НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
ІМЕНІ ІГОРЯ СІКОРСЬКОГО»
Факультет прикладної математики
Кафедра прикладної математики

«До захисту допущено»
Завідувач кафедри
_____ Олег Чертов
«___» _____ 2023 р.

Дипломна робота

на здобуття ступеня бакалавра
за освітньо-професійною програмою «Наука про дані та математичне моделювання»
спеціальності 113 «Прикладна математика»
на тему: «Система розпізнавання повідомлень в соціальній мережі Twitter за принципом бот, троль чи людина»

Виконав:
студент IV курсу, групи КМ-91
Олійник Артем Олегович _____

Керівник:
Старший викладач, канд. техн. наук
Ліскін Вячеслав Олегович _____

Консультант з нормоконтролю:
Старший викладач
Мальчиков Володимир Вікторович _____

Рецензент:
Доцент каф. СПіСКС, канд. техн. наук, доцент
Тарасенко-Клятченко Оксана Володимирівна _____

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

Київ — 2023 року

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет прикладної математики
Кафедра прикладної математики

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 113 «Прикладна математика»

Освітньо-професійна програма «Наука про дані та математичне моделювання»

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Олег Чертов
«___» _____ 2023 р.

ЗАВДАННЯ
на дипломну роботу студенту

Олійнику Артему Олеговичу

1. Тема роботи: «Система розпізнавання повідомлень в соціальній мережі Twitter за принципом бот, троль чи людина»,
керівник роботи Ліскін В. О., старший викладач, канд. техн. наук,
затверджені наказом по університету від «31» травня 2023 р. № 2108-С.
2. Термін подання студентом роботи: «12» червня 2023 р.
3. Вихідні дані до роботи: розроблювана система повинна приймати дані проти “твіт” та його автора і класифікувати його за принципом бот, троль чи людина.
4. Зміст роботи: виконати аналіз існуючих методів розв’язання задачі, вибрати методи перетворення тексту у векторний формат та методи класифікації для багатьох класів, здійснити програмну реалізацію розробленої системи, провести тестування розробленої системи.
5. Перелік ілюстративного матеріалу: приклади роботи метрик, графічні зображення метрик та дії алгоритмів машинного навчання, знімки з результатами роботи програми.
6. Дата видачі завдання: «06» лютого 2023 р.

Календарний план

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітка
1	Огляд літератури за тематикою та збір даних	12.11.2022	
2	Проведення аналізу метрик для оцінки роботи системи для збільшення роздільної здатності зображень	14.12.2022	
3	Проведення аналізу математичних методів та існуючих систем для збільшення роздільної здатності зображень	24.12.2022	
4	Підготовка матеріалів першого розділу роботи	01.02.2023	
6	Підготовка матеріалів другого розділу роботи	15.03.2023	
7	Підготовка матеріалів третього розділу роботи	05.04.2023	
8	Розроблення програмного забезпечення для збільшення роздільної здатності зображень рентгенівських знімків	15.04.2023	
9	Підготовка матеріалів четвертого розділу роботи	03.05.2023	
10	Оформлення пояснювальної записки	01.06.2023	

Студент _____ Артем ОЛІЙНИК

Керівник роботи _____ Вячеслав ЛІСКІН

Анотація

Дипломну роботу виконано на 60 аркушах, вона містить 2 додатки та перелік посилань на використані джерела з 15 найменувань. У роботі наведено 11 рисунків та 1 таблицю.

Метою даної дипломної роботи є виявлення ботів та тролів у соціальній мережі Twitter для якісного спілкування та обговорення тез.

У роботі проведено аналіз існуючих рішень указаної задачі – алгоритмів машинного навчання таких як дерево рішень, випадковий ліс, KNN, наївний Баєсів класифікатор, SVM та згорткових мереж. Виконано їх порівняння базуючись на результатах, отриманих авторами у відповідних роботах. Для розв’язанні задачі в роботі обрано всі вище згадані алгоритми для їх порівняння та виявлення найкращого.

Розроблено, натреновано та протестовано систему, що вирішує обрану задачу.

Ключові слова: бот, троль, машинне навчання, дерево рішень, метрики, обробка даних, трансформер.

ABSTRACT

The thesis is presented in 60 pages. It contains 2 appendixes and bibliography of 15 references. 11 figures and 1 table are given in the thesis.

The aim of this diploma work is to identify bots and trolls on the social media platform Twitter for quality communication and discussion of ideas.

The work includes an analysis of existing solutions to the specified task, such as machine learning algorithms like decision trees, random forests, KNN, naive Bayes classifier, SVM, and convolutional neural networks. A comparison of these algorithms is performed based on the results obtained by the respective authors in their works. All of the mentioned algorithms were selected in this work for comparison and to identify the best one.

A system has been developed, trained, and tested to address the chosen task.

Keywords: bot, troll, machine learning, decision tree, metrics, data processing, transformer.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

DTC – Decision Tree Classifier.

IT (Information Technology) – інформаційні технології.

KNN – K Nearest Neighbor

NaN – Not a Number

SVM – Support Vector Machines

TF-IDF (Term Frequency and Inverse Document Frequency) – частота слова і зворотна частота документа.

UML (Unified Modeling Language) – уніфікована мова моделювання.

ШІ – штучний інтелект.

ВСТУП

За останні роки платформи соціальних мереж змінили спосіб спілкування, обміну інформацією та взаємодії з людьми, ставши невід'ємною частиною нашого повсякденного життя. Twitter вирізняється серед цих платформ як потужний інструмент для поширення новин у реальному часі, вивчення громадської думки та розвитку онлайн-спільнот. Однак Twitter став розсадником кіберзлочинців, таких як тролі та боти, які хочуть контролювати розмови, поширювати неправдиву інформацію та створювати конфлікти в онлайн-екосистемі. Це пояснюється його швидким розвитком і відкритістю.

Підтримка цілісності, надійності та користувацького досвіду Twitter стала критично залежною від виявлення та боротьби з існуванням тролів і ботів. Ці сутності здатні впливати на політичні погляди, поширювати мову ненависті та впливати на громадську думку, а також порушувати суспільний діалог. Тому дуже важливо створити ефективні методи та алгоритми, які зможуть точно виявляти та викривати сумнівні мережі, які працюють за цими обліковими записами.

Це дослідження має на меті дослідити тонкощі ідентифікації Twitter-тролів і ботів шляхом вивчення основних рис, дій і методів, які використовують ці онлайн-актори. Аналізуючи їхні методи, можні створити складні системи виявлення, які покращать здатність платформи запобігати неправдивій інформації та сприяти здоровим онлайн-стосункам.

1 ПОСТАНОВКА ЗАДАЧІ

Метою даної дипломної роботи є розробка математичного та програмного забезпечення для визначення автора повідомлення у соціальній мережі Twitter.

До множини авторів повідомлень, що розпізнаватимуться, віднесено наступні:

а) людина;

б) бот;

в) троль;

При розробленні відповідного забезпечення потрібно розв’язати наступні завдання:

а) проведення порівняльного аналізу існуючих методів класифікації машинного навчання та методів обробки тексту;

б) вибір та адаптація існуючих методів для вирішення задачі класифікації автора повідомлення та їх аналізу;

в) розробка програмного забезпечення на базі вибраних математичних методів;

г) тестування розробленої системи.

Реалізована система має задовольняти такі вимоги:

а) мати високі показники ефективності класифікації;

б) бути спроможною навчатися на різних вибірках.

2 ОГЛЯД ТА АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ КЛАСИФІКАЦІЙ ПОВІДОМЛЕНЬ

2.1 Вступ до огляду існуючих систем розпізнавання ботів та тролів у Twitter

В останній час проблема виявлення ботів та тролів у соціальних мережах стала надзвичайно актуальною. В останні роки активізувалися дослідження у цій сфері, а саме після американських виборів президента 2016 року.

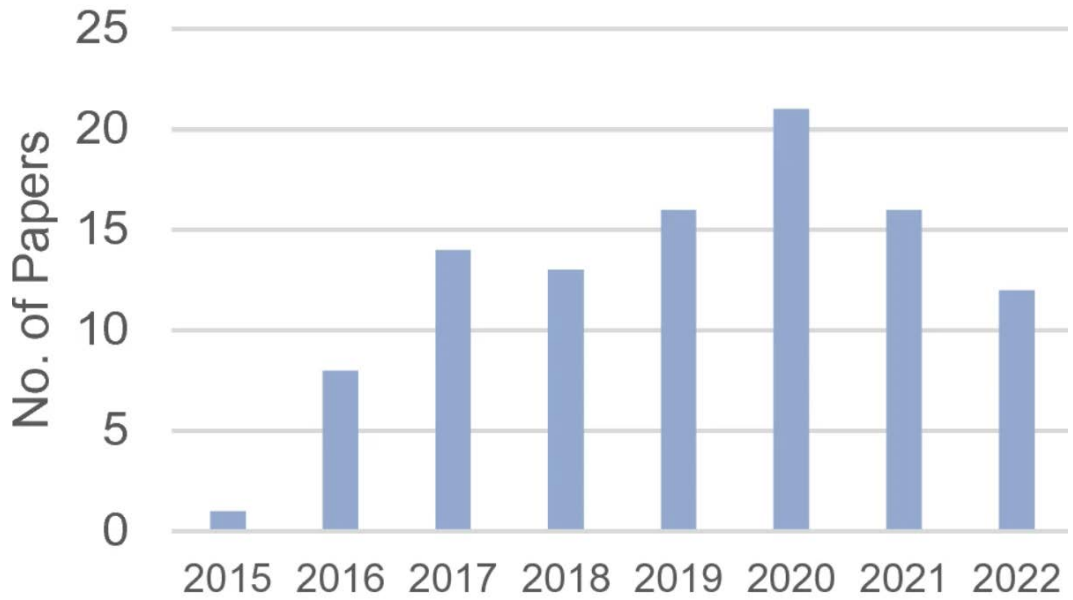


Рисунок 2.1 – Кількість досліджень класифікації ботів та тролів по роках[1]

Але дана проблема є гнучкою та змінюваною, боти та тролі стають кращі, тому існує потреба в постійному дослідженні нових методів вирішення цієї проблеми.

2.2 Огляд існуючих систем класифікації ботів та тролів

Система Bot Sentinel [2] є платформою, яка оцінює проблематику акаунту Twitter оцінюючи їх від “нормальних” до “проблематичних”. Кожному акаунту відноситься процентне значення в цьому рейтингу. Ця система дозволяє аналізувати весь акаунт та окремі повідомлення. Також в них є окрема специфікація під час виборів і вони напряму співпрацюють з журналістами для кращої роботи системи. Хоча ця система і використовує машинне навчання, не вдалось знайти оригінального коду для розуміння принципів роботи цієї система. Тим не менш, розробники стверджують, що їх система класифікує “проблематичність” акаунту з точністю 95%.

Система Hoaxy [3] це прекрасний інструмент візуалізації поширення інформації у Twitter і як поєднані між собою різні акаунти та повідомлення. Як і в системі згаданій вище, цей інструмент оцінює акаунти по шкалі, але в цьому випадку, від 1 (людина) до 5 (бот). Підхід цієї системи спеціалізується на темпоральному та мережевому аналізі. Варто зазначити, що ця система не визначає яка інформація є правдивою, вона визначає лише її поширення. В даний час цей сервіс недоступний через поступове закриття керівництвом Twitter інформації про акаунти.

Система Botometer [4] дозволяє перевіряти як сам акаунт, так і акаунти друзів та підписників. Цей інструмент, як і попередній, оцінює акаунт за шкалою від 1 до 5. Також ця система класифікує самих ботів на підгрупи, такі як спамери, накрутники підписників і так далі. В додаток, цей інструмент пропонує огляд акаунту, включаючи повідомлення за тиждень, відношення твітів до ре-твітів і більше. Також ця система дає доступ для багатьох наборів даних для інших дослідників.

2.3 Математичні методи векторизації тексту

2.3.1 Word2Vec

Word2Vec — це популярна техніка вбудовування слів, яка дозволяє перетворювати слова на великі вектори. Вона встановлює семантичні зв’язки між словами на основі того, як ці слова часто використовуються в великому тексті.

У цьому методі архітектура неглибокої нейронної мережі прямого зв’язку використовується для вивчення вбудовування слів.

Нейронні мережі, також відомі як штучні нейронні мережі, виконують різноманітні завдання за допомогою математичної моделі, яка нагадує біологічні нервові системи. Вона складається зі штучних нейронів, які зв’язані сигналами.

Зазвичай нейронна мережа складається з одного або кількох шарів нейронів. Вхідний шар — це перший шар, до якого подаються вхідні дані. Коли кожен нейрон у шарі отримує суму вагів вхідних сигналів, він застосовує активаційну функцію до цієї суми, а потім видає сигнал. Як вхідний сигнал цей вихід передається наступному шару нейронів. Після цього процес повторюється до того, як нейрони дійдуть до вихідного шару.

Ваги, які визначають силу зв’язку між нейронами, є важливою частиною нейронної мережі. Мережеві ваги змінюються під час навчання, щоб зменшити помилки між вихідними результатами мережі та бажаними вихідними значеннями. Цей процес називається зворотнім поширенням помилки.

Неглибока нейронна мережа прямого зв’язку не має багатошарової структури. Вона складається з одного шару нейронів, який повністю підключений до вхідних даних і вихідних результатів.

Кожен нейрон вхідного шару пов’язаний з кожним нейроном вихідного шару в неглибоких нейронних мережах прямого зв’язку. Вихідний результат може бути змінений кожним зв’язком (вагою) між нейронами. Причина полягає в тому, що в процесі навчання мережа адаптується до конкретного завдання.

Коли вхідні дані мають просту структуру і не потребують складного аналізу або паттернів, ці нейронні мережі можуть бути корисними для простих завдань класифікації або регресії. Порівняно з більш складними нейронними мережами глибокого навчання, вони навчаються та застосовують швидко.

Однак неглибокі нейронні мережі прямого зв’язку нездатні виконувати складні завдання, які вимагають виявлення складних залежностей у вхідних даних.

Коли ви вивчите модель Word2Vec, ви можете створити вбудовування слів для власних текстових даних. Усі вкладення слів є щільними векторами певної довжини, з кожним виміром, який характеризує слово. Після цього ці вектори можна використовувати як вхідні функції для виконання різноманітних завдань машинного навчання, таких як пошук інформації, аналіз настроїв і класифікація тексту.

2.3.2 TF-IDF

Метод векторизації тексту, відомий як TF-IDF (Term Frequency-Inverse Document Frequency), використовується для оцінки важливості слова в документі порівняно з іншими словами в корпусі тексту.

Як часто певне слово з’являється в документі, визначається TF (частота терміну). Що більше його зустрічається, тим більше значення TF.(2.1)

$$TF = \frac{n_i}{\sum_k n_k} \tag{2.1}$$

Термін «інверсний документний частотний зважувач» (IDF) використовується для визначення того, наскільки важливим є слово у всьому корпусі документів. Це значення обернено пропорційне частоті використання слова в корпусі. Слова, які часто використовуються, мають більше значення для IDF.

$$IDF = log\frac{|D|}{|(d_i \supset t_i|} \quad (2.2)$$

Значення TF-IDF можна обчислити шляхом множення значень TF та IDF для кожного слова, що міститься в документі. З цього випливає числовий вектор, який відображає текстовий документ. Багато алгоритмів машинного навчання можуть використовувати цей вектор як вхідну ознаку.

ТФ-IDF дозволяє виділити слова, які мають важливе значення в одному документі, але рідко зустрічаються в інших. Це дозволяє враховувати особливості документа в контексті тексту в цілому.

Для різноманітних завдань обробки тексту, таких як класифікація текстів, кластеризація документів і пошук інформації, цей метод широко використовується.

2.3.3 Bag-of-Words (BoW)

Метод «мішок слів» (BoW) — це техніка векторизації тексту, яка дозволяє представляти документ у вигляді набору слів, незважаючи на їхній порядок і структуру.

Основна концепція «мішка слів» полягає в тому, що ми створюємо словник унікальних слів, які зустрічаються в усьому корпусі тексту. Після цього кожен документ перетворюється на числовий вектор, де кожен елемент вектора показує кількість або присутність слова зі словника в документі.

Токенізація: розбиваємо текст на окремі слова або інші токени, такі як розділові знаки, числа тощо.

Побудова словника: ми створюємо словник із унікальними словами, які зустрічаються в тексті.

Кодування документів: Кожен документ представлений у вигляді вектора, де кожен елемент відповідає слову зі словника. Елемент вектора, який містить слово в

документі, має значення 1. Крім того, він може показувати кількість слов, які містяться в документі.

Нормалізація: Іноді вектори «мішка слів» можна нормалізувати шляхом ділення значення кожного вектора на загальну кількість слів у документі. Це робить можливим зменшення впливу довжини документа на розподіл значень.

У методі «мішка слів» текстові дані можна перетворити на числові вектори, які можна використовувати для навчання моделей машинного навчання. Вектори «мішка слів» використовуються для аналізу настрою, класифікації текстів і аналізу споживчих відгуків.

2.3.4 Transformers

Transformers є потужною архітектурою моделей машинного навчання, яка змінила підхід до обробки послідовностей, зокрема текстових даних. Вони були представлені у роботі "Attention Is All You Need"[5] в 2017 році і стали основою для багатьох сучасних розширень і варіацій.

Основною особливістю transformers є механізм уваги (self-attention), який дозволяє моделі враховувати важливість кожного слова у контексті всього речення або документа. Замість того, щоб пропускати дані послідовності через рекурентні шари, як це робиться у рекурентних нейронних мережах (RNN), transformers можуть враховувати залежності між всіма словами у вхідному тексті одночасно.

Основні компоненти transformers включають:

– Encoder: Використовується для перетворення вхідного тексту на складні представлення, які враховують контекстуальні залежності між словами. Кожне слово в тексті представляється вектором, який може змінюватися залежно від інших слов у тексті.

– Decoder: Використовується для генерації вихідного тексту або послідовності на основі закодованого представлення. Це може бути використано, наприклад, для машинного перекладу або генерації тексту.

– Механізми уваги: Модель transformers використовує механізми уваги для врахування важливих залежностей між словами у тексті. Це дозволяє моделі зосереджуватися на важливих частинах вхідного тексту при генерації представлень або генерації тексту.

– Маскуючі шари: Маскуючі шари використовуються для виключення майбутніх інформаційних залежностей у процесі навчання та генерації тексту.

Transformers стали основою для багатьох відомих моделей, таких як BERT (Bidirectional Encoder Representations from Transformers), GPT (Generative Pre-trained Transformer), T5 (Text-to-Text Transfer Transformer) та багатьох інших. Вони домінують у багатьох завданнях обробки природної мови, таких як машинний переклад, класифікація текстів, побудова чат-ботів та інші. Transformers значно покращили результати в багатьох текстових завданнях і продовжують бути активно використовуваними у сфері машинного навчання.

2.4 Існуючі методи класифікації

2.4.1 Логістична регресія

Логістична регресія є статистичним алгоритмом, який прогнозує ймовірність належності зразка до одного з класів. Зазвичай цей алгоритм використовується для роботи з двома класами. Цей метод використовує логістичну функцію (сигмоїдну функцію) для підрахунку ймовірності того, що зразок належить до того чи іншого класу.

Математично, модель логістичної регресії можна представити наступним чином:

$$p(y = 1|X) = \sigma(\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_n X_n), \quad (2.3)$$

де $p(y = 1|X)$ – ймовірність того, що зразок X належить до позитивного класу,

$X_1, X_2, \ldots, X_n$ – ознаки зразка X,

$\beta_0, \beta_1, \beta_2, \ldots, \beta_n$ – параметри моделі,

σ - логістична (сигмоїдна) функція, яка обмежує значення від 0 до 1.

Оптимізація параметрів моделі відбувається шляхом максимізації функції правдоподібності або мінімізації функції втрати. Це може бути досягнуто за допомогою різних методів оптимізації, таких як градієнтний спуск.

Логістична регресія може бути розширена на випадки багатокласової класифікації, наприклад, за допомогою one-vs-rest або softmax підходів, а також на задачі регресії, застосовуючи лінійну функцію активації замість сигмоїдної функції.

Логістична регресія вважається швидким та ефективним алгоритмом, також вона має просту математичну базу.

2.4.2 Дерево рішень

Дерево рішень (англ. decision tree) є моделлю прийняття рішень. Воно візуалізує прийняття рішень у вигляді дерева, де кожен вузол представляє певну умову або ознаку, а кожне гілка відповідає можливому результату рішення.

Основна ідея дерева рішень полягає у розбитті набору даних на більші і менші підгрупи, шляхом використання різних розбивок і умов на ознаки даних. Цей процес продовжується рекурсивно до досягнення кінцевих рішень або досягнення заданої зупинки.

Для розрахунку дерев рішень використовуються наступні формули:

Нев’язка Джині (Gini Impurity): Для визначення неоднорідності вузлів дерева використовується нев'язка Джині. Формула нев'язки Джині для вузла з к класами виглядає наступним чином:

$$Gini = 1 - \sum_i p_i^2, \tag{2.4}$$

де p_i – це ймовірність наявності класі в даному вузлі.

Інформаційний приріст (Information Gain): Інформаційний приріст вимірює зменшення ентропії (або нев'язки Джині) після розбиття вузла. Вибір оптимального розбиття здійснюється шляхом максимізації інформаційного приросту. Формула для інформаційного приросту виглядає так:

$$IG = E(p) - \sum_i \frac{N_i}{N} * E(c_i), \tag{2.5}$$

де $E(p)$ – ентропія батьківського вузла,

N_i – кількість зразків у дитячому вузлі i,

N – загальна кількість зразків в батьківському вузлі,

$E(c_i)$ – ентропія дитячого вузла i

2.4.3 Випадковий ліс

Випадковий ліс (англ. Random Forest) є ансамблевим методом машинного навчання, який поєднує декілька дерев рішень для вирішення задач класифікації або регресії. Кожне дерево у випадковому лісі навчається на випадковій підвибірці даних та випадкових підвибірках ознак (вхідних змінних), що робить його більш робастним і зменшує схильність до перенавчання.

Основна ідея випадкового лісу полягає у створенні індивідуальних дерев рішень та комбінації їх прогнозів для отримання кінцевого результату. При класифікації, випадковий ліс прогнозує клас, що набуває більшості голосів серед індивідуальних дерев. При регресії, випадковий ліс обчислює середнє або медіанне значення прогнозів дерев.

Процес побудови випадкового лісу включає наступні кроки:

– Бутстреп-семплування: З вихідного набору даних випадково вибирається підвибірка з повторенням. Це створює випадковий піднабір, який буде використовуватись для навчання кожного дерева.

– Побудова дерев: Для кожної підвибірки даних будується окреме дерево рішень з використанням певного критерію розбиття, такого як нев'язка Джині або ентропія. Кожне дерево навчається незалежно і може мати обмежену глибину, щоб запобігти перенавчанню.

– Комбінація прогнозів: При класифікації, випадковий ліс використовує голосування більшості для визначення класу. При регресії, випадковий ліс використовує середнє або медіанне значення прогнозів дерев для визначення кінцевого прогнозу.

$$a(x) = \frac{1}{N}\sum_{i=1}^{N} b_i(x) \tag{2.6}$$

Випадкові ліси є потужними моделями машинного навчання, які виявляються ефективними для багатьох завдань, особливо коли є багато ознак та шумних даних. Вони володіють доброю універсальністю, стійкістю до перенавчання та здатністю оцінювати важливість ознак.

2.4.4 SVM

Метод опорних векторів (SVM - Support Vector Machines) є одним з алгоритмів машинного навчання, який використовується для класифікації та регресії. Він працює шляхом побудови границі прийняття рішень, яка розділяє точки даних у просторі класифікації.

Основна ідея SVM полягає в знаходженні оптимальної гіперплощини, яка максимізує відстань (звану "зазором") між класами даних. Цей зазор визначається опорними векторами - точками, що лежать найближче до границі прийняття рішень.

Гіперплощина є границею, яка розділяє класи у просторі признаків. У двовимірному просторі це може бути проста лінія, а в більш високих вимірах це є гіперплощина.

Формула гіперплощини для бінарної класифікації:

$$w^T * x + b = 0, \tag{2.7}$$

де w - вектор нормалі гіперплощини, x - вхідний вектор признаків, b - зміщення (bias).

Відстань між точкою і гіперплощиною може бути обчислена як модуль проекції вхідного вектора на нормаль гіперплощини:

$$d = \frac{|w^T * x + b|}{||w||}, \tag{2.8}$$

де d – відстань,

w – вектор нормалі гіперплощини,

x – вхідний вектор признаків,

b – зміщення (bias),

$||w||$ – норма вектора w.

Основні поняття, пов'язані з SVM:

– Ядро (Kernel): SVM може використовувати ядра, щоб зробити модель більш гнучкою та здатною розрізняти нелинійні залежності в даних. Ядро змінює просторову форму границі прийняття рішень і дозволяє SVM працювати з необхідними нелинійними залежностями між класами.

– Регуляризація: SVM використовує параметр регуляризації (C), який контролює компроміс між максимізацією зазору та мінімізацією помилок класифікації. Малі значення C призводять до більш широких зазорів, що може призвести до більшої некоректної класифікації, а великі значення C зменшують зазор, що може привести до перенавчання.

– Підтримувальні вектори (Support Vectors): Це точки даних, що лежать найближче до границі прийняття рішень. Вони грають важливу роль у визначенні границі та класифікації.

– Функція рішення: Функція рішення SVM використовується для класифікації нових зразків. Вона використовує ваги опорних векторів та відстань до границі для визначення класу зразка.

$$f(x) = sign(w^T * x + b), \tag{2.9}$$

де $f(x)$ показує класифікований клас, x - вхідний вектор признаків, w і b - параметри моделі.

– М'які межі (Soft Margin): У випадках, коли дані не є повністю лінійно роздільними, можна використовувати м'які межі, які дозволяють деяким точкам даних потрапити в зазор або навіть на неправильну сторону границі. Це допомагає збалансувати ризик перенавчання та недосвідченості.

– Множник Лагранжа (Lagrange Multiplier): Метод SVM використовує оптимізаційну задачу з множником Лагранжа для знаходження оптимальної границі

прийняття рішень. Ця задача полягає в максимізації зазору та мінімізації помилок класифікації.

2.4.5 KNN

Метод k-найближчих сусідів (KNN – k-Nearest Neighbors) є одним з простих і популярних алгоритмів машинного навчання для класифікації та регресії. Він використовується для прогнозування класу або значення цільової змінної на основі найближчих зразків даних з навчального набору.

Основна ідея KNN полягає в тому, щоб призначити новому зразку клас або значення на основі ближніх зразків з відомими класами або значеннями. Коли ми хочемо класифікувати новий зразок, алгоритм KNN знаходить k найближчих зразків з навчального набору та вирішує класифікацію з урахуванням голосування більшості або зваженого голосування.

– Основні поняття, пов'язані з KNN:

– Відстань: Для визначення найближчих сусідів, KNN використовує метрику відстані, таку як Евклідова відстань, Манхеттенська відстань або Косинусна відстань. Це допомагає визначити, наскільки далеко або близько знаходяться зразки один від одного.

– Кількість сусідів (k): Параметр k визначає кількість найближчих сусідів, які використовуються для класифікації або регресії нового зразка. Вибір оптимального значення k залежить від характеристик даних та задачі.

– Голосування більшості: Для класифікації нового зразка, KNN використовує голосування більшості серед k найближчих сусідів. Клас, який набрав більшість голосів, визначається як класифікаційний результат.

– Зважене голосування: У деяких випадках, KNN використовує зважене голосування, де ваги надаються кожному зразку залежно від відстані до нового

зразка. Це означає, що ближчі зразки мають більший вплив на класифікацію або регресію.

– Параметризація: KNN може мати інші параметри, такі як метрика відстані, вагові функції або нормалізація даних, які можуть впливати на його результати. Вибір правильних параметрів може бути важливим для досягнення оптимальної продуктивності моделі.

2.4.6 Наївний Баєсів класифікатор

Наївний баєсів класифікатор Naive Bayes - це статистичний алгоритм машинного навчання, який використовується для класифікації даних. Він базується на теоремі Байєса та припущенні про наївність, згідно з якою всі ознаки вважаються окремими.

Алгоритм працює таким чином: спочатку він аналізує набір тренувальних даних, щоб побудувати модель. Naive Bayes обчислює ймовірність того, що певна точка даних належить до кожного з можливих класів, використовуючи статистику.

Алгоритм використовує ймовірності, отримані раніше, щоб визначити ймовірність належності даних до кожного класу під час класифікації нових даних. Вибирається клас, який має найвищу ймовірність.

Naive Bayes є швидким алгоритмом класифікації та простий у використанні. Використовуйте його часто для класифікації текстів, наприклад, для фільтрації спаму в електронній пошті або для визначення теми тексту.

Спрощене рівняння для класифікації виглядає так:

$$P(\text{Клас A}|\text{Ознака1}, \text{Ознака2}) = \frac{P(\text{Ознака1}|\text{КласA}) * P(\text{Ознака2}|\text{КласA}) * P(\text{КласA})}{P(\text{Ознака1}) * P(\text{Ознака2})} \quad (2.10)$$

На основі ознак 1 і 2, рівняння знаходить ймовірність класу A. Відповідно, дані класу A, ймовірно, містять ознаки 1 і 2.

2.4.7 XGBoost

XGBoost (eXtreme Gradient Boosting) - це потужна бібліотека для машинного навчання, яка базується на алгоритмі градієнтного бустингу дерев рішень. Вона знайшла широке застосування в багатьох задачах, включаючи класифікацію, регресію і ранжування.

Основна ідея XGBoost полягає в побудові ансамблю дерев рішень, де кожне дерево "виправляє" помилки попередніх дерев і додається до сумарного прогнозу. Це досягається шляхом мінімізації функції втрати з додатковою регуляризацією для запобігання перенавчанню.

Основні складові XGBoost включають:

– Градієнтний спуск: XGBoost використовує градієнтний спуск для оптимізації функції втрати. Він обчислює градієнт функції втрати по відношенню до прогнозованої величини і використовує цей градієнт для оновлення параметрів моделі.

– Дерева рішень: XGBoost будує дерева рішень шляхом розбиття набору даних на кожному кроці градієнтного бустингу. Воно використовує критерії інформаційної вигоди, які допомагають вибрати оптимальне розбиття.

– Регуляризація: XGBoost використовує різні методи регуляризації, такі як L1- та L2-регуляризація, для управління складністю моделі та запобігання перенавчанню.

– Функція втрати: XGBoost підтримує різні функції втрати, такі як логарифмічна втрата для класифікації та середньоквадратична втрата для регресії.

Ці функції втрати використовуються для оцінки якості прогнозів і налаштування параметрів моделі.

XGBoost є одним з найпопулярніших алгоритмів для багатокласової класифікації та регресії завдяки своїй високій продуктивності та здатності працювати з великими обсягами даних. Вона підтримує багато розширень і має документацію, яка допомагає використовувати її в різних сценаріях машинного навчання.

2.4.8 LSTM

LSTM (Long Short-Term Memory) - це тип рекурентної нейронної мережі (RNN), який використовується для обробки та аналізу послідовних даних, таких як текст, звук, часові ряди тощо. Він має здатність враховувати залежності в довгостроковій і короткостроковій пам'яті, що робить його особливо корисним при роботі з довгими послідовностями даних.

Основною відмінністю LSTM від звичайних RNN є використання спеціальних блоків пам'яті, які називаються «клітинами пам'яті». Кожна клітина пам'яті має внутрішні стани, які можуть зберігати інформацію протягом тривалого часу, і механізми, які дозволяють контролювати, коли і як ця інформація використовується.

Основні компоненти LSTM включають:

– Ворота: LSTM має три типи воріт – ворота забуття, ворота входу та вихідні ворота. Ворота допомагають регулювати потік інформації, що проходить через клітину пам'яті. Вони визначають, яку інформацію слід зберігати, яку інформацію треба забути та яку інформацію вивести.

– Клітини пам'яті: Клітини пам'яті є центральною частиною LSTM. Вони зберігають інформацію протягом тривалого часу та контролюють її потік.

– Функції активації: LSTM використовує різні функції активації, такі як сигмоїдна функція і гіперболічний тангенс, для регулювання значень воріт і контролю потоку інформації.

LSTM здатний ефективно моделювати довгострокові залежності в послідовних даних, що робить його особливо корисним для завдань, таких як машинний переклад, розпізнавання мови, генерація тексту та аналіз часових рядів. Він став популярним інструментом у сфері обробки природної мови та машинного навчання загалом.

2.5 Висновки до розділу

В цьому розділі були оглянуті існуючі системи класифікації автора повідомлення у Twitter. Так як ці системи не вказують математичні методи та моделі систем, їх наробітки не можуть бути використані у цій роботі. Також були розглянуті основні методи векторизації тексту для використовування їх в моделях, а саме було обрано трансформер, так як це є одним з найсучасніших та найрезультативніших методів обробки тексту. Крім цього було оглянуто велику кількість математичних методів, які включають в себе алгоритми машинного навчання та нейронні мережі. Було вирішено використати деяку кількість цих методів для їх порівняння, аналізу та виявлення найкращого для поставленої задачі.

3 МОДЕЛЬ СИСТЕМИ КЛАСИФІКАЦІЇ ПОВІДОМЛЕННЯ ЗА АВТОРОМ

3.1 Компонентна модель системи класифікації повідомлення за автором

Представимо структуру системи класифікації повідомлення за автором у вигляді діаграми компонентів (рисунок 3.1). Компоненти діаграми формуються за функціональною ознакою. Відношення між компонентами відображається через реалізацію відповідних інтерфейсів.

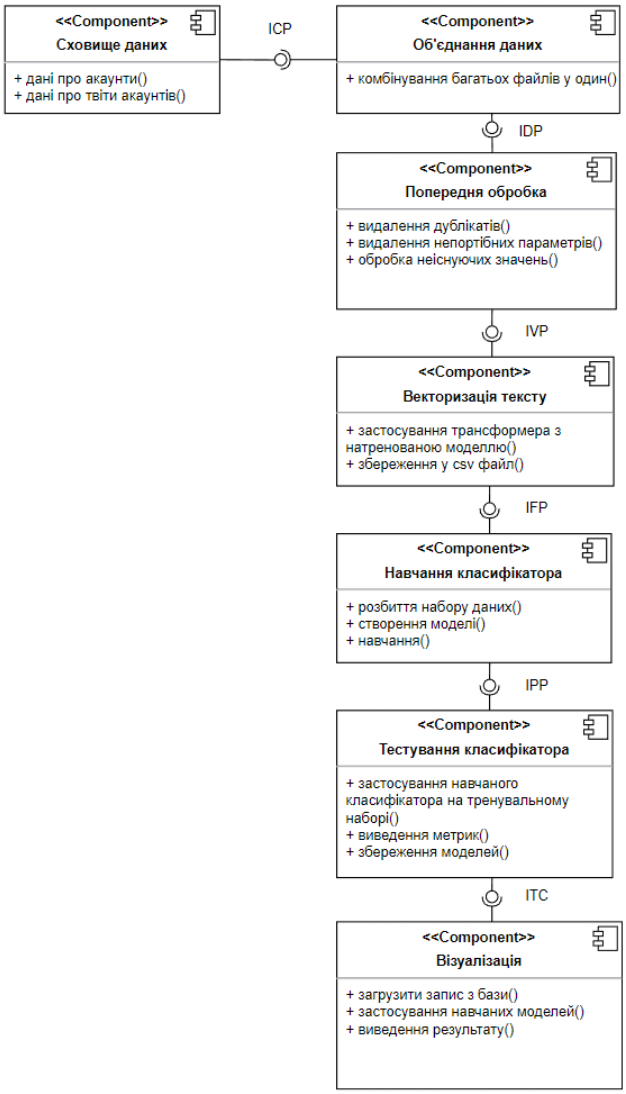


Рисунок 3.1 – Модель системи класифікації повідомлення за автором. Діаграма компонентів у нотації UML

Компонент «Сховище даних» – це сховище, у якому зберігаються набори даних, потрібні для розв’язання задачі класифікації автора повідомлення.

Набір даних cresci-2017 [6] містить в собі, розподілені між класами бот, троль та людина, набори інформації про акаунти та про твіти. Через це створюється потреба їх об’єднати для зручної роботи.

Цей датасет був отриманий на сервісі Botometer згаданому раніше. Саме цей набір даних був обраний через присутність всіх класів для розв’язання обраної задачі класифікації.

Компонент «Об’єднання даних» призначений для об’єднання всіх файлів набору даних cresci-2017 [6] в єдиний файл для подальшої роботи. Загальна кількість записів в отриманому файлі складає майже 10 мільйонів.

Компонент «Попередня обробка» призначений для видалення непотрібної, зайвої для розв’язання задачі класифікації, інформації з даних. У ньому виконуються наступні кроки:

- видалення нульових значень та дублікатів;
- видалення непотрібних параметрів;
- скорочення набору даних та балансування;
- заміна NaN у категоріальних параметрах на 0.

Після видалення дублікатів записів та нульових значень у наборі даних з залишилось близько 7 мільйонів записів, а після випадкового скорочення – близько 360 тис. записів.

Компонент «Векторизація тексту» призначений для переведення текстового параметри набору даних у числовий, для цього був використаний трансформер.

Попередня обробка даних (preprocessing of data) складається з наступних етапів:

- токенізація: текст розбивається на окремі слова – токени;
- видалення стоп-слів: стоп-слова – слова, які не містять лексично важливої інформації (в англійській мові це «a», «the», «this», «is» тощо);
- стемінг: обрізка слова до кореня;

– видалення порожніх рядків: після проведеної вищеописаної процедури обробки в тексті виявились порожні рядки, які мають бути видалені.

Всі ці дії в цьому рішенні виконує бібліотека sentence-transformers [7], а саме клас SentenceTransformer, який в свою чергу використовує вже натреновану модель all-MiniLM-L6-v2, яка відображає речення та абзаци у 384-вимірний щільний векторний простір і може використовуватися для таких завдань, як кластеризація або семантичний пошук.

Компонент «Навчання класифікатора» тренування моделей класифікації розпізнавання повідомлень за автором. Для цього були обрані конкретні алгоритми машинного навчання (дерева прийняття рішень, випадкового лісу, XGBoost, наївного Байєсового класифікатора, SVM).

За допомогою методів бібліотеки Scikit Learn [8] створюються екземпляри моделей класифікації. Набір даних розбивається на тренувальну та тестову вибірки у співвідношенні 3:1. Проводиться навчання моделей на тренувальній вибірці.

Побудова моделей класифікації розпочинається з налаштування гіперпараметрів за допомогою методу GridSearchCV(). Він приймає діапазон значень гіперпараметрів, що налаштовуються, та за допомогою крос-валідації обирає кращу комбінацію гіперпараметрів.

Після цього ініціалізуються та тренуються обрані алгоритми.

Компонент «Тестування класифікатора» призначений для перевірки роботи збережених моделей на тестовій вибірці для порівняльного аналізу та збереження найкращих моделей.

Обчислюються метрики якості класифікації: accuracy, precision, recall, F1-score, Matthew's correlation coefficient. Також для деяких методів були виведені найважливіші параметри при класифікації та матриці невідповідностей.

Зберігаються натреновані моделі.

Компонент «Виведення результатів» відображає результати робот моделей при класифікації окремого прикладу, порівнюючи який клас обрали моделі.

– інтерфейс ICP (Interface Combining Processing) – інтерфейс передання завантажених в оперативну пам’ять сирих наборів даних на вхід процесу об’єднання;

– інтерфейс IDP (Interface Data Processing) – інтерфейс передання завантажених в оперативну пам’ять об’єднаного набору даних на вхід процесу попередньої обробки;

– інтерфейс IVP (Interface Vectorization Processing) – інтерфейс передання вихідних (із компонента попередньої обробки) опрацьованих, очищених даних на вхід процесу обробки текстового параметру;

– інтерфейс IFP (Interface Fitting Process) – інтерфейс передання вихідних (із компонента обробки текстового параметру) опрацьованих даних на вхід процесу ініціалізації моделей;

– інтерфейс IPP (Interface Predecting Process) – інтерфейс передання вихідних (із компонента ініціалізації моделей) даних на вхід процесу збереження кращих моделей;

– інтерфейс ITC (Interface Tweet Classification) – інтерфейс передання кращих моделей на вхід процесу перевірки роботи моделей.

3.2 Висновки до розділу

У даному розділі було побудовано і описано модель системи класифікації повідомлення за автором у вигляді діаграми компонентів. Це діаграма структурного представлення системи, компоненти якої формуються за функціональною ознакою. Відношення між компонентами відображається через реалізацію відповідних інтерфейсів. Описано призначення та зміст компонентів.

4 МАТЕМАТИЧНЕ ЗАБЕЗПЕЧЕННЯ

4.1 Обробка текстового параметру

Для вирішення поставленої задачі один з найважливіших кроків - це перетворити текстові дані у формат зручний для зчитання моделлю, для цього текст буде перетворений у числові дані, а саме вектор, який буде враховувати лексичні зв’язки та довжину тексту. Для цього у вирішенні даної задачі використаний трансформер.

Трансформери та система, відома як архітектура «послідовність-послідовність», обговорюються в статті «Увага – це все, що вам потрібно». Нейронна мережа, що називається Sequence-to-Sequence (також відома як Seq2Seq), перетворює одну послідовність елементів, наприклад, слів у фразі, в іншу послідовність.

Кодер і декодер є основними компонентами моделей Seq2Seq. Кодер перетворює вхідну послідовність у n-вимірний вектор у просторі більшої розмірності. Декодер отримує цей абстрактний вектор і перетворює його у вихідну послідовність. Вихідна послідовність може складатися з символів, іншої мови, дублікату вхідних даних тощо.

Ще один потрібний термін – це увага(attention).

Механізм уваги досліджує вхідну послідовність і визначає, які додаткові елементи послідовності є важливими на кожному етапі. Це звучить абстрактно, але це можна пояснити за допомогою простої ілюстрації: Читаючи цей матеріал, увага завжди зосереджена на окремих словах, які бачить читач, але пам'ять також зберігає основні фрази, які допомагають створити картину.

Також цей механізм розглядає багато входів одночасно для кожного входу, який зчитує LSTM (кодер), і визначає, які входи є важливими, присвоюючи їм різні ваги. Закодоване повідомлення і ваги, отримані механізмом уваги, згодом будуть введені в декодер.

Transformer використовує раніше згаданий механізм уваги. Transformer – це архітектура для перетворення однієї послідовності в іншу за допомогою двох компонентів (кодера і декодера), схожа на LSTM, але відрізняється від раніше описаних/існуючих моделей «послідовність-послідовність», оскільки виключає будь-які рекурентні мережі (GRU, LSTM тощо).

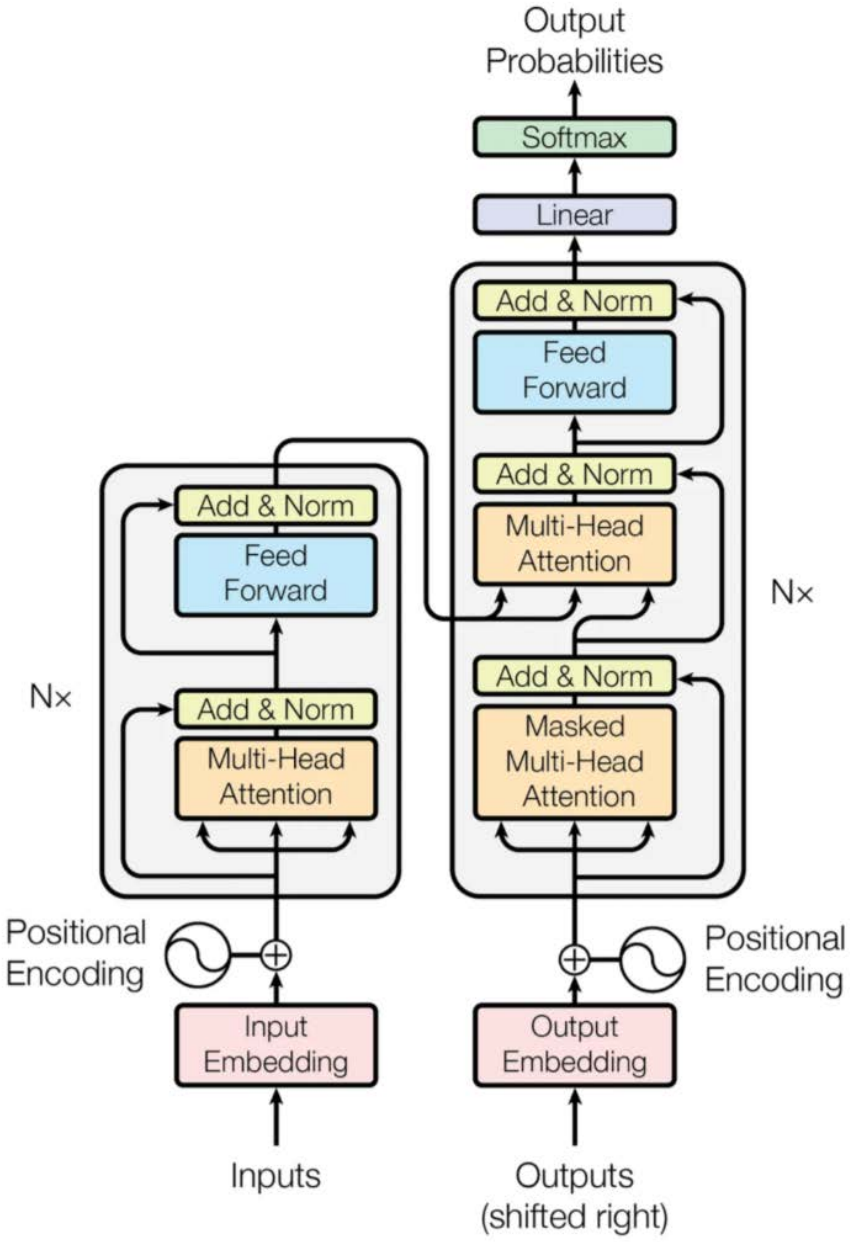


Рисунок 4.1 – Модель архітектури transformera [7]

Ліворуч - кодер, а праворуч - декодер. Відповідно до Nx на зображенні, кодер і декодер складаються з модулів, які можуть бути накладені один на одного кілька разів. Можна побачити, що шари Multi-Head Attention та Feed Forward складають більшість модулів. Оскільки не можна використовувати рядки безпосередньо, входи і виходи (цільові речення) спочатку вбудовуються в n-вимірний простір.

Позиційне кодування різних слів є невеликим, але важливим компонентом моделі. Оскільки послідовність залежить від порядку її складових, потрібно якось призначити кожному слову або компоненту в нашій послідовності відносну позицію, оскільки відсутні рекурентні мережі, які могли б згадати, як послідовності подаються в модель. Вбудоване представлення (n-вимірний вектор) кожного слова розширюється, щоб включити ці місця.

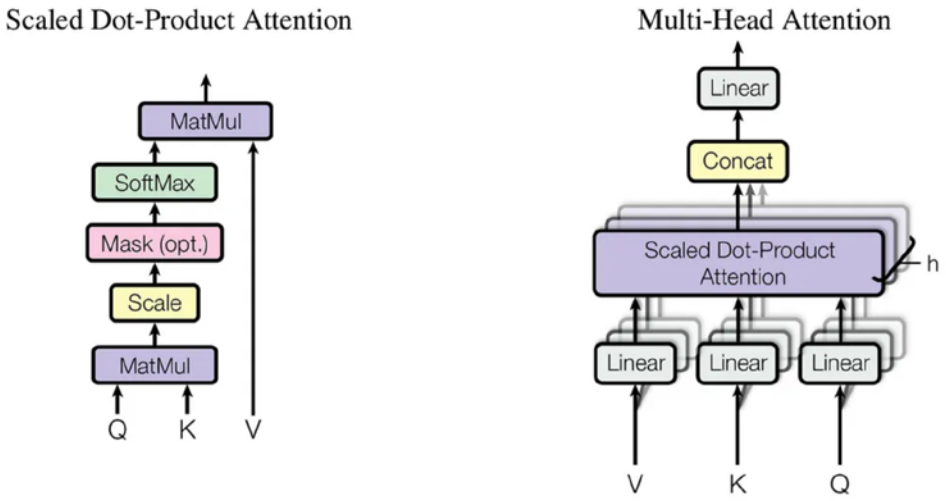


Рис. 4.2 – Багатоголові модулі уваги (Multi-Head Attention) [7]

Для подальших формул введемо формулу softmax:

$$softmax(z)_i = \frac{e^{z_i}}{\sum_{k=1}^{K} e^{z_k}} \tag{4.1}$$

Отже, введемо поняття уваги(attention):

$$Attention(Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V, \quad (4.2)$$

Де Q – матриця, яка містить запит (векторне представлення одного слова в послідовності),

K – всі ключі (векторні представлення всіх слів у послідовності),

V – це значення, які знову ж таки є векторними представленнями всіх слів у послідовності.

Послідовність слів у V ідентична послідовності слів у Q для кодера і декодера, двох багатоголових модулів уваги. Однак V відрізняється від послідовності, представленої Q для модуля уваги, який враховує послідовності кодера та декодера.

Можна сказати, що значення в V перемножуються і додаються з деякими вагами уваги a, де наші ваги визначаються:

$$a = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right), \quad (4.3)$$

Вплив кожного слова в послідовності (позначається Q) на всі інші слова в послідовності (позначається K) визначає вагу a, яка визначається цим словом. Крім того, функція SoftMax використовується для розподілу ваг a між 0 і 1. Потім, використовуючи ті ж самі вектори, що і Q для кодера і декодера, але різні для модуля з входами кодера і декодера, ці ваги застосовуються до кожного слова у послідовності, яка вводиться у V.

Як цей механізм уваги можна розпаралелити на різні механізми, які можна використовувати одночасно, показано на рис. 4.1 праворуч. Лінійні проекції Q, K і V використовуються для багаторазового повторення механізму уваги. В результаті система може отримати вигоду від навчання на основі різних представлень Q, K і V. Шляхом множення Q, K і V на вагові матриці W, отримані під час навчання, створюються ці лінійні репрезентації.

Залежно від того, чи знаходяться модулі уваги в кодері, декодері або десь посередині між кодером і декодером, матриці Q, K і V відрізняються для кожної позиції модулів уваги в структурі. Далі треба зосередитися або на всій вхідній послідовності кодера, або на певній ділянці вхідної послідовності декодера, яка є основним драйвером. Кодер і декодер з'єднані багатоголовим модулем уваги, який гарантує, що вхідні послідовності від обох кодерів розглядаються до певного місця.

Точковий шар прямого перетворення слідує за Multi-Attention Heads кодера і декодера, що мають багатофункціональну увагу. Ця маленька мережа прямого зв'язку, яку можна розглядати як окреме, ідентичне лінійне перетворення кожного елемента з заданої послідовності, має ідентичні параметри в кожній точці.

4.2 Методи машинного навчання розв’язання задачі класифікації

4.2.1 Дерево рішень

Алгоритм дерева рішень - це метод машинного навчання, що використовується для прийняття рішень на основі вхідних даних. Воно будує дерево з рішеннями шляхом послідовного розбиття даних на підмножини на основі різних критеріїв. Кожен вузол дерева представляє тест на певний атрибут даних, а кожна гілка представляє рішення на основі цього тесту.

Основні кроки алгоритму дерева рішень:

– Вибір атрибута: Задається питання про те, який атрибут даних найкраще розподіляє вхідні дані на підмножини. Для цього використовуються різні метри, такі як індекс Джині, ентропія або інформаційний приріст.

– Розбиття даних: Вибраний атрибут використовується для розбиття вхідних даних на підмножини на основі значень цього атрибута. Кожна гілка дерева відповідає одному значенню атрибута.

– Побудова піддерев: Для кожної підмножини даних алгоритм рекурсивно застосовується, починаючи з кроку 1, аж до виконання певної умови зупинки. Ця умова може бути, наприклад, досягнення певного рівня глибини дерева або досягнення певної кількості елементів у підмножині.

– Листові вузли: Коли досягнута умова зупинки, створюється листовий вузол дерева, який представляє рішення. Це може бути категорія, клас або значення, залежно від типу задачі (класифікація або регресія).

– Критерії зупинки: У деяких випадках можуть використовуватися додаткові критерії зупинки для уникнення перенавчання, наприклад, обмеження максимальної глибини дерева, мінімальної кількості елементів у листовому вузлі або мінімального збільшення інформації при розбитті.

– Класифікація та прогнозування: Після побудови дерева можна використовувати його для класифікації нових зразків або прогнозування значень на основі їх атрибутів. Це відбувається шляхом спуску по дереву, тестування значень атрибутів і переходу від вузла до вузла відповідно до результатів тестів.

4.2.2 Випадковий ліс

Алгоритм випадкового лісу (Random Forest) є ансамблем дерев рішень і використовується для задач класифікації та регресії. Він поєднує декілька дерев рішень, що навчаються на різних підмножинах даних та з використанням різних підмножинів атрибутів. Кожне дерево рішень у випадковому лісі прогнозує клас (у випадку класифікації) або значення (у випадку регресії), і після цього голосування або середнє значення використовуються для отримання кінцевого прогнозу.

Основні кроки алгоритму випадкового лісу:

– Вибір заміщення: З кожного набору даних створюється випадковий піднабір, який вибирається заміщенням (з повторенням) з вихідного набору даних.

– Вибір атрибутів: Загалом, для кожного дерева рішень, вибирається випадковий піднабір атрибутів з доступного пулу атрибутів. Це дозволяє зменшити кореляцію між деревами та зробити їх більш незалежними.

– Побудова дерев рішень: За допомогою вибраних підмножинів даних та атрибутів будується кожне дерево рішень. Зазвичай використовується алгоритм дерева рішень, такий як CART (Classification and Regression Trees) або C4.5.

– Прогнозування: Коли всі дерева рішень побудовані, вони застосовуються до нових зразків для прогнозування. У випадку класифікації, результат може бути визначений голосуванням більшості, а у випадку регресії - середнім значенням.

4.2.3 XGBoost

Алгоритм методу XGBoost (eXtreme Gradient Boosting) є потужним і широко використовуваним методом машинного навчання для задач класифікації, регресії та ранжування. Він базується на ідеї градієнтного підсилення (gradient boosting) і використовує ансамбль дерев'яних моделей.

Основні кроки алгоритму методу XGBoost:

– Побудова базових моделей: Початково побудовуються базові моделі, які можуть бути простими деревами рішень. Кожна базова модель вирішує частину задачі і намагається покращити прогноз.

– Оцінка помилки: Для кожної базової моделі обчислюється помилка між прогнозами і справжніми значеннями. Ця помилка використовується для налаштування наступної базової моделі.

– Обчислення градієнту та стохастичного градієнтного спуску: Градієнт обчислюється, що вказує напрямок найшвидшого зменшення помилки моделі. Застосовується алгоритм стохастичного градієнтного спуску, який покращує модель, змінюючи параметри базових моделей.

– Додавання базової моделі: Покращена базова модель додається до ансамблю. Це виконується шляхом зменшення градієнту помилки з кожною ітерацією.

– Регуляризація: Застосовується регуляризація для контролю перенавчання. Це може включати обмеження на глибину дерев, швидкість навчання (learning rate) та інші параметри.

– Фінальний прогноз: Після побудови ансамблю базових моделей, вони комбінуються, щоб отримати фінальний прогноз для нових зразків.

4.2.4 SVM

Алгоритм методу опорних векторів (Support Vector Machine, SVM) є одним із найпоширеніших методів машинного навчання для задач класифікації і регресії. Він працює шляхом розбиття простору ознак за допомогою гіперплощини, що максимально розділяє два класи даних.

Основні кроки алгоритму методу SVM:

– Побудова опорних векторів: Задача SVM полягає в пошуку оптимальної гіперплощини, що максимально розділяє два класи даних. Опорні вектори є точками даних, що лежать найближче до цієї гіперплощини. Початкова гіперплощина визначається таким чином, щоб вона максимізувала відстань до найближчих точок двох класів.

– Вибір ядра: Для вирішення не-лінійних задач SVM використовує ядровий метод. Ядро перетворює простір ознак в більш високорозмірний простір, де дані можуть бути лінійно розділені. Популярні ядра включають лінійне, поліноміальне та гаусове (RBF) ядра.

– Оптимізація: Задача SVM може бути сформульована як задача квадратичного програмування, яку слід вирішити для знаходження оптимальної

гіперплощини. Це включає мінімізацію функції втрат, що враховує як роздільну здатність, так і регуляризацію для забезпечення балансу між точністю класифікації і розміром опорних векторів.

– Класифікація та прогнозування: Після навчання SVM може бути використаний для класифікації нових зразків. Для цього вхідний зразок перетворюється за допомогою вибраного ядра, а потім оцінюється його положення відносно оптимальної гіперплощини. Залежно від знаку цього оцінювання, зразок класифікується в один із двох класів.

4.2.5 KNN

Алгоритм методу k-найближчих сусідів (k-Nearest Neighbors, KNN) є одним з найпростіших та популярних методів машинного навчання для класифікації та регресії. Він базується на припущенні, що подібні зразки мають схожі класи або значення.

Основні кроки алгоритму методу KNN:

– Вибір значення k: Початково задається значення k, що визначає кількість найближчих сусідів, які будуть використовуватися для класифікації або регресії нового зразка.

– Визначення відстані: Використовується метрика відстані (наприклад, Евклідова відстань або Манхеттенська відстань) для вимірювання відстані між зразками в просторі ознак. Ця відстань використовується для визначення близькості між зразками.

– Пошук найближчих сусідів: Для нового зразка обчислюються відстані до всіх інших зразків у навчальному наборі даних. Вибираються k найближчих зразків з найменшими відстанями.

– Вибір класу або значення: Для задач класифікації, клас нового зразка визначається голосуванням більшості класів серед його k найближчих сусідів. Для задач регресії, значення нового зразка може бути обчислене як середнє значення або медіана залежно від значень його k найближчих сусідів.

– Класифікація та прогнозування: Після побудови моделі KNN може бути використаний для класифікації нових зразків або прогнозування значень. Для цього обчислюються відстані до всіх зразків у навчальному наборі даних, вибираються k найближчих сусідів, і визначається клас або значення на основі їх голосування або середнього значення.

4.2.6 Наївний Баєсів класифікатор

Алгоритм методу Naive Bayes (Наївний Баєсівський класифікатор) є простим та ефективним методом машинного навчання для класифікації. Він базується на теоремі Баєса та припущенні про наївність (незалежність) ознак.

Основні кроки алгоритму методу Naive Bayes:

– Побудова моделі: На початку виконується побудова моделі на основі навчального набору даних. Для кожного класу обчислюються апріорні ймовірності P(C) - ймовірність належності зразка до конкретного класу.

– Обчислення умовних ймовірностей: Для кожної ознаки (атрибуту) обчислюються умовні ймовірності P(X|C) –- ймовірність виникнення конкретного значення ознаки, при умові, що зразок належить до певного класу. Враховуються статистичні оцінки, такі як частота виникнення значення ознаки у класі.

– Класифікація: При надходженні нового зразка, обчислюються умовні ймовірності P(C|X) – ймовірність належності зразка до кожного класу, використовуючи теорему Баєса. Класифікація здійснюється шляхом вибору класу з найбільшою умовною ймовірністю P(C|X).

4.3 Метрики оцінки якості моделі

Accuracy (точність) – доля правильних відповідей алгоритму. Обчислюється за формулою:

$$accuracy = \frac{TP+TN}{TP+TN+FP+FN}, \tag{4.4}$$

де TP (True Positive) – правильне спрацьовування алгоритму;

TN (True Negative) – алгоритм правильно не спрацював на хибне значення;

FP (False Positive) – хибне спрацьовування алгоритму;

TP (True Positive) – алгоритм хибно не спрацював на правильне значення.

Precision (влучність) – здатність алгоритму відрізняти потрібний клас від решти:

$$precision = \frac{TP}{TP+FP} \tag{4.5}$$

Recall (повнота) – здатність алгоритму виявляти потрібний клас узагалі:

$$recall = \frac{TP}{TP+FN} \tag{4.6}$$

Метрики precision та recall (влучність і повнота) можуть бути визначені для багатокласового розбиття. Тоді їх значення обчислюється усереднено за всіма класами в два способи:

macro average – середнє арифметичне значення за всіма класами;

weighted average – зважене значення:

$$weighted\ average\ recall = \frac{1}{\sum TP_i} \sum TP_i \cdot R_i, \tag{4.7}$$

де R_i – recall (повнота) для класу i.

F1-score – середнє гармонічне precision та recall (влучності та повноти):

$$F_1 = \frac{2}{\frac{1}{precision}+\frac{1}{recall}} = \frac{2 \cdot precision \cdot recall}{precision+recall} \quad (4.8)$$

Так само, як і у випадку з precision та recall, для вибірки з багатьма класами обчислюється усереднене значення F1-score: macro-average F1-score та weighted average F1-score.

Matthew's correlation coefficient (MCC, коефіцієнт кореляції Метьюса), на відміну від precision, recall, F1-score, у своїй формулі враховує значення TN (True Negative), що запобігає зміщенню (bias) оцінки:

$$MCC = \frac{TP \cdot TN - FP \cdot FN}{\sqrt{(TP+FP)(TP+FN)(TN+FP)(TN+FN)}} \quad (4.9)$$

На відміну від інших метрик, значення яких лежать у межах [0; 1], $MCC \in [-1; 1]$, де –1 відповідає поганій моделі, +1 – ідеальному випадку.

4.4 Висновки до розділу

У цьому розділі було розглянуто математичні методи обробки текстів та збалансування набору даних, які використовуються відповідними компонентами моделі класифікації автора за повідомленням. Для попередньої обробки текстів та їх векторизації використовується transformer, який є state-of-the-art рішенням при роботі з текстовими даними, так як він зберігає контекст та загальні характеристики тексту.

Також описано математичне забезпечення методів машинного навчання розв’язання задачі класифікації, а саме: дерев прийняття рішень, випадкового лісу, наївного Байєсового класифікатора, методу SVM. Розглянуто метрики для оцінки якості класифікації цими методами, а саме:: accuracy, precision, recall, F1-score, MCC. Описано математичне забезпечення наведених метрик.

5 ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ

5.1 Об’єднання даних

Для створення цілісного набору даних з усіма потрібними класами (0 – людина, 1 – бот, 2 – троль) потрібно об’єднати розкиданий по різним каталогам датасет cresci-2017 [6], для цього була використана бібліотеку pandas, а саме методи read_csv та merge. Перед цим кожному класу був присвоєний новий параметр з класом для кожного запису.

5.2 Попередня обробка даних

Для того щоб моделі машинного навчання видавали гарні результати, треба обробити дані належним чином. Для цього були видалені непотрібні параметри, наприклад такі як посилання на аватар акаунта і так далі. Після цього були видалені дуплікати методом drop_duplicates()[10]. Наступним кроком стало заміна чи виділення NaN (Not a Number) зі значень параметрі. Для категоріальних змін виявилось, що NaN краще замінити на нуль, а для інших – видалити відповідні записи з набору даних. Після попередньої обробки параметри набору даних мають наступний вигляд:

- Text – текст твіту у форматі текстових даних
- Retweet count – кількість людей, які переслали цей твіт
- Reply count – кількість відповідей на твіт
- Favorite count – кількість вподобань твіта
- Possibly sensitive – логічний параметр, який означає чи може вміст твіту бути шкідливим
- Num hashtags – кількість хештегів у твіті

- Num mentions – кількість згадувань цього твіта у інших
- Followers count – кількість підписників автора твіта
- Friends count – кількість друзів автора твіта
- Favourites count – кількість вподобаних твітів інших авторів
- Verified – наявність підтвердження акаунта адміністрацією
- Class – тип автора коментаря (0 – людина, 1 – бот, 2 – троль)

5.3 Обробка тексту

Для роботи з текстом була використана бібліотека sentence-transformers. Її клас – SentenceTransformer має змогу самому зробити попередню обробку тексту та виявити ембедінги, які подалі будуть використані при навчанні моделі. Для отримання ембедінгів була використана вже натренована модель allMiniLM-L6-v2, яка відображає речення та абзаци у 384-вимірний щільний векторний простір. Далі параметр тексту був замінений на отриманий векторний простір.

5.4 Навчання моделей

Більшість з використаних моделей машинного навчання в цій реалізації було взято з бібліотеки sklearn, а саме реалізації дерева рішень, випадкового лісу, KNN, SVM та наївного Баєсівського класифікатору. Для моделі XGBoost була використана бібліотека xgboost[9]. При реалізації цих моделей були знайдені оптимальні гіперпараметри, які видають гарний результат. Вони наведені нижче:

– Дерево рішень: {'ccp_alpha': 0.0, 'class_weight': None, 'criterion': 'gini', 'max_depth': None, 'max_features': None, 'max_leaf_nodes': None,

'min_impurity_decrease': 0.0, 'min_samples_leaf': 1, 'min_samples_split': 2, 'min_weight_fraction_leaf': 0.0, 'random_state': None, 'splitter': 'best'}

– Випадковий ліс: {'bootstrap': True, 'ccp_alpha': 0.0, 'class_weight': None, 'criterion': 'gini', 'max_depth': None, 'max_features': 'sqrt', 'max_leaf_nodes': None, 'max_samples': None, 'min_impurity_decrease': 0.0, 'min_samples_leaf': 1, 'min_samples_split': 2, 'min_weight_fraction_leaf': 0.0, 'n_estimators': 100, 'n_jobs': None, 'oob_score': False, 'random_state': None, 'verbose': 0, 'warm_start': False}

– XGBoost: {'objective': 'multi:softprob', 'use_label_encoder': None, 'base_score': None, 'booster': None, 'callbacks': None, 'colsample_bylevel': None, 'colsample_bynode': None, 'colsample_bytree': 0.5, 'early_stopping_rounds': None, 'enable_categorical': False, 'eval_metric': 'auc', 'feature_types': None, 'gamma': None, 'gpu_id': None, 'grow_policy': None, 'importance_type': None, 'interaction_constraints': None, 'learning_rate': 0.01, 'max_bin': None, 'max_cat_threshold': None, 'max_cat_to_onehot': None, 'max_delta_step': None, 'max_depth': 5, 'max_leaves': None, 'min_child_weight': None, 'missing': nan, 'monotone_constraints': None, 'n_estimators': 5000, 'n_jobs': None, 'num_parallel_tree': None, 'predictor': None, 'random_state': None, 'reg_alpha': None, 'reg_lambda': None, 'sampling_method': None, 'scale_pos_weight': None, 'subsample': 0.5, 'tree_method': None, 'validate_parameters': None, 'verbosity': 1}

– KNN: для цього методу головний параметр n_components=4, це було виявлено за допомогою методу ліктя.

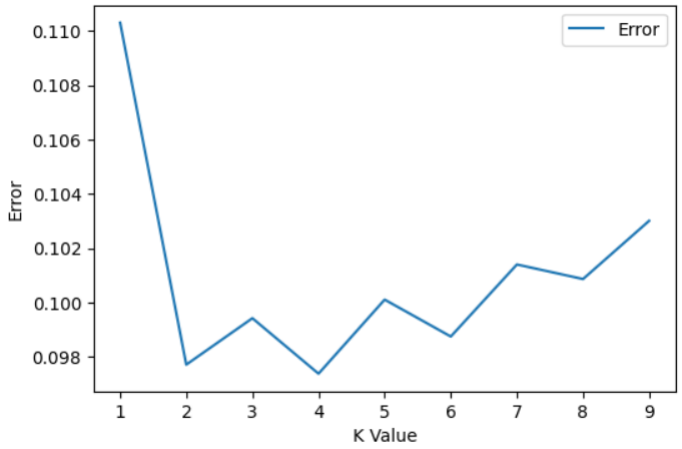


Рисунок 5.1 – Метод ліктя

– SVM: {'C': 1.0, 'break_ties': False, 'cache_size': 200, 'class_weight': None, 'coef0': 0.0, 'decision_function_shape': 'ovr', 'degree': 3, 'gamma': 'scale', 'kernel': 'rbf', 'max_iter': -1, 'probability': False, 'random_state': None, 'shrinking': True, 'tol': 0.001, 'verbose': True}

– Naive Bayes: {'priors': None, 'var_smoothing': 1e-09}

5.5 Результати випробування

5.5.1 Навчання дерева рішень

При роботі натренованої моделі дерева рішень на тестовій вибірці були отримані наступні результати:

Predicted	0	1	2	All
True				
0	33897	34	111	34042
1	37	34631	758	35426
2	98	603	37902	38603
All	34032	35268	38771	108071

Рисунок 5.2 – Матриця невідповідностей дерева рішень

Метрики:

- Accuracy: 0.9848155379333956
- Mcc: 0.9771930129773184
- F1 : [0.99588683 0.97974368 0.97970895]
- Recall : [0.99574056 0.97755886 0.98184079]
- Precision: [0.99603315 0.9819383 0.97758634]

Найважливіші параметри:

- favourites_count 0.429955
- friends_count 0.151361
- statuses_count 0.121685
- favorite_count 0.099903
- listed_count 0.039904

5.5.2 Випадковий ліс

При роботі натренованої моделі випадкового лісу на тестовій вибірці були отримані наступні результати:

Predicted	0	1	2	All
True				
0	33835	138	69	34042
1	27	35254	145	35426
2	1299	570	36734	38603
All	35161	35962	36948	108071

Рисунок 5.3 – Матриця невідповідностей випадкового лісу

Метрики:

- Accuracy: 0.9791988600086979
- Mcc: 0.969048287506215
- F1 : [0.97784778 0.987673 0.97242922]
- Recall : [0.99391928 0.99514481 0.95158407]
- Precision: [0.96228776 0.98031255 0.99420808]

Найважливіші параметри:

- favourites_count 0.147543
- friends_count 0.071947
- statuses_count 0.069337
- followers_count 0.048248
- listed_count 0.027117

5.5.3 XGBoost

При роботі натренованої моделі XGBoost на тестовій вибірці були отримані наступні результати:

Predicted	0	1	2	All
True				
0	33795	208	39	34042
1	2	35163	261	35426
2	1700	1574	35329	38603
All	35497	36945	35629	108071

Рисунок 5.4 – Матриця невідповідностей XGBoost

Метрики:

- Accuracy: 0.9649859814381286
- Mcc: 0.9482906655002101
- F1 : [0.97197256 0.97174283 0.95185365]
- Recall : [0.99274426 0.99257607 0.91518794]
- Precision: [0.95205229 0.95176614 0.99157989]

Найважливіші параметри:

- favourites_count 0.082181
- 369 0.052783 (параметр тексту)
- favorite_count 0.036404
- num_mentions 0.031194
- friends_count 0.029803

5.5.4 KNN

При роботі натренованої моделі KNN на тестовій вибірці були отримані наступні результати:

Predicted	0	1	2	All
True				
0	34035	0	7	34042
1	0	35426	0	35426
2	19	2	38582	38603
All	34054	35428	38589	108071

Рисунок 5.5 – Матриця невідповідностей KNN

Метрики:

- Accuracy: 0.999740911067724
- Mcc: 0.9996108405826314
- F1 : [0.99961817 0.99995766 0.99965023]
- Recall : [0.999765 1. 0.99948191]
- Precision: [0.9994714 0.99991532 0.99981861]

5.5.5 SVM

При роботі натренованої моделі SVM на тестовій вибірці були отримані наступні результати:

[LibSVM]Predicted	0	1	2	All
True				
0	27245	4230	2567	34042
1	5508	27636	2282	35426
2	6128	5359	27116	38603
All	38881	37225	31965	108071

Рисунок 5.6 – Матриця невідповідностей SVM

Метрики:

- Accuracy: 0.7587326849941243
- Mcc: 0.6417888775307715
- F1 : [0.74722653 0.76078788 0.76850697]
- Recall : [0.80033488 0.78010501 0.70243245]
- Precision: [0.70072786 0.7424043 0.84830283]

5.5.6 Наївний баєсів класифікатор

При роботі натренованої моделі наївного Басівського класифікатору на тестовій вибірці були отримані наступні результати:

Predicted	0	1	2	All
True				
0	27067	6132	843	34042
1	296	34685	445	35426
2	2175	11046	25382	38603
All	29538	51863	26670	108071

Рисунок 5.7 – Матриця невідповідностей наївного Баєсівського класифікатору

Метрики:

- Accuracy: 0.8062662508906182
- Mcc: 0.7306369450401775
- F1 : [0.85143127 0.7947164 0.7777182]
- Recall : [0.79510605 0.97908316 0.65751366]
- Precision: [0.91634505 0.66878121 0.95170604]

5.6 Аналіз отриманих результатів

Для зручного порівняння результатів роботи різних методів можна скомбінувати вивід їх метрик у окрему таблицю:

Таблиця 5.1 – Метрики кожного методу на тестовій вибірці

	Дерево рішень	Випадковий ліс	XGBoost	KNN	SVM	Naive Bayes
Accuracy	0.9848	0.9792	0.9649	0.9997	0.7587	0.8062
Mcc	0.9771	0.969	0.9482	0.9996	0.6417	0.7306
F1	0.9958 0.9797 0.9797	0.9778 0.9876 0.9724	0.9719 0.9717 0.9518	0.9996 0.9999 0.9996	0.7472 0.7607 0.7685	0.8514 0.7947 0.7777
Recall	0.9927 0.9925 0.9151	0.9939 0.9951 0.9515	0.9927 0.9925 0.9151	0.9997 1. 0.9994	0.8003 0.7801 0.7024	0.7951 0.979 0.6575
Precision	0.996 0.9819 0.9775	0.9622 0.9803 0.9942	0.952 0.9517 0.9915	0.9994 0.9999 0.9998	0.7007 0.7424 0.8483	0.9163 0.6687 0.9517

Для загального аналізу візьмемо за основну метрику accuracy, так як в нас збалансований набір даних і ми не даємо ніякому класу більше ваги ніж іншим. Тоді можна побачити, що всі алгоритми показали себе добре. Але гіршим чином виділяються SVM та Наївний Баєсів класифікатор, а найкращим методом є KNN з точністю 99.97%

5.7 Висновки до розділу

У цьому розділі було описано програмне забезпечення обробки текстів, попередню обробку та збалансування набору даних, яке використовується відповідними компонентами моделі класифікації автора за повідомленням. Для

обробки текстів використовується клас SentenceTransformer бібліотеки sentence-transformers, який сам обробляє текст та видає ембедінги для подальшої роботи.

Також описано програмне забезпечення методів машинного навчання розв’язання задачі класифікації, а саме: дерев прийняття рішень, випадкового лісу, XGBoost, наївного Байєсового класифікатора, методу SVM. Проведений аналіз отриманих результатів. Встановлено, що найкращими класифікаторами для задачі класифікації автора за повідомленням виявились дерево прийняття рішень, випадковий ліс, XGBoost та KNN, як найкраща реалізація. Найгіршими з шести класифікаторів є наївний Баєсівський класифікатор та SVM. Проте, він дає точність не менше 75% на тестовій вибірці, що також вважається гарним результатом.

ВИСНОВКИ

У результаті виконаної роботи:

1. Було розглянуто наявні методи розв’язання поставленої задачі класифікації автора за повідомленням у соціальній мережі Twitter (штучні нейронні мережі, методи машинного навчання) та програмні рішення, у яких реалізовані описані методи. Усі ці методи є достатньо різними, та не всі програмні рішення є у відкритому доступі, тому їх не можна порівняти. У даній роботі проблема встановлення автора повідомлення розглядається як задачі класифікації, тому обрані методи машинного навчання розв’язання задачі: дерева прийняття рішень, випадковий ліс, XGBoost, наївний Байєсів класифікатор, метод SVM;

2. Було побудовано систему класифікації автора за повідомленням у вигляді діаграми компонентів. Її компоненти формуються за функціональною ознакою, а відношення між компонентами відображається через реалізацію відповідних інтерфейсів. Описано призначення та зміст компонентів;

3. Було розглянуто математичні методи, які використовуються відповідними компонентами. Серед них було виділено метод трансформера, так як він є одним з найактуальніших рішень на сьогоднішній день. Також описано математичне забезпечення методів машинного навчання розв’язання задачі класифікації, а саме: дерев прийняття рішень, випадкового лісу, XGBoost, наївного Байєсового класифікатора, методу SVM та KNN. Встановлено, що для оцінки якості класифікації цими методами можна використати наступні метрики: accuracy, precision, recall, F1-score, MCC. Описано математичне забезпечення наведених метрик;

4. Було розроблено програмне забезпечення системи класифікації автора за повідомленням. Описано програмне забезпечення методів машинного навчання розв’язання задачі класифікації, а саме: дерев прийняття рішень, випадкового лісу, XGBoost, наївного Байєсового класифікатора, методу SVM та KNN. Проведено

аналіз отриманих результатів. Встановлено, що найкращими класифікаторами для задачі класифікації автора за повідомленням KNN з результатами accuracy 99.97%. Найгіршим з шести класифікаторів є SVM класифікатор з точністю в 75.87%.

5. В ході виконання роботи та зіткненням з поточними проблема було виявлено ще декілька напрямків потенціального покращення математично та програмного забезпечення для вирішення даної задачі, а саме: покращення набору даних, перевірка інших методів попередньої обробки даних, використання інших моделей трасформера чи створення спеціального для даної задачі, додавання до аналізу методів машинного навчання нейронні мережі та інші алгоритми.

ПЕРЕЛІК ПОСИЛАНЬ

1. Machine learning-based social media bot detection: a comprehensive literature review / Malak Aljabri [та ін.] // Social Network Analysis and Mining. – 2023. – Т. 13, № 1.

2. Bot Sentinel [Електронний ресурс] // BotSentinel. – Режим доступу: https://hoaxy.osome.iu.edu/.

3. Hoaxy: How claims spread online [Електронний ресурс] // Hoaxy. – Режим доступу: https://hoaxy.osome.iu.edu/.

4. Botometer by OSoMe [Електронний ресурс] // Botometer® by OSoMe. – Режим доступу: https://botometer.osome.iu.edu/.

5. Attention is all you need / A. Vaswani [та ін.] // Advances in Neural Information Processing Systems. – 2017. – Т. 30, – С. 1-15.

6. Social fingerprinting: detection of spambot groups through DNA-inspired behavioral modeling / Stefano Cresci [та ін.] // IEEE Transactions on Dependable and Secure Computing. – 2018. – Т. 15, №. 4, – С. 561-576.

7. SentenceTransformers Documentation – Sentence-Transformers documentation [Електронний ресурс]. – Режим доступу: https://www.sbert.net/.

8. scikit-learn: machine learning in Python – scikit-learn 1.2.2 documentation [Електронний ресурс]. – Режим доступу: https://scikit-learn.org/stable/index.html.

9. XGBoost Documentation – xgboost 1.7.5 documentation [Електронний ресурс]. – Режим доступу: https://xgboost.readthedocs.io/en/stable/.

10. pandas documentation – pandas 2.0.2 documentation [Електронний ресурс] // pandas - Python Data Analysis Library. – Режим доступу: https://pandas.pydata.org/pandas-docs/stable/index.html.

11. Brown J. B. Classifiers and their Metrics Quantified / J. B. Brown // Molecular Informatics. – 2018. – Т. 37, № 1-2. – С. 1700127.

12. Ali D. Multiclass Event Classification from Text / Daler Ali, Malik Muhammad Saad Missen, Mujtaba Husnain // Scientific Programming. – 2021. – Т. 2021. – С. 1–15.

13. Zhu C. Influence of Data Preprocessing / Changming Zhu, Daqi Gao // Journal of Computing Science and Engineering. – 2016. – Т. 10, № 2. – С. 51–57.

14. Multiclass patent document classification / Chaitanya Anne [та ін.] // Artificial Intelligence Research. – 2017. – Т. 7, № 1.

Додаток А

Лістинги програм

Лістинг файлу data_preprocess.ipynb — збір та обробка даних

```
import pandas as pd
chunk_list = []             # used for storing dataframes
for chunk in pd.read_csv('genuine_accounts/tweets.csv', chunksize=1000, encoding = "ISO-8859-1"):          # each chunk is a dataframe
    chunk_list.append(chunk)
# concat all the dfs in the list into a single dataframe
df_concat = pd.concat(chunk_list)
h_users = pd.read_csv('genuine_accounts/users.csv', encoding = "ISO-8859-1")
# rename feature to concat
h_users = h_users.rename(columns={'id':'user_id'})
humans = df_concat.merge(h_users, on='user_id')
# dropping useless features
sub = humans[['text', 'retweet_count', 'reply_count', 'favorite_count', 'possibly_sensitive', 'num_hashtags', 'num_mentions',
          'statuses_count', 'followers_count', 'friends_count', 'favourites_count', 'listed_count', 'lang', 'location',
          'verified']]
# remained features
list(sub.columns)
# humans class
sub['class'] = 0
# reading trolls datasets
ss1_tweets = pd.read_csv('social_spambots_1/tweets.csv', encoding = "ISO-8859-1")
ss2_tweets = pd.read_csv('social_spambots_2/tweets.csv', encoding = "ISO-8859-1")
ss3_tweets = pd.read_csv('social_spambots_3/tweets.csv', encoding = "ISO-8859-1")
ss1_users = pd.read_csv('social_spambots_1/users.csv', encoding = "ISO-8859-1")
ss2_users = pd.read_csv('social_spambots_2/users.csv', encoding = "ISO-8859-1")
ss3_users = pd.read_csv('social_spambots_3/users.csv', encoding = "ISO-8859-1")
# concatinating datasets
ss_tweets = pd.concat([ss1_tweets, ss2_tweets, ss3_tweets])
ss_users = pd.concat([ss1_users, ss2_users, ss3_users])
ss_users = ss_users.rename(columns={'id':'user_id'})
sbots = ss_tweets.merge(ss_users, on='user_id')
sub2 = sbots[['text', 'retweet_count', 'reply_count', 'favorite_count', 'possibly_sensitive', 'num_hashtags', 'num_mentions',
          'statuses_count', 'followers_count', 'friends_count', 'favourites_count', 'listed_count', 'lang', 'location',
          'verified']]
# troll class
sub2['class'] = 2
# reading bots datasets
ts_tweets = pd.read_csv('traditional_spambots_1/tweets.csv', encoding = "ISO-8859-1")
ts1_users = pd.read_csv('traditional_spambots_1/users.csv', encoding = "ISO-8859-1")
ts2_users = pd.read_csv('traditional_spambots_2/users.csv', encoding = "ISO-8859-1")
ts3_users = pd.read_csv('traditional_spambots_3/users.csv', encoding = "ISO-8859-1")
ts4_users = pd.read_csv('traditional_spambots_4/users.csv', encoding = "ISO-8859-1")
ts_users = pd.concat([ts1_users, ts2_users, ts3_users, ts4_users])
ts_users = ts_users.rename(columns={'id':'user_id'})
tbots = ts_tweets.merge(ts_users, on='user_id')
sub3 = tbots[['text', 'retweet_count', 'reply_count', 'favorite_count', 'possibly_sensitive', 'num_hashtags', 'num_mentions',
          'statuses_count', 'followers_count', 'friends_count', 'favourites_count', 'listed_count', 'lang', 'location',
          'verified']]
# bot class
sub3['class'] = 1
# concating all datasets to one
data = pd.concat([sub, sub2, sub3])
# checking NaN values
data.isna().sum()
# dropping rows with NaN in 'text'
data = data[data['text'].notna()]
# dropping random rows to reduce dataset
import numpy as np
np.random.seed(10)

remove_n = 4636846
drop_indices = np.random.choice(data.index, remove_n, replace=False)
data = data.drop(drop_indices)
# drop additional columnns
```

```
data = data.drop(columns=['location', 'lang'])
# checking class distribution
data['class'].value_counts()
# filling NaN s in 'possibly_sensitive' with 0
data['possibly_sensitive'] = data['possibly_sensitive'].fillna(0)
# filling NaN s in 'verified' with 0
data['verified'] = data['verified'].fillna(0)
# drop duplicates
data = data.drop_duplicates()
# setting all text to lowercase
data['text'] = data['text'].str.lower()
# deleting URLs from text
data['text'] = data['text'].str.replace('http\S+', ' ', regex=True)
# leaving only words
data['text'] = data['text'].str.replace('[^A-Za-z0-9]+', ' ', regex=True)
# removing stopwords
import nltk
from nltk.corpus import stopwords
nltk.download('stopwords')
stopwords = stopwords.words('english')
data['text'] = data['text'].apply(lambda words: ' '
                                  .join(word.lower() for word in words.split()
                                      if word not in stopwords))
# format text feature
data['word_number'] = data['text'].str.split().str.len()
# TF-IDF tokenization
# from sklearn.feature_extraction.text import TfidfVectorizer
# vectorizer = TfidfVectorizer()
# X = vectorizer.fit(data['text'])
# vectorizer.get_feature_names_out()
# test = vectorizer.transform(data['text'])
import keras
from keras.models import Model
from keras.layers import LSTM, Activation, Dense, Dropout, Input, Embedding
from keras.optimizers import RMSprop
from keras.preprocessing.text import Tokenizer
from keras.preprocessing import sequence
from keras.utils import to_categorical, pad_sequences
from keras.callbacks import EarlyStopping
from sklearn.utils import shuffle
maxlen = 280
maxwords = 40000

# receiving embeddings
from sentence_transformers import SentenceTransformer
model = SentenceTransformer('all-MiniLM-L6-v2')
sentences = pd.Series(data['text']).tolist()
embeddings = model.encode(sentences=sentences, show_progress_bar=True, batch_size=5)

# creating df of text replaced by vectors
feature_names = ['x' + str(i) for i in range(padded.shape[1])]
sequence_df = pd.DataFrame(padded, columns=feature_names)
# replacing text with vectors
data = data.drop(columns=['text'])
data = data.reset_index(drop=True)
# concatinating
data = pd.concat([data, sequence_df], axis=1)
# write dataset content in data.csv
data.to_csv('data.csv', index=False)
```

Лістинг файлу models.ipynb — побудова та тестування моделей

```
import pandas as pd
# reading dataset
data = pd.read_csv('data.csv')
# checking class distribution
data['class'].value_counts()
# creating X and y for model fitting
X = data.drop(columns=['class'])
y = data['class']
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn import metrics
# test train validation split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=1)
```

```
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size=0.25, random_state=1)
# create a Decision Tree Classifier
clf = DecisionTreeClassifier()
# train the model
clf.fit(X_train, y_train)
# DTC results
y_pred = clf.predict(X_test)

y_true = y_test

matrix = pd.crosstab(y_true, y_pred, rownames=['True'],
                     colnames=['Predicted'], margins=True)
print(matrix)
print("Accuracy:", metrics.accuracy_score(y_test, y_pred))
print("Mcc:", metrics.matthews_corrcoef(y_test, y_pred))
print("F1 :", metrics.f1_score(y_test, y_pred, average=None))
print("Recall :", metrics.recall_score(y_test, y_pred, average=None))
print("Precision:", metrics.precision_score(y_test, y_pred, average=None))

feature_imp = pd.Series(
        clf.feature_importances_,
        index=data.columns.drop('class')).sort_values(ascending=False)
print(feature_imp)
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, precision_score, recall_score, ConfusionMatrixDisplay
from sklearn.model_selection import RandomizedSearchCV, train_test_split
from scipy.stats import randint
# random forest initialization and results
rf = RandomForestClassifier()
rf.fit(X_train, y_train)

y_pred = rf.predict(X_test)

# Model Accuracy, how often is the classifier correct?
y_true = y_test

matrix = pd.crosstab(y_true, y_pred, rownames=['True'],
                     colnames=['Predicted'], margins=True)
print(matrix)
print("Accuracy:", metrics.accuracy_score(y_test, y_pred))
print("Mcc:", metrics.matthews_corrcoef(y_test, y_pred))
print("F1 :", metrics.f1_score(y_test, y_pred, average=None))
print("Recall :", metrics.recall_score(y_test, y_pred, average=None))
print("Precision:", metrics.precision_score(y_test, y_pred, average=None))

feature_imp = pd.Series(
        rf.feature_importances_,
        index=data.columns.drop('class')).sort_values(ascending=False)
print(feature_imp)
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import make_pipeline
from sklearn.svm import SVC
import numpy as np
# standart scaling for SVM
scaler = StandardScaler()
scaler.fit(X)

X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)
# SVM fitting and results
svm = SVC()
svm.fit(X_train, y_train)

y_pred = svm.predict(X_test)

y_true = y_test

matrix = pd.crosstab(y_true, y_pred, rownames=['True'],
                     colnames=['Predicted'], margins=True)
print(matrix)
print("Accuracy:", metrics.accuracy_score(y_test, y_pred))
print("Mcc:", metrics.matthews_corrcoef(y_test, y_pred))
print("F1 :", metrics.f1_score(y_test, y_pred, average=None))
print("Recall :", metrics.recall_score(y_test, y_pred, average=None))
print("Precision:", metrics.precision_score(y_test, y_pred, average=None))
from sklearn.neighbors import KNeighborsClassifier
```

```
# KNN fitting
knn = KNeighborsClassifier(n_neighbors=6)
knn.fit(X_train, y_train)
# KNN results
y_pred = knn.predict(X_test)

y_true = y_test

matrix = pd.crosstab(y_true, y_pred, rownames=['True'],
                colnames=['Predicted'], margins=True)
print(matrix)
print("Accuracy:", metrics.accuracy_score(y_test, y_pred))
print("Mcc:", metrics.matthews_corrcoef(y_test, y_pred))
print("F1 :", metrics.f1_score(y_test, y_pred, average=None))
print("Recall :", metrics.recall_score(y_test, y_pred, average=None))
print("Precision:", metrics.precision_score(y_test, y_pred, average=None))

feature_imp = pd.Series(
       knn.feature_importances_,
       index=data.columns.drop('class')).sort_values(ascending=False)
print(feature_imp)
# elbow method to pick 'n_neighbors' parameter
error_rate = []

for k in range(1,10):
   knn_model = KNeighborsClassifier(n_neighbors=k)
   knn_model.fit(X_train,y_train)

   y_pred = knn_model.predict(X_test)

   error_rate.append(np.mean(y_pred != y_test))
# elbow graph
plt.figure(figsize=(6,4),dpi=100)
plt.plot(range(1,10),error_rate,label='Error')
plt.legend()
plt.ylabel('Error')
plt.xlabel("K Value")
# log transform for Naive Bayes
from sklearn.naive_bayes import GaussianNB
from sklearn.preprocessing import FunctionTransformer

def log_transform(x):
   return np.log(x + 1)

transformer = FunctionTransformer(log_transform)
transformer.fit(X)
X_train = transformer.transform(X_train)
X_test = transformer.transform(X_test)

# Naive Bayes fitting
gnb = GaussianNB().fit(X_train, y_train)
# Naive Bayes results
y_pred = gnb.predict(X_test)

y_true = y_test

matrix = pd.crosstab(y_true, y_pred, rownames=['True'],
                colnames=['Predicted'], margins=True)
print(matrix)
print("Accuracy:", metrics.accuracy_score(y_test, y_pred))
print("Mcc:", metrics.matthews_corrcoef(y_test, y_pred))
print("F1 :", metrics.f1_score(y_test, y_pred, average=None))
print("Recall :", metrics.recall_score(y_test, y_pred, average=None))
print("Precision:", metrics.precision_score(y_test, y_pred, average=None))
```

Додаток Б

Ілюстративний матеріал

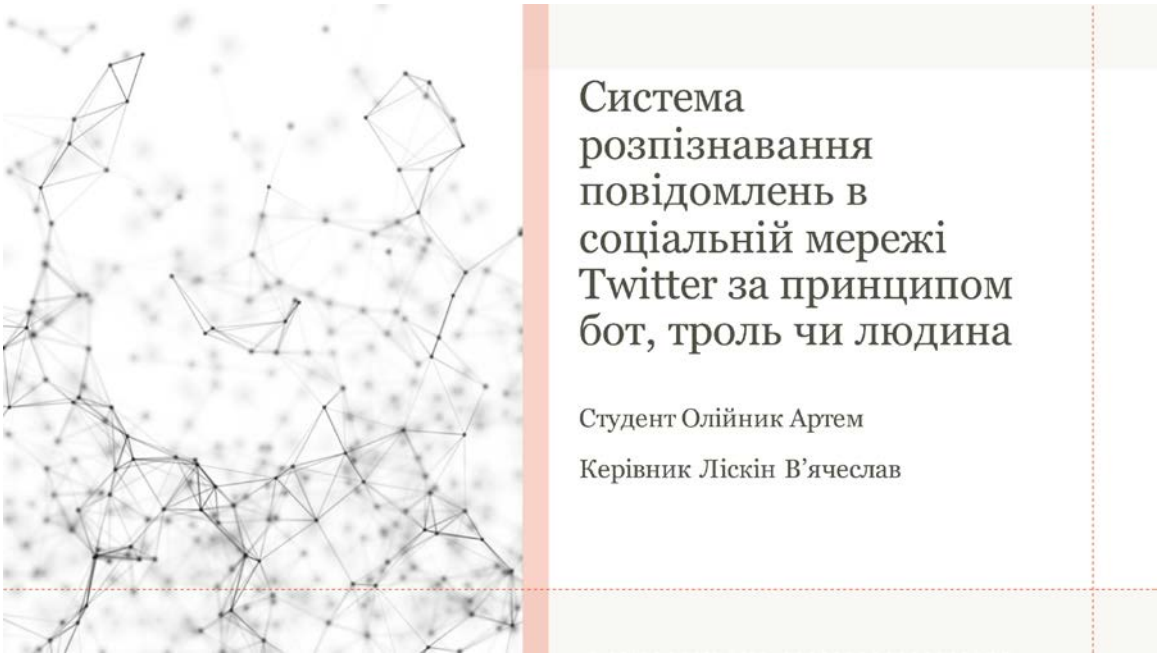


Рисунок Б.1 – Слайд 1

Рисунок Б.2 – Слайд 2

3

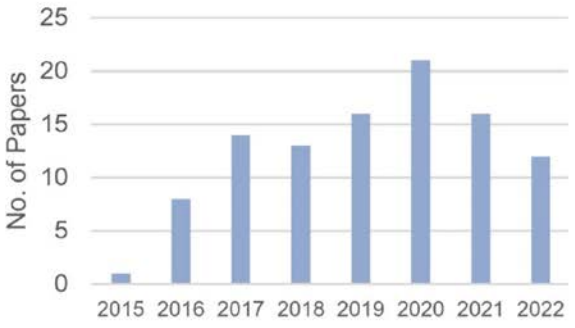
Постановка задачі

- **Об'єктом дослідження** є найбільш відомі методи обробки неструктурованих текстів та класифікації, існуючі системи та програмні засоби вирішення задачі класифікації.
- **Предметом дослідження** є математичне та програмне забезпечення системи для визначення автора коментаря за типом людина, бот чи "троль".
- **Метою роботи** є системна інженерія та дослідження математичного і програмного забезпечення для визначення автора коментаря за типом людина, бот чи "троль".
- **Кінцевим результатом роботи** є математичне та програмне забезпечення системи, що визначає автора коментаря за типом людина, бот чи "троль" та аналіз результатів роботи різних алгоритмів.

Рисунок Б.3 – Слайд 3

4

Наявні програмні рішення



- Кількість досліджень по роках

Рисунок Б.4 – Слайд 4

5

Існуючі системи

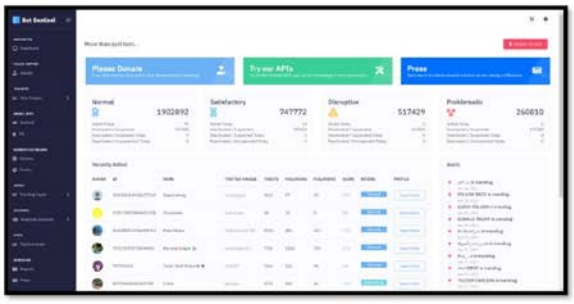


Рисунок Б.5 – Слайд 5

6

Методи векторизації тексту

- Word2Vec
- TF-IDF
- Bag-of-Words
- Transformers

Рисунок Б.6 – Слайд 6

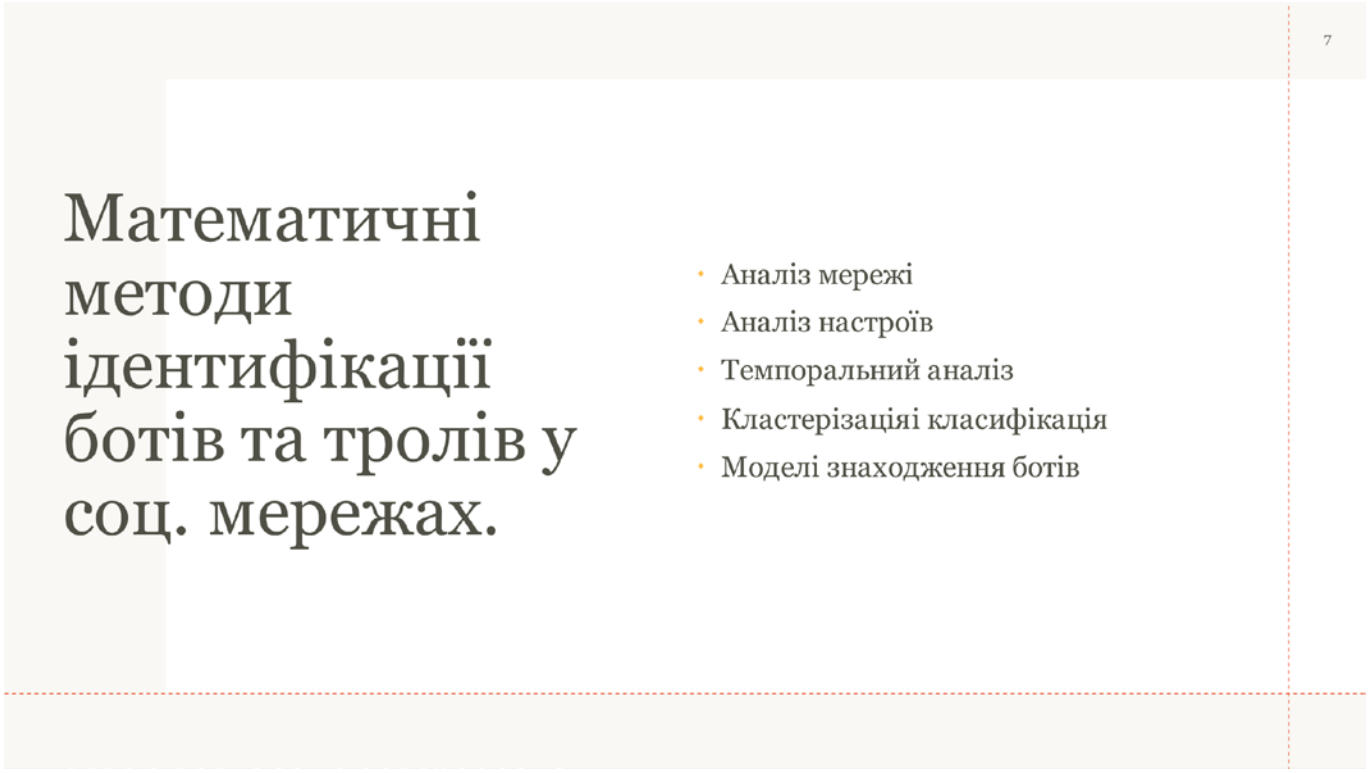


Рисунок Б.7 – Слайд 7

Рисунок Б.8 – Слайд 8

9

Модель системи

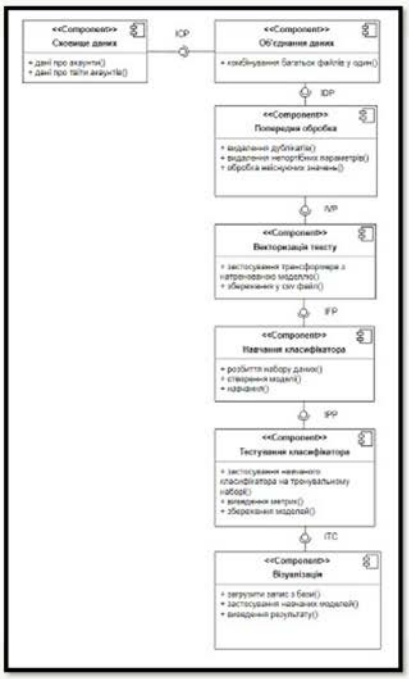


Рисунок Б.9 – Слайд 9

10

Набір даних

- Cresci-2017
  - Text – текст твіту у форматі текстових даних
  - Retweet count – кількість людей, які переслали цей твіт
  - Reply count – кількість відповідей на твіт
  - Favorite count – кількість вподобань твіта
  - Possibly sensitive – логічний параметр, який означає чи може вміст твіту бути шкідливим
  - Num hashtags – кількість хештегів у твіті
  - Num mentions – кількість згадувань цього твіта у інших
  - Followers count – кількість підписників автора твіта
  - Friends count – кількість друзів автора твіта
  - Favourites count – кількість вподобаних твіпів інших авторів
  - Verified – наявність підтвердження акаунта адміністрацією
  - Class – тип автора коментаря (0 – людина, 1 – бот, 2 – троль)

Рисунок Б.10 – Слайд 10

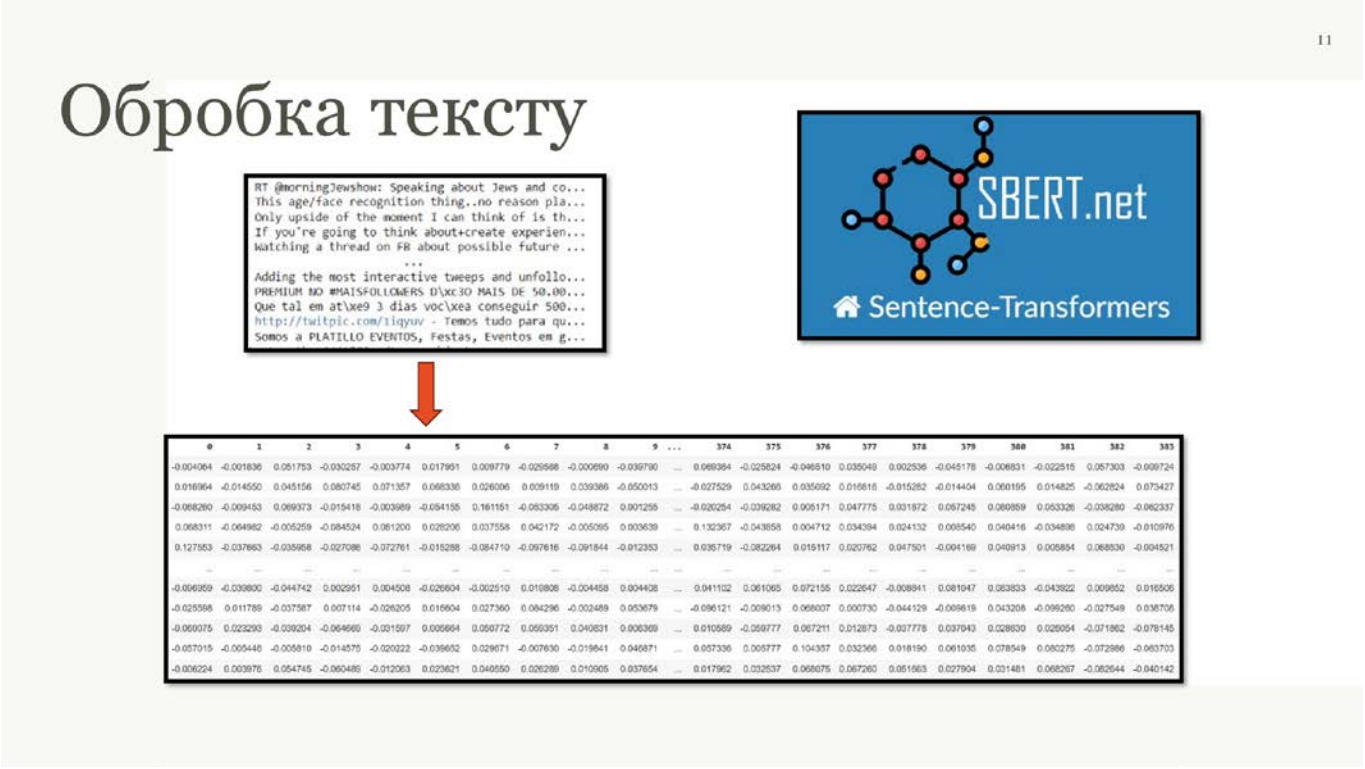


Рисунок Б.11 – Слайд 11

Рисунок Б.12 – Слайд 12

13

Результати

	Дерево рішень	Випадковий ліс	XGBoost	KNN	SVM	Naive Bayes
Accuracy	0.9848	0.9792	0.9649	0.9997	0.7587	0.8062
Mcc	0.9771	0.969	0.9482	0.9996	0.6417	0.7306
F1	0.9958	0.9778	0.9719	0.9996	0.7472	0.8514
	0.9797	0.9876	0.9717	0.9999	0.7607	0.7947
	0.9797	0.9724	0.9518	0.9996	0.7685	0.7777
Recall	0.9927	0.9939	0.9927	0.9997	0.8003	0.7951
	0.9925	0.9951	0.9925	1.	0.7801	0.979
	0.9151	0.9515	0.9151	0.9994	0.7024	0.6575
Precision	0.996	0.9622	0.952	0.9994	0.7007	0.9163
	0.9819	0.9803	0.9517	0.9999	0.7424	0.6687
	0.9775	0.9942	0.9915	0.9998	0.8483	0.9517

Рисунок Б.13 – Слайд 13

14

Можливі вдосконалення

- Більш глибока обробка даних
- Комбінування різних наборів даних
- Використання інших алгоритмів класифікації
- Інші методи перетворення тексту

Рисунок Б.14 – Слайд 14

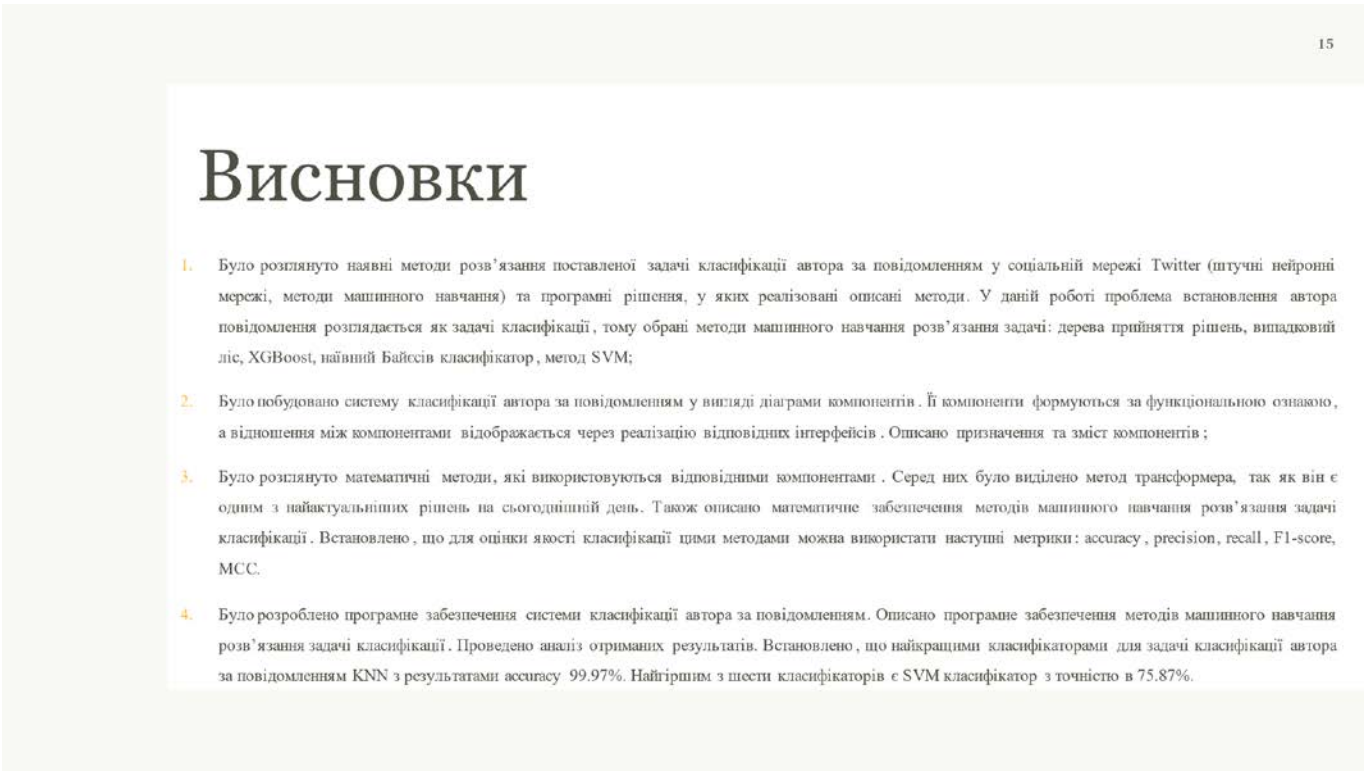


Рисунок Б.15 – Слайд 15