні основних теоретичних положень дослідження;
- аналіз існуючих систем виявлення аномалій на основі методів машинного навчання;
- цілеспрямоване експериментальне дослідження методів машинного навчання: машини опорних векторів та штучних нейронних мереж в якості детектора аномалій.

Наукова новизна дослідження полягає в тому, що запропоновано та експериментально обгрунтовано новий підхід (на основі методу однокласової класифікації машиною опорних векторів) до розв'язання проблеми виявлення вторгнень, який забезпечує ефективний захист мережі підприємства, стійкий до нових видів атак при мінімальному втручанні системних адміністраторів; запропоновано оптимізовану архітектуру штучної нейронної мережі в задачах безпарольної автентифікації користувачів на основі аналізу клавіатурного ритму, яка забезпечує достатній рівень захисту доступу.

Практичне значення дослідження визначається тим, що:

- розроблено ефективний програмний засіб безпарольної автентифікації на основі клавіатурного ритму, який може бути використаний для захисту доступу до окремих програмних зазстосунків, автентифікації на онлайн-ресурсах тощо.
- розроблено методику використання машини опорних векторів в задачах вивлення вторгнень.

Апробація результатів дослідження здійснювалась впровадженням системи безперольної автентифікації в ситему документообігу Приватного Акціонерного товариства «Газинтек» м. Київ.

РОЗДІЛ 1. Основні типи задач виявлення нетипової поведінки та методи їх вирішення

1.1 Проблема вибору моделі даних

Віртуально всі алгоритми виявлення аномалій створюють модель нормального патерна даних, нормальної поведінки системи, і потім обчислюють «ступінь аномальності» певної точки даних (набору точок) на основі відхилення від цього патерна. Наприклад, модель даних може бути статистичною регресійною моделлю, або моделлю близькості. Ці моделі мають різне тлумачення «нормальної поведінки» даних. Ступінь аномальності досліджуваної (нової) точки даних оцінюють обчисленням подібності точки даних і моделі. В деяких випадках модель може бути задана алгоритмічно. Наприклад, алгоритм виявлення викидів на основі методу найближчих сусідів моделює дані в термінах розподілу відстані поміж n-найближчих сусідів. В цьому випадку, викиди знаходяться на великій відстані від більшості даних

Зрозуміло, що вибір моделі даних є важливим завданням. Некоректно вибрана модель даних може бути причиною незадовільних результатів роботи алгоритму. Наприклад, модель лінійної регресії може давати результати низької якості, якщо дані, на яких побудована ця модель, кластеризовані випадковим чином.

Правильний вибір моделі даних вимагає добре розуміння предметної галузі. Наприклад, регресійні моделі добре працюють у методах виявлення викидів на даних розподілених лінійно (рис. 1.1). Для розподілу поданому на рис. 1.2, більше підходять кластерні моделі.

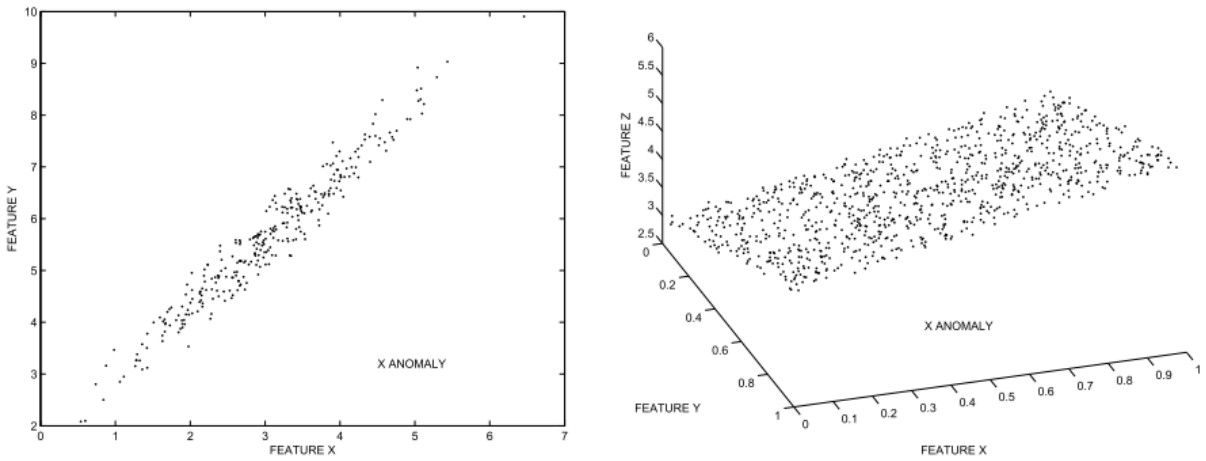


Рис. 1.1 Приклад даних з лінійним розподілом

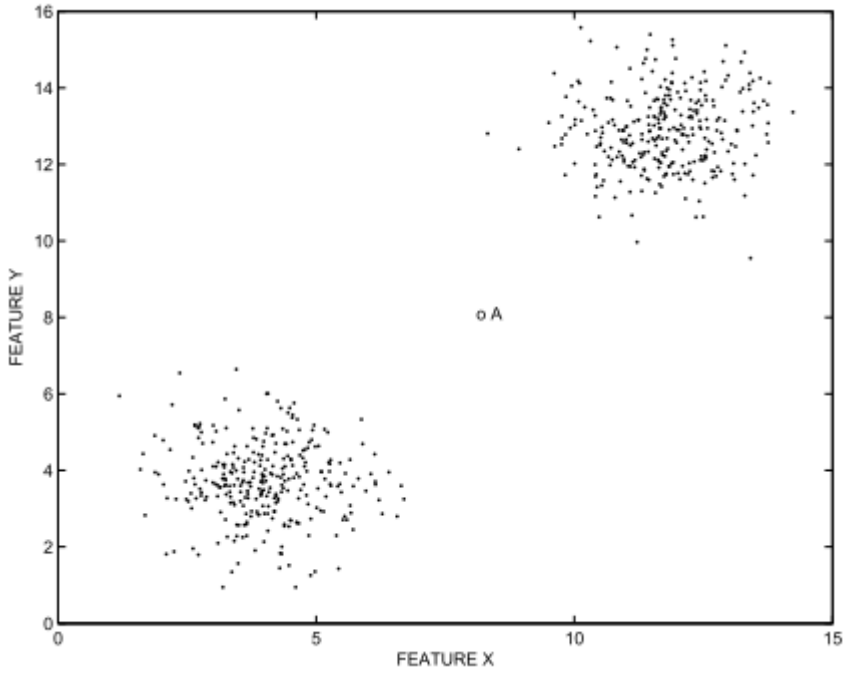


Рис. 1.2 Приклад даних, на яких аномалій можуть визначатись кластерним аналізом.

Також важливою складовою є правильний вибір складності (узагальнення) моделі. Складна модель, з високим рівнем узагальнення, з надто великою кількістю параметрів буде «перенавчатись», і знайде спосіб «пристосуватися» до викидів. Простіша модель, побудована з хорошим розумінням даних, швидше за все, призведе до кращих результатів.

Надмірно спрощена модель, погано «допасована» до даних, ймовірно прийме нормальні патерни за викиди.

1.2 Огляд основних моделей

1.2.1 Аналіз Екстремальних значень

Найбільш просто формою виявлення аномалій є аналіз екстремумів даних розмірності 1. Цей метод не є предметом нашого дослідження проте його розгляд має важливі особливості для розуміння взагалі поняття аномалій, викидів, нетипової поведінки.

В найбільш простому випадку, значення певної величини, які або надто великі або надто малі, приймають за викиди. Проте розділ статистики який займається аналізом екстремальних значень може дати зовсім інші тлумачення викидів на цьому ж ряді даних. Наприклад, розглянувши набір даних {1, 2, 2, 50, 98, 98, 99} можна зробити припущення, що значення 1 та 99 є викидами. Разом з тим, більшість ймовірнісних моделей, та моделей які базуються на аналізі щільності розподілу вкажуть значення 50 як «найсильніший» викид. Невірне тлумачення екстремальних значень як викідів зустрічається часто, особливо в контексті мультиваріативних даних.

Моделі аналізу ектремальних значень також відіграють важливу роль на останніх етапах багатьох алгоритмів виявлення аномалій. Це тому, що більшість з цих алгоритмів обчислюють відхилення точки даних від нормального патерну у формі числового значення. Це значення як правило одновимірне і його екстремальні значення належать викидам. Також методи вивчення багатовимірних екстремумів дозволяють отримати об’єднаний одномірний числовий вимір аномальності на багатомірних даних [13].

1.2.2 Статистичні та ймовірнісні моделі

Хоча ці моделі також не є предметом дослідження, їх огляд є важливим для розуміння багатьох методів машинного навчання. В цих методах дані моделюють у формі розподілу ймовірностей з параметрами які навчаються.

Наприклад, модель суміші функцій Гауса – породжувальна модель [36] яка характеризує дані в формі генеративного процесу що містить суміш гаусовських кластерів. Параметри цього розподілу навчаються з використанням EM-алгоритму (Expectation-Maximization) [11] на колекції даних. Результатом роботи алгоритму за цим методом є ймовірність приналежності точки даних певному кластеру, а також густина розподілу «допасування» точки даних до моделі (fit). Це забезпезчує природній спосіб виявлення викидів, як точок з найнижчим рівнем «допасування».

- *Ключове припущення*: Нормальні точки даних розміщені в регіонах статистичого розподілу з високими значеннями ймовірностей, тоді коли аномалії «лежать» в регіонах статистичного розподілу з низькими значеннями щільності ймовірності.
- *Загальний підхід:* Оцінити статистичний розподіл вихідних даних, зробити оцінку чи тестові дані належать до цього розподілу чи ні.
  - *Якщо дане спостереження лежить на відстані більшій за 3 стандартні відхилення від середнього значення вибірки, вважаємо його за аномальним.*
  - *На аномальних даних наступний вираз приймає нетипово великі значення:*

$$T^2 = \frac{n}{n+1}(\mathbf{X} - \bar{\mathbf{X}})' \boldsymbol{S}^{-1} (\mathbf{X} - \bar{\mathbf{X}})$$

Основною перевагою ймовірнісних моделей є можливість їх застосування до будь-якого типу даних (або змішаного типу даних), для якого існує відповідна породжувальна модель. Наприклад, для категорійних (нечислових) даних дискретний розподіл Бернуллі може бути застосований для кожного компоненту суміші. Так як розглядувані моделі працюють з ймовірностями, проблема нормалізації (буде розглянута нижче) вже врахована.

Основним недоліком ймовірнісних моделей є намагання «пристосувати» колекцію даних до певного виду розподілу, який часто не є прийнятний для типу досліджуваних даних. Коли кількість параметрів зростає, дослідники зустрічаються з проблемою перенавчання – модель нормального патерну даних може допасовувати і викиди. Також параметри цих моделей важко інтерпретувати в термінах досліджуваної галузі – це виключає виконання одного з завдань аналізу аномалій – виявлення причини аномальної поведінки.

1.2.3 Лінійні моделі

Лінійні методи [32] моделюють дані в підпросторі меншої розмірності з використанням лінійних кореляцій [17]. Так у випадку поданому на рис. 1.1 дані розміщені вздовж 1-мірної лінії в 2-вимірному просторі(ліва схема) і вздовж двомірної площини розміщеної в 3-вимірному просторі (права схема). Оптимальне розміщення цієї лінії (або площини) визначається методами регресійного аналізу [34].

Для 2-мірного випадку на рис. 1 лінійна модель колекції точок даних $\{(x_i, y_i)\},\ i \in \{1 \dots N\}$ може бути подана так:

$$y_i = a * x_i + b + \epsilon_i \quad \forall i \in \{1 \dots N\}$$

тут ϵ_i – залишковий член, похибка яка є по суті помилкою моделювання. Коефіцієнти *a* та *b* повинні бути визначені методом навчання з на даних, і повинні мінімізувати помилку найменших квадратів яка визначається як:

$$error = \sum_{i=1}^{N} \epsilon_i^2$$

Ця задача відома як мінімізація опуклої функції в нелінійному програмуванні, і може бути вирішена аналітично з допомогою операцій над матрицями та методом градієнтного спуску.

Отримані залишкові члени можуть використовуватися в поєднанні з методами екстремального аналізу для пошуку викидів.

Різновидом лінійного моделювання даних є методи пониження розмірності [9] та метод головних компонент (PCA - Principal Component Analasis) [22]. Метод головних компонент полягає на визначенні засобами багатовимірного регресійного аналізу площини яка оптимізує найменшу квадратичну похибку вибірки в термінах відстані до цієї площини. Іншими словами визначається підпростір, такий, на який проекція даних відбувається з найменшою сумарною квадратичною помилкою. При цьому набір даних з можливими кореляційними залежностями поміж окремих розмірностей конвертується в набір даних з лінійно некорельованими полями які і називають головними компонентами. Абсолютний розмір сумарної помилки може бути проаналізований для визначення аномалій (викидів). Точка даних, яка має велику помилку, швидше за все буде аномальною, оскільки вона не «відповдає» нормальному підпростору патернів даних.

Додатково методи PCA можуть бути застосовані для корекції шумів, [2], методом зміни атрибутів точок даних для зменшення шумів в даних.

Недоліком методів пониження розмірності та регресійного аналізу є складність інтерпретації, інтуїтивного розуміння, особливо коли величина розмірностей в даних висока.

1.2.4 Спектральні моделі

Багато методів матричних розкладів, такі як PCA також використовуються на даних у вигляді графів та мереж. Головною відмінністю

є спосіб створення матриці для розкладу. Такі моделі відомі як спектральні моделі. Спектральні моделі в головному застосовуються для кластерного аналізу набору даних у вигляді графів а також використовуються для пошуку аномалій в часових рядах.

1.2.5 Моделі аналізу подібності

Ідея покладена в основу цих моделей пошуку викидів – знайти точки які ізольовані від решти даних. Відомі 3 підвиди цих методів: кластерний аналіз, аналіз густини розподілу, аналіз «найближчих сусідів». При кластерному аналізі та аналізу густини розподілу виділяють щільні регіони даних і аномальними вважають ті дані які не потрапляють у ці регіони. Головною відмінністю між кластерним аналізом та аналізом густини розподілу є те, що кластерні методи сегментують точки, а методи які базуються на густині – простір.

В методі k - найближчих сусідів (k-nearest neighbor) [26], [10], визначається відстань кожної точки даних до своїх k найближчих сусідів. Невелика група точок, які близькі одна до одної але далека від решти вважається викидами. Це припущення є справедливим адже мала група пов’язаних точок може бути наслідком роботи аномального процесу. Наприклад, великий кластер даних з 4000 точок, поданий на рис. 1.3, і мала множина ізольованих проте близько розміщених аномалій. В розглянутому випадку аномальні точки близькі одна до одної, тож не можуть бути визначені на базі моделі 1- найближчий сусід.

Такого роду аномалії важко відрізнити від шумів та іншого роду спотворень з використанням кластерних алгоритмів та алгоритмів що базуються на густині розподілу.

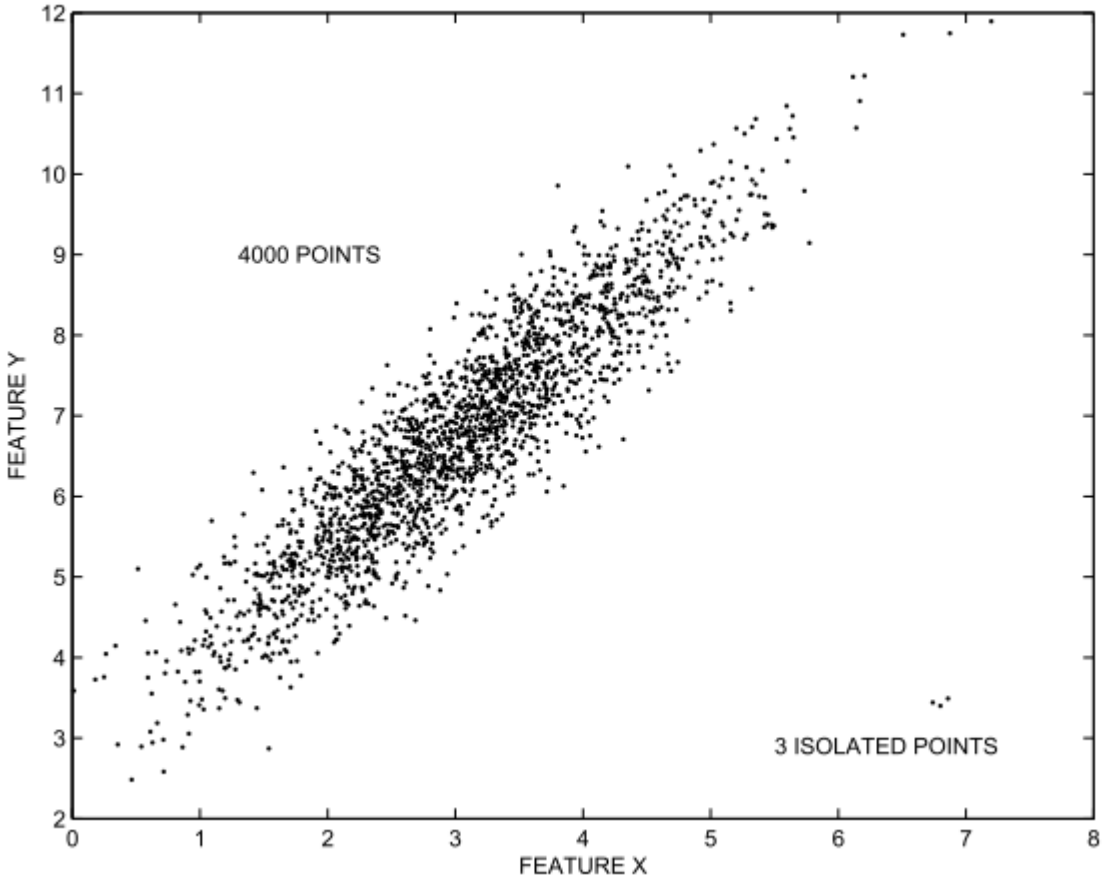


Рис. 1.3 Приклад аномалій які не можуть бути виявлені методом 1-го найближчого сусіда

Недоліком алгоритмів які базуються на методі к-найближчих сусідів є висока їх обчислювальна складність. Без використання ефективних методів індексації вони потребують $O(N^2)$ часу для набору даних з N точок.

У випадку кластерних моделей, першим кроком є алгоритм кластеризації для виявлення регіонів скупчення в наборі даних. Наступним кроком є певне вимірювання «допасування» точки даних до різних кластерів. На основі цього вимірювання можна зробити висновок про рівень «аномальності» цієї точки. Наприклад, у алгоритмі k-means (к-середніх), відстань точки даних до найближчих центроідів може бути мірою нетиповості поведінки. Одним із важливих завдань, які необхідно виконати досліднику перед застосуванням алгоритмів на основі кластерних моделей, є необхідність явно вказати форму кластера, та правильно вибрати функцію відстані між точками даних.

Підходи що базуються на кластеризації мають високий рівень інтерпретованості. Розріджені регіони набору даних можуть бути подані в термінах комбінації оригінальних атрибутів даних.

1.2.6 Теоретико-інформаційні моделі

Більшість розглянутих вище моделей виявлення аномалій використовують певну форму підсумовування (узагальнення) даних - у виді параметрів імовірнісних генеративних моделей, кластерів, гіпер-площин проекцій нижчої розмірності. Дані які «відхиляються» від цього узагальнення позначають як викиди. Теоретико-інформаційні моделі загалом базуються на цих же принципах, проте викидами позначають дані які збільшують мінімальну довжину коду що описують колекцію даних. Наприклад, розглянемо два рядки:

ABABABABABABABABABABABABABAB

*ABA**C**ABABABABABABABABABABABAB*

Другий рядок тієї ж довжини що і перший, проте відрізняється однією позицією, що містить унікальний символ «С». Перший рядок може бути коротко описаний як «15 разів АВ», проте наявність нового символу в алфавіті другого рядка не дозволяє описати його так само коротко. Також не важко замітити, що символ С є в цьому випадку аномалією, викидом.

Теоретико-інформаційні моделі подібні до звичайних моделей аналізу аномалій - в обох випадках застосовують скорочене подання набору даних як базу для порівняння. Так, наприклад, у випадку набору багаторозмірних даних ці моделі використовують різні способи скороченого (узагальненого опису) опису:

- ймовірнісні моделі описують набір даних породжувальними (генеративними) параметрами, такими як параметри суміші розподілів гауса;

- кластерні меделі узагальнюють набір даних описом кластерних форм, гістограм тощо;

- моделі PCA описують дані в підпросторах меншої розмірності.

В загальному, аномалії збільшують довжину опису в термінах цих «згущених» компонент щоб досягти того ж рівня наближення. Відповідно в теоретико-інформаційних моделях, ключовою є ідея побудувати словник кодування який відображає дані, і описувати викиди як такі дані, при видаленні яких з досліджуваної колекції, максимальна довжина опису зменшується найбільше. Конкретний тип кодування вибирається евристично в залежності від даних. В багатьох випадках кодування не задається явно, а використовуються міри такі як ентропія або складність Колмогорова що би оцінити рівень нерівномірності в специфічному сегменті даних. Сегменти з більшою нерівномірністю можуть бути розглянуті як аномальні.

Такого роду моделі даних найчастіше використовують для аналізу часових рядів. Перевіряються певної довжини сегменти даних і вимірюють довжину «описового коду». Сегменти з найбільшими змінами матимуть більшу довжину опису.

1.3 Виявлення аномалій у колекціях даних великої розмірності

Виявлення нетипової поведінки системи яка «генерує» дані великої розмірності є особливо складним. Багатовимірні дані часто є розрідженими і всі пари точок даних стають рівновіддаленими один від одного. [7]. В цьому випадку ми не можемо приймати відхилення у дистанції між точками даних за міру аномальності. Зашумленість та розрідження даних великих розмірностей робить точки даних дуже подібними одна одній.

Характерні спостереження підтверджують, що справжні аномалії можуть бути виявлені аналізом даних в локальному підпросторі меншої

розмірності. В цьому випадку, викиди часто скриті в нетиповій поведінці підпростору меншої розмірності, і ці відхилення в поведінці маскуються при аналізі повновимірного простору. Тому явний пошук відповідних підпросторів, в яких можуть бути виявлені аномалії часто дає плідні результати. Однак одночасний пошуку підпросторів та аномалій в цих підпросторах є не простою задачею в термінах обчислювальної складності. Еволюційні евристики такі як генетичні алгоритми можуть стати в нагоді для дослідження великої кількості підпросторів.

1.4 Базові типи даних

Більшість розглянутих вище моделей працюють з багатовимірними числовими даними. Крім того, припускається що окремі записи (рядки) даних незалежні. Проте, на практиці, вихідні дані можуть бути більш складними, як в термінах виду атрибутів так і в зв'язках між окремими записами.

1.4.1 Категорійні, текстові та змішані атрибути

Колекції даних в реальних застосунках можуть містити категорійні атрибути, які приймають дискретні значення, що не можуть бути впорядковані. Такого роду дані вимагають спеціальних технік аналізу які дозволяють розширити на них вже розглянуті моделі такі як, наприклад, метод найближчих сусідів чи класифікатор базований на густині розподілу. Для таких даних будують спеціальні концепції подібності, проте головним є сконструювати функцію відстані, яка залишатиметься семантично значущою на такого роду дискретних даних.

Моделі регресії можуть обмежено застосовуватись на категорійних даних, при невеликому числі можливих значень атрибуту. Типовим методом є конвертувати дискретні дані в бінарний вид створюючи один бінарний атрибут для кожного категорійного значення. Моделі регресії, такі наприклад як, аналіз головних компонент (PCA) можуть бути застосовані до цієї колекції бінарних даних.

Ці ж методи можуть поширюватися і на текстові дані, яким притаманне впорядкування частотності слів. В цьому випадку кореляції поміж частотами появи слів у тексті можуть бути використані при створенні моделей лінійної регресії. Іншими методами якими часто послуговуються на текстових даних є кластеризація текстів та методи аналізу подібності.

1.4.2 Врахування залежності поміж точками даних

Більшість вище розглянутих методів припускає, що окремі записи даних розглядають незалежно один від одного. На практиці, різні значення даних можуть бути пов'язаними один з одним часово, просторово чи через явні посилання поміж елементами даних. Присутність таких залежностей суттєво змінює проблему виявлення аномалій, тому що природа цих залежностей відіграє критичну роль у процесі виявлення викидів. В цьому випадку, на «очікуване» значення елемента даних впливає їх контекстна залежність, і виявлення викидів повинно базуватись на відхиленнях від цієї конекстної моделі. Якщо окремий елемент даних (наприклад, значення з часової серії) задекларовано як аномальний, через його відношення до пов'язаних елементів даних, він називається контекстним викидом. Такі аномалії також часто згадуються як умовні викиди [31].

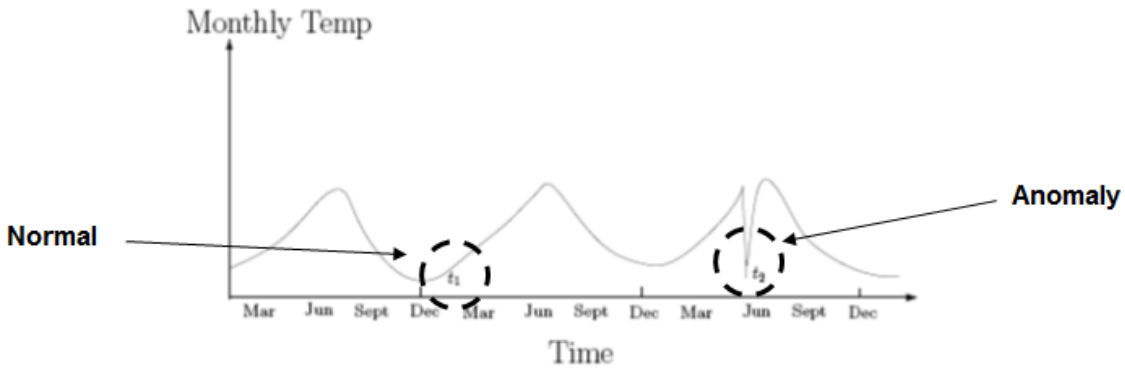


Рис. 1.4 Приклад контекстного викиду

В залежності від типу набору даних існує декілька шляхів моделювання таких аномалій. Розглянемо деякі з них.

1.4.3 Часові ряди (Time Series) та потоки даних (Data Streams)

Часовий ряд – набір значень, який зазвичай створений послідовними вимірюваннями в часі. Тому величини в послідовних часових мітках змінюються не значно або змінюються плавно. Випадкові значні зміни в записах даних можуть бути розглянуті як аномальні. Зазвичай такі події викликані раптовими змінами в досліджуваній системі і можуть становити інтерес для дослідника.

Розглянемо часовий ряд значень разом з відповідними часовими мітками, що неявно визначені індексом точки даних:

3, 2, 3, 2, 3, 87, 86, 85 87, 89, 86, 3, 84, 91, 86, 91, 88

Графік цього часового ряду подано на рис. 4

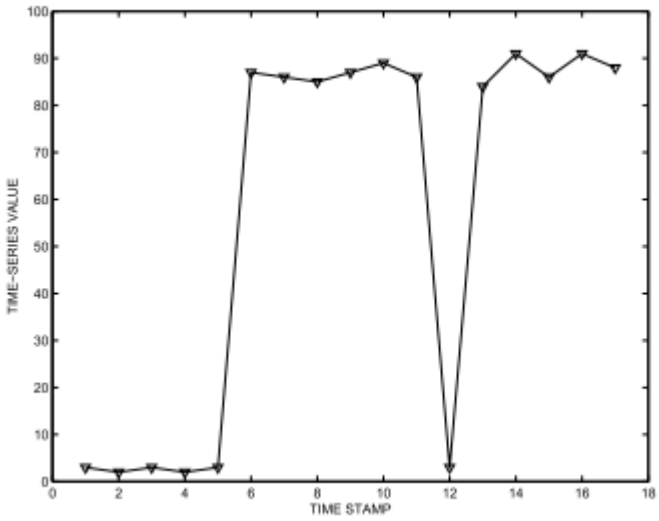


Рис. 1.5 Приклад часового ряду

Проаналізувавши ряд, можна відзначити раптову зміну значень на часових мітках 6, 12, 14. Зміна на мітці 6 можу бути сприйнята як викид, аномалія, проте дані стабілізувалися в цій точці і встановили нове «нормальне» значення. В часовій мітці 12 дані знову «падають» до відмітки 3. Не дивлячись на те, що це значення вже траплялося в серії та вважалося

«нормальним», воно вважається нетиповим, тому що є раптовою зміною в послідовних значення величини.

З вищенаведеного прикладу видно, що оцінювання значень незалежно одне від одного не дає можливості виявити аномальність коректно. Коли в серії зустрічається нові значення, вони називаються новинками (novelties). Проте виявлення аномалій відноситься до будь яких раптових змін, не тільки новинок, які є певними формами аномалій.

Необхідно відмітити, що виявлення змін в часових рядах та виявлення аномалій у часових рядах близькі поняття, проте необов'язково ідентичні. Зміни в часових рядах можуть відбуватися одним із двох способів:

- значення та тренди потоку даних змінюються повільно в часі. Такі з «плавні» зміни можуть бути виявлені лише детальним аналізом протягом тривалого часу і в багатьох випадках не є очевидними.

- значення та тренди потоку даних змінюються різко, відразу викликаючи підозру, що механізм який генерує ці дані зазнав фундаментальних змін.

Перший сценарій не обов'язково відноситься до аномалій, тоді як другий сценарій більш відповідає задачі виявлення аномалій.

Важливим завданням аналізу часових рядів та потоків даних є виявлення аномалій у реальному часі, як тільки нові дані зафіксовано. Зазвичай онлайн аналіз застосовують для виявлення змін поведінки, в той час як офлайн аналіз може «відкрити» інші нетипові аспекти даних.

1.4.4 Дискретні послідовності

Багато застосунків, такі як, виявлення вторгнень або виявлення шахрайства мають виражений темпоральний характер [37]. Такі моделі можуть бути розглянуті як категорійний або дискретний аналог часового ряду. Дискретна послідовність не завжди є темпоральною (часовим рядом), а

може базуватися на взаємному розташуванню по відношенню один до одного. Прикладом є біологічні дані, де послідовність задана на базі взаємного розташування.

Аномальна подія на дискретній послідовності може бути виявлена відхиленням від нормальних патернів шляхом аналізу підпослідовностей в різних часових мітках. Це є аналогом виявлення незвичних форм в часових рядах, і нетипова поведінка є набором колективних викидів.

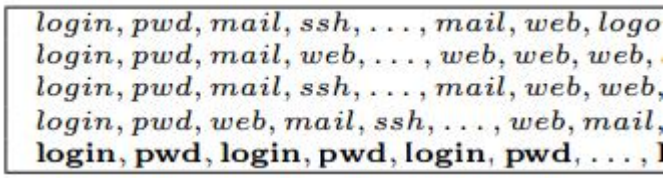


Рис. 1.6. Приклад дискретних послідовностей в системі виявлення вторгнень

Дискретні послідовності подібні до часових рядів. Проте числові часові ряди містять значення, що можуть бути упорядковані, і тому можуть «осмислено» порівнюватися між собою протягом усього спектру. 2 різних дискретних значення не можуть порівнюватись у такий спосіб. Відмінне представлення даних вимагає інші функцію подібності, структури даних, більш складні методи передбачення, такі як марковські моделі [35] або методи рухомого вікна (Sliding Windows).

1.4.5 Просторові дані

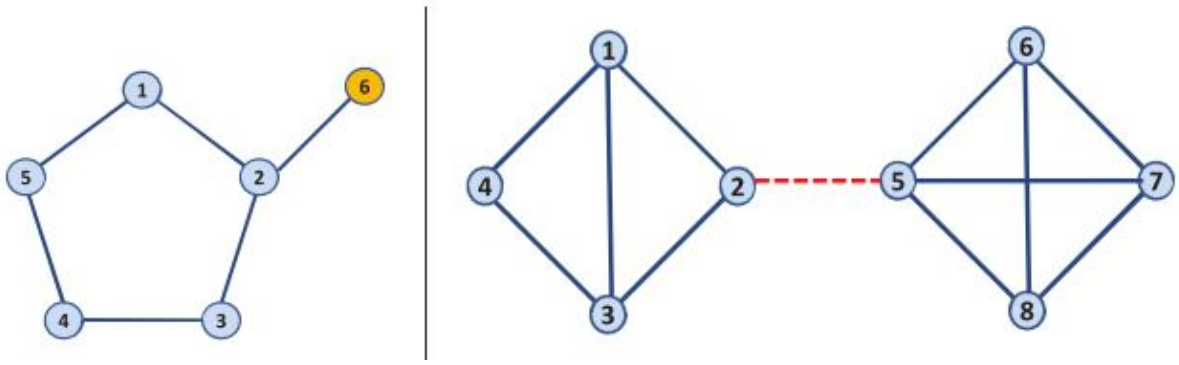
В просторових даних, багато не просторових атрибутів (температура, тиск, інтенсивність кольору пікселя зображення) вимірюють в точках простору. Незвичні локальні зміни таких значень рахуються викидами.

Методи виявлення аномалій в просторових даних подібні до таких в часових рядах [21]. В обох випадках досліджувані атрибути мають певний рівень безперервності. Наприклад, розглянемо вимірювання температури, при якому вимір асоціюється з точкою в часі і просторовими координатами.

Подібно, як очікується, що температура в послідовних часових мітках не змінюється занадто сильно (часова безперервність), також очікується, що температура в просторово близьких локаціях не змінюється надто сильно (просторова безперервність). Нетипова просторова варіативність температури і тиску близько поверхні морів дозволяють викрити аномальні події такі як формування циклонів тощо.

Мережеві дані та графи

В мережевих даних чи графах значення даних відносяться до вузлів, в той час як зв'язки між окремими точками даних відносяться до ребер мережі чи графу. В таких випадках аномалії можуть виявлятися в ірегулярності залежності між вузлами або ірегулярності в ребрах (рис. 1.7)



а) аномальний – вузол *б) аномальне - ребро*

Рис. 1.7. Приклад аномалій в мережі

Графи також можуть бути темпоральними за природою. В цьому випадку дані можуть мати структурну та часову залежність. Моделі виявлення аномалій на таких структурах повинні комбінувати аналіз змін в структурі та часі.

1.5 Виявлення аномалій на розмічених даних (Supervised Outlier Detection)

Аналіз аномалій у сценаріях, при яких наявні попередні приклади нетипової поведінки, називають виявлення аномалій на розмічених даних або ж виявлення аномалій з вчителем. Мітки аномальних ділянок в попередніх даних використовують для навчання моделей виявляти аномалії певного типу.

Аналіз аномалій на розмічених даних є специфічним (більш складним) випадком задачі класифікації. Характеристичною ознакою є значна незбалансованість відносної присутності міток нормальної та аномальної поведінки. Нормальні дані зазвичай легко зібрати і тому вони присутні в наборі даних у великій кількості. З іншого боку, випадки аномалій, викидів дуже розріджені. В класичних працях з машинного навчання ця проблема відома як виявлення класів з низькою частотністю (Rare Class Detection) [3]. Дисбаланс міток (позитивних-негативних, нормальних – аномальних) часто призводить до

Tid	SrcIP	Start time	Dest IP	Dest Port	Number of bytes	Attack
1	206.135.38.95	11:07:20	160.94.179.223	139	192	No
2	206.163.37.95	11:13:56	160.94.179.219	139	195	No
3	206.163.37.95	11:14:29	160.94.179.217	139	180	No
4	206.163.37.95	11:14:30	160.94.179.255	139	199	No
5	206.163.37.95	11:14:32	160.94.179.254	139	19	Yes
6	206.163.37.95	11:14:35	160.94.179.253	139	177	No
7	206.163.37.95	11:14:36	160.94.179.252	139	172	No
8	206.163.37.95	11:14:38	160.94.179.251	139	285	Yes
9	206.163.37.95	11:14:41	160.94.179.250	139	195	No
10	206.163.37.95	11:14:44	160.94.179.249	139	163	Yes

Рис. 1.7 Приклад розмічених даних

проблеми перенавчання (over-training, overfitting). Крім того, відомо кілька варіацій проблем класифікації у відповідності до рівня повноти розмітки:

- обмежена кількість позитивних прикладів, тоді коли «нормальні» приклади можуть містити деяку невідому пропорцію аномалій [8]. Цей випадок відомий як класифікація на позитивно-нерозмічених даних (Positive-Unlabeled Classification). В тому випадку, коли нерозмічена навчаюча вибірка не в повній мірі відображає тестову вибірку, доцільно відкинути з навчальної вибірки нерозмічені дані і розглядати цю задачу як проблему класифікації одного класу (one-class problem).
- в навчаючій вибірці присутні лише деякі види аномалій, які можуть зустрітися в тестовій вибірці. Наприклад, у вибірці, на якій навчається модель виявлення вторгнень в комп'ютерну мережу, присутні аномалії у вигляді певного роду атак. Проте в робочому режимі система повинна реагувати і на нові види атак, що раніше не зустрічались.

Зазначені проблеми вимагають спеціальних підходів до налаштування стандартних алгоритмів машинного навчання. Так, наприклад, класифікатор налаштовується у спосіб коли помилка класифікації аномального класу штрафується більше ніж помилка класифікації класу більшості. Ідея полягає в тому, що краще помилково передбачити нормальну поведінку за аномальну (false positive) ніж пропустити дійсно аномальну ситуацію (false negative). Такого роду «компроміси» регулюються з використанням методів кривих Precision-Recall(PR) та Receiver Operating Characteristics, які будуть розглянуті нижче.

1.7 Метод опорних векторів: загальний огляд

Метод опорних векторів (**Support Vector Machines (SVMs))** є одним з найбільш потужніших методів машинного навчання. Хоча він розроблявся в головному для задач класифікації, його модифікації успішно використовуються в задачах виявлення аномалій. Метод опорних векторів є дискримінаційним класифікатором: він «проводить» межу поміж кластерами даних.

Основними перевагами методу є:

- висока ефективність на даних великих розмірностей
- залишається ефективним навіть у випадках коли розмірність даних перевищує кількість записів в навчаючій вибірці
- ефективне використання оперативної пам'яті так як метод застосовує підмножину навчаючих точок в вирішуючій функції (опорні вектори)
- універсальність: у вирішуючій функції можуть використовуватись різноманітні функції ядра (будуть розглянуті нижче).

Недоліками методу опорних векторів є:

- класифікатор не дає ймовірнісних оцінок, проте в разі необхідності ці оцінки можуть бути розраховані методом крос-валідації (хоча цей процес не є ефективним в термінах обчислювальної складності).

Математичне формулювання

Як зазначено вище, метод опорних векторів будує гіперплощину (або набір гіперплощин) в просторі великої розмірності. Інтуїтивно зрозуміло, що кращий розподіл досягається гіперплощиною яка проходить на найбільшій відстані до найближчих точок даних будь-якого класу. Ця відстань називається функціональною маржою (functional margin), і в загальному випадку більша функціональна маржа забезпечує меншу узагальнюючу похибку класифікатора.

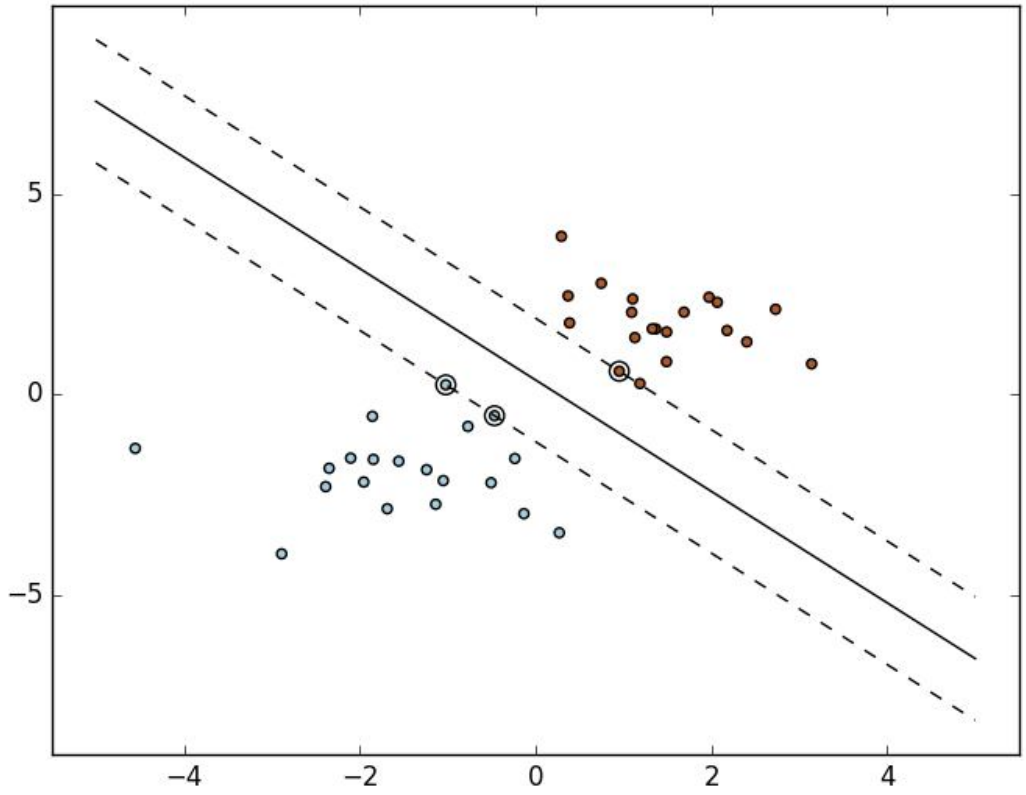


Рис. 1.8 Гіперплощина і функціональна маржа

Формалізуємо задачу класифікації набору даних методом опорних векторів. Точку даних розглядатимемо як *p*-вимірний вектор. Задача класифікації полягає в розділені заданого набору точок $x_i \in \mathbb{R}^p$ (*p-1*)-вимірною гіперплощиною. Існує багато різних варіантів гіперпорщин, які могли би розділяти одні й тіж дані. Варіантом найкращої гіперплощини є такий, який пропонує найбільший проміжок розділення. Якщо така площина існує її називають *максимально розділовою гіперплощиною (maximum margin hyperplane).*

Лінійна опорно-векторна модель

Для заданих навчаючих векторів

$x_i \in \mathbb{R}^p$ *i=1,..., n,*

і вектора

$$y \in \{1, -1\}^n,$$

кожен компонент якого вказує клас (1 або -1) до якого належить точка x_i,

розділяючі гіперплощини можуть бути описані рівнянням:

$$w \cdot x - b = 0$$

Де *w* – вектор нормалі до цієї гіперплощини, параметер $\frac{b}{\|w\|}$ визначає зсув гіперплощини від початку координат вздовж вектора нормалі *w*.

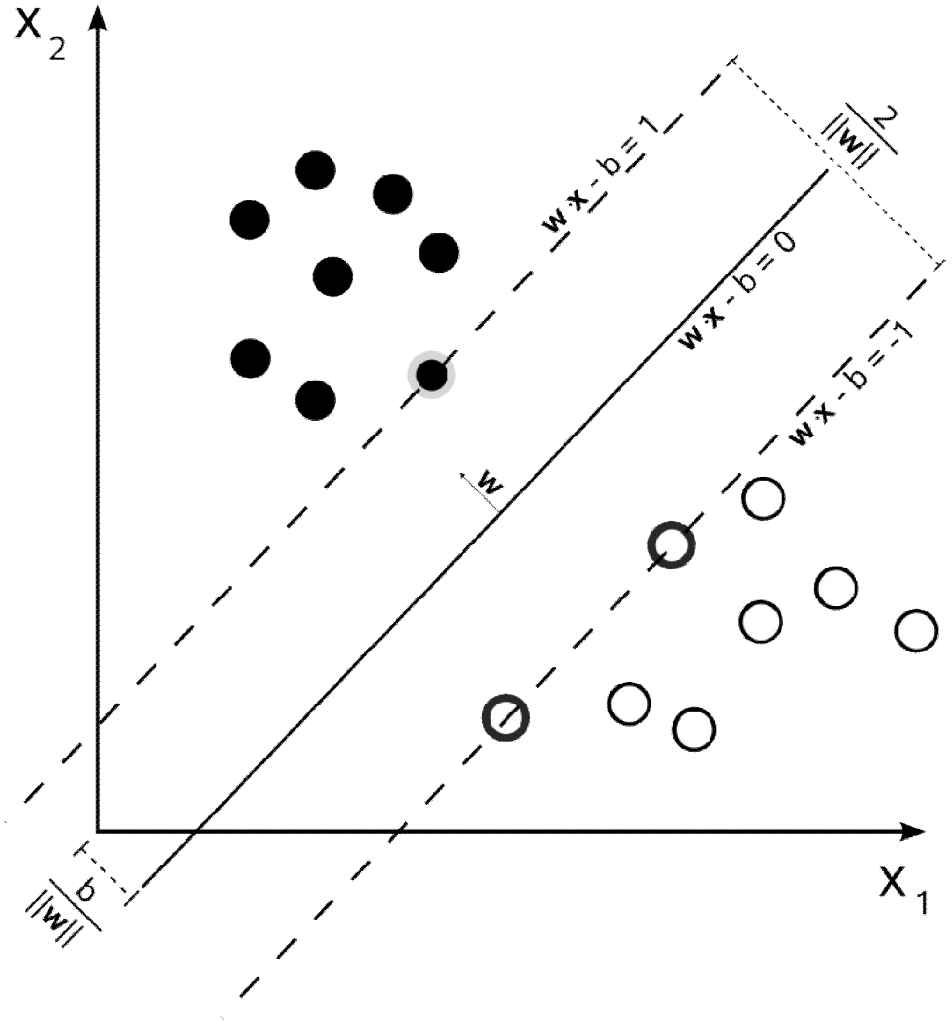


Рис. 1.9 Лінійна опорно-векторна модель

Якщо дані вибірки є лінійно роздільні, то паралельні гіперплощини які розділяють дані таким чином, що відстань між ними є якомога більшою описуються рівняннями:

$$\boldsymbol{w \cdot x - b = 1}$$

$$\boldsymbol{w \cdot x - b = -1}$$

З геометричної точки зору відстань між цими двома гіперплощинами є

$$\frac{\boldsymbol{2}}{\boldsymbol{\|w\|}}$$

Отже, для максимізації відстані між ними потрібно мінімізувати $\|w\|$ [[33]]

Додавши наступні нерівності ми обмежуємо попадання точок даних до розділення:

$$\boldsymbol{w \cdot x - b \geq 1}, \qquad \text{якщо } \boldsymbol{y_i = 1}$$

$$\boldsymbol{w \cdot x - b \leq -1}, \qquad \text{якщо } \boldsymbol{y_i = -1},$$

для кожного *i*

Ці обмеженні стверджують, що кожна точка повинна лежати з «правильного» боку розділення.

Останні вирази можна переписати так:

$$\boldsymbol{y(w \cdot x - b) \geq 1} \text{ для всіх } \boldsymbol{1 \leq i \leq n} \qquad (1)$$

і в загальному вигляді оптимізаційна задача матиме вигляд:

«Мінімізувати $\|w\|$ за умови $y \cdot (w \cdot x - b) \geq 1$ для $i = 1..n$»

*w,*та *b,* які розв'язують цю задачу, визначають наш класифікатор:

$$\boldsymbol{x \rightarrow sgn(w \cdot x - b)}$$

Важливим наслідком наведеної геометричної інтерпретації є те, що максимально розділова гіперплощина повністю визначається тими векторами *x*, які лежать найближче до неї. Ці вектори і називають *опорними векторами.*

М’яке розділення

У випадку, коли дані не є лінійно роздільними, вводиться наступна функція втрат:

$$\max\,(0{,}1 - y_i(w \cdot x - b))$$

Дана функція набуває нульових значень, коли задовольняється рівність (1). Для даних які лежать з «неправильного» боку розділення значення цієї функції є пропорційними до відстані від розділення

Тоді задача мінімізації розширюється так

«мінімізувати» $\left[\frac{1}{n}\sum_{i=1}^{n}\max\left(0{,}1 - y_i(w \cdot x - b)\right)\right] + \lambda\|w\|^2$

де регуляційний параметер λ визначає компроміс між збільшенням розміру розділення та забезпеченням того що x_i лежить з правильного боку розділення.

Нелінійна класифікація

У випадку коли множини, які потрібно розрізнювати, не є в заданому просторі лінійно роздільними, запропоновано відображати первинний простір до простору вищої розмірності, роблячи розділення можливим у тому просторі. Отриманий алгоритм є формально аналогічним лінійно роздільному, за вийнятком того, що кожен скалярний добуток замінено нелінійною ядровою функцією:

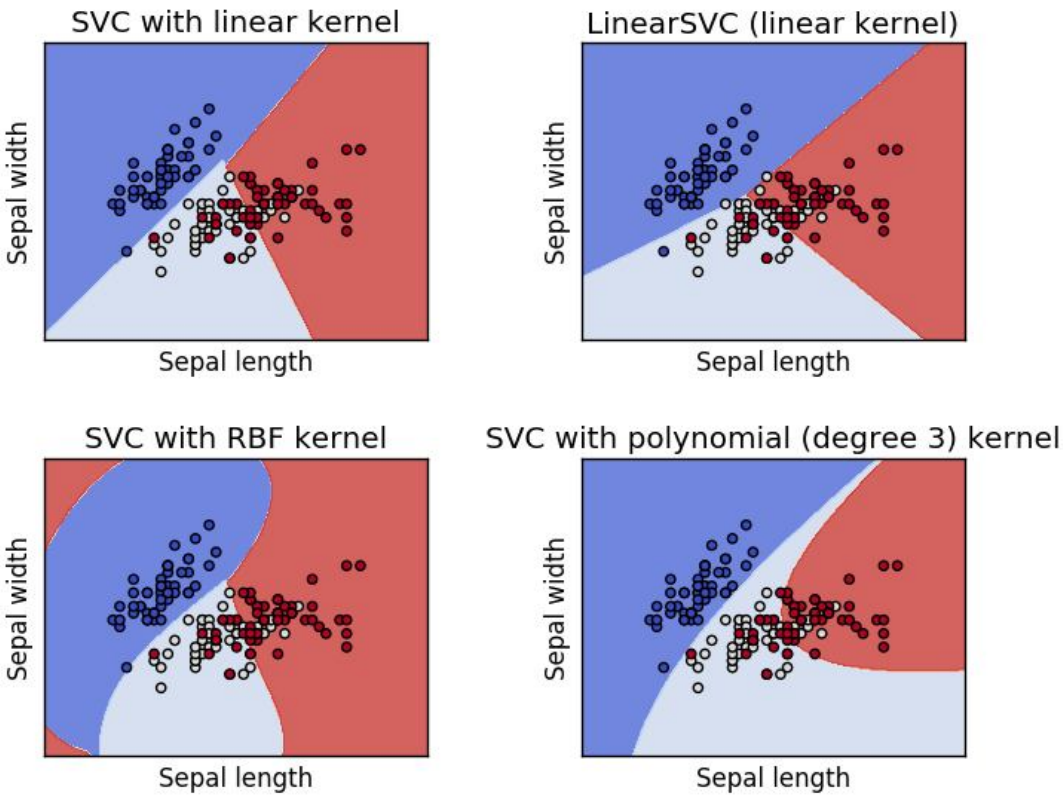


Рис. 1.10 SVM класифікатор з різними ядровими функціями

До деяких напоширеніших ядер належать:

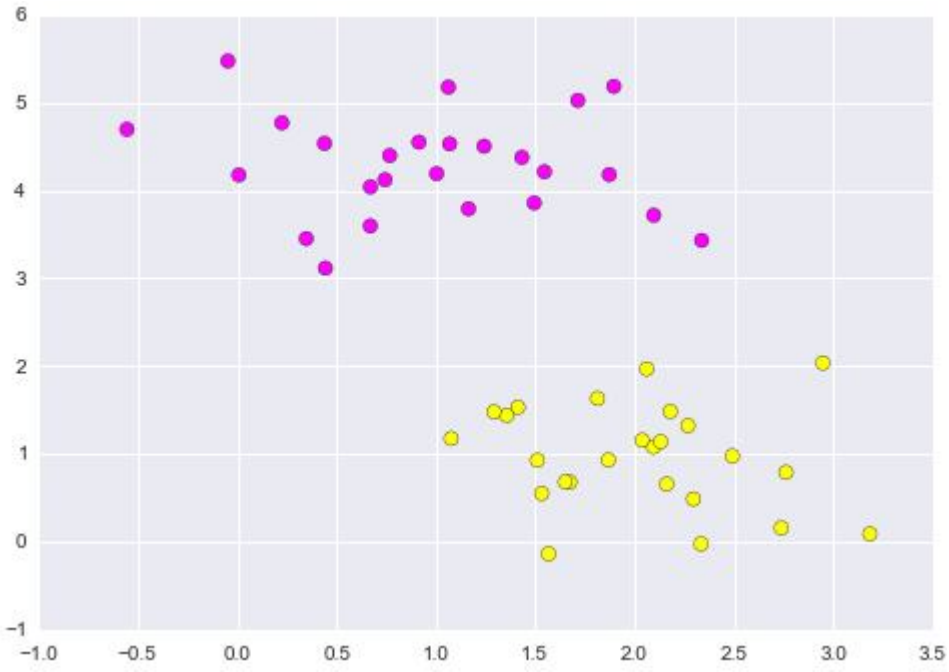
- поліноміальне: $k(x_i, x_j) = (x_i \cdot x_j + r)^d$
- гаусова радіально-базисна функція: $k(x_i, x_j) = \exp\left(-\gamma \|x_i - x_j\|^2\right)$ для $\gamma > 0$
- сігмоїдна функція: $k(x_i, x_j) = tanh(kx_i \cdot x_j + c)$ для деяких $\kappa > 0$ та $c < 0$

Практичний розгляд методики проведемо з використанням мови програмування Python та бібліотеки машинного навчання Scikit-Learn (http://scikit-learn.org).

```
Імпортуємо бібліотеки для побудови графіків та роботи з матрицями
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
from scipy import stats
```

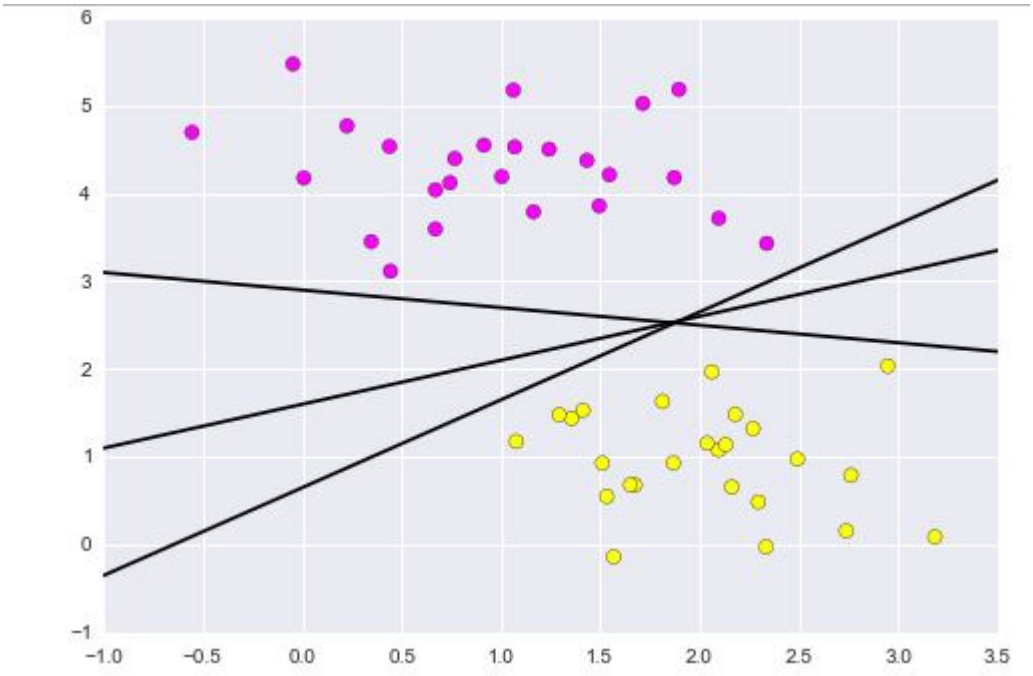
```
# use seaborn plotting defaults
import seaborn as sns; sns.set()
```

```
# Імпортуємо вбудований (бібліотечний) набір даних
from sklearn.datasets.samples_generator import make_blobs
X, y = make_blobs(n_samples=50, centers=2,
                  random_state=0, cluster_std=0.60)
plt.scatter(X[:, 0], X[:, 1], c=y, s=50, cmap='spring');
```



Дискримінантний класифікатор лінійно розділяє набір даних, проте варіантів розділення може бути декілька:

```
xfit = np.linspace(-1, 3.5)
plt.scatter(X[:, 0], X[:, 1], c=y, s=50, cmap='spring')

for m, b in [(1, 0.65), (0.5, 1.6), (-0.2, 2.9)]:
    plt.plot(xfit, m * xfit + b, '-k')

plt.xlim(-1, 3.5);
```



Навчимо класифікатор на основі методу опорних векторів здійснити це розділення:

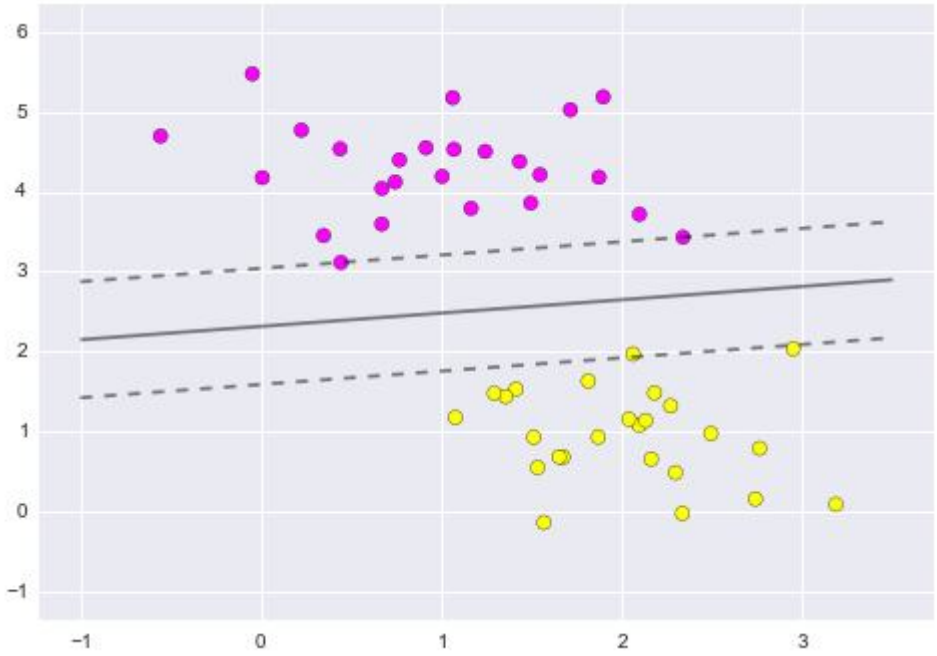
```
from sklearn.svm import SVC # "Support Vector Classifier"
clf = SVC(kernel='linear')
clf.fit(X, y)
```

Для кращої візуалізації створимо додаткову функцію, яка буде відображати розділяючі гіперплощини:

```
def plot_svc_decision_function(clf, ax=None):
    """Відорбражає графік вирішуючої функції (decision function) для 2D"""
    if ax is None:
        ax = plt.gca()
    x = np.linspace(plt.xlim()[0], plt.xlim()[1], 30)
    y = np.linspace(plt.ylim()[0], plt.ylim()[1], 30)
    Y, X = np.meshgrid(y, x)
    P = np.zeros_like(X)
    for i, xi in enumerate(x):
        for j, yj in enumerate(y):
            P[i, j] = clf.decision_function([xi, yj])

    # plot the margins
    ax.contour(X, Y, P, colors='k',
               levels=[-1, 0, 1], alpha=0.5,
               linestyles=['--', '-', '--'])
```

Для нашого випадку:

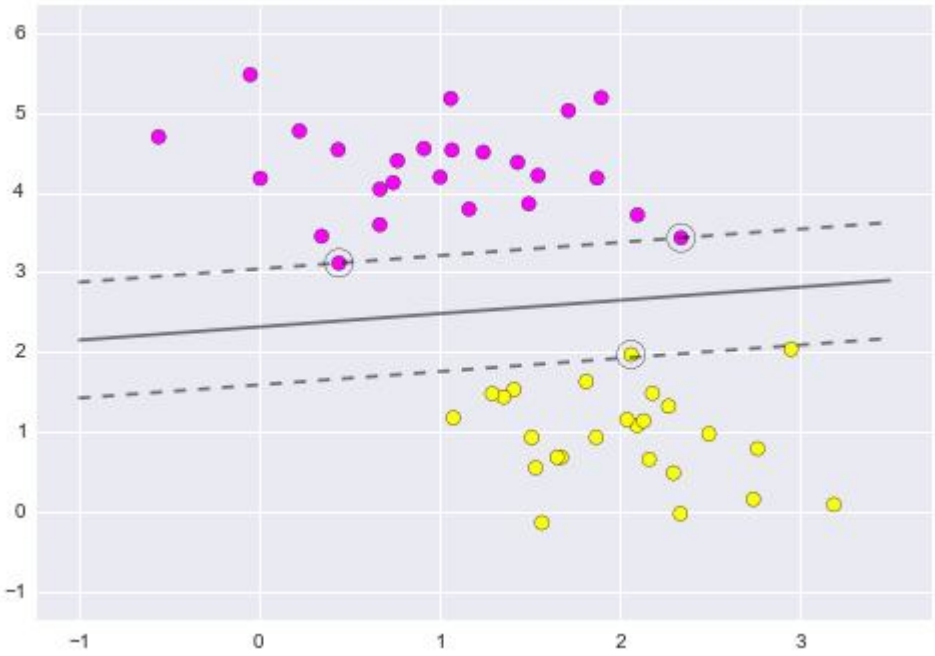
```
plt.scatter(X[:, 0], X[:, 1], c=y, s=50, cmap='spring')
plot_svc_decision_function(clf);
```



Бачимо, що лінії розподілу проходять через найближчі точки – опорні вектори. В бібліотеці scikit-learns вони зберігаються в атрибуті support_vectors класифікатора.

Позначимо ці точки:

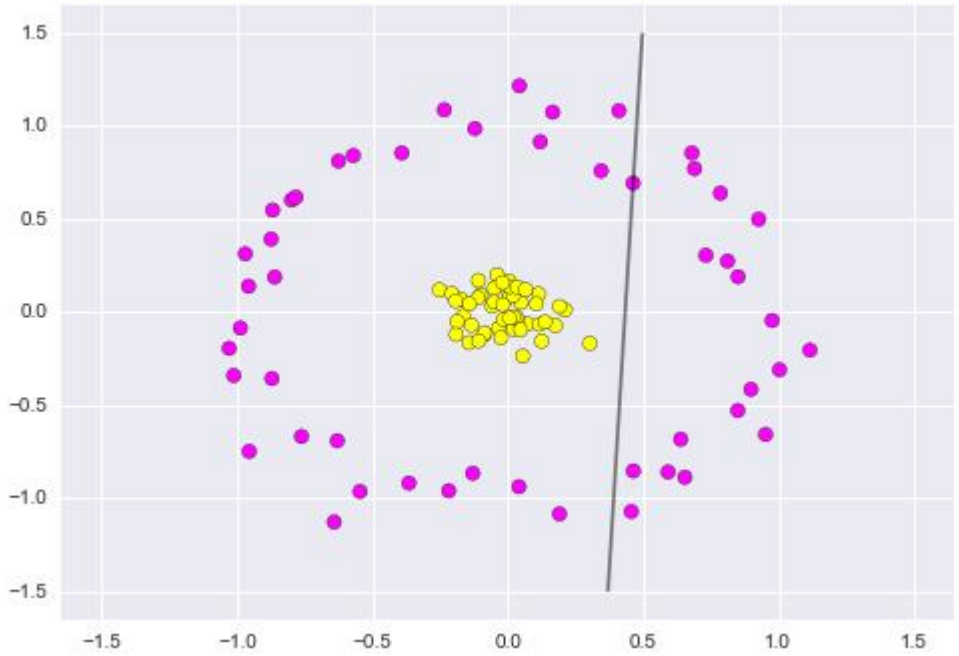
```
plt.scatter(X[:, 0], X[:, 1], c=y, s=50, cmap='spring')
plot_svc_decision_function(clf)
plt.scatter(clf.support_vectors_[:, 0], clf.support_vectors_[:, 1],
            s=200, facecolors='none');
```



Лінійно нероздільні випадки

Спробуємо навчити класифікатор з лінійним ядром:

```
from sklearn.datasets.samples_generator import make_circles
X, y = make_circles(100, factor=.1, noise=.1)

clf = SVC(kernel='linear').fit(X, y)

plt.scatter(X[:, 0], X[:, 1], c=y, s=50, cmap='spring')
plot_svc_decision_function(clf);
```

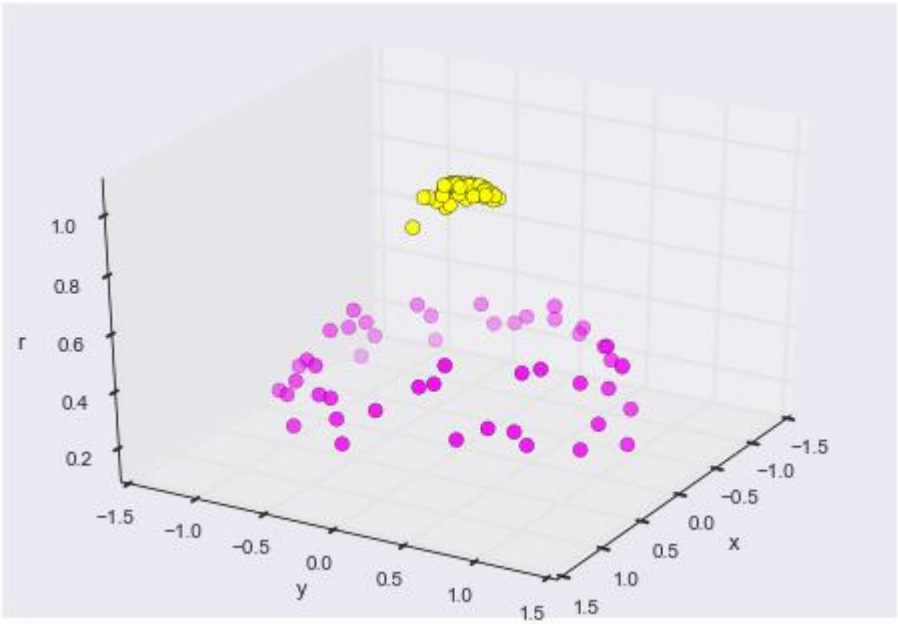


Додамо ядрову функцію (радіально – базисна функція):

```
r = np.exp(-(X[:, 0] ** 2 + X[:, 1] ** 2))
```

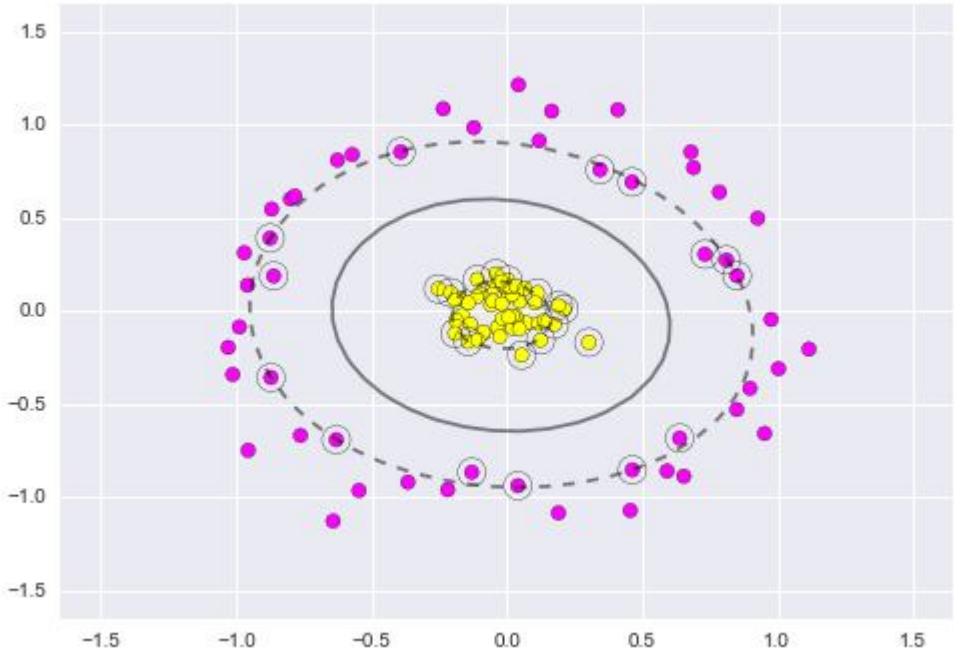
Спробуємо відобразити наші дані перетворивши їх з допомогою ядрової функції:

```
from mpl_toolkits import mplot3d

def plot_3D(elev=30, azim=30):
    ax = plt.subplot(projection='3d')
    ax.scatter3D(X[:, 0], X[:, 1], r, c=y, s=50, cmap='spring')
    ax.view_init(elev=elev, azim=azim)
    ax.set_xlabel('x')
    ax.set_ylabel('y')
    ax.set_zlabel('r')

plot_3D()
```



Бачимо, що додавши одну додаткову розмірність, дані стали лінійно розділювальними. Використання ядра описується параметром *kernel='rbf'*:

```
clf = SVC(kernel='rbf')
clf.fit(X, y)

plt.scatter(X[:, 0], X[:, 1], c=y, s=50, cmap='spring')
plot_svc_decision_function(clf)
plt.scatter(clf.support_vectors_[:, 0], clf.support_vectors_[:, 1],
            s=200, facecolors='none');
```



1.8 Метод опорних векторів: однокласова класифікація

Однокласова класифікація за допомогою машини опорних векторів використовується для виявлення аномалій, викидів, нетипової поведінки, детекції «новинок» в датасеті. На навчаючому наборі даних (нормальних), метод будує м'яку межу (soft boundary) цього набору і на базі неї класифікує нові точки даних – належать вони до цього набору чи ні.

Припустимо, що заданий набір даних характеризується розподілом ймовірностей P і необхідно «оцінити» підмножину S вхідного простору таким чином, що ймовірність тестової точки з розподілу P лежати поза S рівна деякому апріорно заданому значенню від 0 до 1. Метод пропонує спосіб підійти до цієї проблеми, намагаючись встановити функцію f яка позитивна на S і негативна на доповненні.

Функціональна форма f задається розкладом ядра в термінах потенційно невеликої підмножини навчальних даних. Вона регуляризується

шляхом контролю довжини вектора вагових коефіцієнтів у відповідному просторі ознак. Коефіцієнти розкладання шукаються шляхом вирішення задачі квадратичного програмування. Проводиться послідовна оптимізація по парам вхідних патернів. Алгоритм є природнім продовженням алгоритму опорних векторів на випадок нерозмічених даних.

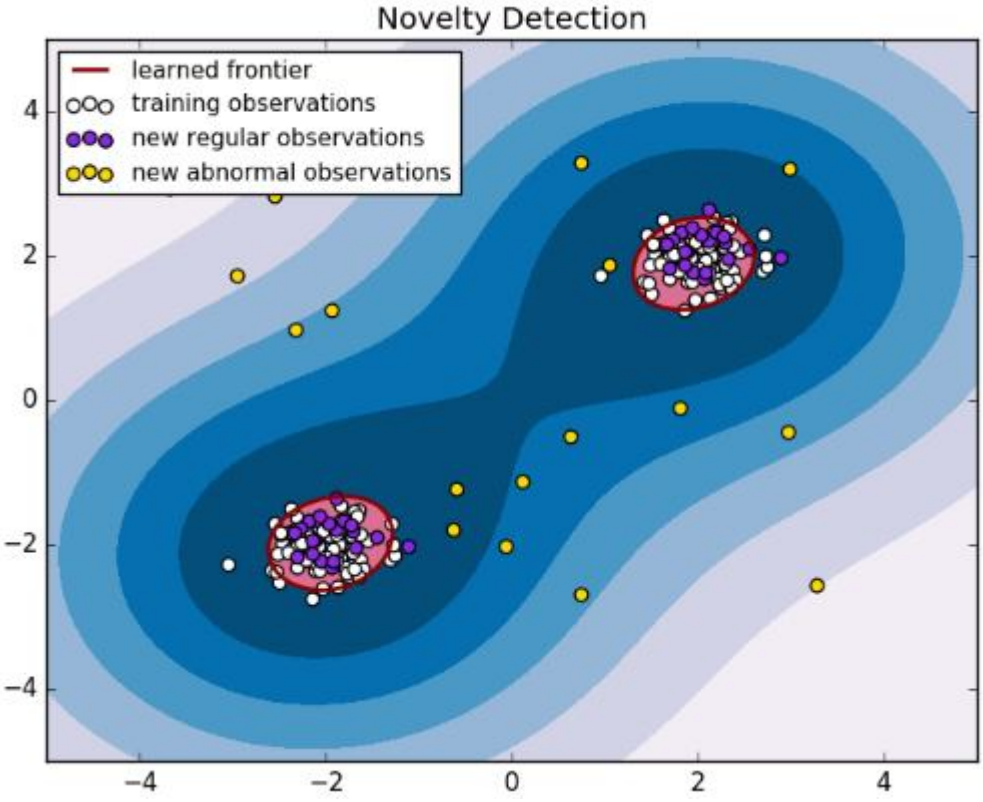


Рис. 1.11 Виявлення аномалій методом опорних векторів

1.9 Штучні нейронні мережі в задачах виявлення аномалій

Пригадаємо формальне означення задачі класифікації, як одного з розділів машинного навчання. Нехай X – множина описів об'єктів, Y – множина номерів (або міток класів). Існує невідома цільова залежність – відображення: $y^* : X \leftarrow Y$, значення якої відомі лише на об'єктах навчаючої вибірки $X^m = \{(x_1, y_1), \dots, (x_m, y_m)\}$. Необхідно побудувати алгоритм $\alpha : X \leftarrow Y$, здатний класифікувати довільний об'єкт $x \in X$. Якщо множина Y

складається лише з двох елементів, то має місце бінарна (двокласова) класифікація. В цьому випадку умовно вважають, що $Y = \{-1,1\}$ і один з класів - позитивним другий негативним.

Розглянемо випадок бінарної класифікації, коли множина X^m має структуру $\{(x_1, y_1), \dots, (x_m, y_1)\}$, тобто відома тільки така звана *позитивна* вибірка. Ціль залишається незмінною. В цьому випадку має місце *однокласова* класифікація, або *класифікація з навчанням на основі тільки позитивних прикладів*.

Одним з способів реалізації однокласового класифікатора є використання багатошарового перцептрона в якості адаптивного фільтра. Перецептрон – це штучна нейронна мережа прямого поширення. Навчання нейронних мереж на основі тільки позитивних прикладів було вперше розглянуто в роботах [5] і [19].

Нехай існує набір векторів V_n, кожен з яких є *позитивним* і має розмірність *m*. Виберемо деяку метрику *M*, яка буде описувати відстань між векторами (в якості метрики можна взяти відстань евклідову чи Чебишева). Побудуємо штучну нейронну мережу прямого поширення з *m* вхідними нейронами, *h* нейронами прихованого шару та *m* вихідними нейронами. При цьому прихований шар має сигмоїдну функцію активації, вихідний - лінійну.

Для навчання мережі методом зворотного поширення помилки будемо використовувати навчаючу вибірку X^n виду: $\{(x_1, x_1), \dots, (x_n, x_n)\}$ (для кожного прикладу, в якості вектора результатів використано той самий вектор). Тобто побудована штучна нейронна мережа буде працювати як *адаптивний фільтр*, який, отримавши на вхід сигнал (вектор), повинен без спотворення подати його на вихід. Отже, для векторів, «подібних» на вектори навчаючої вибірки, відстань $M(x_i, y_i) \leftarrow 0$, де y_i – вихідний вектор ШНМ при подачі на вхід вектора x_i. Після навчання мережі, необхідно пропустити всю навчаючу вибірку через ШНМ і отримати порогове значення метрики $treshold = max_{\forall i \in (1,n)} M(x_i, y_i)$. Після цього, для отримання рішення чи є довільний вектор позитивним, достатньо визначити засобом побудованої та

навченої ШНМ вектор y та перевірити – чи не перевищує значення $M(x, y)$ значення *threshold*.

ШНМ, що реалізує роботу детектора аномалій подано на рис 1.11

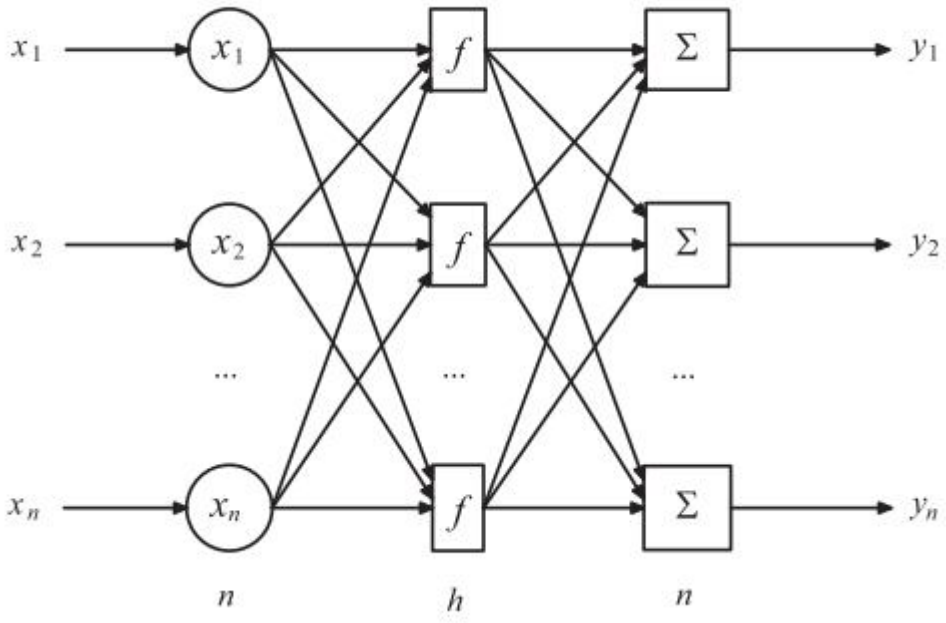


Рис. 1.11 Архітектура штучної нейронної мережі для задач виявлення аномалій

На ефективність роботи такого детектора аномалій впливають наступні параметри, які будуть експериментально встановлені в нашому дослідженні:

- *h* – кількість нейронів прихованого шару, що напряму впливає на обсяг «пам’яті» ШНМ; якщо *h* буде більше або рівне *m,* то виникає ризик «перенавчання», якщо *h* буде надто малим, то можливе «недонавчання» мережі (позитивний клас буде недостатньо визначений;
- *lr* – коефіцієнт швидкості навчання (lr, learning rate), що впливає як на швидкість навчання, так і на ефективність процесу «сходження»; при великих значеннях *lr* процес навчання ШНМ може не на завершитись, при малих - модель може навчатись надто довго або «перенавчитись» (втратити здатність «генералізувати» результати на дані що не належать навчаючій вибірці).

- α - параметер регуляризації (регуляризація типу L2) – допомагає уникати «перенавчання» шляхом штрафування нейро-коефіцієнтів з великими значеннями.

1.9 Методи оцінки результатів роботи програмних додатків виявлення аномалій

Результати роботи алгоритмів машинного навчання в задачах виявлення нетипової поведінки можуть мати такі види:

- **Мітки –** кожен екземпляр тестової вибірки отримує мітку *нормальний* або *аномальний*. Цей вид особливо притаманний системам, що базуються на моделях класифікації.
- **Оцінки** – кожному екземпляру тестової вибірки ставиться у відповідність оцінка аномальності. Це дозволяє ранжувати вихідні дані, проте потребує додаткового порогового параметру.

Оскільки випадки аномальної поведінки системи за визначенням є рідкими, задача оцінки результатів роботи алгоритмів машинного навчання є особливо складною. В загальному випадку більш кращий алгоритм визначається такою базовою оцінкою:

$$Base\ Rate = \ argmax \frac{1}{l} \sum_{i=1}^{l} [y_0 = \ y_i]$$

l – кількість членів тестової вибірки
y_i - результат передбачення
y_0 – значення елемента даних визначене експертом

Для оцінки результатів роботи окремих алгоритмів машинного навчання використовують різні підходи. Більшість з них базується на побудові (на основі «прогону» тестовою вибіркою) наступної матриці (Confusion Matrix):

Таблиця 1.2 Confusion Matrix

Клас визначений експертом (актуальні дані)	Передбачені класи	
	AC	**NC**
AC	**TP**	**FN**
NC	**FP**	**TN**

AC – аномальний клас

NC – нормальний клас

TP – True Positive - правильно передбачений позитивний клас (аномалія)

FP – False Positive – помилково передбачений позитивний клас (передбачено аномальність насправді нормальна поведінка)

FN - False Negative – помилково передбачена нормальність, насправді точка даних є викидом-аномалією

TN – True Negative – правильно передбачена нормальність даних.

Різні оцінки алгоритмів будуються на різних співвідношеннях елементів даної матриці.

Найпростіша з них – Accuracy – точність – відношення кількість правильно передбачених випадків до загального числа випадків (довжини вибірки):

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

То, що дана міра не є характеристичною метрикою якості систем виявлення аномалій, легко бачити на такому прикладі:

Вибірка з трафіку з мережевого шлюзу містить 99.9% нормальних даних і 0.1% даних що є результатом вторгнення. Розглянемо тривіальний

класифікатор який просто позначає будь-який елемент даних за нормальний. Згідно вищенаведеної формули точність роботи цього класифікатора сягне 99.9%!

Більш адекватні оцінки базуються на використанні мір *Precision* і *Recall.*

Міра *Precision* (точність) задається наступним відношенням:

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

і формально описує рівень довіри класифікатору - процент знайдених знайдених викидів, які дійсно є викидами.

Міра *Recall* (повнота) задається відношенням:

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

і описує як багато об'єктів знаходить алгоритм які є дійсними аномаліями – процент дійсних аномалій серед всіх які система визначила за аномалії.

Чим вища точність - тим менше хибних спрацювань, чим вища повнота – тим менше хибних попусків. Змінюючи параметри алгоритму (чи порогове значення для алгоритмів які в результаті роботи видають не мітку класу а оцінку приналежності) можна побудувати криву залежності точності і повноти. Зазначимо, що ця крива не обов'язково є монотонною, також високі значення точності можуть відповідати низьким рівням повноти і навпаки.

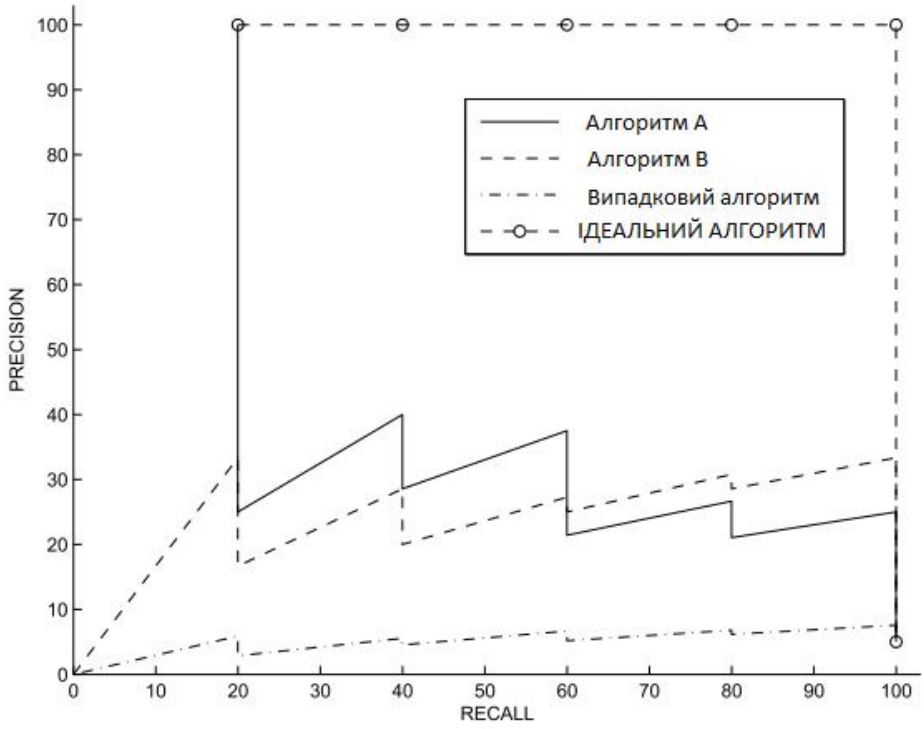


Рис. 1.11 Приклади кривих Точності-Повноти

Міра яка враховує співвідношення і повноту називається F-міра, та обчислюється згідно наступного виразу:

$$F = \frac{2 * Precision * Recall}{Precision + Recall}$$

Мірою якості певного алгоритму також є площа під кривою точності-повноти і відома як AUC-PRC – Area Under Precision-Recall Curve.

Інша крива - *Receiver Operating Characteristics Curve* (ROC) тісно пов'язана з кривою PRC, проте в деяких випадках більш інтуїтивно зрозуміла. В цьому випадку будують графік залежності *True Positive Rate* від *False Positive Rate.*

True Positive Rate (Detection Rate) - обчилюється так само як повнота (*Recall)* і визначає співвідношення між числом коректно виявлених аномалій та загального числа аномалій

False Positive Rate (False Alarm):

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

- визначає співвідношення між числом записів даних які будучи нормальними система передбачила за аномальні та загального числа записів «нормального» класу.

Параметер TPR – визначає чутливість алгоритму, параметер FPR – визначає специфічність алгоритму (виборча характеристика).

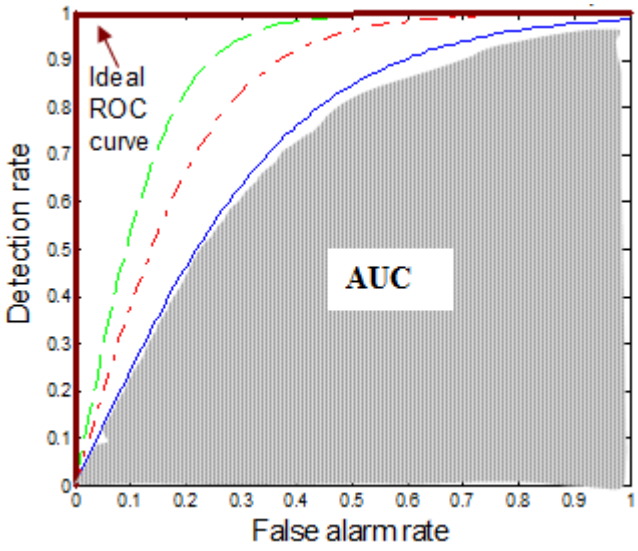


Рис. 1.12 ROC криві різних алгоритмів виявлення аномалій

Площа під кривою ROC відома як *AUC-ROC (Area Under ROC Curve)* змінюється в діапазоні від 0.5 до 1 і формально визначає ймовірність, що випадково взятий об'єкт класу 1 (аномалія) отримує оцінку вищу за випадково взятий об'єкт класу 0. Випадковий алгоритм (випадково приписує мітку нормальний чи аномальний) на графіку ROC буде зображений діагональною лінією від (0,0) до (1,1). Графіки вище цієї лінії будуть відповідати алгоритмам кращої якості за випадковий – чим вище графік тим більша точність алгоритму.

Перевага ROC кривої над PRC в її монотонності та більш простої інтерпретації, проте часом рівновага між точністю і повнотою спостерігається більш детально на PR кривій.

1.10 Висновки до розділу

В розділі розглянуто основні моделі даних які застосовуються в задачах виявлення нетипової поведінки: аналіз екстремальних значень, статистичні та ймовірнісні моделі, лінійні моделі, спектральні моделі, моделі аналізі подібності, теоретико інформаційні моделі.

Обґрунтовано важливість вибору правильної моделі даних при розробці систем виявлення аномалій. Розглянуто специфіку проблеми виявлення аномалій у колекціях даних великої розмірності.

Розглянуто базові типи даних які використовуються при моделюванні задач виявлення нетипової поведінки системи: категорійні, текстові та змішані атрибути, часові ряди та потоки даних, дискретні послідовності, просторові дані.

Розглянуто специфіку виявлення аномалій на розмічених даних

Детально розглянутий метод опорних векторів як один з найбільш потужних засобів машинного навчання, та його розширення на випадок завдань виявлення аномалій.

Розглянуто існуючі архітектури класифікаторів на основі штучних нейронних мереж для роботи в режимі виявлення аномалій.

Проведено опис основних методів оцінки роботи програмних додатків виявлення аномалій, роглянуто метрики якості: точності-повноти, та ROC – Робоча характеристика приймача.

РОЗДІЛ 2. Системи виявлення вторгнень на основі засобів машинного навчання

2.1 Системи виявлення вторгнень – традиційний підхід

Виявлення вторгнень – процес моніторингу подій в комп'ютерній системі чи мережі та аналіз їх на предмет несанкціонованого доступу. Під вторгненням будемо розуміти спробу обійти механізми безпеки комп'ютера чи мережі.

У зв'язку з широким поширенням мережі Інтернет все більше та більше організацій стають уразливими для кібер-атак. Складність механізмів атак а також масштаб загроз невпинно зростають.

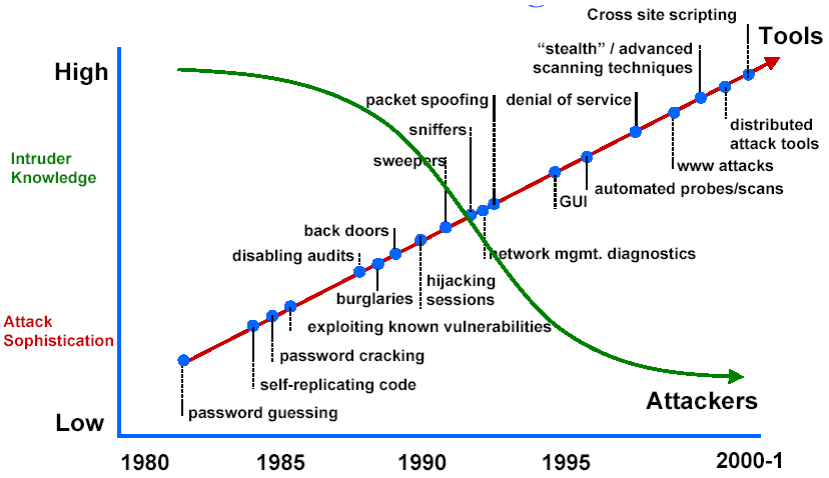


Рис. 2.1 Складність атак та рівень технічних знань зловмисників (джерело: www.cert.org/archive/ppt/cyberterror.ppt)

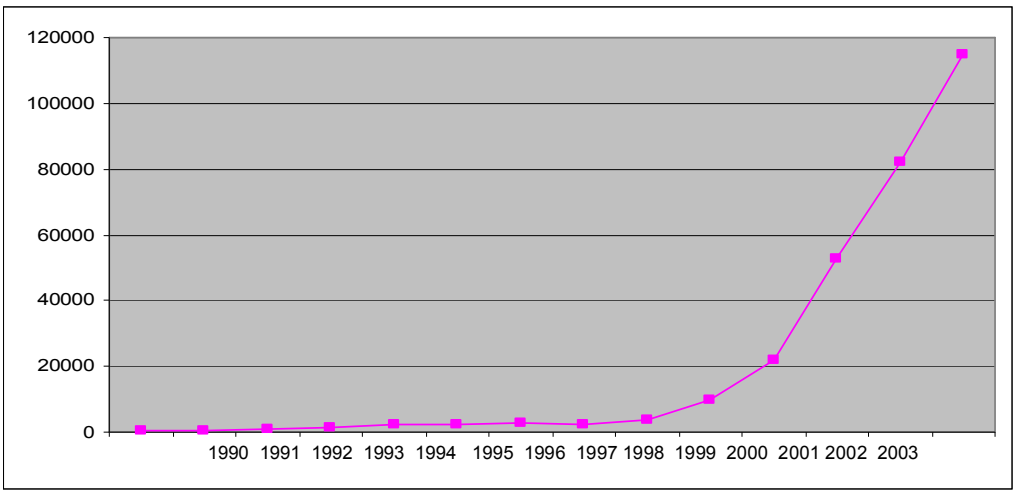


Рис. 2.2 Кількість повідомлень про атаки на обчислювальні системи до Computer Emergency Response Team/Coordination Center (CERT/CC)

Механізми безпеки завжди мають неминучі невикриті уразливості. Мережеві екрани(фаєрволи) не є достатніми для захищеності мереж. Також складним є викриття атак «зсередини» (Insider attacks).

Традиційні системи виявлення вторгнень такі як, наприклад, широко розповсюджена система SNORT, базуються на сигнатурах відомих атак. Сигнатури в системі SNORT задаються правилами виду (MS-SQL “Slammer” worm):

```
any -> udp port 1434 (content:"|81 F1 03 01 04 9B
81 F1 01|"; content:"sock"; content:"send")
```

Такі правила повинні постійно оновлюватись для кожного нового виду атак. Це робить їх неефективними при застосуванні зловмисником нових способів кібератак.

Іншою проблемою традиційного підходу є суттєва затримка в розгортанні новостворених сигнатур на комп’ютерні системи.

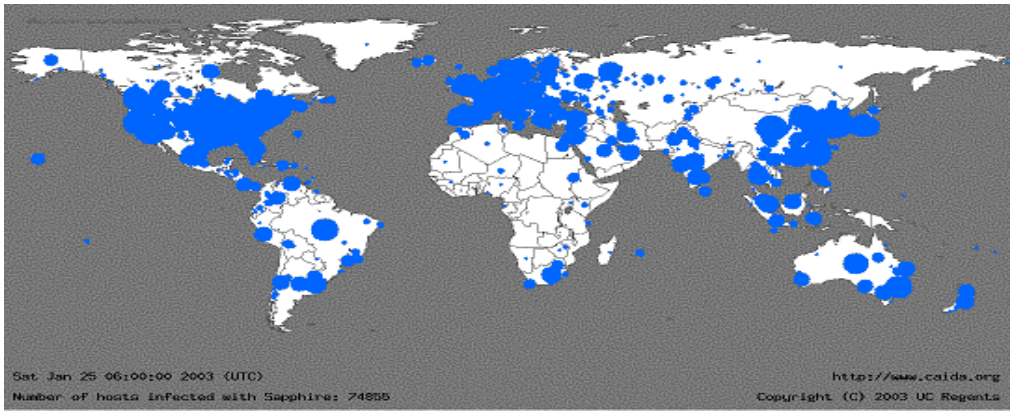


Рис. 2.3 Географія розповсюдження віруса Sapphire/Slammer через 30 хв після запуску у мережу

2.2 Виявлення вторгнень – підхід з використанням методів машинного навчання

Методи знаходження нетипової поведінки, аномалій, що базуються на моделях «нормальної» поведінки користувачів, обчислювальних вузлів, мереж, виявляють атаки як значне відхилення від цих моделей. Основною перевагою такого підходу є потенційна здатність системи розпізнати невідомі раніше атаки. Також ці системи можуть виявляти дії зловмисника які важко описати сигнатурами.

Підходи з використанням методів машинного навчання до проблеми виявлення вторгнень можна поділити на 2 класи:

1) **виявлення неправильного використання (Misuse detection)** – побудова прогностичної моделі на основі розмічених даних (екземпляри помічені як «нормальні» або «проникні»)
   - демонструють високу точність виявлення великої кількість відомих атак
   - не можуть виявити невідомі та нові атаки
2) **виявлення аномалій**

- система може виявляти нові атаки як відхилення від «нормальної» поведінки

- можливий високий рівень хибних «спрацьовувань», так як виявлене відхилення не завжди являють собою реальне вторгнення.

У ході дослідження розроблена система виявлення аномалій для захисту мережі та робочих станцій від різноманітних проникних/підозрілих активностей.

2.3 Набір даних (Dataset)

В 1998 році міжнародна організація DARPA (Управління перспективних дослідницьких проектів міністерства оборони США) у співпраці з MIT LincoLn Labs приготували документ – Intrusion Detection Evaluation Program, який регулює принципи оцінки роботи систем захисту мереж від вторгнень зловмисників. Документ описує стандартний набір даних, і включає широке різноманіття змодельованих вторгнень. Розроблювана нами система використовує подібну структуру даних. Це дає змогу нам тестувати запропоновану методику на цьому стандартному датасеті а також порівнювати якість її роботи з іншими системами виявлення вторгнень.

LincoLn налаштували середовище для збору TCP-dump мережевих даних протягом 7-ми місяців. Було опрацьовано близько 5-ти мільйонів записів про з'єднання. Подібно було зібрані тестові дані (2 тижні ти близько 2 мільйони записів).

Ми поступили подібним чином, проте строки які ми могли виділити під цей експеримент були дещо коротші. Дані для навчання моделей збирались протягом 3-х тижнів, дані для тесту – протягом тижня.

Кожен запис стандартного набору DARPA для оцінки систем захисту складається з близько 100 байтів інформації і промарковано як нормальний або зловмисний (атака).

Всі тики атак можуть бути поділені на 4 основні категорії:

- DOS: denial-of-service - syn flood;

- R2L: неавторизований доступ з віддаленої машини – підбір паролів;
- U2R: неавторизований доступ до локальних привілеїв суперкористувача (root) – різноманітні атаки типу «переповнення буфера»
- Probing: спостереження та інші зондування – наприклад, сканування портів.

Важливо відмітити, що тестовий набір MIT LoncoLn Labs не з того ж ймовірнісного розподілу що і навчаючий набір, і включає специфічні види атак, які не присутні в навчаючому наборі.

Основні ознаки

Поля даних, що відображають основні властивості з'єднання подано в таблиці 2.1

Назва	Опис	Тип
Duration	довжина (число секунд) з'єднання	безперервний
protocol_type	тип протоколу: tcp, udp, icmp	дискретний
Service	мережевий сервіс: http, telnet, smtp тощо	дискретний
Flag	статус з'єднання: нормально чи помилка	дискретний
src_bytes	кількість байтів з джерела до пункту призначення	безперервний
dst_bytes	кількість байтів з пункту призначення до джерела	безперервний
Land	1 якщо з'єднання від/до того ж host/ip	дискретний
wrong_fragment	кількість помилкових фрагментів (wrong fragments)	безперервний
Urgent	кількість термінових пакетів	безперервний

Таблиця 2.1 Осовні ознаки даних про вторгнення

Контекстні ознаки

На відміну від вищезгаданих ознак які збираються безпосередньо з шлюзового вузла мережі, наступні властивості є предметом конструювання (feature engineering) експертами галузі і допомагають повніше відобразити ознаки несанкціонованого доступу.

Назва	Опис	Тип
hot	кількість індикаторів «hot»	безперервний
num_failed_logins	кількість невдалих логінів	безперервний
logged_in	1 якщо вдалий вхід в систему	дискретний
num_compromised	кількість «скомпрометованих» умов	безперервний
root_shell	1 якщо права root отримано	дискретний
su_attempted	1 якщо спроба виконати "su root" команду	дискретний
num_root	кількість входів як root	безперервний
num_file_creations	кількість операцій створення файлів	безперервний
num_shells	кількість командних стрічок	безперервний
num_access_files	кількість операцій доступу до системних файлів	безперервний
num_outbound_cmds	кількість outbound команд протягом ftp сесії	безперервний
is_hot_login	1 якщо логін належить «гарячому» списку	дискретний
is_guest_login	1 якщо спроба гостьового входу	дискретний

Таблиця 2.2 Контекстні ознаки визначені експертом галузі

Ознаки трафіку засновані на часі (time-based)

Наступні ознаки обчислені методом рухомого часового вікна. Так ознаки «same host» перевіряються тільки для з'єднань протягом останніх 2 секунд які були адресовані тому ж хосту, і обчислюють статистичні характеристики пов'язані з поведінкою протоколу, служб тощо.

Подібно ознаки «same service» перевіряються для з'єднань протягом останніх 2 секунд і які призначені тій ж службі. Дані ознаки подано в таблиці

.

Назва	Опис	Тип
count	кількість з'єднань до того ж хосту	безперервний
наступні ознаки відносяться до з'єднань same-host		
serror_rate	% кількості з'єднань що мають «SYN» помилку	безперервний
rerror_rate	% кількості з'єднань що мають «REJ» помилку	безперервний
same_srv_rate	% кількості з'єднань до тієї ж служби	безперервний

diff_srv_rate	% кількості з’єднань до інших служб	безперервний
srv_count	кількість з’єднань до того ж сервісу на протязі 2-х с.	
	наступні ознаки відносяться до з’єднань same-service	
srv_serror_rate	% кількості з’єднань що мають «SYN» помилку	безперервний
srv_rerror_rate	% кількості з’єднань що мають «REJ» помилку	безперервний
srv_diff_host_rate	% кількості з’єднань до інших хостів	безперервний

Таблиця 2.3 Ознаки обчислені використовуючи 2-х секундне віко

2.4 Діаграма потоків даних проектованої системи

Діаграма потоків даних проектованої системи подана на рис 2.4

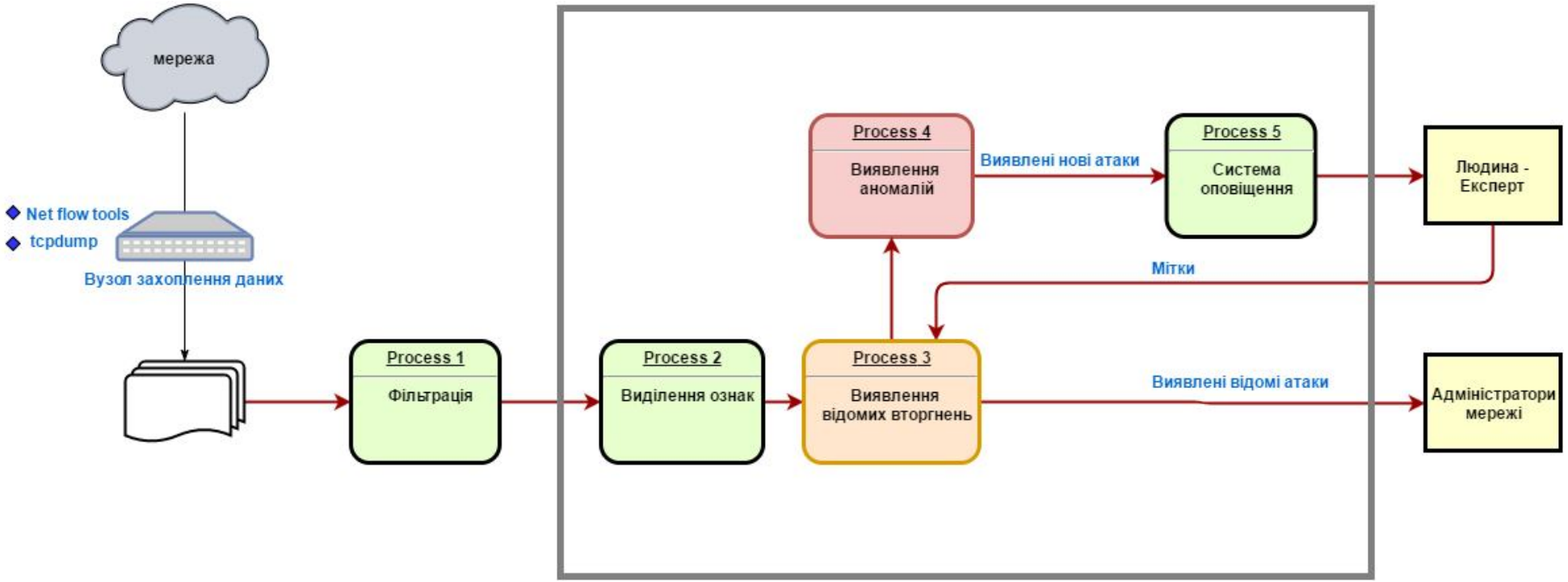


Рис. 2.4 Діаграма потоків даних системи виявлення вторгнень на базі засобів машинного навчання

2.5 Протокол дослідження моделей - Машини Опорних Векторів та Однокласової Машини Опорних векторів на наборі даних вторгнень LincoLn Labs

Дослідження проводитимо з використанням бібліотеки машинного навчання Scikit-learn, в середовищі Ipython Notebook.

Імпорт необхідних модулів:

```
# модулі побудови графіків
%matplotlib inline
import matplotlib.pyplot as plt
import seaborn as sns; sns.set()

# модулі машинного навчання та функції оцінювання моделей
from sklearn.svm import SVC, OneClassSVM
from sklearn.metrics import confusion_matrix
from sklearn.metrics import precision_recall_fscore_support
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score, recall_score

from sklearn.metrics import f1_score
from sklearn.metrics import roc_curve, auc

# модулі предпідготовки набору даних
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split

# модулі паралельних обчислень
from ipyparallel import require
from ipyparallel import Client

# модулі матричних обчислень та маніпуляції наборами даних
import numpy as np
import pandas as pd


# допоміжні модулі
import time
import itertools
```

Допоміжна функція для друку часу виконання коду:

```
# Функція приймає аргумент час виконання коду в секундах та вивиодить на друк у форматі
гг:хв:сс
def print_time(t):
    m, s = divmod(t, 60)
    h, m = divmod(m, 60)
    print("Час виконання: %d г: %02d хв : %02d с" % (h, m, s))
```

*** Зчитуємо набір даних з файла:***

```
# Для зручного і швидкого тестування моделей обмежимось лише 10% усіх наявних даних
data = pd.read_csv('e:/Projects/Dyplom/kddcup.data_10_percent1.txt')
#data=data.sample(frac=0.5)
```

Оцінимимо розмір вибірки:

```
print('data({0[0]},{0[1]})'.format(data.shape))
```

```
data(494019,42)
```

Додатковий масив з назвами полів даних:

```
data.columns=['duration',
'protocol_type',
'service',
'flag',
'src_bytes',
'dst_bytes',
'land',
'wrong_fragment',
'urgent',
'hot',
'num_failed_logins',
'logged_in',
'num_compromised',
'root_shell',
'su_attempted',
'num_root',
'num_file_creations',
'num_shells',
'num_access_files',
'num_outbound_cmds',
'is_host_login',
'is_guest_login',
'count',
'srv_count',
'serror_rate',
'srv_serror_rate',
'rerror_rate',
'srv_rerror_rate',
'same_srv_rate',
'diff_srv_rate',
'srv_diff_host_rate',
'dst_host_count',
'dst_host_srv_count',
'dst_host_same_srv_rate',
'dst_host_diff_srv_rate',
'dst_host_same_src_port_rate',
'dst_host_srv_diff_host_rate',
'dst_host_serror_rate',
'dst_host_srv_serror_rate',
'dst_host_rerror_rate',
'dst_host_srv_rerror_rate','state']
```

Додатковий масив з мітками класів даних:

```
#'normal' - "нормальні" безпечні з'єднання, та класи порушень-вторгнень
classes=['normal','back','buffer_overflow','ftp_write','guess_passwd','imap','ipsweep','land',
'loadmodule','multihop','neptune',

'nmap','perl','phf','pod','portsweep','rootkit','satan','smurf','spy','teardrop','warezclient'
,'warezmaster']
```

Додатковий словник, що описує типи полів даних у вибірці:

```
dataDescr={'duration': 'continuous',
'protocol_type': 'symbolic',
'service': 'symbolic',
'flag': 'symbolic',
'src_bytes': 'continuous',
'dst_bytes': 'continuous',
'land': 'symbolic',
'wrong_fragment': 'continuous',
'urgent': 'continuous',
```

```
'hot': 'continuous',
'num_failed_logins':'continuous',
'logged_in': 'symbolic',
'num_compromised': 'continuous',
'root_shell': 'continuous',
'su_attempted': 'continuous',
'num_root': 'continuous',
'num_file_creations': 'continuous',
'num_shells': 'continuous',
'num_access_files': 'continuous',
'num_outbound_cmds': 'continuous',
'is_host_login': 'symbolic',
'is_guest_login': 'symbolic',
'count': 'continuous',
'srv_count': 'continuous',
'serror_rate':' continuous',
'srv_serror_rate': 'continuous',
'rerror_rate': 'continuous',
'srv_rerror_rate': 'continuous',
'same_srv_rate': 'continuous',
'diff_srv_rate': 'continuous',
'srv_diff_host_rate': 'continuous',
'dst_host_count': 'continuous',
'dst_host_srv_count': 'continuous',
'dst_host_same_srv_rate': 'continuous',
'dst_host_diff_srv_rate': 'continuous',
'dst_host_same_src_port_rate': 'continuous',
'dst_host_srv_diff_host_rate': 'continuous',
'dst_host_serror_rate': 'continuous',
'dst_host_srv_serror_rate': 'continuous',
'dst_host_rerror_rate': 'continuous',
'dst_host_srv_rerror_rate': 'continuous'}
```

Переглянемо перші 5 записів вибірки:

```
data.head()
```

```
data.head()
```

	duration	protocol_type	service	flag	src_bytes	dst_bytes	land	wrong_fragment	urgent	hot	...	dst_host_srv_count	state
0	0	tcp	http	SF	239	486	0	0	0	0	...	19	normal.
1	0	tcp	http	SF	235	1337	0	0	0	0	...	29	normal.
2	0	tcp	http	SF	219	1337	0	0	0	0	...	39	normal.
3	0	tcp	http	SF	217	2032	0	0	0	0	...	49	normal.
4	0	tcp	http	SF	217	2032	0	0	0	0	...	59	normal.

5 rows × 42 columns

Виведемо на друк кількість "нормальних" записів (безпечних з'єднань) та кількість аномальних (вторгнення, сканування,...тощо):

```
print("Нормальних даних: %d, вторгнень: %d" % (len(data[data.state=='normal.']),
len(data[data.state!='normal.'])))
```

```
Нормальних даних: 97276, вторгнень: 396743
```

Виділимо окремо вектор з мітками класів та матрицю даних:

```
y=data["state"]
X=data.drop(labels=["state"], axis=1)
```

```
# видаляємо зайвий символ в мітці класів
y=y.str.rstrip('.')
```

Предпідготовка даних

Визначимо назви стовпців що мають символьний тип:

```
[k for k in dataDescr.keys() if dataDescr[k] == 'symbolic']

['protocol_type',
 'is_guest_login',
 'flag',
 'is_host_login',
 'land',
 'service',
 'logged_in']
```

Виділимо окремо матрицю з даними символьного типу (дискретні):

```
X_symbolic=X[['flag','service','protocol_type']]
```

	flag	service	protocol_type
0	SF	http	tcp
1	SF	http	tcp
2	SF	http	tcp
3	SF	http	tcp
4	SF	http	tcp

Виділимо окремо матрицю з данимим числового типу (безпрервні):

```
X_continuous =
X.drop(labels=["flag","service","protocol_type","land","logged_in","is_guest_login"], axis=1)
```

Кодуємо та переформатовуємо символьні значення в бінарну матрицю:

```
le_flag=LabelEncoder()
le_service=LabelEncoder()
le_protocol=LabelEncoder()

# кодуємо символьні значення числами від 0 до n_classes-1.
x_flag=le_flag.fit_transform(X_symbolic['flag'])
x_service=le_service.fit_transform(X_symbolic['service'])
x_protocol=le_protocol.fit_transform(X_symbolic['protocol_type'])

x_flag=x_flag.reshape(len(x_flag),1)
x_service=x_service.reshape(len(x_service),1)
x_protocol=x_protocol.reshape(len(x_protocol),1)

# Збираємо все в одну матрицю
X_numeric=np.hstack((x_flag,x_service,x_protocol))

n_1 = len(X_symbolic['flag'].unique())
n_2 = len(X_symbolic['service'].unique())
n_3 = len(X_symbolic['protocol_type'].unique())


# кодуємо категорійні числові значення за схемою one-of-K.
```

```
enc = OneHotEncoder(n_values=[n_1, n_2, n_3],sparse=False)
X_binary=enc.fit_transform(X_numeric)

#Збираємо всі дані в одну матрицю ознак
#(ознаки "land","logged_in","is_guest_login" є бінарними тому не вимагають  додаткової
обробки)

X_transformed = np.hstack((X_binary,X_continuous,X[["land","logged_in","is_guest_login"]]))
```

Розбиваємо вибірку на навчаючу та тестову:

```
X_train, X_test, y_train, y_test = train_test_split(X_transformed, y, test_size=0.3,
random_state=0)
```

Нормалізація даних

Оскільки моделі машини опорних векторів чутливі до абсолютних значень в окремих полях даних, нормалізуємо усі числові ознаки до одного діапазону [0..1]:

```
# Нормалізуємо числові поля вибірки використовуючи відповідну фукцію бібліотеки scikit-learn
scaler=StandardScaler()
X_train=scaler.fit_transform(X_train)
X_test=scaler.transform(X_test)
```

Навчання моделі класифікатора на основі методу опорних векторів

Створюємо екземпляр об'єкта класифікатора:

```
#створення об'єкта класифікатора використовуючи параметри за замовченням
clf=SVC(C=1, kernel='rbf',random_state=241)
```

Навчання моделі на даних навчаючої вибірки з заміром часу навчання:

```
startTime = time.time()
clf.fit(X_train,y_train)
print_time((time.time() - startTime))
```

```
Час виконання: 0 г: 04 хв : 21 с
```

Використовуючи навчену модель, виконаємо передбачення класу даних тестової вибірки:

```
y_predicted=clf.predict(X_test)
```

Визначимо базову оцінку якості моделі класифікатора:

```
accuracy=accuracy_score(y_test,y_predicted)
print("Accuracy %.4f" % accuracy)
```

```
Accuracy 0.9991
```

Визначимо загальну кількість даних у тестовій вибірці:

```
print("Довжина тестової вибірки: %s" % len(y_test))
```

```
Довжина тестової вибірки: 148206
```

Визначимо загальне число помилок класифікації:

```
print("Загальне число помилок класифікації: %s" % len(y_test[y_test!=y_predicted]))
```

```
Загальне число помилок класифікації: 139
```

Надрукуємо класи атак які модель не виявила ("пропустила"):

```
suma_compromized=0
for i in range(len(y_predicted)):
    if y_test.iloc[i]!=y_predicted[i]:
        if y_predicted[i] =='normal':
            suma_compromized+=1
            print("клас що передбачено: %s, дійсний клас: %s" % (y_predicted[i],
y_test.iloc[i]))

print('Кількість записів які передбачено що є нормальними а насправді є вторгненнями: %s' %
suma_compromized)
```

```
клас що передбачено: normal, дійсний клас: rootkit
клас що передбачено: normal, дійсний клас: warezclient
клас що передбачено: normal, дійсний клас: satan
клас що передбачено: normal, дійсний клас: satan
клас що передбачено: normal, дійсний клас: portsweep
клас що передбачено: normal, дійсний клас: buffer_overflow
клас що передбачено: normal, дійсний клас: warezclient
клас що передбачено: normal, дійсний клас: buffer_overflow
клас що передбачено: normal, дійсний клас: buffer_overflow
клас що передбачено: normal, дійсний клас: satan
клас що передбачено: normal, дійсний клас: portsweep
клас що передбачено: normal, дійсний клас: warezmaster
клас що передбачено: normal, дійсний клас: warezclient
клас що передбачено: normal, дійсний клас: portsweep
клас що передбачено: normal, дійсний клас: loadmodule
клас що передбачено: normal, дійсний клас: nmap
клас що передбачено: normal, дійсний клас: warezclient
клас що передбачено: normal, дійсний клас: smurf
клас що передбачено: normal, дійсний клас: land
клас що передбачено: normal, дійсний клас: neptune
клас що передбачено: normal, дійсний клас: back
клас що передбачено: normal, дійсний клас: back
клас що передбачено: normal, дійсний клас: ipsweep
клас що передбачено: normal, дійсний клас: smurf
клас що передбачено: normal, дійсний клас: warezclient
клас що передбачено: normal, дійсний клас: rootkit
клас що передбачено: normal, дійсний клас: back
клас що передбачено: normal, дійсний клас: warezclient
клас що передбачено: normal, дійсний клас: warezclient
клас що передбачено: normal, дійсний клас: warezclient
клас що передбачено: normal, дійсний клас: multihop
клас що передбачено: normal, дійсний клас: warezclient
клас що передбачено: normal, дійсний клас: warezclient
клас що передбачено: normal, дійсний клас: multihop
клас що передбачено: normal, дійсний клас: guess_passwd
клас що передбачено: normal, дійсний клас: nmap
клас що передбачено: normal, дійсний клас: ipsweep
клас що передбачено: normal, дійсний клас: warezclient
клас що передбачено: normal, дійсний клас: warezclient
клас що передбачено: normal, дійсний клас: warezclient
клас що передбачено: normal, дійсний клас: satan
клас що передбачено: normal, дійсний клас: warezclient
```

```
клас що передбачено: normal, дійсний клас: warezclient
клас що передбачено: normal, дійсний клас: warezmaster
клас що передбачено: normal, дійсний клас: multihop
клас що передбачено: normal, дійсний клас: perl
клас що передбачено: normal, дійсний клас: satan
клас що передбачено: normal, дійсний клас: nmap
клас що передбачено: normal, дійсний клас: warezclient
клас що передбачено: normal, дійсний клас: spy
клас що передбачено: normal, дійсний клас: satan
клас що передбачено: normal, дійсний клас: portsweep
клас що передбачено: normal, дійсний клас: nmap
клас що передбачено: normal, дійсний клас: ipsweep
клас що передбачено: normal, дійсний клас: ipsweep
клас що передбачено: normal, дійсний клас: satan
клас що передбачено: normal, дійсний клас: loadmodule
клас що передбачено: normal, дійсний клас: nmap
клас що передбачено: normal, дійсний клас: back
клас що передбачено: normal, дійсний клас: buffer_overflow
клас що передбачено: normal, дійсний клас: rootkit
клас що передбачено: normal, дійсний клас: satan
клас що передбачено: normal, дійсний клас: warezclient
клас що передбачено: normal, дійсний клас: satan
клас що передбачено: normal, дійсний клас: warezclient
клас що передбачено: normal, дійсний клас: portsweep
клас що передбачено: normal, дійсний клас: perl
клас що передбачено: normal, дійсний клас: warezclient
клас що передбачено: normal, дійсний клас: warezclient
клас що передбачено: normal, дійсний клас: warezclient
клас що передбачено: normal, дійсний клас: warezmaster
клас що передбачено: normal, дійсний клас: warezclient
клас що передбачено: normal, дійсний клас: warezclient
клас що передбачено: normal, дійсний клас: warezclient
клас що передбачено: normal, дійсний клас: nmap
клас що передбачено: normal, дійсний клас: warezclient
клас що передбачено: normal, дійсний клас: warezclient
клас що передбачено: normal, дійсний клас: nmap
Кількість записів які передбачено що є нормальними а насправді є
вторгненнями: 78
```

Як бачимо система пропустила деякі суттєві загрози як от buffer_overflow, чи warezclient. Необхідна оптимізація параметрів моделі. Для більш детального аналізу надрукуємо матрицю помилок (Confusion Matrix).

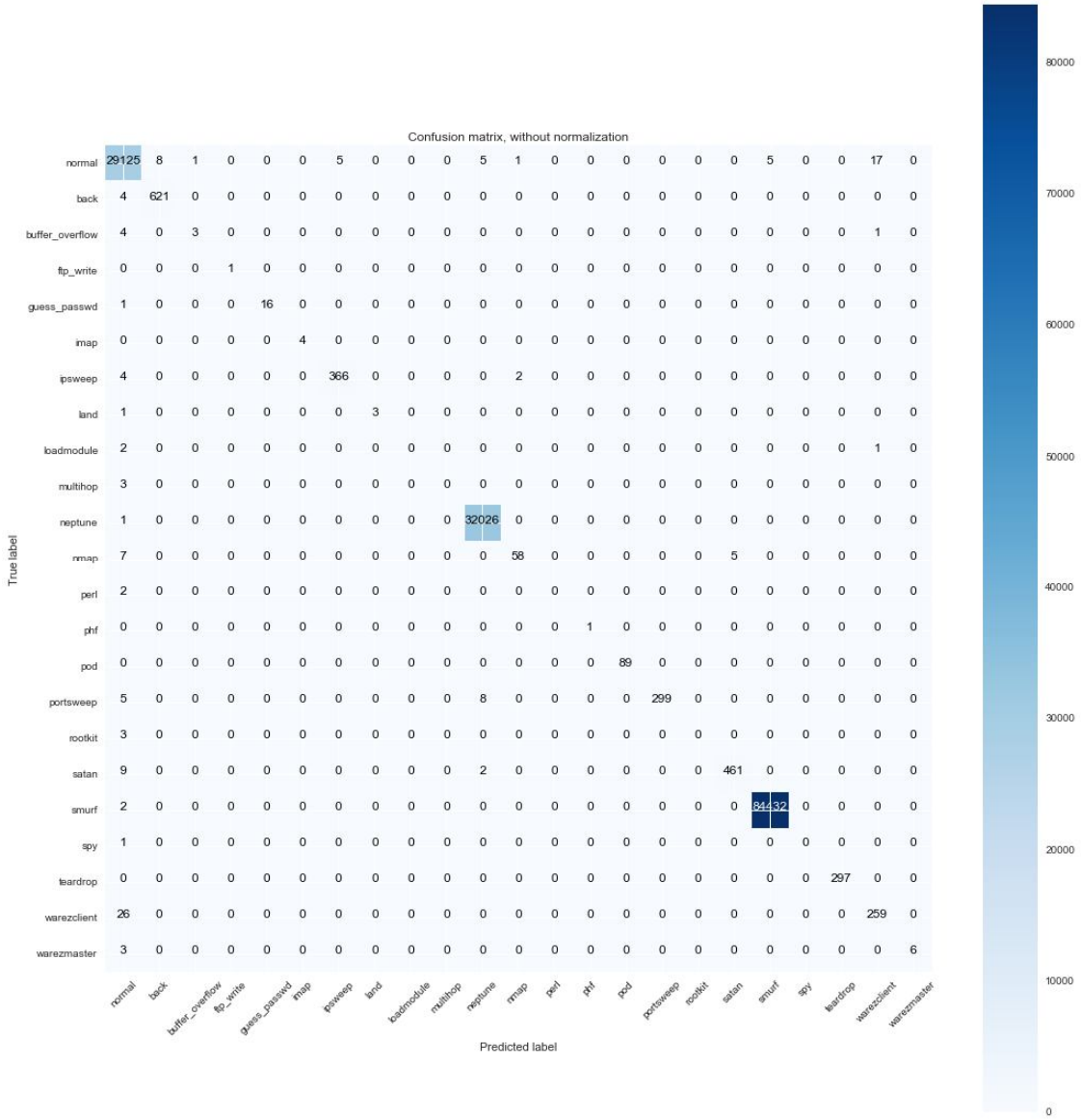
```
def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)
```

```
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')

    #print(cm)

    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, cm[i, j],
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
```

Confusion matrix:

```
cnf_matrix = confusion_matrix(y_test, y_predicted,labels=classes)
np.set_printoptions(precision=3)

# Plot non-normalized confusion matrix
plt.figure(figsize=(15,15))
plot_confusion_matrix(cnf_matrix, classes=classes,
                      title='Confusion matrix, without normalization')

# Plot normalized confusion matrix
#plt.figure()
#plot_confusion_matrix(cnf_matrix, classes=classes, normalize=True,
#                      title='Normalized confusion matrix')

plt.show()
```

```
Confusion matrix, without normalization
```



Бачимо що найбільше помилок система припустила у класифікації атак buffer_overflow, warezclient, nmap

Визначення міри якості F-міри для кожного з класів:

```
f1_score(y_test,y_predicted,labels=classes, average=None)
```

```
array([ 0.998,  0.99 ,  0.5  ,  1.   ,  0.97 ,  1.   ,  0.985,  0.857,
        0.   ,  0.   ,  1.   ,  0.885,  0.   ,  1.   ,  1.   ,  0.979,
        0.   ,  0.983,  1.   ,  0.   ,  1.   ,  0.92 ,  0.8  ])
```

Типовий звіт з результатів класифікації створений з використанням бібліотечної функції пакету Scikit Learn:

```
from sklearn.metrics import classification_report
print(classification_report(y_test,y_predicted))
```

```
                precision    recall  f1-score   support

          back       0.99      0.99      0.99       625
buffer_overflow       0.75      0.38      0.50         8
      ftp_write       1.00      1.00      1.00         1
   guess_passwd       1.00      0.94      0.97        17
          imap       1.00      1.00      1.00         4
        ipsweep       0.99      0.98      0.99       372
           land       1.00      0.75      0.86         4
     loadmodule       0.00      0.00      0.00         3
       multihop       0.00      0.00      0.00         3
        neptune       1.00      1.00      1.00     32027
           nmap       0.95      0.83      0.89        70
         normal       1.00      1.00      1.00     29167
           perl       0.00      0.00      0.00         2
            phf       1.00      1.00      1.00         1
            pod       1.00      1.00      1.00        89
      portsweep       1.00      0.96      0.98       312
        rootkit       0.00      0.00      0.00         3
          satan       0.99      0.98      0.98       472
          smurf       1.00      1.00      1.00     84434
            spy       0.00      0.00      0.00         1
       teardrop       1.00      1.00      1.00       297
    warezclient       0.93      0.91      0.92       285
    warezmaster       1.00      0.67      0.80         9

    avg / total       1.00      1.00      1.00    148206
```

Задамо параметр моделі, який врахує нерівномірність даних з вторгнень

(у вибірці точок даних класу normal більше за даних які згенеровані внаслідок дій зловмисників)

```
'''навчаємо класифікатор з параметором class_weight,
   З документації scikit-learn:
   Set the parameter C of class i to class_weight[i]*C for SVC.
   If not given, all classes are supposed to have weight one.
   The “balanced” mode uses the values of y to automatically adjust weights inversely
proportional to
   class frequencies in the input data as n_samples / (n_classes * np.bincount(y))
'''

clf2=SVC(C=10,kernel='rbf',random_state=241, class_weight = {'normal':1}, probability = True)
clf2.fit(X_train,y_train)
```

```
#параметер probability = True вказує що модель повинна давати оцінки приналежності класу у
виді ймовірності
#що дана точка належить до певного класу

# Передбачення нової моделі:
y_predicted_balanced=clf2.predict(X_test)


accuracy=accuracy_score(y_test,y_predicted_balanced)

print("Accuracy %.4f" % accuracy)
```

```
Accuracy 0.9993
```

Як бачимо загальна оцінка якості моделі зросла

```
suma_compromized=0
for i in range(len(y_predicted)):
    if y_test.iloc[i]!=y_predicted_balanced[i]:
        if y_predicted_balanced[i] =='normal':
            suma_compromized+=1
            #print("клас що передбачено: %s, дійсний клас: %s" % (y_predicted[i],
y_test.iloc[i]))

print('Кількість записів які передбачено що є нормальними а насправді є вторгненнями: %s' %
suma_compromized)
```

```
Кількість записів які передбачено що є нормальними а насправді є
вторгненнями: 59
```

Як бачимо кількість пропусків зменшилася

```
print("Загальне число помилок класифікації: %s" % len(y_test[y_test!=y_predicted_balanced]))
```

```
Загальне число помилок класифікації: 97
```

Визначимо показники точності, повноти та F-міри оптимізованого класифікатора:

```
y_pred_bal_proba=clf2.predict(X_test)
y_pred_bal_log_proba=clf2.predict_log_proba(X_test)

accuracy=accuracy_score(y_test,y_pred_bal_proba)

precision=precision_score(y_test,y_pred_bal_proba, average='micro')

recall=recall_score(y_test,y_pred_bal_proba, average='micro')

F=f1_score(y_test,y_pred_bal_proba, average='micro')

print("%.4f %.4f %.4f %.4f" % (accuracy, precision, recall, F))
```

```
0.9993 0.9993 0.9993 0.9993
```

Для подальшої оптимізації параметрів моделі бінаризуємо вектор Y результатів:

```
# Бінаризуємо вектор міток класів з допомогою бібліотечної функції пакету scikit learn
from sklearn.preprocessing import label_binarize
y_b = label_binarize(y, classes)
n_classes = y_b.shape[1]
```

```
n_classes
```

```
23
```

```
y_b
```

```
array([[1, 0, 0, ..., 0, 0, 0],
       [1, 0, 0, ..., 0, 0, 0],
       [1, 0, 0, ..., 0, 0, 0],
       ...,
       [1, 0, 0, ..., 0, 0, 0],
       [1, 0, 0, ..., 0, 0, 0],
       [1, 0, 0, ..., 0, 0, 0]])
```

Як бачимо кожній точці даних тепер ставиться у відповідь не мітка класу а бінарний вектор приналежності даної точки до певного класу

Проведемо навчання нового мультикласового класифікатора

Навчання проводитимо за схемою один проти всіх

```
#Розбиваємо вибірку на навчаючу та тестову

X_train, X_test, y_train, y_test = train_test_split(X_transformed, y_b, test_size=.3,
                                                    random_state=101)



X_train=scaler.fit_transform(X_train)
X_test=scaler.transform(X_test)

from sklearn.multiclass import OneVsRestClassifier

# Задіємо 4 ядра процесора (параметер n_jobs = 4)
classifier = OneVsRestClassifier(SVC(kernel='linear', probability=True,
                                     random_state=0), n_jobs = 4)

#Навчання проведемо з заміром часу виконання
startTime = time.time()
y_score = classifier.fit(X_train, y_train).decision_function(X_test)
print_time((time.time() - startTime))
```

```
Час виконання: 1 г: 29 хв : 39 с
```

Обчислимо та побудуємо графік ROC(Receiver Operating Characteristic):

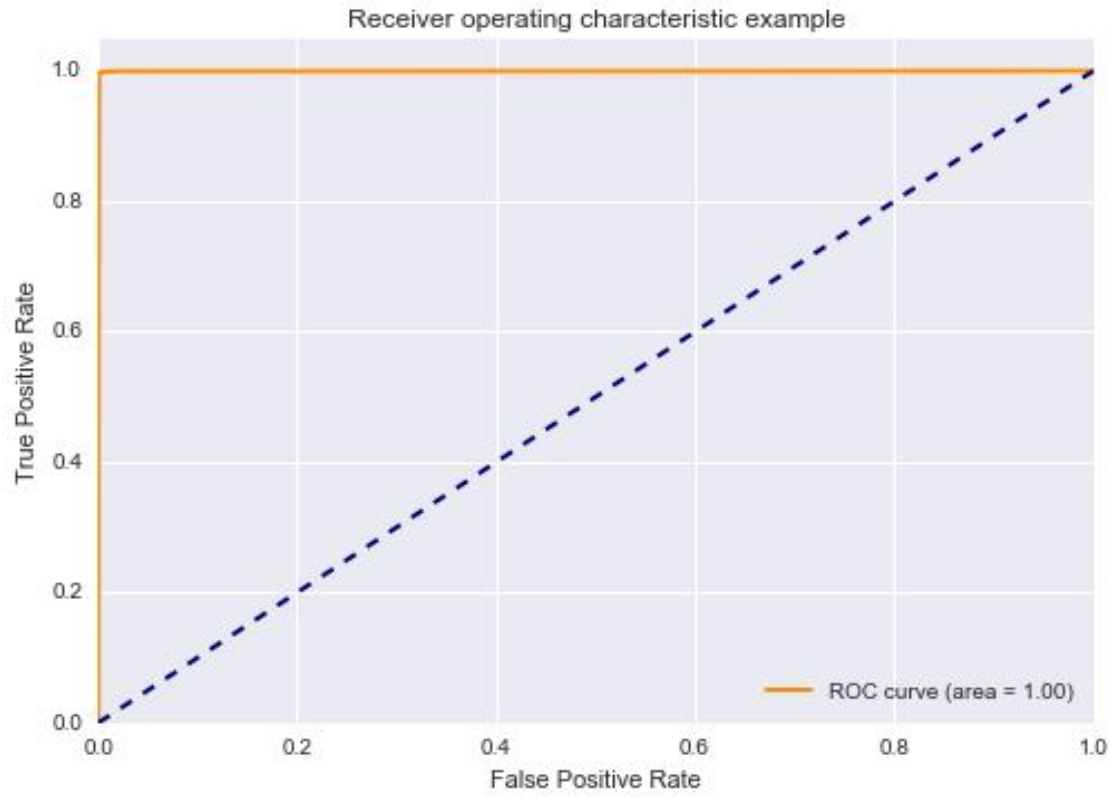
```
# Обчислимо ROC-криву та площу під ROC-кривою ляо кожного класу
fpr = dict()
tpr = dict()
roc_auc = dict()
for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve(y_test[:, i], y_score[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

# Обчислимо усереднену ROC криву і ROC площу використовуючи micro-average усереднення
# http://scikit-learn.org/stable/modules/model_evaluation.html
fpr["micro"], tpr["micro"], _ = roc_curve(y_test.ravel(), y_score.ravel())
roc_auc["micro"] = auc(fpr["micro"], tpr["micro"])

roc_auc
```

```
{0: 0.99955579521502624,
 1: 0.99997161991556238,
```

```
 2: 0.99973346828609988,
 3: 0.99578621359747366,
 4: 0.99907371792708943,
 5: 1.0,
 6: 0.99997911715349508,
 7: 0.99999662613530549,
 8: 0.9961657062657725,
 9: nan,
 10: 0.99999999865225775,
 11: 0.98511348806002508,
 12: nan,
 13: nan,
 14: 0.99998061997802279,
 15: 0.99470946874386612,
 16: 0.6664979791232295,
 17: 0.9936521320458036,
 18: 0.99998432529653103,
 19: nan,
 20: 1.0,
 21: 0.99989559318962007,
 22: 0.99992240111202579,
 'micro': 0.99982459696861115}
```

Побудуємо графік ROC для класу Normal:

```
#Plot of a ROC curve for a specific class
plt.figure()
lw = 2
plt.plot(fpr[0], tpr[0], color='darkorange',
         lw=lw, label='ROC curve (area = %0.2f)' % roc_auc[0])
plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic example')
plt.legend(loc="lower right")
plt.show()
```



Спрощення моделі методом Аналізу Головних Компонент (PCA)

```
Спробуємо спростити модель шляхом зменшення кількості ознак які використовуються при побудові
класифікатора, та порівняти якість прийняття рішення таким класифікатором.

Визначимо наявну кількість ознак у навчаючому датасеті:

print('Наявна кількість ознак: %s' % X_transformed.shape[1])

Наявна кількість ознак: 118

Зменшимо кількість ознак використовуючи Метод Головних Компонент (PCA):

from sklearn.decomposition import PCA
coder = PCA(n_components=50)
train = coder.fit_transform(X_transformed)

print('Кількість ознак після пониження розмірності: %s' % train.shape[1])

Кількість ознак після пониження розмірності: 50

#Розбиваємо вибірку на навчаючу та тестову

X_train, X_test, y_train, y_test = train_test_split(train, y_b, test_size=.5,
                                                    random_state=0)

# Нормалізуємо числові поля вибірки використовуючи відповідну фукцію
бібліотеки scikit-learn
scaler=StandardScaler()
X_train=scaler.fit_transform(X_train)
X_test=scaler.transform(X_test)
```

На основі отриманого датасету проведемо навчання класифікатора:

```
classifier = OneVsRestClassifier(SVC(kernel='linear', probability=True,
                                     random_state=0))

#Навчання проведемо з заміром часу виконання
startTime = time.time()
y_score = classifier.fit(X_train, y_train).decision_function(X_test)
print_time((time.time() - startTime))
```

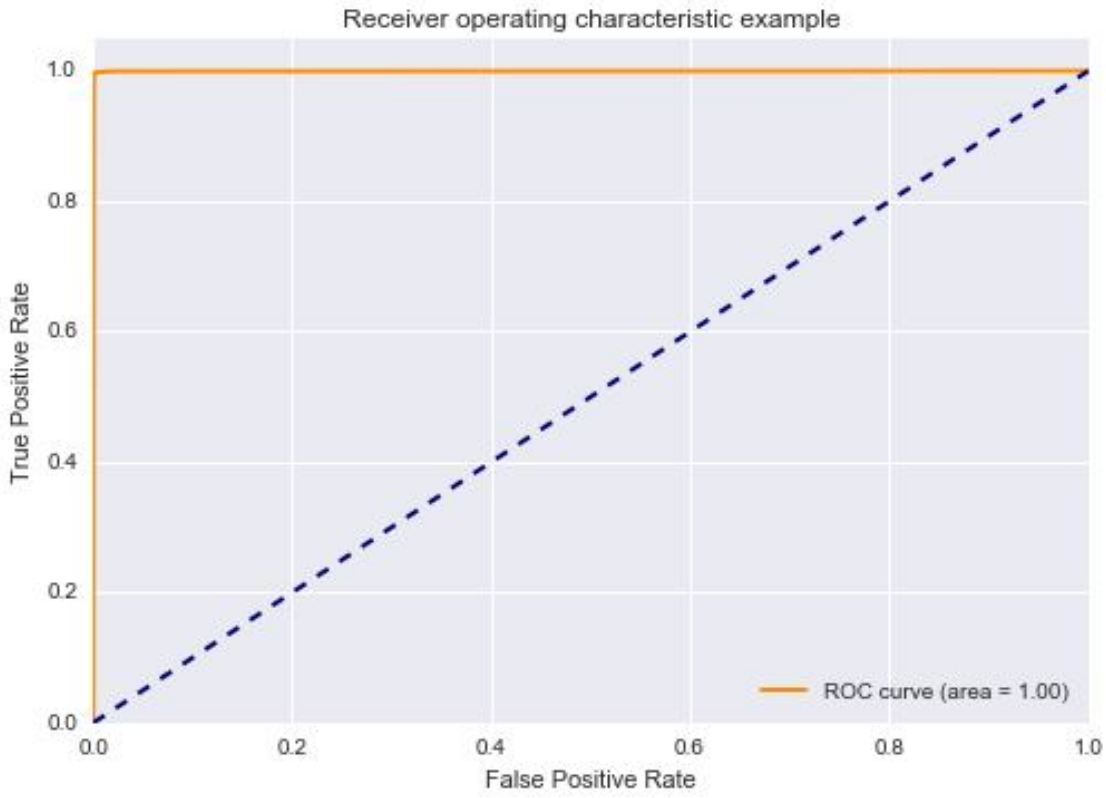
```
Час виконання: 1 г: 10 хв : 25 с
```

```
# Compute ROC curve and ROC area for each class
fpr = dict()
tpr = dict()
roc_auc = dict()
for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve(y_test[:, i], y_score[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

# Compute micro-average ROC curve and ROC area
fpr["micro"], tpr["micro"], _ = roc_curve(y_test.ravel(), y_score.ravel())
roc_auc["micro"] = auc(fpr["micro"], tpr["micro"])

roc_auc
```

```
{0: 0.99952547017500026,
 1: 0.99994787171429711,
 2: 0.93744458367409711,
 3: 0.99896358792903805,
 4: 0.99999436049127932,
 5: 0.99999190303069552,
 6: 0.99996926810754894,
 7: 0.99963967611336013,
 8: 0.99935224245564869,
 9: 0.98061172850751999,
 10: 0.99998244985389695,
 11: 0.99984399202453023,
 12: 0.99972875372457559,
 13: 0.99997975765868985,
 14: 0.99998330868851137,
 15: 0.99638299875884118,
 16: 0.79681382967956116,
 17: 0.99521013771051903,
 18: 0.99999999699328923,
 19: 0.99991093441939372,
 20: 0.99999998367572951,
 21: 0.99989264006990886,
 22: 0.99971877578083312,
 'micro': 0.99986082662890874}
```

```
#Plot of a ROC curve for a specific class
plt.figure()
lw = 2
plt.plot(fpr[0], tpr[0], color='darkorange',
         lw=lw, label='ROC curve (area = %0.2f)' % roc_auc[0])
plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
```

```
plt.title('Receiver operating characteristic example')
plt.legend(loc="lower right")
plt.show()
```



Пошук аномалій

```
Замінюємо дані вибірки - нормальні випадки ('normal') позначаємо -1, решта випадків
(вторгнення) позначаємо 1.

y_bin=y.map(lambda x: -1 if x == 'normal' else 1)
```

Розбиваємо вибірку на навчаючу та тестову

```
X_train, X_test, y_train, y_test = train_test_split(X_transformed, y_bin.values,
test_size=0.3, random_state=101)
```

Начання проводитимо тільки на "нормальних" даних, тому залишаємо в навчаючій вибірці тільки нормальні дані. Сформуємо також тестову вибірку у якій тільки викиди (вторгнення)

```
X_train_normal=X_train[y_train==-1]
X_outliers=X_test[y_test == 1]

len(X_train)
```

345813

```
len(X_train_normal)
```

67974

```
len(y_test)
```

148206

```
len(X_outliers)
```

118904

```
clf=OneClassSVM(nu=0.2, kernel="rbf", gamma=0.1)

startTime = time.time()

clf.fit(X_train_normal)

print_time((time.time() - startTime))
```

Час виконання: 1 г: 26 хв : 02 с

```
# передбачення на тестовій вибірці
y_pred_test = clf.predict(X_test)

# передбачення на  вибірці - "тільки вторгнення"
y_pred_outliers = clf.predict(X_outliers)
```

```
#оскільки метод predict повертає значення 1 якщо нормальний елемет та -1 якщо викид,
#необхідне переформатування (у вибірці 1 позначено за викид, -1 за нормальне значення)
y_pred_test=np.where(y_pred_test ==-1, 1,-1)
y_pred_outliers=np.where(y_pred_outliers ==-1, 1,-1)

n_error_test = y_pred_test[y_pred_test!=y_test].size
n_error_outliers = y_pred_outliers[y_pred_outliers == -1].size

print("Помилок на тестовій вибірці: %d/148206 ; "
    "помилок на вибірці - тільки вторгнення: %d/148206"
    % (n_error_test, n_error_outliers))
```

Помилок на тестовій вибірці: 24680/148206 ; помилок на вибірці - тільки вторгнення: 0/148206

```
FP=0
FN=0
TP=0

for i in range(len(y_pred_test)):
    if y_test[i] == -1 and y_pred_test[i] == 1:
        FP+=1
    if y_test[i]== 1 and y_pred_test[i] == - 1:
        FN+=1
    if y_test[i]== -1 and y_pred_test[i] == -1:
        TP+=1


print("Передбачено: normal, насправді: вторгнення: %s" % (FN))
print("Передбачено: вторгнення, насправді: normal: %s" % (FP))
print ('Total error: %s' % (FP+FN))
```

Передбачено: normal, насправді: вторгнення: 0
Передбачено: вторгнення, насправді: normal: 24680
Total error: 24680

```
precision, recall, f1, support = precision_recall_fscore_support(y_test,
y_pred_test,labels=[1],pos_label=1)

precision , recall, f1
```

```
(array([ 0.828]), array([ 1.]), array([ 0.906]))
```

Навчання моделі з підбором параметра використовуючи паралельні обчислення

Для більш точного моделювання методом OneClassSVM, необхідно підібрати параметер nu, який забезпечить оптимальні співвідношення точності (precision) та повноти (recall) на наявних даних. Даний параметер приймає значення в інтервалі (0,1] і в документації бібліотеки scikit-learn описаний так: nu - "An upper bound on the fraction of training errors and a lower bound of the fraction of support vectors" Для підбору параметра проведемо навчання 10-ти класифікаторів з параметром nu зі списку [0.1,0.2,0.3...1]

```
nu_s = np.arange(0.1,1.1,0.1)
```

Оскільки навчання одного класифікатора займає суттєвий обсяг процесорного часу, для навчання класифікаторів використаємо паралельні обчислення. Потужна та гнучка архітектура IPython дозволяє розпаралелювати обчислювальні задачі як між кількома процесорними ядрами так і між кількома процесорами та навіть між різними вузлами кластера.

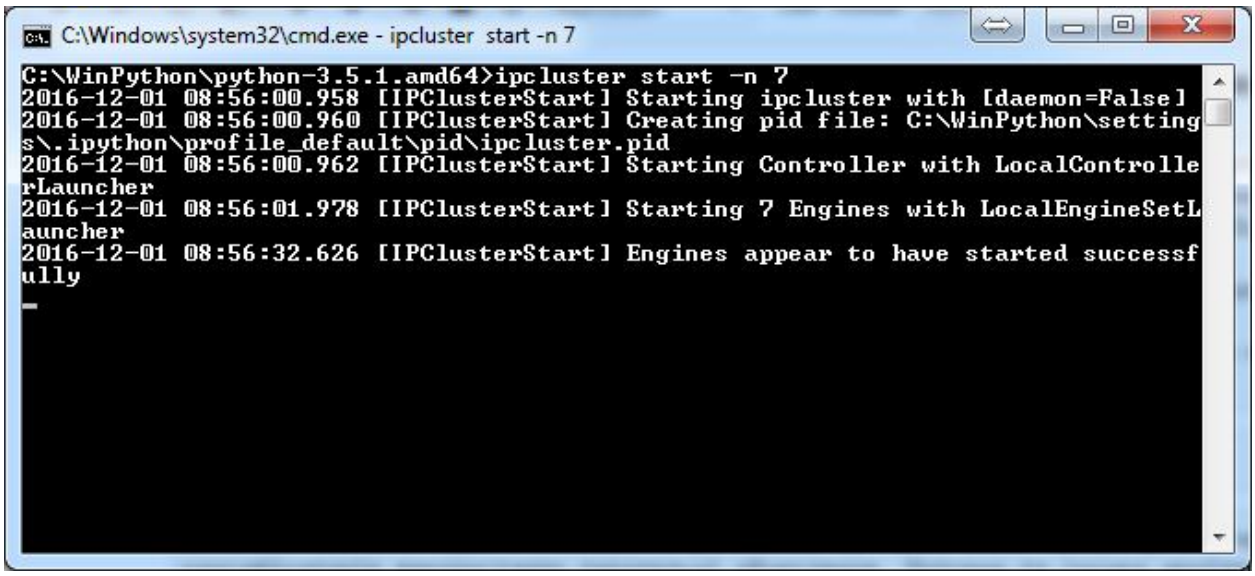
Код для розпаралелювання обислювальних потоків пишеться з використанням бібліотеки IPyparalell: https://ipyparallel.readthedocs.io

Перш ніж імпортувати та створити екземпляр класу Client який відповідає за доступ до обчислювальних вузлів, необхідно з консолі запустити контролер та один або декілька інстансів обчислювальних вузлів. Кількість обчислювальних вузлів виберемо з розрахунку кількість ядер комп'ютера - 1.

```
# Визначаємо кількість обчислювальних ядер які встановлено на даній робочій станції:
import multiprocessing

print("CPU count: %s" %  multiprocessing.cpu_count())

CPU count: 8
```

Для старту контролера та обчислювальних вузлів в консолі запускаємо команду:

$ ipcluster start -n 4



Після старту контролера та вузлів можемо імпортувати клас та створити екземпляр клієнта

```
rc = Client()
#dv - колекція DirectView для доступу до окремих обчислювальних вузлів
dv=rc[:]
dv
```

```
<DirectView [0, 1, 2, 3]>
```

Переносимо дані вибірки в обчислювальні вузли. Створюємо в кожному вузлі об'єкт класифікатора

```
dv['X_n'] = X_train_normal
dv['X_test'] = X_test
dv['y_test'] = y_test

# імпортуємо необхідні функції у всі обислювальні ядра
with dv.sync_imports():
    from sklearn.metrics import precision_recall_fscore_support
    import os
    import numpy
    from sklearn.svm import OneClassSVM
    import timeit

importing precision_recall_fscore_support from sklearn.metrics on engine(s)
importing os on engine(s)
importing numpy on engine(s)
importing OneClassSVM from sklearn.svm on engine(s)
importing timeit on engine(s)

#описуємо функцію - навчання одного класифікатора, та обчислення метрик якості
#важливо відмітити, що у сформатованій нами вибірці позитивний клас (вторгнення) позначено міткою -1
@require(OneClassSVM)
def f_clf(nu):
    clf = OneClassSVM(kernel="rbf")
    clf.nu=nu
    clf.fit(X_n)

    y_pred_test=clf.predict(X_test)

    #оскільки метод presict повертає значення 1 якщо нормальний елемет та -1 якщо викид,
    #необхідне переформатування (у вибірці 1 позначено за викид, -1 за нормальне значення)
    y_pred_test=numpy.where(y_pred_test ==-1, 1,-1)
```

```
    precision, recall, f1, support = precision_recall_fscore_support(y_test,
y_pred_test,labels=[1],pos_label=1)

    #визначаємо також ідентифікатор процесу який проводить дане обчислення
    pid = os.getpid()
    return (nu, precision, recall,f1,support,pid)
```

Проводимо обчислення за заміром часу виконання

```
startTime = time.time()
# Виконуємо "мапінг" функції з коефіцієнтами зі списку по різним обчислювальним ядрам
scores = dv.map_sync(f_clf,nu_s)

print_time((time.time() - startTime))
```

```
Час виконання: 5 г: 14 хв : 38 с
```

```
#переформатуємо результати в зручніший вид:
nu_s1, precisions,recals,f1s,pids = list(),list(),list(),list(),list()
for x in scores:
    nu_s1.append(x[0])
    precisions.append(x[1])
    recals.append(x[2])
    f1s.append(x[3])
    pids.append(x[5])

for n, p in zip(nu_s1, pids):
    print('nu: %.1f, PID: %i' %(n, p))
```

```
nu: 0.1, PID: 5724
nu: 0.2, PID: 5724
nu: 0.3, PID: 5724
nu: 0.4, PID: 9452
nu: 0.5, PID: 9452
nu: 0.6, PID: 9452
nu: 0.7, PID: 10452
nu: 0.8, PID: 10452
nu: 0.9, PID: 9676
nu: 1.0, PID: 9676
```

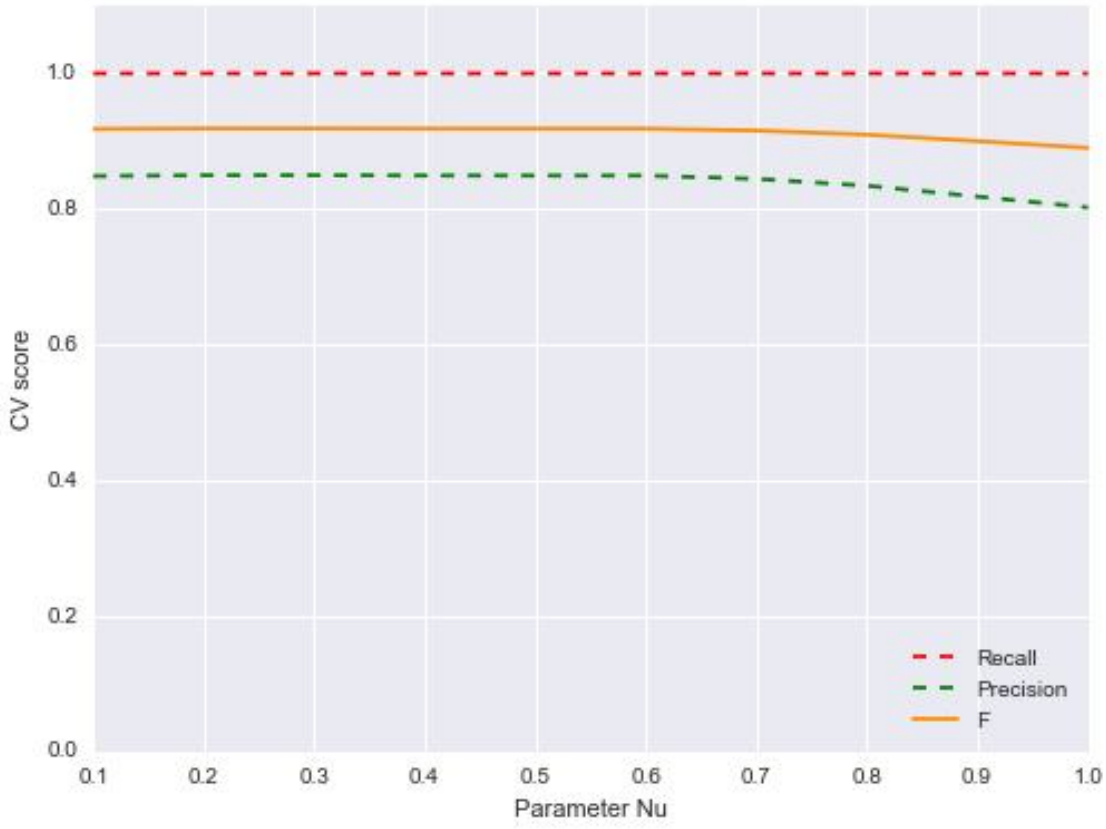
```
Як бачимо обчислення велося різними обчислювальними ядрами
```

Побудуємо графіки залежності оцінок моделі від параметра nu

Оскільки для систем вявлення вторгнень найбільш важливою ознакою є здатність не пропустити дані дані - основним критерієм якості буде recall (повнота) - як багато об''єктів знаходить алгоритм що є класом 1 (в нашому випадку це "вторгнення"). Чим вища повнота тим менше хибних пропусків.

```
lw=2
plt.figure(1, figsize=(8, 6))
plt.clf()
plt.plot(nu_s, recals, 'r--', label='Recall')
plt.plot(nu_s, precisions, 'g--', label="Precision")
plt.plot(nu_s, f1s, color='darkorange', label="F")
plt.xlim([0.1, 1.0])
plt.ylim([0.0, 1.05])
plt.ylabel('CV score')
plt.xlabel('Parameter Nu')
plt.ylim(0, 1.1)
```

```
plt.legend(loc="lower right")
plt.show()
```



2.6 Висновки до розділу

Подано аналіз традиційних підходів до вирішення завдань виявлення вторгнень в комп'ютерну систему та їх недоліки, зокрема, їх нездатність розпізнавати нові типи атак. Розглянуто методику виявлення вторгнень з використанням методів машинного навчання. Запропоновано архітектру системи.

Проведено детальне дослідження моделей – машини опорних векторів та однокласової машини опорних векторів на наборі даних вторгнень.

Порівняльні результати коректно розпізнаних атак TP (True positive, %) та хибно відкинутих позитивних точок даних (атак) FN (False negative, %) на основі запропонованої моделі та моделей інших систем, подано в таблиці:

Модель	**TP**	**FN**
Запропонована модель	0.89	1.12×10^{-5}
Геометричний підхід [1]	0.98	0.1
Кластеризація [2]	0.667	0.021
Штучна нейронна мережа [3]	0.967	1.33×10^{-5}

Згідно наведених вище даних видно, що запропонована модель показує кращі результати в під-задачі «не пропуску» загрозливого трафіку, проте доля хибних спрацювань досить висока. Подальша оптимізація моделі повинна проводитись саме в напрямку зменшення хибних спрацювань.

1. Eskin E., Arnold A., Portnoy L. A Geometric Framework for Unsupervised Anomaly Detection: Detecting Intrusions in Unlabeled Data. New York: Columbia University, 2001. 14 p.
2. Eskin E., Portnoy L. Intrusion Detection withUnlabeledDataUsingClustering.NewYork: Columbia University, 1999. 13 p.
3. Большов А.К., Яновский В.В. Применение нейронных сетей для обнаружения вторжений в компьютерные сети // Вестник Санкт-Петербургского университета. - Санкт-Петербург, 2010. -С.129-135.

РОЗДІЛ 3. Застосування алгоритмів виявлення аномалій в задачі біометричної автентифікації

3.1 Біометрика клавіатурного введення - загальний огляд

Біометричні технології автентифікації зазвичай поділяють на фізіологічні (відбиток пальців, обличчя, сітківки ока) та поведінкові (підпис, клавіатурна динаміка). Разом з поширенням інтернет-технологій у різні галузі діяльності, та необхідності віддаленої ідентифікації користувачів спостерігається значний ріст інтересу до поведінкової біометрики серед дослідників та IT індустрій в цілому. Це є наслідком простоти використання кінцевим користувачем, прозорості та великої кількості потенційних способів застосування.

Біометрика клавіатурного введення (keystroke biometrics, keystroke dynamics) – метод автоматичної автентифікації/ідентифікації що базується на розпізнаванні патернів (ритму) клавіатурного набору. На відміну від інших біометричних систем, що можуть базуватись на дорогому обладнанні для аналізу, аналіз клавіатурного введення не потребує значних вкладень.

Перші дослідження з використання набутих патернів в клавіатурному введенні бали проведені Гейнсом ще у 80 роках минулого століття [23]. Експеримент проводився з невеликою вибіркою даних отриманих внаслідок клавіатурного введення семи секретарів. Тестування на статистичну незалежність їх профілів було проведено використовуючи *T-Test* з гіпотезою що середні значення часу дограф в різних сесіях одинакові але з різною дисперсією. Подібні дослідження проводились Леджетом [14] за участю 17-ти програмістів, проте з іншим підходом до автентифікації. Дослідником використовувався так званий метод безперервної автентифікації, коли аналіз клавіатурного введення проводився на протязі всієї робочої сесії.

Протягом останнього десятиріччя запропоновано велику кількість алгоритмів систем автентифікації на основі аналізу клавіатурного введення. Проте кожен з дослідників використовує власні методи оцінки якості запропонованого ними підходу і як наслідок порівняння різних методологій поміж собою утруднене. Так, наприклад, використовують різну довжину паролей, кількість повторень введення на етапі навчання та тестування, різні методики фільтрації даних що значно відхиляються від середньо статистичних на етапі збору, різні методики подання кількості хибних та позитивних спроб.

В дослідженні [15] автори розробили уніфіковану методику тестування якості роботи алгоритмів автентифікації на базі аналізу динаміки клавіатурного введення. В якості парольної фрази було вибрано стрічку яка відповідає вимогам надійності паролів, та згенерована доступним публічно генератором паролів: **.tie5Roanl**.

Для збору даних був використаний портативний комп'ютер з підключеною зовнішньою клавіатурою. В дослідженні прийняли участь 50 людей різного віку. Кожен з них повторив введення паролю 400 разів, під час розбитих на кілька днів сесій. 200 пакетів даних використано для навчання системи, 200 для тестування. Тестування проводилось на основі даних 200 введень оригінального користувача та 5 перших введень кожного з 50-ти піддослідних.

Фіксувались наступні ознаки:

- Press-Press (Keydown - Keydown) – час між послідовними натискуваннями;
- Release-Press (Keyup - Keydown) – час між відпусканням клавіші та натискуванням наступної
- Hold – час утримування клавіші.

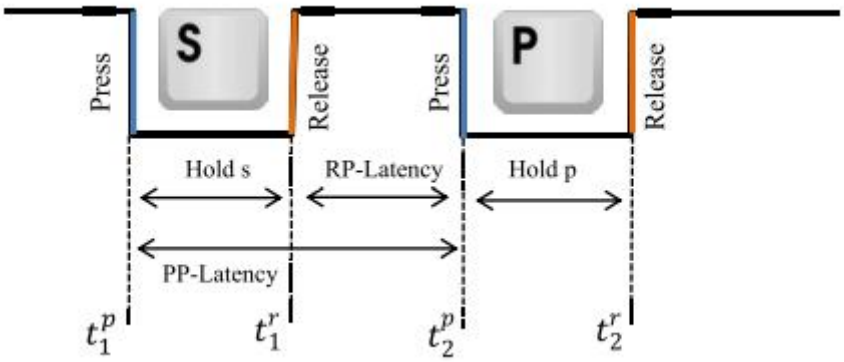


Рис. 3.1 Найбільш популярні ознаки що застосовуються в моделях динаміки клавіатурного введення

3.2 Огляд популярних моделей

Euclidean

Класичний алгоритм виявлення аномалій. Кожний клавіатурний набор пароля моделюється точкою в p-вимірному просторі, де p – кількість ознак в часовому векторі. Навчаючі дані розглядаються як хмара точок, і обчислюється ступінь аномальності тестового вектора базуючись на віддаленості цього вектора від центру цієї хмари. Формально кажучи під час навчаючої фази, обраховується середній вектор часових векторів вибірки. В тестовій фазі ступінь аномальності обраховується як квадрат евклідової відстані між тестовим вектором та середнім вектором.

Euclidean (normed) - нормалізований бінарний класифікатор на базі метрики мінімальної відстані

В навчаючій фазі середній вектор розраховується подібно як в стандартному евклідовому детекторі. В тестовій фазі обчислюється квадрат евклідової відстані між тестовим та середнім вектором, проте ступінь аномальності розраховують «нормалізуючи» цю відстань шляхом ділення на добуток норм цих двох векторів:

Якщо $\boldsymbol{x}$ – середній вектор, $\boldsymbol{y}$ - тестовий вектор, то ступінь аномальності:

$$score = \frac{d}{\|x\|\|y\|}$$

Manhattan

Модель подібна до стандартного евклідового детектора, проте замість евклідової метрики відстані використана *манхетеновська відстань* (відстань міських кварталів).

Manhattan (filtered)

В навчаючій фазі обчислюється середній вектор і стандартне відхилення кожної ознаки. Будь який часовий вектор який різниться на більш ніж три стандратні відхилення від середнього видаляється з вибірки і обчислюється новий середній вектор вже не враховуючи цих екстремальних ознак. В тестовій фазі манхетеновська відстань від цього нового вектора є мірою аномальності.

Manhattan (scaled)

Міра аномальності обчислюється подібно до манхетеновської проте кожен вимір масштабується на середню величину абсолютного відхилення кожної ознаки від середнього значення:

$$\sum_{i=1}^{p} \frac{|x_i - y_i|}{a_i},$$

тут a_i – середнє абсолютне відхилення ознаки

Mahalanobis

Міра відстані у цій моделі більш складна за евклідову. Враховується кореляція поміж ознаками. Під час навчаючої фази обчислюються середній вектор та коваріаційна матриця часових векторів. В тестовій фазі міру аномальності обчислюють як відстань Махаланобіса між середнім та тестовими векторами:

$$(x - y)^T S^{-1} (x - Y)$$

Найближчих сусідів (Nearest-neighbor) (Mahalanobis)

У цій моделі міру аномальності обчислюють як відстань Махаланобіса між тестовим вектором та найближчим вектором з навчаючої вибірки.

Нейронна мережа (стандартна)

Цей детектор описаний Хайдером [25] і побудований на базі штучної нейронної мережі прямого поширення з навчанням за допомогою алгоритму бекпропагації. Мережа містить p вхідних нейронів (за кількістю ознак – довжиною часового вектора), один вихідний нейрон та [2p/3] прихованих нейрони. Детектор «натренований» видавати 1 на кожному навчаючому прикладі. (мережа навчається лише на «нормальних» даних тобто даних оригінального користувача. Протягом тестової фази ступінь аномальності обчислюється як 1–s, де s – величина отримана на виході мережі. Якщо s наближається до 1.0 – тестовий вектор «подібний» до векторів з навчаючого набору.

Нейронна мережа (адаптивний фільтр)

Нейронна мережа в режимі виявлення аномалій, описана вище.

Fuzzy-logic

Детектор реалізує досить складний алгоритм на основі моделі нечіткої логіки і описаний в досліджені Хайдера [25] .

Підрахунок викидів (Outlier-counting) – z-score

Статистичний метод. Під час навчаючої фази детектор обчислює середнє значення та стандартне відхилення кожної часової ознаки. В тестовій фазі детектор обчислює абсолютне значення z-score кожної ознаки за формулою:

$$|x_i - y_i|/s_i$$

Де x_i, y_i - i-ті ознаки середнього та тестового векторів відповідно, s_i - стандартне відхилення від навчаючої фази. Міра аномальності

розраховуються як кількість z-score що перевищили певний пороговий рівень.

SVM (однокласова)

Алгоритм на базі однокласової машини опорних векторів. Детально описаний в розділі 2 даного дослідження.

К-середніх (k-means)

Метод використовує алгоритм кластеризації k-means для ідентифікації кластерів серед навчаючих векторів. Під час тестової фази алгоритм визначає близкість вектора до одного з кластерів. Ступінь аномальності визначається як евклідова відстань тестового вектора до найбижчого з центроїдів.

3.3 Методи аналізу ефективності біометричних алгоритмів автентифікації

Для вимірювання ефективності моделей використовувались ROC – криві (Receiver Operating Characteristic – робоча характеристика приймача). Рейтинг «влучання» - оцінка якості визначення користувача як зловмисника (1 – рейтинг пропусків). Рейтинг False Alarm (False Positive) – рейтинг помилкового визначення «нормального» користувача як зловмисника.

Проводилось обчислення даних рейтингів при різному рівні treshhold – порогу, та будувались відповідні графіки. ROC крива - розповсюджений метод оцінки точності детекторів, та дозволяє проводити різні оцінки такі як, наприклад, співвідношення хибних спрацювань та пропусків.

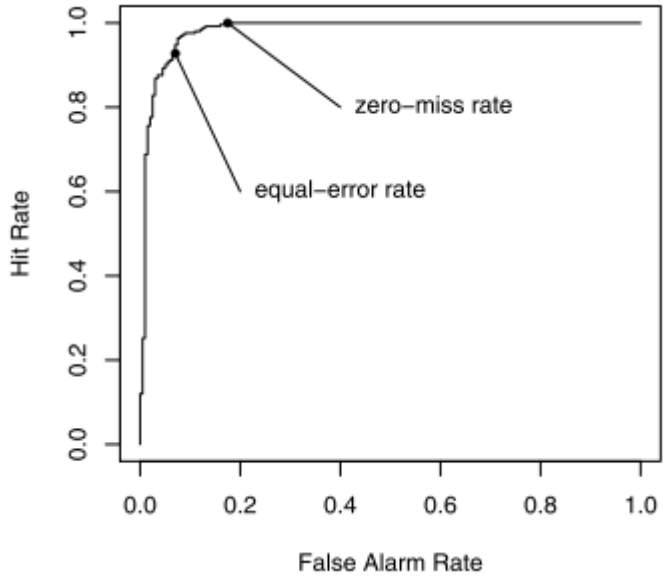


Рис. 3.2 Приклад ROC кривої

Для порівняння роботи різних алгоритмів використані оцінки *equal-error rate* та *zero-miss false-alarm rate*. Для обчислення *equal-error rate* поріг вибраний таким чином, щоб рейтинг пропусків та помилкових спрацювань були рівними. Для обчислення zero-miss false-alarm rate поріг вибраний так, щоб кількість хибних спрацювань була мінімальною при кількості пропусків рівних нулю. Описані оцінки є різні міри якості детектора, але обоє є рейтингом помилок (тобто менше значення є кращим).

3.3 Протокол дослідження моделей безпарольної автентифікації на основі клавіатурного ритму

Дослідження проводитимо з використанням бібліотеки машинного навчання Scikit-learn, в середовищі Ipython Notebook.

Імпорт необхідних модулів:

```
# модулі побудови графіків
%matplotlib inline
import matplotlib.pyplot as plt
import seaborn as sns; sns.set()

# модулі машинного навчання та функції оцінювання моделей
from sklearn.svm import OneClassSVM
from sklearn.metrics import precision_recall_fscore_support
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score, recall_score

from sklearn.metrics import f1_score
from sklearn.metrics import roc_curve, auc

# модулі передпідготовки набору даних
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# модулі матричних обчислень та маніпуляції наборами даних
import numpy as np
import pandas as pd
from scipy import stats

# допоміжні модулі
import time
import itertools

# Налаштування параметрів виводу на друк
np.set_printoptions(precision=3,suppress=True)
```

Допоміжна функція для друку часу виконання коду:

```
# Функція приймає аргумент час виконання коду в секундах та вивиодить на друк у форматі
гг:хв:сс
def print_time(t):
    m, s = divmod(t, 60)
    h, m = divmod(m, 60)
    print("Час виконання: %d г: %02d хв : %02d с" % (h, m, s))
```

Зчитуємо набір даних з файла:

```
#http://www.cs.cmu.edu/~keystroke/
data = pd.read_csv('e:/Projects/Dyplom/DSL-StrongPasswordData.csv')
```

Оцінимимо розмір вибірки:

```
print('data({0[0]},{0[1]})'.format(data.shape))
```

```
data(20400,34)
```

Фраза пароль для дослідження: .tie5Roanl

Переглянемо перші 5 записів вибірки:

```
data.head()
```

	subject	sessionIndex	rep	H.period	DD.period.t	UD.period.t	H.t	DD.t.i	UD.t.i	H.i	...	H.a	DD.a.n	UD.a.n	H.n	DD.n.l	UD.n.
0	s002	1	1	0.1491	0.3979	0.2488	0.1069	0.1674	0.0605	0.1169	...	0.1349	0.1484	0.0135	0.0932	0.3515	0.258
1	s002	1	2	0.1111	0.3451	0.2340	0.0694	0.1283	0.0589	0.0908	...	0.1412	0.2558	0.1146	0.1146	0.2642	0.149
2	s002	1	3	0.1328	0.2072	0.0744	0.0731	0.1291	0.0560	0.0821	...	0.1621	0.2332	0.0711	0.1172	0.2705	0.153
3	s002	1	4	0.1291	0.2515	0.1224	0.1059	0.2495	0.1436	0.1040	...	0.1457	0.1629	0.0172	0.0866	0.2341	0.147
4	s002	1	5	0.1249	0.2317	0.1068	0.0895	0.1676	0.0781	0.0903	...	0.1312	0.1582	0.0270	0.0884	0.2517	0.163

5 rows × 34 columns

```
# Назви стовпців
data.columns
```

```
Index(['subject', 'sessionIndex', 'rep', 'H.period', 'DD.period.t',
       'UD.period.t', 'H.t', 'DD.t.i', 'UD.t.i', 'H.i', 'DD.i.e', 'UD.i.e',
       'H.e', 'DD.e.five', 'UD.e.five', 'H.five', 'DD.five.Shift.r',
       'UD.five.Shift.r', 'H.Shift.r', 'DD.Shift.r.o', 'UD.Shift.r.o', 'H.o',
       'DD.o.a', 'UD.o.a', 'H.a', 'DD.a.n', 'UD.a.n', 'H.n', 'DD.n.l',
       'UD.n.l', 'H.l', 'DD.l.Return', 'UD.l.Return', 'H.Return'],
      dtype='object')
```

Принцип назв полів даних:

- H.period - час утримування клавіші .
- DD.period.t - час між натискуваннями клавіш . та t
- UD.period.t - час між відпусканням клавіші . та натискуванням клавіші t

Візуалізація

Визначаємо кількість піддослідних:

```
n_subject = len(data.subject.unique())
print('Кількіст піддослідних у вибірці: %i' % n_subject)
```

```
Кількіст піддослідних у вибірці: 51
```

Підготовка даних:

```
# кодуємо мітку піддослідного номером
le_subject = LabelEncoder()
data.subject = le_subject.fit_transform(data.subject)

# для більш чіткої візуалізації беремо до розрахунку тільки 4 сесії
# і тільки 20 спроб у кожній сесії і тільки 5 піддослідних
data_small = data[(data.sessionIndex <= 4) & (data.rep <= 20) & (data.subject <= 4)]

target=data_small.subject

X=data_small.drop(labels=["subject","sessionIndex","rep"], axis=1)

# додатковий масив ознак часу між відпусканням клавіші та натискуванням - на випадок коли ми
захочемо видалити ці ознаки з вибірки.
UD_labels=['UD.period.t', 'UD.t.i', 'UD.i.e', 'UD.e.five', 'UD.five.Shift.r',
           'UD.Shift.r.o', 'UD.o.a','UD.a.n', 'UD.n.l', 'UD.l.Return']

# Видаляємо ці ознаки так як вони корелюють з ознаками типу DD (натискування -натискування) та
H (утримування)
X=X.drop(labels=UD_labels,axis=1)
```

```
# Оцінимо розмір отриманої вибірки даних
X.shape
```

```
(400, 21)
```

Розраховуємо центроїди (середні вектори):

Обчислюємо центроїд *c* для кожної групи *1, 2, ..., k*, де *к* кількість піддослідних

$$c_{ij} = \frac{1}{m}\sum_{l=1}^{m_i} x_{lj}, j = 1,2,\dots d$$

де m_i - кількість векторів у групі, d - кількість компонент вектора (кількість ознак)

```
#array для зберігання середніх векторів (центроїдів)
centroids=pd.DataFrame(np.zeros((5,X.shape[1])),columns=X.columns,dtype='float')

#пробігаємось по всіх групах і знаходимо центроїд  групи
for j in range(len(target.unique())):
    centroids.loc[j,:]=X[target==j].sum()/len(X[target==j])

centroids.shape
```

```
(5, 21)
```

```
#Додамо обчислені центроїди у кінець виббірки для того щоб наступні перетворення дотичили і їх

X=pd.concat([X,centroids])
```

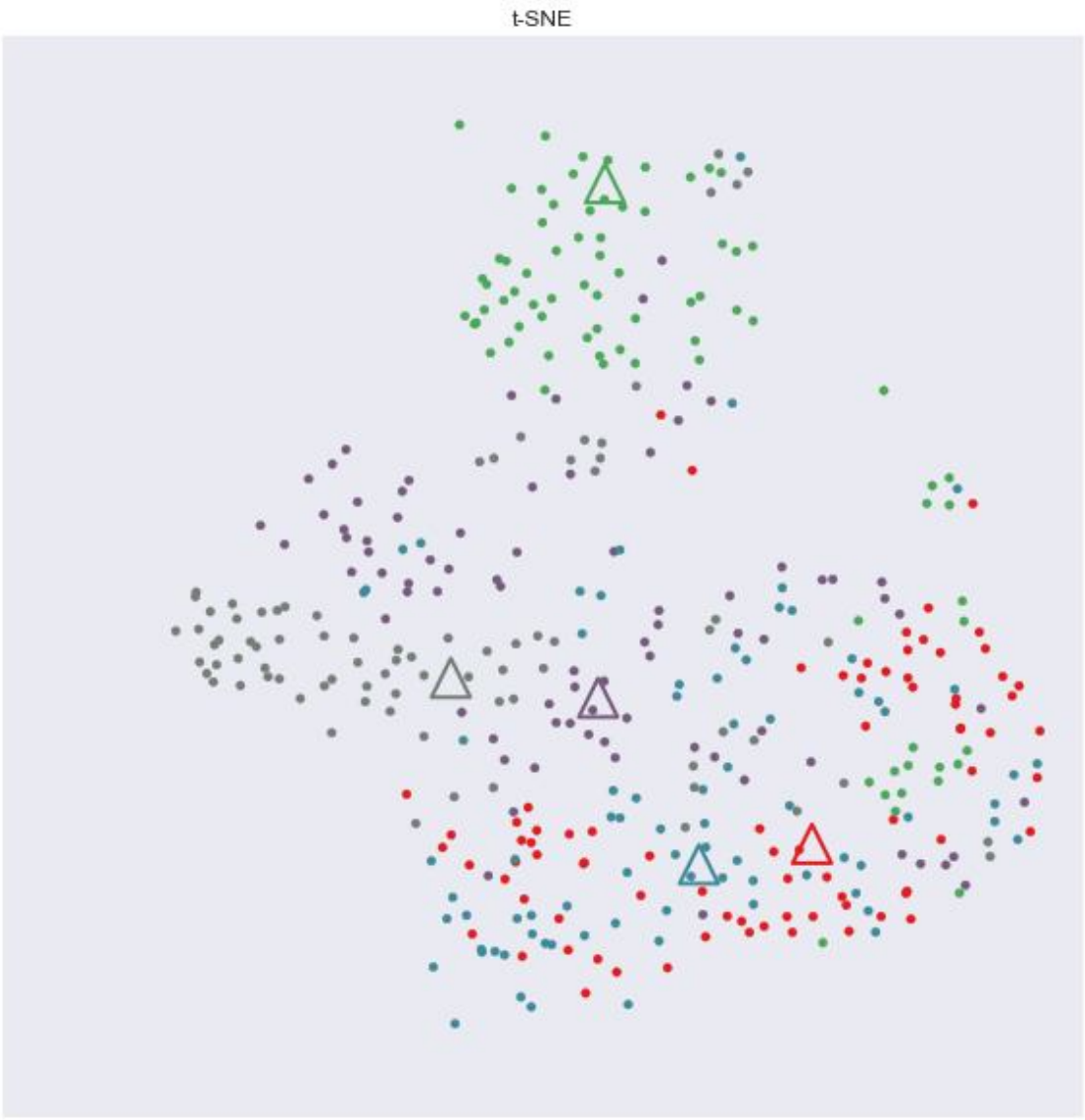
Перетворюємо дані (понижуємо розмірність) методом t-distributed Stochastic Neighbor Embedding (t-SNE):

```
from sklearn import manifold

t0 = time.time()
model = manifold.TSNE(n_components=2, init='pca', random_state=0)
X_2 = model.fit_transform(X)
t1 = time.time()
print_time(t1 - t0)
```

```
Час виконання: 0 г: 00 хв : 02 с
```

Візуалізуємо отримані дані:

```
plt.figure(figsize=(10,10))
plt.scatter(X_2[:400, 0], X_2[:400, 1],color=plt.cm.Set1(target*20))
plt.scatter(X_2[400:, 0], X_2[400:, 1], s=400, marker='^',facecolors='none',
edgecolors=plt.cm.Set1(target.unique()*20),linewidths=2)
plt.title("t-SNE")
plt.xticks([]), plt.yticks([])
plt.show()
```



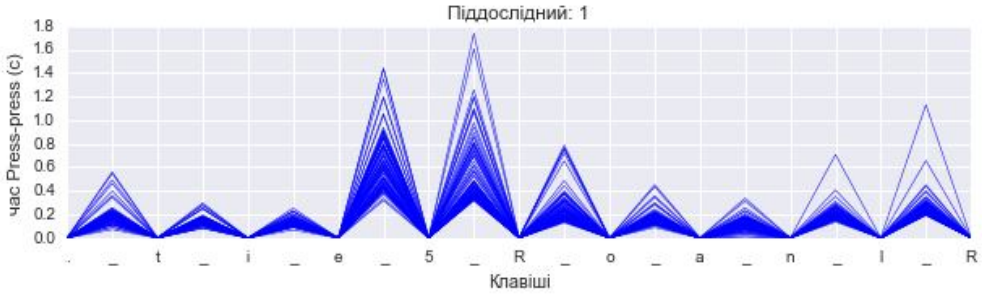
Побудуємо графік часового ряду для 5-ти піддослідних

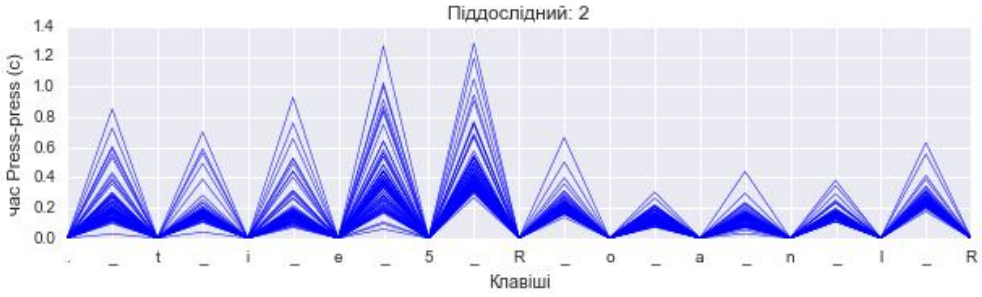
```
# ознаки які беремо до уваги при побудові графіку
cols=['DD.period.t', 'DD.t.i', 'DD.i.e', 'DD.e.five','DD.five.Shift.r', 'DD.Shift.r.o',
      'DD.o.a', 'DD.a.n', 'DD.n.l', 'DD.l.Return']
# обмежимо розмір вибірки
X_3=X[cols]

#допоміжний код для красивих графіків - додаємо перед кожним стовпцем стовпець 0.
df=pd.DataFrame()
for i in X_3.columns:
    X_3.insert(X_3.columns.get_loc(i),value=0, column=i+'_')

X_3.insert(len(X_3.columns),value=0, column='_')
```

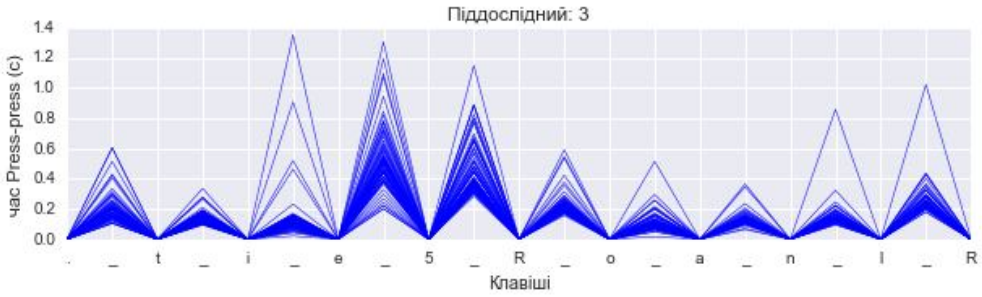
Графік розподілу часу між послідовним натискуванням клавіш для кожного з 5-ти піддослідних:

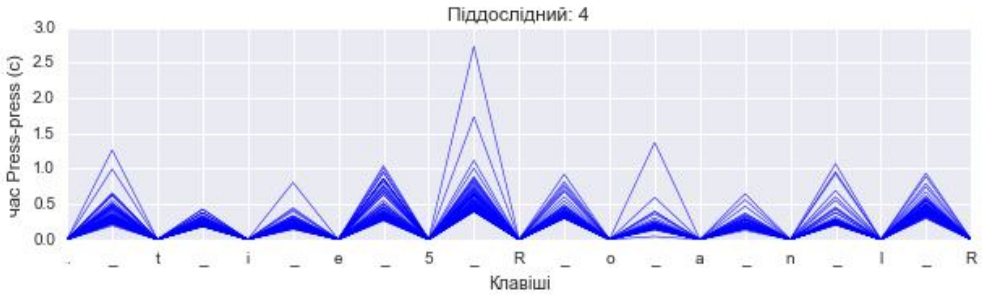
```
plt.figure(figsize=(10,20))
x_t=[x for x in range(0,21)]

for fig in range(5):
```

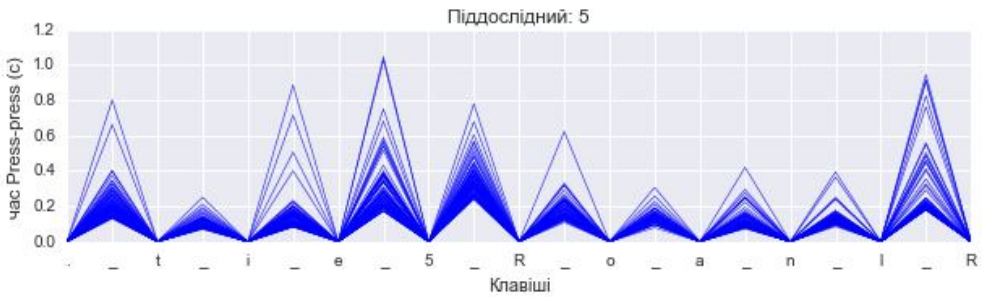
```
    plt.subplot(5,1,fig+1)

    for idx, row in X_3.loc[target==fig].iterrows():
        plt.plot(x_t,row,lw=0.5,color='b')

    plt.title("Піддослідний: %s" %(fig+1))
    passwd='.tie5RoanlR'
    x_labels=list('_'.join(list(passwd)))
    plt.xticks(x_t,x_labels)

    ax = plt.gca()
    ax.set_ylabel("час Press-press (c)")

    ax.set_xlabel("Клавіші")

plt.subplots_adjust(hspace=1)
plt.show()
```


Піддослідний: 1
час Press-press (c)
1.8
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
. _ t _ i _ e _ 5 _ R _ o _ a _ n _ l _ R
Клавіші


Піддослідний: 2
час Press-press (c)
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
. _ t _ i _ e _ 5 _ R _ o _ a _ n _ l _ R
Клавіші


Піддослідний: 3
час Press-press (c)
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
. _ t _ i _ e _ 5 _ R _ o _ a _ n _ l _ R
Клавіші


Піддослідний: 4
час Press-press (c)
3.0
2.5
2.0
1.5
1.0
0.5
0.0
. _ t _ i _ e _ 5 _ R _ o _ a _ n _ l _ R
Клавіші


Піддослідний: 5
час Press-press (c)
1.2
1.0
0.8
0.6
0.4
0.2
0.0
. _ t _ i _ e _ 5 _ R _ o _ a _ n _ l _ R
Клавіші

Euclidean

Попередня підготовка даних:

```
target=data.subject

X=data.drop(labels=["subject","sessionIndex","rep"], axis=1)
#X=X.drop(labels=UD_labels,axis=1)

***Розраховуємо середні значення векторів:***

n_subject=len(target.unique())

centroids = np.zeros((n_subject,X.shape[1]))
for j in target.unique():
    centroids[j,:]=X[target==j].sum()/len(X[target==j])


print(centroids.shape)
```

```
(51, 31)
```

Розраховуємо середню відстань кожної точки у групі до відповідного цетроїда і на основі цього розраховуємо граничне значення відстані:

```
# Граничне значення відстані розраховуємо для кожного з піддослідних окремо
tresholds=np.zeros(len(target.unique()))

for j in target.unique():
    distances=np.linalg.norm(X[target==j]-centroids[j],axis=1)**2
    dist_mean=distances.mean()
    std_dev=distances.std()
    # print('mean: %.3f, std: %.3f, max: %.3f' % (dist_mean,std_dev,distances.max()))

    # задаємо поріг - як 2 стандартних відхилення від середнього значення
    tresholds[j]=dist_mean + 2*std_dev
```

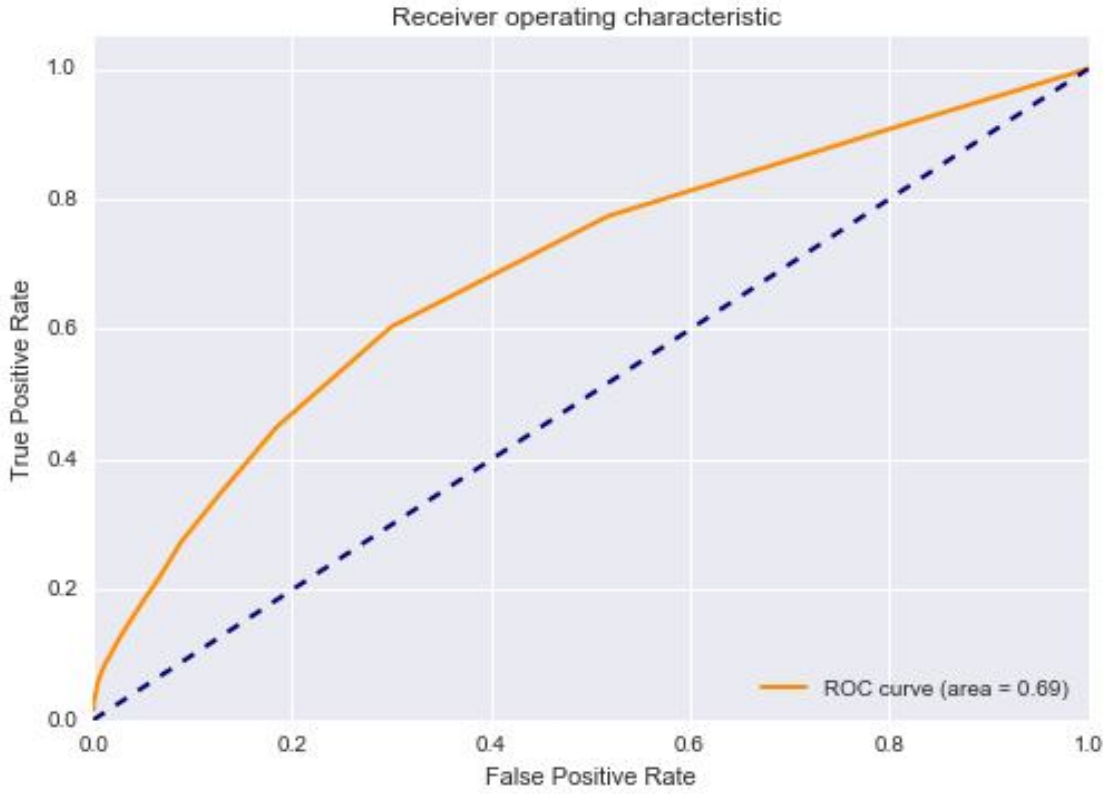
Функція передбачення:

```
def prediction(X_test, subject, centroids, treshold):
    return np.linalg.norm(X_test-centroids[subject], axis=1) < tresholds[subject]
```

Перебираємо різні значення treshold і будуємо графік:

```
def MY_ROC(X_t,y_t,centroids,tresholds):
    tpr=list()
    fpr=list()
    tpr=[]
    fpr=[]
    for tr in range(-20,20,1):
        tp = 0
        fp = 0
        len_true_pos = 0
        len_false_pos = 0
        #print('tr: %s' % (tr))
        for i in target.unique():
            cur_treshold = tresholds[i] + tresholds[i]*tr/10

            #tp - true positive  - кількість «влучень» - правильного визначення користувача як
зловмисника
            tp = tp + np.count_nonzero(np.linalg.norm(X_t[y_t!=i]-centroids[i], axis=1) >
cur_treshold)
            len_true_pos = len_true_pos + len(X_t[y_t!=i])
```

```
            #fp - false positive - кількість  помилковj сприйнятих "нормальних" користувачів
за зловмисника.
            fp = fp + np.count_nonzero(np.linalg.norm(X_t[y_t==i]-centroids[i], axis=1) >
cur_treshold)
            len_false_pos = len_false_pos + len(X_t[y_t==i])


        tpr.append(tp/len_true_pos)
        fpr.append(fp/len_false_pos)
    return (tpr,fpr)

# знахидомо усереднені дані tpr, fpr по всій вибірці
tpr,fpr = MY_ROC(X,target,centroids,tresholds)
roc_auc = auc(fpr, tpr)

#Будуємо ROC криву
plt.figure()
lw = 2
plt.plot(fpr, tpr, color='darkorange',
         lw=lw, label='ROC curve (area = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic ')
plt.legend(loc="lower right")
plt.show()
```



З форми графіка ROC-кривої робимо висновок про низьку якість передбачення на основі побудованої моделі.

Використання штучних нейронних мереж для розв'язку задачі

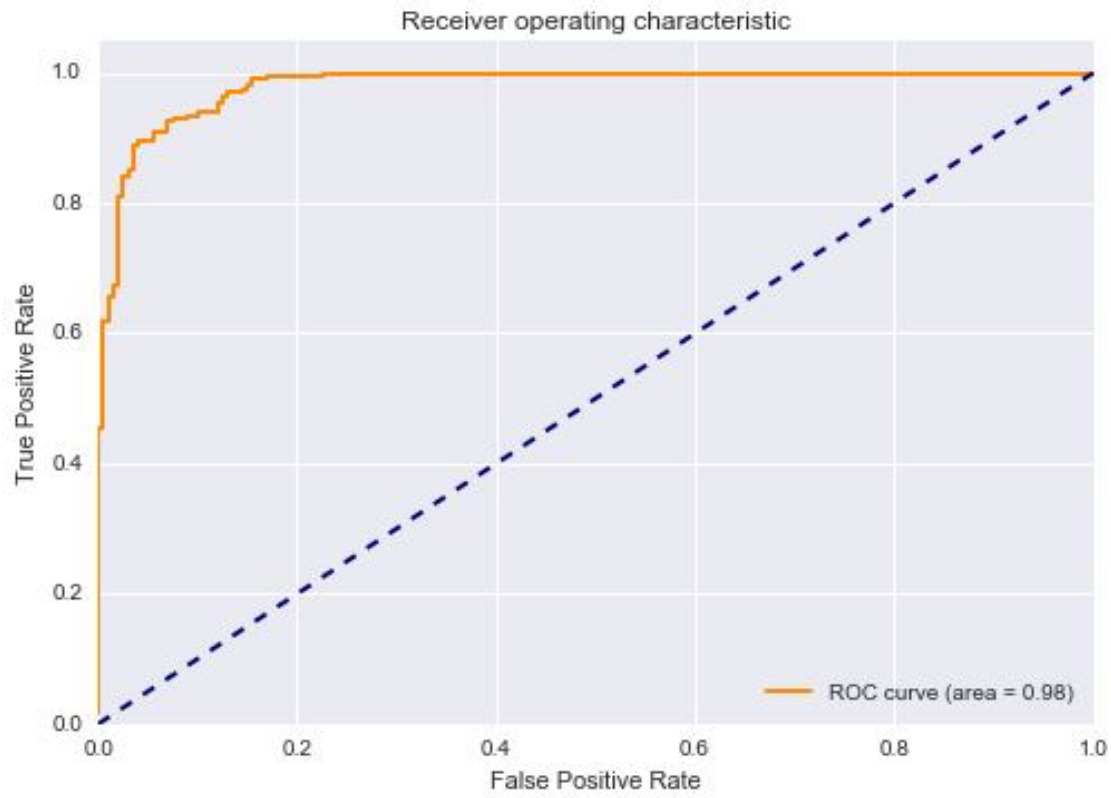
Спочатку спробуємо використати стандартний нейромережевий класифікатор:

```
from sklearn.neural_network import MLPClassifier

# будуємо класифікатор
clf = MLPClassifier(solver='lbfgs', activation='logistic', alpha=1e-4,
learning_rate_init=0.0001, max_iter=1000,
                    hidden_layer_sizes=(10), random_state=1)

fpr = dict()
tpr = dict()
roc_auc = dict()
y_test_all=np.array([])
y_predicted_all=np.array([])

for s in target.unique():
    print("Опрацьовуємо користувача: %s" % s)
    X_0=X[target==s]

    #X_train, X_test = train_test_split(X_0, test_size=0.5, random_state=0)

    # Розбиваємо на рівні частини згідно з описаною методологією тестування
    X_train=X_0[:200]
    X_test=X_0[200:]


    X_train_neg=X[target!=s]

    # 1 - пароль введений не оригінальним користувачем, 0  - пароль введений користувачем
    y_train = np.hstack((np.zeros(X_train.shape[0]),np.ones(X_train_neg.shape[0]) ))

    # загальна навчаюча вибірка з позитивними та негативними екземплярами
    X_train=np.vstack((X_train,X_train_neg))



    #ToDo - переробити:  взяти 5 ряків з кожного користувача
    # Обмежуємо кількість негативних даних для тестування
    X_test_neg=X[target!=s].sample(n=len(X_test))

    y_test = np.hstack(( np.zeros(X_test.shape[0]), np.ones(X_test_neg.shape[0]) ))

    # загальна тестова вибірка з позитивними та негативними екземплярами
    X_test=np.vstack((X_test,X_test_neg))


    #навчаємо класифікатор
    clf.fit(X_train,y_train)

    #Робимо передбачення на тестових даних
    predict=clf.predict_proba(X_test)

    #
    y_test_all = np.hstack((y_test_all, y_test))
    y_predicted_all = np.hstack((y_predicted_all, predict[:,1].ravel()))

    # Будуємо ROC криву
    fpr[s], tpr[s], _ = roc_curve(y_test, predict[:,1])
    roc_auc[s] = auc(fpr[s], tpr[s])

    print('AUC: %.3f' % (roc_auc[s]))


# Обчислимо усереднену ROC криву і ROC площу використовуючи micro-average усереднення
```

```
# http://scikit-learn.org/stable/modules/model_evaluation.html
fpr["micro"], tpr["micro"], _ = roc_curve(y_test_all, y_predicted_all)
roc_auc["micro"] = auc(fpr["micro"], tpr["micro"])
```

```
Опрацьовуємо користувача: 0
AUC: 0.924
Опрацьовуємо користувача: 1
AUC: 0.981
Опрацьовуємо користувача: 2
AUC: 0.989
Опрацьовуємо користувача: 3
AUC: 0.979
Опрацьовуємо користувача: 4
AUC: 0.857
Опрацьовуємо користувача: 5
AUC: 0.879
Опрацьовуємо користувача: 6
AUC: 0.999
Опрацьовуємо користувача: 7
AUC: 0.973
Опрацьовуємо користувача: 8
AUC: 0.981
Опрацьовуємо користувача: 9
AUC: 0.877
Опрацьовуємо користувача: 10
AUC: 0.911
Опрацьовуємо користувача: 11
AUC: 0.961
Опрацьовуємо користувача: 12
AUC: 0.993
        … … … …

Опрацьовуємо користувача: 39
AUC: 0.968
Опрацьовуємо користувача: 40
AUC: 0.977
Опрацьовуємо користувача: 41
AUC: 0.982
Опрацьовуємо користувача: 42
AUC: 0.986
Опрацьовуємо користувача: 43
AUC: 0.979
Опрацьовуємо користувача: 44
AUC: 0.957
Опрацьовуємо користувача: 45
AUC: 1.000
Опрацьовуємо користувача: 46
AUC: 0.998
Опрацьовуємо користувача: 47
AUC: 0.934
Опрацьовуємо користувача: 48
AUC: 0.983
Опрацьовуємо користувача: 49
AUC: 0.839
Опрацьовуємо користувача: 50
AUC: 0.830
```

```
plt.figure()
lw = 2
plt.plot(fpr[1], tpr[1], color='darkorange',
         lw=lw, label='ROC curve (area = %0.2f)' % roc_auc[1])
plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic')
plt.legend(loc="lower right")
plt.show()
```




```
#Обчислюємо загальну accuracy
pred = np.where(y_predicted_all < 0.5,0,1)
accuracy=accuracy_score(y_test_all,pred)
print(accuracy)

0.773284313725
```

З графіка ROC-кривої робимо висновок, що дана модель є більш якісною за модель Euclidean. Проте використання моделі в робочій системі контролю доступу на основі біометрії клавіатурного введення є неефективним: при додаванні нового користувача, нейронну мережу потрібно повністю перенавчати на всій вибірці.

Модель на основі штучної нейронної мережі з архітектурою детектора аномалій

```
# допоміжна функція для видалення викидів з вибірки. Використаємо для цього однокласову машину опорних векторів
 # видалимо 10% даних найбільш "віддалених".
OUTLIER_FRACTION=0.1
def is_inlier(X1):
    clf_1 = OneClassSVM(kernel="linear")
    clf_1.fit(X1)
    dist_to_border = clf_1.decision_function(X1).ravel()
    threshold = stats.scoreatpercentile(dist_to_border,
                100 * OUTLIER_FRACTION)

    return dist_to_border > threshold

# визначаємо сигмоїдну функцію
def sigmoid(X):
    s=np.zeros(np.shape(X))
    s = 1/(1 + np.exp(-X))
    return s

# для побудови мережі використовуємо MultiLayerPerceptron Regressor
from sklearn.neural_network import MLPRegressor

# оптимізатор sgd - stochastic gradient descent (стохастичний градієнтний спуск),
# активація прихованого шару - сигмоїдна функція

clf = MLPRegressor(solver='sgd', activation='logistic', alpha=0.0001, tol=0.00001,
                   learning_rate_init=0.0001, max_iter=20000, momentum = 0.0003,
                    hidden_layer_sizes=(20), random_state=1, verbose=False,shuffle=True)

fpr = dict()
tpr = dict()
roc_auc = dict()
tresholds = dict()
acc = []

y_test_all=np.array([])
y_predicted_all=np.array([])

# описуїмо скалер для подальшого масштабуваня даних
scaler = StandardScaler()

for s in target.unique():

    print("Опрацьовуємо користувача: %s" % s)

    # елементи того ж класу - паролі введені оригінальним користувачем
    is_same_class = (target.as_matrix()==s)
    X_0=X[is_same_class]

    #проводимо розбиття згідно методології тестування
    X_train=X_0[:200]
    X_test=X_0[200:]


    # попередня підготовка даних - масштабуємо  в діапазон [0..1]

    scaler.fit(X_train)
    X_train = scaler.transform(X_train)


    # Опціонально фільтруємо навчаючу вибірку
    #is_good = is_inlier(X_train)
```

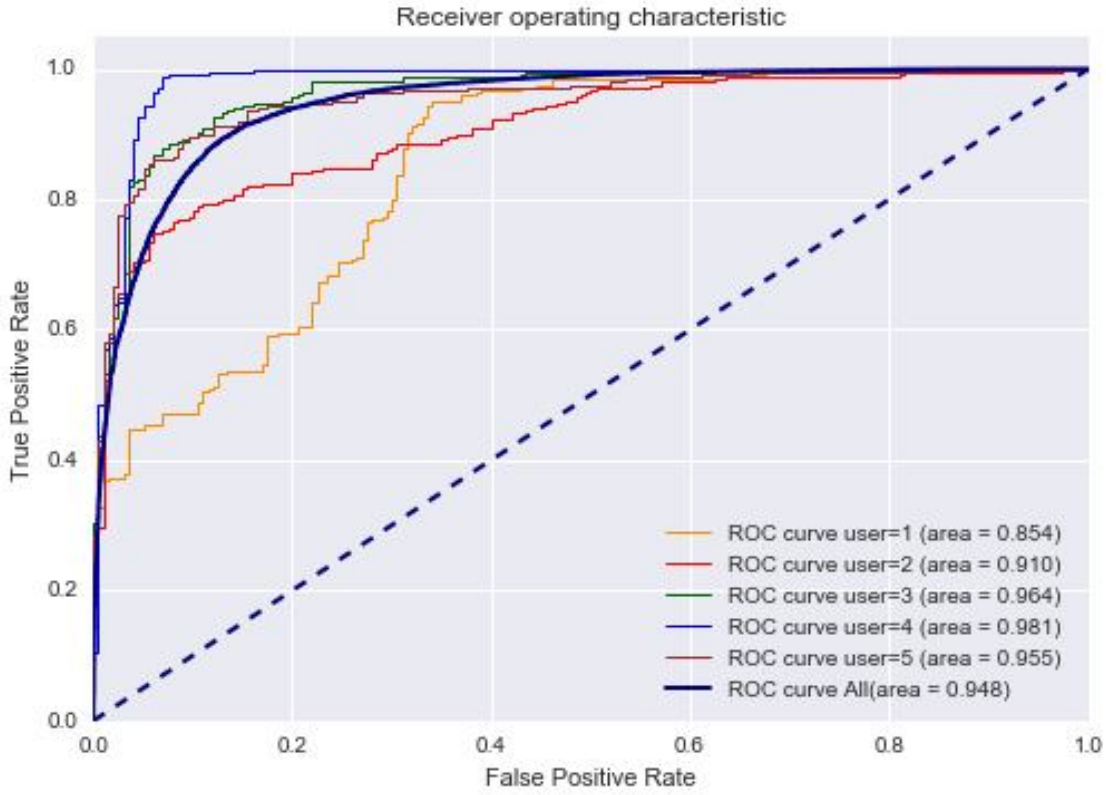
```
    #X_train=X_train[is_good]

    X_test_neg=np.empty((0,X_norm.shape[1]))

    # формуємо вибірку позитивну - беремо перші 5 введень кожного користувача відмінного від
оригінального
    for u in target.unique():
        if u!=s:
            X_test_neg=np.vstack((X_test_neg, X[target.as_matrix()==u][:5]))


    y_test = np.hstack((np.zeros(X_test.shape[0]), np.ones(X_test_neg.shape[0]) ))

    # загальна тестова вибірка з позитивними та негативними екземплярами
    X_test=np.vstack((X_test,X_test_neg))

    # застосовуємо масштабування і до тестових даних
    X_test = scaler.transform(X_test)


    #навчаємо детектор -  в якості вектора результатів підсавляємо ветор X
    # навчаємо тільки на нормальних даних - біометрика паролей введених оригінальним
користувачем
    clf.fit(X_train,X_train)

    #для визначення treshold визначимо відстані на тренувальній вибірці
    dist_train=np.linalg.norm(X_train-clf.predict(X_train),axis=1)

    # В якості порогу вибираємо значення відстані 95% екземплярів вибірки.
    # ця величина потрібна лише щоб обчислити accuracy, і в побудові ROC не бере участі
    treshold=stats.scoreatpercentile(dist_train, 95)
    print('Treshold: %.3f' % treshold)

    #Робимо передбачення на тестових даних
    PREDICTED=clf.predict(X_test)

    # обчислюємо відстань поміж тестовими часовими  векторами та векторами отриманими на
виході нейронної мережі
    # чим більша відстань тим більша ймовірність що дані не належать ритму друку оригінального
користувача
    dist = np.linalg.norm(X_test - PREDICTED,axis=1)

    # робимо передбачення на основі порогової величини
    predicted=np.where(dist < treshold,0,1)

    # обчислюєму загальну точність
    accuracy=accuracy_score(y_test,predicted)

    #масив значень accuracy по кожному з 50-ти користувачів
    acc.append(accuracy)

    print('accuracy: %.3f' % accuracy)

    # Будуємо ROC криву
    # dist - як міра оцінки - чим більша - тим більша ймовірність що пароль введено
зловмисником
    fpr[s], tpr[s], tresholds[s] = roc_curve(y_test, dist)
    roc_auc[s] = auc(fpr[s], tpr[s])

    print('AUC: %.3f' % (roc_auc[s]))

    #збираємо дані по всім користувачам для обчислення загальної статистики
    y_test_all = np.hstack((y_test_all, y_test))
    #y_predicted_all = np.hstack((y_predicted_all, predicted_proba))

    y_predicted_all = np.hstack((y_predicted_all, dist))
```

```
# Обчислимо усереднену ROC криву і ROC площу використовуючи micro-average усереднення
# http://scikit-learn.org/stable/modules/model_evaluation.html
fpr["avrg"], tpr["avrg"], _ = roc_curve(y_test_all, y_predicted_all)
roc_auc["avrg"] = auc(fpr["avrg"], tpr["avrg"])

print('Average AUC : %.3f' % (roc_auc["avrg"]))

print('Average Accuracy : %.3f' % (sum(acc)/len(acc)))
```

```
Опрацьовуємо користувача: 0
Treshold: 10.105
accuracy: 0.689
AUC: 0.852
Опрацьовуємо користувача: 1
Treshold: 9.488
accuracy: 0.822
AUC: 0.904
Опрацьовуємо користувача: 2
Treshold: 9.720
accuracy: 0.833
AUC: 0.961
Опрацьовуємо користувача: 3
Treshold: 9.716
accuracy: 0.816
AUC: 0.981
 …  …  …

Опрацьовуємо користувача: 46
Treshold: 9.086
accuracy: 0.951
AUC: 0.989
Опрацьовуємо користувача: 47
Treshold: 10.358
accuracy: 0.762
AUC: 0.967
Опрацьовуємо користувача: 48
Treshold: 9.883
accuracy: 0.976
AUC: 0.997
Опрацьовуємо користувача: 49
Treshold: 9.879
accuracy: 0.804
AUC: 0.950
Опрацьовуємо користувача: 50
Treshold: 8.704
accuracy: 0.889
AUC: 0.966
Average AUC: 0.946
Average Accuracy: 0.812
```

```
plt.figure()
lw = 1
plt.plot(fpr[0], tpr[0], color='darkorange',
         lw=lw, label='ROC curve user=%s (area = %0.3f) ' % (1, roc_auc[0]))

plt.plot(fpr[1], tpr[1], color='red',
         lw=lw, label='ROC curve user=%s (area = %0.3f) ' % (2, roc_auc[1]))

plt.plot(fpr[2], tpr[2], color='darkgreen',
         lw=lw, label='ROC curve user=%s (area = %0.3f) ' % (3, roc_auc[2]))
```

```
plt.plot(fpr[3], tpr[3], color='blue',
         lw=lw, label='ROC curve user=%s (area = %0.3f) ' % (4, roc_auc[3]))

plt.plot(fpr[4], tpr[4], color='brown',
         lw=lw, label='ROC curve user=%s (area = %0.3f) ' % (5, roc_auc[4]))


lw = 2
plt.plot(fpr["avrg"], tpr["avrg"], color='navy',
         lw=lw, label='ROC curve All(area = %0.3f)' % roc_auc["avrg"])

plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic')
plt.legend(loc="lower right")
plt.show()
```



3.4 Висновки до розділу

Проведено аналіз систем біометричної автентифікації на основі ритму клавіатурного введення. Розглянуті методи аналізу ефективності біометричних алгоритмів автентифікації.

Проведено детальне дослідження моделей безпарольної автентифікації. Оптимізовано структуру та параметри штучної нейронної мережі з досягненням метрики «площа під ROC кривою» - 0.948.

Детальне порівняння запроектованої системи та існуючих систем буде подано в розділі 4.

РОЗДІЛ 4. Програмна реалізація та тестування

4.1 Модель життєвого циклу

Для реалізації системи біометричної автентифікації було обрано спіральну модель розробки. Головним завданням було отримати робочий варіанти системи на кожному циклі ітерації для тестування та корегування моделі даних.

Оскільки загальний термін розробки досить стислий, було обрано довжину циклу в 4 дні. На кожному циклі система обговорювалася з керівництвом підприємства та системними адміністраторами.

4.2 Інженерія вимог

Для проектування системи обрано об’єктно орієнтований підхід. Проведено аналіз проблем, цілей, сценаріїв, об’єктів. На основі цього проведено фіксацію ролей акторів, створено use-case діаграму.

Також на цьому етапі сформульовано **нефункціональні вимоги**: надійність, зручність використання, мінімальний рівень обслуговування.

Діаграма активностей:

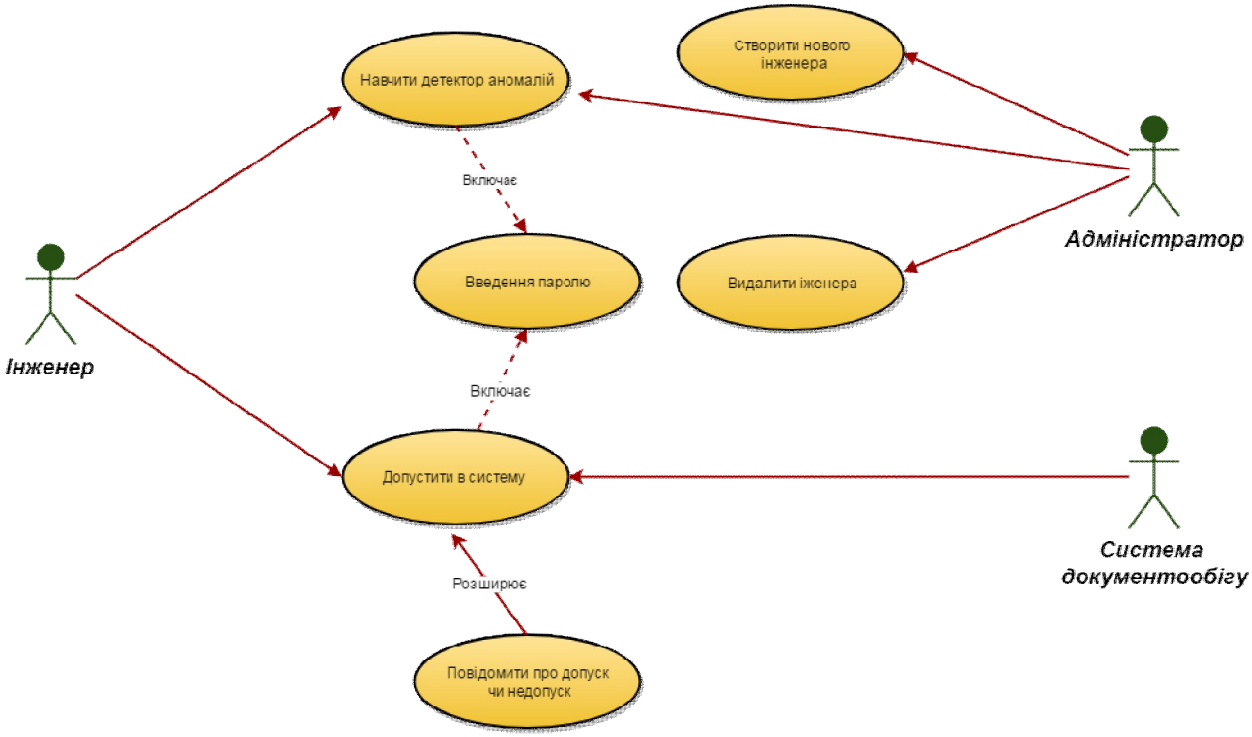


Рис. 4.1 Діаграма активностей системи біометричної автентифікації

Специфікація прецидентів

Прецидент «Навчити детектор аномалій»

Прецидент дає змогу вибрати ім’я користувача, детектор аномалій якого будемо навчати, та здійснити ряд послідовних введень заданого паролю для реєстрації клавіатурного ритму. Дія відбувається під наглядом **Адміністратора.**

Прецидент «Допустити в систему»

Надає можливість введення ім’я користувача та пароля для доступу в систему документообігу. На основі аналізу клавіатурного ритму система здійснює допуск у систему та додактово сповіщує користувача в разі відхилення доступу.

Прецидент «Створити нового інженера»

Надає можливість адміністратору системи створити нового інженера у списку користувачів системи

Прецидент «Видалити інженера»

Надає можливість адміністратору системи видалити інженера зі списку користувачів системи.

4.3 Проектування ПЗ

Для проектування системи обрано об’єктно-орієнтований підхід

4.3.1 Визначення системних архітектур

Система створена за клієнт серверною архітектурою.

Серверна частина. Система управління базами даних Interbase Firebird 1.0 проінстальована на сервері x86 архітектури. Операційна система сервера Microsoft Windows 2003.

Клієнтська частина. Основні програмні модуль системи – виконувані *.exe файли 32-бітної архітектури x86. Клієнтські ПК працюють під управлінням ОС Windows 7. На клієнтських ПК також повинна бути встановлена клієнтська ODBC бібліотека доступу до СУБД-сервера Firebird.

4.3.2 Декомпозиція системи на окремі функціональні блоки

Оскільки проектований програмний засіб не реалізує складних бізнес процесів, для декомпозиції системи вирішили обмежитись лише діаграмою потоків даних та діаграмою класів.

Діаграма потоків даних

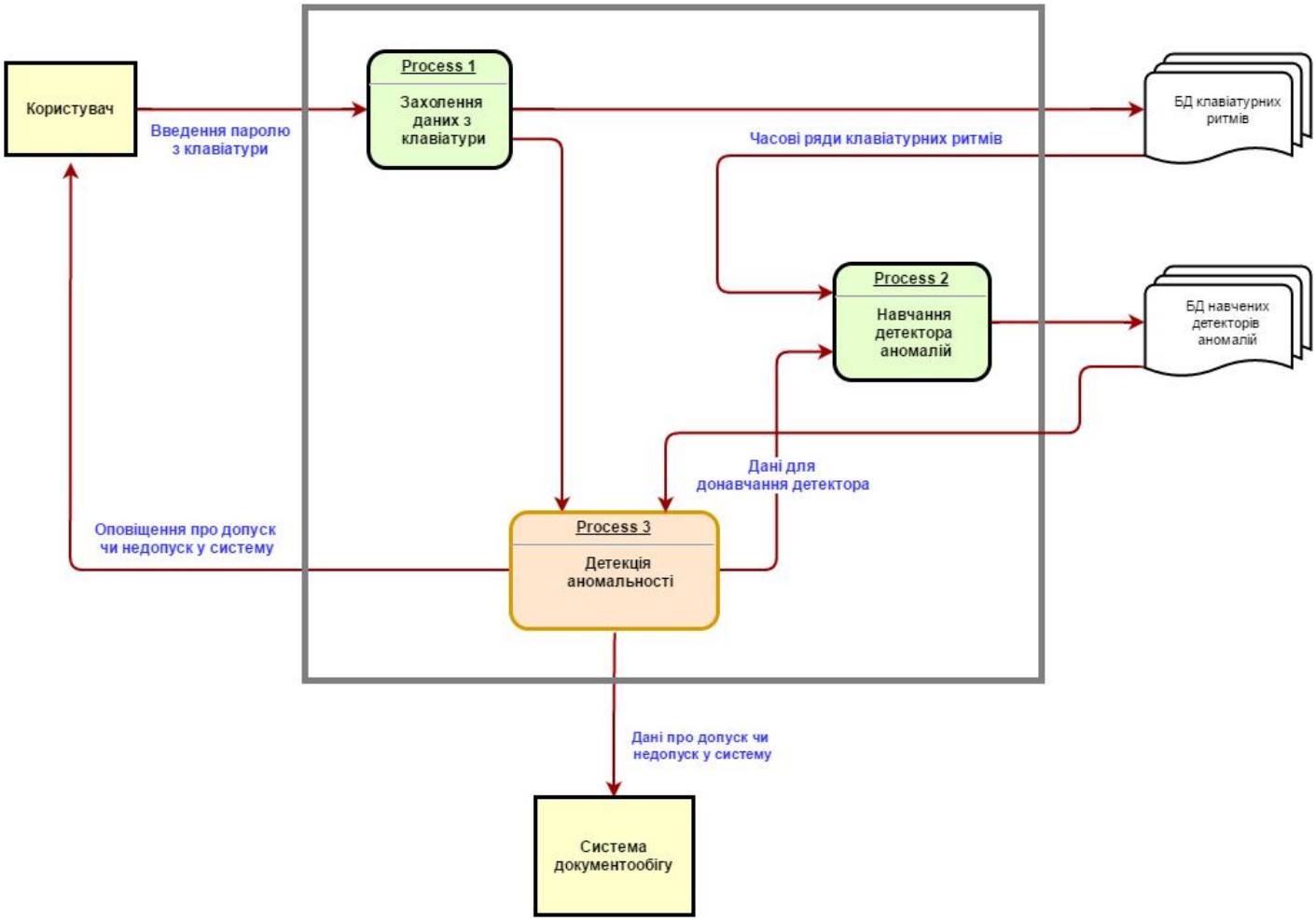


Рис. 4.2 Діаграма потоків даних системи біометричної автентифікації

Діаграма класів

Попередня діаграма класів створювалась на етапі інженерії вимог та уточнювалась протягом наступних циклів проектування системи. Остаточна діаграма класів подана на рис. 4.3.

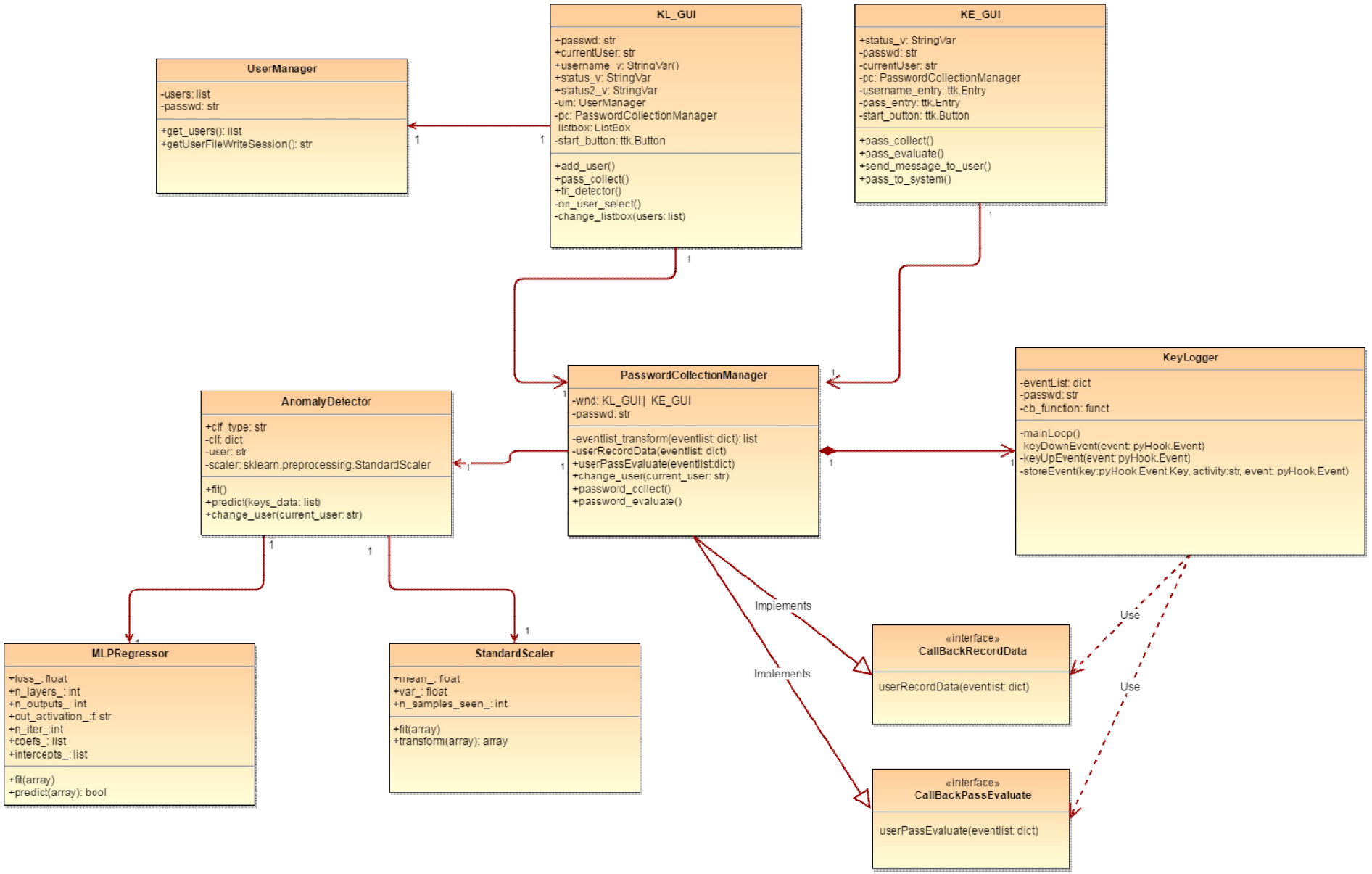


Рис. 4.3 Діаграма класів системи біометричної автентифікації

4.4 Реалізація ПЗ в коді

Для управління версіями та зберіганням коду використано систему GitHub. Код написаний мовою програмування Python з використанням бібліотеки машинного навчання Scikit-Learn, та модуля tkinter для побудови віконного інтерфейсу. Розробка велась згідно з методологією TDD (Test-Driven Development).

4.5 Тестування

Проводились наступні види тестів:

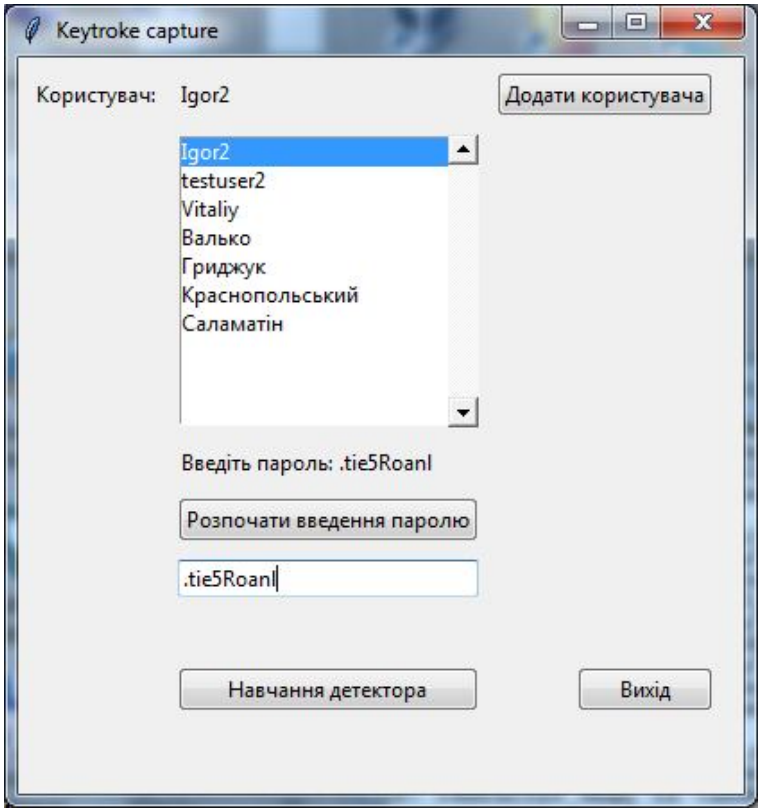
- тестування окремих елементів
- інтеграційне тестування
- тестування системи

Unit-тестування окремих елементів програмного коду мовою Python велося з використанням фреймворку **pytest.**

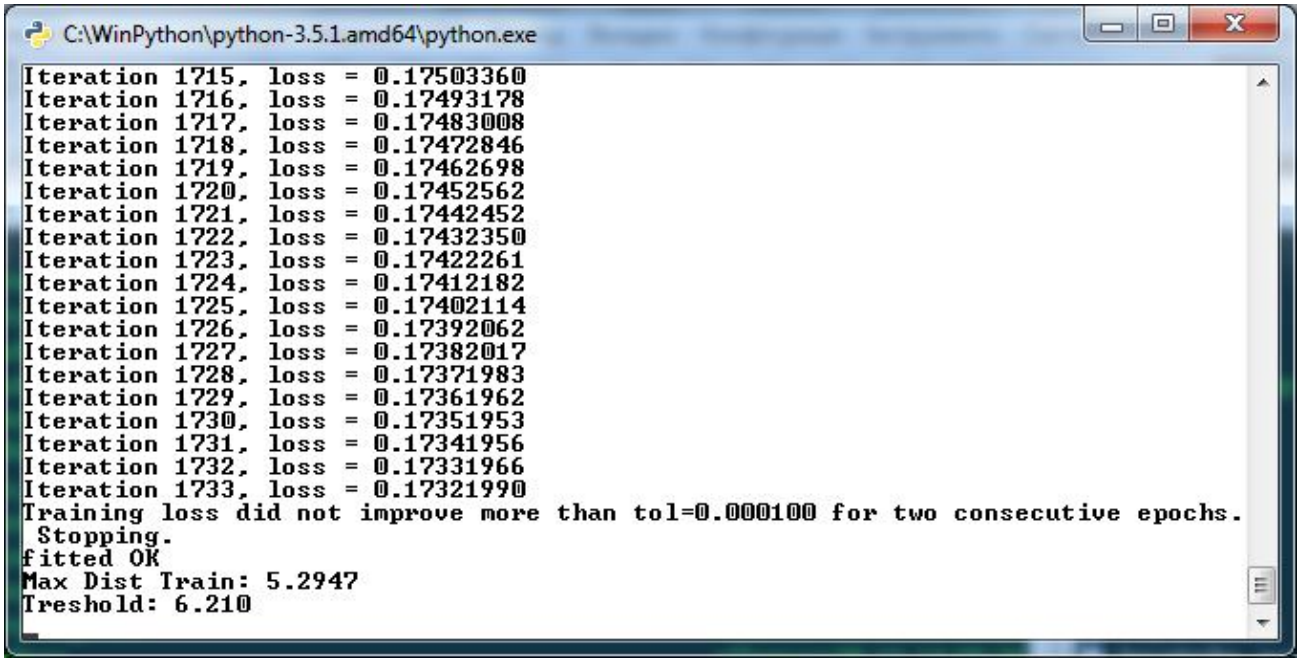
Для загального тестування системи використовувались метрики якості алгоритмів машинного навчання.

4.6. Копії екранів

Підмодуль додавання користувачів та навчання детектора:

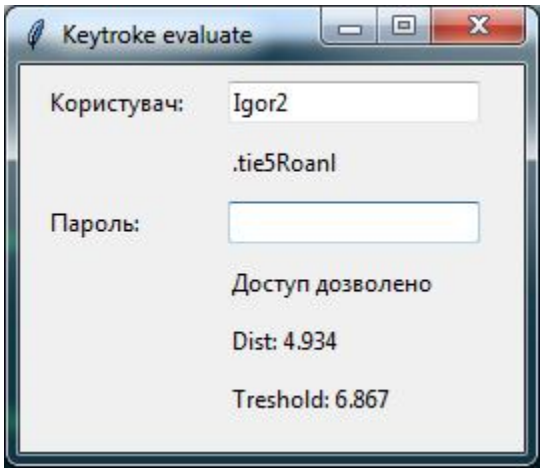


Адміністративна консоль з результатами навчання детектора:



(обчислений поріг 6.210, та максимальна відстань між вхідним вектором та вектором отриманим на виході навченої мережі 5.2947)

Підмодуль автентифікації в систему документообігу:



Видно, що на основі аналізу ритму клавіатурного введення, нейронна мережа обчислила відсань між оригінальним вектором даних клавіатурного ритму та вектором на виході з штучного нейронного детектора: 4.934. Дане значення є меншим за порогове Treshold: 6.867, отже, прийнято рішення про допуск у систему.

4.7 Висновки до розділу, порівняння результатів з аналогами

Система біометичної автентифікації порівнювалась з системами описаними в досліджені: Kevin S., Roy A.. Comparing Anomaly-Detection Algorithms for Keystroke Dynamics [18]. Рузультати порівняння подано в таблиці нижче.

Таблиця середніх значень Equal Error Rate:

	Детектор	Equal Error Rate	Стандартне відхилення
1.	**Manhattan (scaled)**	**0.096**	**0.069**
2.	**Nearest Neighbor (Mahalanobis)**	**0.100**	**0.064**
3.	**Outlier Count (z-score)**	**0.102**	**0.077**
4.	**Досліджувана система**	0.102	0.071
5.	SVM (one-class)	0.102	0.065
6.	Mahalanobis	0.110	0.065
7.	Mahalanobis (normed)	0.110	0.065
8.	Manhattan (filter)	0.136	0.083
9.	Manhattan	0.153	0.092
10.	Neural Network (auto-assoc)	0.161	0.080
11.	Euclidean	0.171	0.095
12.	Euclidean (normed)	0.215	0.119
13.	Fuzzy Logic	0.221	0.105
14.	k-Means	0.372	0.139
15.	Neural Network (standard)	0.828	0.148

Детектори подано у порядку з найбільш якісних до найменш якісних

Таблиця середніх значень zero-miss false-alarm rate:

	Детектор	Equal Error Rate	Стандартне відхилення
1.	**Nearest Neighbor (Mahalanobis)**	**0.468**	**0.272**
2.	**Mahalanobis**	**0.482**	**0.273**
3.	**Досліджувана система**	**0.483**	**0.271**
4.	Mahalanobis (normed)	**0.484**	**0.273**
5.	SVM (one-class)	**0.504**	**0.316**
6.	Manhattan (scaled)	0.601	0.337
7.	Manhattan (filter)	0.757	0.282
8.	Outlier Count (z-score)	0.782	0.306
9.	Manhattan	0.843	0.242
10.	Neural Network (auto-assoc)	0.859	0.220
11.	Euclidean	0.875	0.200
12.	Euclidean (normed)	0.911	0.148
13.	Fuzzy Logic	0.935	0.108
14.	k Means	0.989	0.040
15.	Neural Network (standard)	1.000	0.000

Як видно з таблиці **zero-miss false-alarm rate,** при роботі детектора у режимі 100 відсоткової детекції паролів введених зловмисником, доля хибних спрацювань (пароль введений оригінальним користувачем проте віхилений системою становить близько 50%. Це становить незручність для користувачів, змушуючи їх до повторного введення пароля. Тому досліджувана система потребує подальшого доопрацювання та вдосконалення. Есперимент показав, що збільшення кількості ознак (в даному випадку – довжини пароля) позитивно впливає на якість детектора. Тому подалші дослідження доцільно проводити у цьому напрямку.

ВИСНОВКИ

Проблема виявлення аномалій в даних знаходить застосування в багатьох областях в яких бажано встановити незвичні події в діяльності що генерує такі дані. Ядро всіх методів виявлення аномалій є створення імовірнісної, статистичної або алгоритмічної моделі, яка характеризує нормальну поведінку системи. Методи машинного навчання та датамайнінг пропонують ряд алгоритмів ефективного навчання параметрів цих моделей. Відхилення від цих моделей використовуються для визначення викидів.

Хороша обізнаність з галуззю з якої отримані вихідні дані часто має вирішальне значення для розробки простих та точних моделей, які не перенавчаються на тестових вибірках. Проблема виявлення аномалій стає особливо складною коли існують значні зв'язки між різними точками даних. В часових рядах та мережах даних «патерни» взаємовідносин поміж окремими точками (темпоральні чи структурні) відіграють ключову роль в визначенні викидів.

В проведеному дослідженні запропоновано методику використання Машини опорних векторів в задачах виявлення вторгнень в інформаційні системи. Запропонована модель дозволлила досягти нижчої в порівнянні з аналогами частки пропусків загрозливого трафіку ((1.12×10^{-5}). Проте доля хибних спрацювань залишається високою і може бути предметом подальших досліджень.

Запроектовано та виконано програмну реалізацію системи біометричної класифікації на основі аналізу аномалій клавіатурного ритму. Отримані результати еквіваленої частки помилок (рівної долі хибних спрацювань та хибних допусків в систему) - 0.102. Хоча даний результат є одним з найкращих серед систем які працюють з використанням інших моделей, її практичне застосування можливе лише в якості допоміжного засобу перевірки користувачів, при допуску до програмних систем чи онлайн

ресурсів, як наприклад, системи дистанційного навчання (перевірка самостійності виконаного завдання користувачем).

Аналіз аномалій засобами машинного навчання має великий простір подальших досліджень, особливо в області структурно-темпорального аналізу.

ЛІТЕРАТУРА

1. Aggarwal C.C., Charu C. Data Classification Algorithms and Applications. 2015: Chapman & Hall /CRC.

2. Aggarwal C.C., Charu C. Outlier Analysis. Springer, 2013.

3. Aggarwal C.C., Hinneburg A., Keim A. On the Surprising Behavior of Distance Metrics in High Dimensional Space // ICDT Conference. 2001.

4. Aggarwal C.C. On the E□ects of Dimensionality Reduction on High Dimensional Similarity Search // ACM PODS Conference. 2001.

5. Agyemang M., Barker K., Alhajj R. A Comprehensive Survey of Numeric and Symbolic Outlier Mining Techniques // Intelligent Data Analysis, No. 10(6), 2006. pp. 521–538.

6. Barnett V., Lewis T. Outliers in Statistical Data. Wiley, 1994.

7. Beckman R., Cook R. Outliers // Technometrics, No. 25(2), 1983. pp. 119–149.

8. Chandola V., Banerjee A., Kumar V. Anomaly Detection for Discrete Sequences: A Survey // IEEE Transactions on Knowledge and Data Engineering, No. 24(5), 2012. pp. 823–839.

9. Chandola V., Banerjee A., Kumar V. Anomaly Detection: A Survey // ACM Computing Surveys, No. 41(3), 2009.

10. Dimensionality reduction [Електронний ресурс] URL: https:// en.wikipedia.org/wiki/Dimensionality_reduction

11. Dunnung T., Friedman E. Practical Machine Learning: A New Look at Anomaly Detection. O'Reilly Media, 2004.

12. Elkan C., Noto K. Learning Classifiers from only Positive and Unlabeled Data // ACM KDD Conference. 2008.

13. EM-алгоритм [Електронний ресурс] URL: https://uk.wikipedia.org/wiki/EM-алгоритм

14. Gaines R., Lisowski W., Press S., Shapiro N. Authentication by keystroke timing: some prelimary results. // Technical Report R-2526-NSF, RAND Corporation, 1980.

15. Haider S., Abbas A., Zaidi A.K. A multi-technique approach for user identification through keystroke dynamics. // IEEE International Conference on Systems, Man and Cybenetics. 2000. pp. 1336–1341.

16. Hawkins D. Identification of Outliers. Chapman and Hall, 1980.

17. Hwang B., Cho S. Characteristics of auto-associative MLP as a novelty detector. // In Proceedings of the IEEE International Joint Conference on Neural Networks. Washington, DC. 10–16 July,1999. Vol. 5, pages 3086–3091.

18. Kevin S. Comparing Anomaly-Detection Algorithms for Keystroke Dynamics // IEEE, 2009.

19. Knorr E., Ng R. Algorithms for Mining Distance-based Outliers In Large Datasets // VLDB Conference. 1998.

20. Laurikkala J., Juholal M., Kentala E. Informal Identification of Outliers in Medical Data 2000.

21. Leggett J., Williams G. Verifying identity via keystroke characteristics., 1988.

22. Manevitz L. M. Y.M. Document Classification on Neural Networks Using Only Positive Examples // SIGIR. 2000.

23. Markou M., Singh S. Novelty detection: A Review, Part 2: Neural Network-based Approaches // Signal Processing, No. 83(12), 2003. pp. 2481–2497.

24. Markou M..S.S. Novelty detection: A Review, Part 1: Statistical Approaches // Signal Processing, No. 83(12), 2003. pp. 2481–2497.

25. Peacock A., Ke X., Wilkerson M. Typing patterns: A key to user identification // IEEE Security and Privacy, Vol. 2, no.5, pp.40–47, Sep. 2004.

26. Principal component analysis [Електронний ресурс] URL: https:// en.wikipedia.org/wiki/Principal_component_analysis

27. Ramaswamy S., Rastogi R., Shim K. E□cient Algorithms for Mining Outliers from Large Data Sets // ACM SIGMOD Conference. 2000. pp. 427–438.

28. Rousseeuw P., Leroy A. Robust Regression and Outlier Detection. Wiley, 2003.

29. Schölkopf B., Platt J.C. Estimating the support of a high-dimensional distribution // Neural Computation, No. 13(7). pp. 1443–1472.

30. Shelestov A., Skakun S., Kussul O. Complex neural network model of user behavior in distributed systems // International Conference «Knowledge-Dialogue-Solutions». 2007.

31. Song X., Wu M., Jermaine C., Ranka S. Conditional Anomaly Detection // IEEE Transaction on Knowledge and Data Engineering, No. 19(5), 2007. pp. 631–645.

32. Sun P., Chawla S. On Local Spatial Outliers // IEEE ICDM Conference. 2004.

33. Беляев А., Петренко С. Системы обнаружения аномалий: новые идеи в защите информации // Экспресс-Электроника, No. 2, 2004.

34. Большев А.К., Яновский В.В. Применение нейронных сетей для обнаружения вторжений в компьютерные сети // Вестник Санкт-Петербургского университета, No. Сер. 10., 2010.

35. Воронцов К.В. Лекции по методу опорных векторов [Електронний ресурс] URL: http://www.ccas.ru/voron/download/SVM.pdf

36. Гамаюнов Д.Ю. Обнаружение компьютерных атак на основе анализа поведения сетевых обьектов. М. 2007. Диссертация на соискание ученой степени кандидата физико-математических наук.

37. Загальна лінійна модель [Електронний ресурс] URL: https:// uk.wikipedia.org/wiki/Загальна лінійна модель

38. Клионский Д.М. Применение искусственных нейронных сетей в задачах обнаружения аномалий в поведении сложных динамических объектов // Нейрокомпьютеры: разработка, применение, No. 11, 2011.

39. Колодчак О.М. Сучасні методи виявлення аномалій в системах виявлення вторгнень // Lviv Polytechnic National University Institutional Repository http://ena.lp.edu.ua, 2012.

40. ЛЕВОНЕВСКИЙ Д.К., ФАТКИЕВА Р.Р. Разработка системы обнаружения аномалий сетевого трафика // Научный вестник НГТУ, Т. 56, № № 3, 2014. С. 108–114.

41. Лінійна регресія [Електронний ресурс] URL: https://uk.wikipedia.org/wiki/ Лінійна_регресія

42. Максименко Г.А. Метод обнаружения аномалий потоков данных в сетях // Економічна та інформаційна безпека, 2009.

43. Марковська модель [Електронний ресурс] URL: https://uk.wikipedia.org/ wiki/Марковська_модель

44. Обнаружение аномалий в данных: современный подход [Електронний ресурс] [2014]. URL: http://datareview.info/article/novyie-podhodyi-k-obnaruzheniyu-anomaliy/

45. Породжувальна модель [Електронний ресурс] URL: https:// uk.wikipedia.org/wiki/Породжувальна_модель

46. Рубан І.В., Мартовицький В.О., Партика С.О. Класифікація методів виявлення аномалій в інформаційних системах // Системи озброєння і військова техніка, No. 3(47), 2016.

47. Рудик І.І. Виявлення аномалій в комп'ютерній мережі на основі нейромережних технологій // Штучний інтелект, No. 2, 2012.

48. Статистический метод обнаружения аномалий в eBay [Електронний ресурс] [2015]. URL: https://habrahabr.ru/company/io/blog/265571/

49. Темпоральна логіка [Електронний ресурс] URL: https://uk.wikipedia.org/ wiki/Темпоральна_логіка

50. Эксперимент в Яндексе. Как идентифицировать взломщика с помощью машинного обучения [Електронний ресурс] [2014]. URL: https:// habrahabr.ru/company/yandex/blog/230583/