КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

ІМЕНІ ТАРАСА ШЕВЧЕНКА

Факультет комп'ютерних наук та кібернетики

Кафедра математичної інформатики

Кваліфікаційна робота

на здобуття ступеня бакалавра

за спеціальністю 122 Комп'ютерні науки

на тему:

МЕТОДИ КЛАСИФІКАЦІЇ ТЕКСТІВ ПРИРОДНОЮ МОВОЮ

Виконала студентка 4-го курсу

Нечаєва Вероніка Валентинівна _____

Науковий керівник:

професор кафедри МІ

Марченко Олександр Олександрович _____

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

Роботу розглянуто й допущено до захисту
на засіданні кафедри математичної інформатики
«___» _____ 2023 р.,
протокол № ___
Завідувач кафедри
Терещенко В. М. . _____

Київ - 2023

РЕФЕРАТ

Обсяг роботи 43 сторінки, 9 ілюстрацій, 6 таблиць, 12 джерел посилань.
ПОСЛІДОВНА НЕЙРОННА МЕРЕЖА, РЕКУРЕНТНА НЕЙРОННА МЕРЕЖА, ПОРІВНЯННЯ МЕТОДІВ, МАШИННЕ НАВЧАННЯ, ГЛИБОКЕ НАВЧАННЯ, КЛАСИФІКАЦІЯ ТЕКСТІВ НАТУРАЛЬНОЮ МОВОЮ, ІНФОРМАЦІЙНІ ТЕХНОЛОГІЇ.

Об’єктом роботи є процес класифікації текстів, написаних натуральною мовою, на прикладі бінарної класифікації англомовних веб-ресурсів масової інформації на предмет достовірності або фейковості.

Метою роботи є порівняння різних методів класифікації на основі нейронних мереж для дослідження їхньої ефективності.

Методи розроблення: код Python, виконаний у середовищі Colab Notebook, з використанням бібліотек Tensorflow та Keras для створення моделей, а також Scikit-learn, Matplotlib, та Spacy для аналізу та передобробки вхідних даних.

Результати розроблення: розглянуто декілька рішень класифікації текстів на основі SNN та LSTM та їхня ефективність за умовою різних варіантів передобробки або кількості наявних ресурсів.

Створені моделі можуть бути використані для широкого кола прикладних, дослідницьких та навчальних задач.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАЧЕННЯ

BERT - Bidirectional Encoder Representations from Transformers

CNTK - Microsoft Cognitive Toolkit

GPU - Graphics Processing Unit

GRU - Gated Recurrent Unit

LSTM - Long-Short Term Memory

RNN - Recurrent Neural Network

SNN - Sequential Neural Network

TF-IDF - Term Frequency-Inverse Document Frequency

TPU - Tensor Processing Unit

ВСТУП

Актуальність роботи. В сучасному інформаційному суспільстві швидкість поширення інформації та доступ до неї досягли небачених раніше рівнів. У зв'язку зі стрімким розвитком цифрових медіа та соціальних мереж зловживання і фальсифікація інформації, зокрема, поширення фейкових новин, мають потенціал серйозно вплинути на громадську думку, політичні процеси та суспільство в цілому.

Розробка автоматизованих методів для розпізнавання фейкових новин стає одним із викликів для наукової спільноти. В такому середовищі, задачі автоматичного аналізу, класифікації та фільтрації текстів стають важливими як ніколи.

Метою даної роботи є розробка та вдосконалення моделей класифікації текстів природною мовою для досягнення високої курації та якості інтернет-контенту. Дана тема розглядається на прикладі задачі розпізнавання фейкових, неправдивих та недостовірних новин, яка є класичним прикладом задачі бінарної класифікації.

Застосування нейромереж для класифікації текстів природною мовою надає потенціал для розв'язання цієї задачі шляхом аналізу лінгвістичних особливостей та контексту інформації. Основний внесок цієї роботи полягає в дослідженні різних підходів до обробки та аналізу текстових даних, виборі відповідних алгоритмів та параметрів нейромережі, а також удосконаленні методів класифікації з використанням важливих лінгвістичних ознак та контекстуальної інформації.

Об'єкт, методи й засоби дослідження і розробки. Ми ставимо перед собою завдання розпізнавання та відокремлення фейкових новин від справжніх

засобами машинного навчання та аналізу тексту. Для виконання даної задачі ми використовуємо бібліотеки та фреймворки з машинним навчанням, такі як TensorFlow та Keras, для реалізації наших моделей. Ми також використовуємо інструменти для попереднього аналізу та очищення текстових даних, такі як Spacy та Scikit-learn, які допомагають нам підготувати дані для навчання моделей. Всі написання та запуски коду були здійснені у Google Colab з використанням віртуальних середовищ, що використовують GPU- та TPU-процесори, для пришвидшення аналізу.

Можливі сфери застосування. Сфери застосування розроблених моделей є широкими і включають як практичні, так і наукові області, такі, як медіа та журналістика, модерація соціальних мереж та платформ, політика та громадське обговорення, а також подальші наукові дослідження у галузі обробки природної мови, машинного навчання та соціальних наук.

Взаємозв’язок з іншими роботами. Результати досліджень можуть бути використані для розвитку нових моделей та алгоритмів, спрямованих на боротьбу з дезінформацією та фейковими новинами.

РОЗДІЛ 1 ЗАДАЧА КЛАСИФІКАЦІЇ ТЕКСТІВ ПРИРОДНОЮ МОВОЮ

1.1 Методи класифікації текстів

Існує багато методів класифікації текстів, які використовуються в завданнях обробки природної мови. Виділимо декілька основних.

Статистичні методи базуються на використанні статистичних характеристик тексту для класифікації. Вони можуть використовувати такі підходи, як байєсівський [7], н-грами (послідовності з найчастіших слів або символів), аналіз частотності слів (наприклад, TF-IDF [3]) та інші.
Використання статистичних методів у класифікації текстів може бути ефективним для простих задач (таких, як детекція спам-повідомлень) або у випадках, коли немає достатньої кількості тренувальних даних для складніших моделей. Однак, вони можуть бути менш точними у врахуванні семантики, контексту та глибшого розуміння тексту порівняно з методами, які використовують нейронні мережі або інші складні моделі.

Машинне навчання на основі ознак (feature-based machine learning) використовує різні ознаки (функції), які вилучаються з тексту. Ознаки можуть включати слова, лексичні або синтаксичні властивості, частоту зустрічі слів, граматичні структури тощо. За допомогою методів, таких як наївний байєсівський класифікатор, логістична регресія, дерева рішень або метод опорних векторів, моделі навчаються на тренувальних даних і застосовуються до нових текстів для їх класифікації.

Рекурентні нейронні мережі (RNN) стали потужним інструментом для класифікації текстів. RNN, зокрема LSTM (моделі довгочасної-короткочасної пам'яті) і GRU (керовані рекуренні блоки) дозволяють моделям зберігати та враховувати контекстуальну інформацію. Такі моделі здатні моделювати залежності та контекст у послідовностях даних. Вони особливо корисні для аналізу даних, де порядок слів має значення.

Моделі-трансформери (Transformer-based Models), такі як модель BERT (Bidirectional Encoder Representations from Transformers), використовують механізми уваги для моделювання відносин між словами в тексті. Вони демонструють потужну здатність до розуміння семантичного змісту та контексту тексту. Застосування моделей-трансформерів дозволяє використовувати контекстуальні векторні представлення слів та розуміти семантику тексту на більш глибокому рівні.

Ансамблі моделей: Нарешті, одним з найефективніших підходів є поєднання моделей класифікації для отримання кращої точності. Наприклад, можуть використовуватися ансамблі дерев рішень, згорткових нейронних мереж або комбінації різних алгоритмів навчання.

В даній роботі ми порівнюємо декілька різних підходів, насамперед зосереджуючись на статистичних методах, навчанні на основі ознак, та LSTM, і робимо висновки щодо ефективності кожного з них.

2.2 Задача розпізнавання фейкових новин

Постановка задачі. Задача розпізнавання фейкових новин є частковим випадком задачі класифікації як такої. У найпростішому випадку, задача розпізнавання фейкових новин може бути поставлена наступним чином:
Дано:

- Набір неперервних текстових документів $D = \{d_1, d_2, ..., d_n\}$, де n -- кількість документів.
- Кожен документ d_i представляється як послідовність токенів $d = \{t_1, t_2, ..., t_m\}$, де m -- кількість токенів у документі.
- Для кожного документа d_i відома мітка y_i, яка позначає його класифікацію: $y_i \in \{0, 1\}$, де 0 відповідає фейковій новині, а 1 - правдивій новині.

Задача: Побудувати модель f, яка буде приймати на вхід текстовий документ d_i і прогнозувати його мітку y_i.

Для вимірювання якості прогнозування будемо використовувати функцію втрат (loss), яку ми намагаємося мінімізувати з використанням методів оптимізації, таких як стохастичний градієнтний спуск (SGD) для знаходження оптимальних параметрів моделі f. Стандартною функцією мінімізації втрат для випадку бінарної класифікації є бінарна крос-ентропія:

$$L(y_1, y) = -(y\,log(y_1) + (1 - y)\,log(1 - y_1))$$

де y_1 -- прогнозована ймовірність фейкової новини, а y -- фактична мітка документа [3].

Проблема тренувального датасету. Задача розпізнавання фейкових новин стикається зі значними проблемами і перешкодами, які ускладнюють її вирішення. Основні проблеми виникають ще на стадії формування датасету. Вони полягають у:

- *Суб'єктивність визначення фейкових новин*: Визначення того, що розглядається як фейкова новина, може бути суб'єктивним процесом. Іноді виявлення фейкових новин вимагає глибокого аналізу контексту, перевірки фактів і порівняння інформації з різних джерел. Оскільки фейкові новини можуть бути витонченою формою маніпуляції, важко встановити однозначні критерії для їх розпізнавання. [8]

- *Недостатність адекватних тренувальних даних*: Для успішного навчання моделей розпізнавання фейкових новин необхідна велика кількість анотованих даних, тобто новин, які вже класифіковані як фейкові або правдиві. Однак, отримання таких наборів даних є складним завданням, оскільки розпізнавання фейкових новин є суб'єктивним процесом, а експертні оцінки можуть варіюватись. Недостатність тренувальних даних може обмежувати точність і загальну ефективність моделей.

- *Проблема збалансованості даних*: Набори даних для розпізнавання фейкових новин можуть бути незбалансованими, тобто може бути значна різниця в кількості фейкових та правдивих новин. Це може призвести до некоректної класифікації або перекрученого навчання моделей.

В даній роботі ми використовуємо поєднання декількох датасетів з різних джерел (таких як Kaggle, McIntire, Reuters, BuzzFeed Political) для того, щоб досягти високого об’єму даних, водночас не поступаючись якістю розмітки. Дані датасети містять велику кількість джерел -- як надійних, так і відомих за публікацію свідомо некоректних, хибних або вигаданих подій чи оцінок. Завдяки високій загальній кількості ми уникаємо перетренування на стилістиці конкретних ресурсів, замість цього виділяючи ознаки, які притаманні сумнівним джерелами як таким.

Незважаючи на те, що даний підхід не уникає повністю певної суб'єктивності тренувальних даних, він дає достатньо надійну і якісну базу, яка викликає мінімум заперечень і добре підходить для задач класифікації, у тому числі машинного навчання.

РОЗДІЛ 2 NLP-МОДЕЛІ; ІНСТРУМЕНТИ СТВОРЕННЯ МОДЕЛЕЙ NLP

1. Використані фреймворки

Існує кілька популярних бібліотек для реалізації класифікації текстів та навчання моделей глибокого навчання. Серед них ми надали увагу наступним:

TensorFlow [6] є однією з найпопулярніших бібліотек глибокого навчання, яка надає широкий спектр інструментів для роботи з текстовими даними. Вона має вбудовані функції для побудови моделей, оптимізаторів, функцій втрат та інших компонентів, необхідних для класифікації текстів. Вона (а також її близькі аналоги Theano та CNTK) часто використовується у поєднанні з **Keras**, який надає простий інтерфейс для розробки моделей класифікації текстів.

Spacy є бібліотекою для обробки природної мови, яка надає різноманітні інструменти для роботи з текстовими даними. Вона має функції для токенізації, лематизації, видалення стоп-слів та багато інших операцій, які можна використовувати для попередньої обробки тексту перед класифікацією.

Scikit-learn є бібліотекою машинного навчання, яка містить багато алгоритмів класифікації, включаючи ті, що використовуються для аналізу текстових даних. Вона має інструменти для попередньої обробки тексту, векторизації та вибору ознак, а також реалізацію різних класифікаторів.

Дана робота використовує Keras [5] для побудови моделей з використанням Tensorflow як бекенду. Ми також використовуємо Spacy для передобробки та

первинного аналізу датасету і Scikit-learn для рандомізації датасету, а також створення наївного класифікатору у якості baseline.

2. Моделі нейромереж. SNN, LSTM

В даній роботі ми порівнюємо моделі двох типів - SNN (Sequential Neural Network) та LSTM (Long-Short Term Memory Neural Network).

Sequential Neural Network, або послідовна нейромережа - це тип нейромережі, де шари розташовані послідовно один за одним. Це найпростіший тип нейромережі, де інформація передається через шари у прямому напрямку без циклічних зв'язків. Кожен шар приймає вихідні дані від попереднього шару і передає свої вихідні дані наступному шару. Це дозволяє моделі автоматично вивчати репрезентацію даних на різних рівнях абстракції.

Основною перевагою SNN є її простота. Модель побудована у вигляді послідовного списку шарів, що полегшує створення, навчання та використання. Крім того, SNN є досить ефективною для багатьох задач, зокрема тих, де дані можна розглядати як послідовності.

Серед недоліків можна виділити втрату контексту слів, проблеми з розумінням семантики, а також проблеми роботи з рідкісною або спеціалізованою термінологією. Також можуть виникати проблеми з обробкою текстів значного розміру - чим довше текст, тим більше пам'яті потрібно для обробки повною мірою.

LSTM (Long Short-Term Memory) - це тип рекурентних нейромереж, який був розроблений для розв'язання проблеми зниклого градієнту в класичних рекурентних нейромережах. Вони є потужним інструментом для моделювання послідовностей та обробки тексту.

У LSTM використовується спеціальна архітектура, що дозволяє моделі зберігати, оновлювати та використовувати інформацію на тривалий період часу. Вони мають внутрішні структури, які називаються "воротами" (gates), що дозволяють контролювати потік інформації через нейромережу. [2][3]

Перевага LSTM порівняно з SNN полягає в можливості більш ефективної обробки контексту. "Ворота" дозволяють контролювати, яку інформацію треба зберігати в пам'яті, яку інформацію треба забути і яку інформацію треба вивести з моделі. Однак LSTM мають вищу обчислювальну складність порівняно з SNN, а також відрізняється значною чутливістю до перенавчання.

3. Архітектура використаних моделей

Перша модель, яку ми тренуємо та тестуємо, має таку структуру:

Таблиця 2.3.1. SNN - зменшена версія

Назва шару	Вихідна форма	Кількість параметрів шару
Embedding	(None, 512, 16)	8000
GlobalAveragePooling1D	(None, 16)	0
Dense	(None, 6)	102
Dense	(None, 1)	7
Всього параметрів		= 8109

В даній таблиці бачимо, що модель складається з чотирьох шарів. Розглянемо роль кожного з них.

Embedding Layer (Вбудовувальний шар): Цей шар використовується для перетворення вхідного тексту (послідовності токенів) в векторну форму.

Кожен токен тексту буде представлений як вектор фіксованої довжини, що дозволяє моделі враховувати семантику та синтаксис тексту. У даному випадку вбудовувальний шар має розмірність виходу (Output Shape) (None, 512, 16), де None відповідає розмірності партії (batch size), 512 - кількості токенів у вхідній послідовності, а 16 - розмірності векторів вбудовування.
Global Average Pooling1D Layer (Глобальний середній пулінг): Цей шар виконує операцію глобального середнього пулінгу на виході з вбудовувального шару. Він робить усереднення значень вбудовування для кожного текстового документа та повертає вектор фіксованої довжини. У даному випадку вихідний розмір (Output Shape) становить (None, 16). Тут None відповідає розмірності партії (batch size) - кількість прикладів даних, які одночасно передаються моделі для обчислення градієнтів і виконання одного кроку навчання, і може змінюватися в залежності від параметрів тренування. У свою чергу, 16 відповідає розмірності глобально усередненого вектора.
Dense Layer (Повнозв'язний шар): Цей шар є повнозв'язним шаром нейронів, який приймає на вхід глобально усереднений вектор і виконує лінійну трансформацію за допомогою вагових параметрів. У даному випадку вихідний розмір (Output Shape) становить (None, 6), де None відповідає розмірності партії, а 6 - кількості вихідних нейронів у повнозв'язаному шарі.
Dense Layer (Повнозв'язний шар): Цей шар також є повнозв'язним шаром, але з одним вихідним нейроном, оскільки ми маємо завдання бінарної класифікації (фейкова/правдива новина). Він використовує активаційну функцію для визначення ймовірності фейкової новини на основі виходу з попереднього повнозв'язаного шару.
Аналогічна за дизайном модель більшого розміру виглядає таким чином:

Таблиця 2.3.2. SNN - збільшена версія

Назва шару	Вихідна форма	Кількість параметрів шару
Embedding	(None, 512, 64)	32000
GlobalAveragePooling1D	(None, 16)	0
Dense	(None, 16)	1040
Dense	(None, 4)	68
Dense	(None, 1)	5
Всього параметрів		= 33113

LSTM-модель, яку ми використовуємо, має наступну структуру:

Таблиця 2.3.3. LSTM - зменшена версія

Назва шару	Вихідна форма	Кількість параметрів шару
Embedding	(None, 512, 16)	8000
Bidirectional	(None, 512, 8)	672
Bidirectional	(None, 4)	176
Dense	(None, 2)	10
Dense	(None, 1)	3
Всього параметрів		= 8861

Розглянемо роль кожного з шарів:

Embedding Layer (Вбудовувальний шар): для перетворення вхідного тексту (послідовності токенів) в векторну форму. Кожен токен тексту буде представлений як вектор фіксованої довжини. В даному випадку розмірність виходу (Output Shape) (None, 512, 16), де None відповідає розмірності партії (batch size).

Bidirectional Layer (Багатонасичений шар): Цей шар є багатонасиченим (послідовним) шаром, який оперує над вбудовуваннями в обидва напрямки (вперед і назад). Він допомагає моделі захоплювати контекст та залежності між словами у тексті. У даному випадку вихідний розмір (Output Shape) становить (None, 512, 32), де None відповідає розмірності партії, 512 - кількості токенів, а 32 - розмірності вихідних векторів. Ми маємо два таких шари в нашій моделі.

Dense Layer (Повнозв'язний шар): Перший шар є повнозв'язним шаром нейронів, який виконує лінійну трансформацію. Другий шар виконує активаційну функцію.

Збільшена модель:

Таблиця 2.3.4. LSTM - збільшена версія

Назва шару	Вихідна форма	Кількість параметрів шару
Embedding	(None, 512, 32)	16000
Bidirectional	(None, 512, 64)	16640
Bidirectional	(None, 8)	2208
Dense	(None, 8)	72
Dropout	(None, 8)	0

Dense	(None, 1)	9
Всього параметрів		= 34929

Дана версія відрізняється тим, що також використовує шар **Dropout (Викидання)** для фільтрації. Dropout є методом регуляризації, в якому випадковим чином вимикаються певні нейрони у кожному кроці навчання. Це означає, що в процесі навчання деякі нейрони тимчасово призупиняють свою роботу, але з урахуванням випадковості, тому модель не може надмірно спиратися на конкретні нейрони. [9]

Це потрібне для уникання перенавчання для даної моделі, а також для зменшення залежності від окремих нейронів.

Незважаючи на те, що LSTM класично показують кращі результаті при більшій кількості параметрів, ніж та, яку ми розглядаємо, моделі були обрані приблизно однакового розміру для "чистоти" порівняння, а також через технічні обмеження за пам’яттю та максимальною довжиною безперервної роботи, що підтримується у середовищи. Як ми побачимо далі, даного розміру достатньо для того, щоб досягти високих практичних результатів.

РОЗДІЛ 3 ПОРІВНЯННЯ МОДЕЛЕЙ КЛАСИФІКАТОРІВ ТА ЇХНЬОЇ ЕФЕКТИВНОСТІ

3.1 Аналіз вхідних даних

Датасет, який ми використовуємо, є поєднанням декількох вільних для використання датасетів [10][11][12] і містить понад 70,000 новин. Спробуємо визначити деякі характеристики нашого датасету.

```
import seaborn
import matplotlib.pyplot as plt
plt.figure(figsize = (20,6))
seaborn.histplot(df['text'].apply(lambda x: len(x.split())), bins =
range(1, 3000, 50), palette = 'Set1', alpha = 0.8)
plt.ylabel("кількість текстів")
plt.xlabel("довжина тексту")
plt.show()
```

Бачимо, що середня довжина новини має близько 450 слів. На графіку можемо побачити розподіл довжин вхідних текстів у повному датасеті.

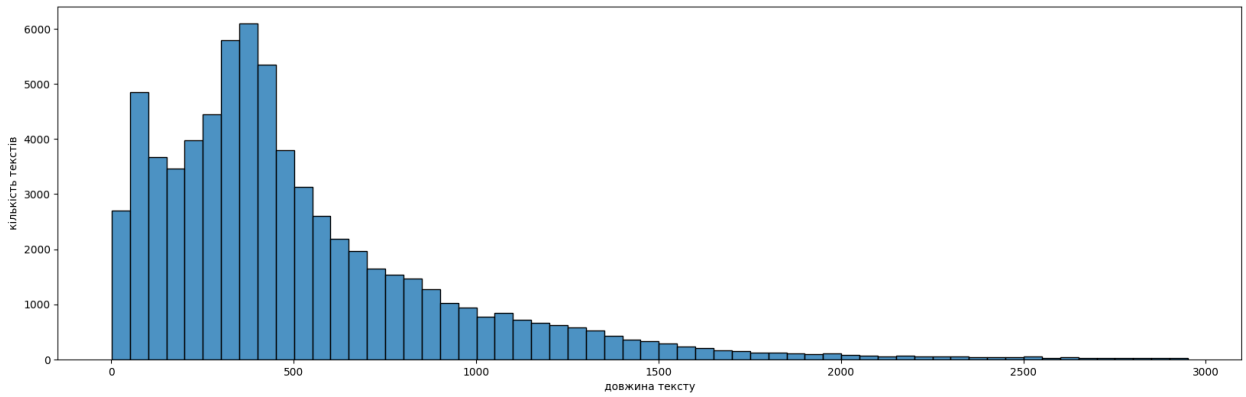

Рисунок 2.1 - Розподіл довжин текстів новин згідно з їхньою кількістю

Визначимо розподіл міток.

```
df[df['label'] == 1].info()
```

повертає нам

Int64Index: 36509 entries, 0 to 71536

Тобто ми бачимо, що серед них близько 36,000 новин є правдивими, що означає, що наш датасет є збалансованим. Збалансований датасет забезпечує рівномірне представлення кожної категорії, дозволяючи моделі краще оцінювати і узагальнювати важливі ознаки для кожного класу; він також дозволяє запобігати викривленню результатів, коли модель стає надто схильною приписувати одну й ту саму категорію до всіх вхідних даних підряд.

Спробуємо визначити найпоширеніші терміни для фейкових та правдивих новин. Для цього, використовуючи інструменти spacy, виділимо найпоширеніші токени, викинувши з них стоп-слова і знаки пунктуації. Стоп-слова - це загальні слова, які не несуть суттєвої інформації про контекст тексту, такі як "і", "в", "на" і т.д., і тому очікувано містяться в текстах обох видів у великої кількості.

```
fake_tokens = [token for token, label in zip(all_tokens, df['label']) if
label == 0]
real_tokens = [token for token, label in zip(all_tokens, df['label']) if
label == 1]

fdist_fake = FreqDist(fake_tokens)
fdist_real = FreqDist(real_tokens)

top_fake_terms = fdist_fake.most_common(100)

for term, frequency in top_fake_terms:
   if term.lower() not in stopwords and any(c.isalpha() for c in term):
       print(term, ":", frequency)

top_real_terms = fdist_real.most_common(100)
for term, frequency in top_real_terms:
   if term.lower() not in stopwords and any(c.isalpha() for c in term):
```

```
        print(term, ":", frequency)
```

Отримуємо такі списки:

Таблиця 3.3.1 - Найпоширеніші терміни залежно від типа новини

Справжні новини		**Фейкові новини**	
Токен	**Частота**	**Токен**	**Частота**
s	288	s	302
said	174	said	170
Trump	127	Trump	118
t	75	t	69
people	69	people	57
like	61	Mr.	56
Mr.	61	like	48
Clinton	48	years	46
years	46	Clinton	42
U.S.	44	time	40
United	37	Hillary	40
government	37	government	35
		de	34

		Donald	33

Бачимо, що в середньому справжні та фейкові новини використовують дуже схожі словники. Це означає, що, скоріше за все, буде складно визначити якість новини за одними лише її найпоширенішими словами, і нейромережі буде потрібно звернути увагу на більш детальні ознаки.
Перевіримо це припущення, створивши наївний байєсівський класифікатор, який у якості вхідних даних використовує bag-of-words ("мішок слів"), тобто дані, які лише підраховують кількість найпопулярніших слів без урахування їхнього порядку в тексті. Це дасть уявлення, яких результатів можна досягти методами одного лише статистичного аналізу тексту і допоможе оцінити кількісний приріст, що досягається за допомогою рішень, що використовують нейронні мережі.

```
vectorizer = CountVectorizer(max_features=N_WORDS)
X = vectorizer.fit_transform(data['text']).toarray()

X_train, X_test, y_train, y_test = train_test_split(X, data['label'],
test_size=0.1, random_state=42)
```

```
naive_bayes = MultinomialNB()
naive_bayes.fit(X_train, y_train)

y_pred = naive_bayes.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

Ми отримуємо точність у 85%. Це достатньо високий результат, що означає, що за межами найпоширеніших слів існує достатня різниця, щоб за даними

ознаками можна було судити щодо істинності або фейковісті новини. Але такого результату буде недостатньо для більшості задач, тому нейромережевий підхід буде тут корисним. Тому наш датасет повинен достатньо добре підходити для перевірки ефективності різних моделей.

3.2 Підготовка датасету

Для підготовки датасету ми розбиваємо його на три категорії - тренування, валідацію та тестування.

```
TRAIN_SHARE, TEST_SHARE, VALIDATION_SHARE = 0.9, 0.05, 0.05
train_len = int(df.shape[0] * TRAIN_SHARE)
test_len = int(df.shape[0] * TEST_SHARE)
val_len = int(df.shape[0] * VALIDATION_SHARE)
```

```
x, x_test, y, y_test = train_test_split(df[['title', 'text']],
df['label'], test_size=0.1, train_size=0.9)
x_train, x_val, y_train, y_val = train_test_split(x, y, test_size = 0.5,
train_size =0.5)
```

Цей підхід має певні переваги порівняно з класичним підходом, де ми розбиваємо датасет лише на дві категорії - а саме, це дозволяє нам уникнути перенавчання на рівні гіперпараметрів, де ми перебираємо можливі моделі, доки не знайдемо таку, яка показує добрі результати саме на нашому датасеті. Третя категорія дозволяє нам мати додаткові "зовнішні" дані, які дають нам змогу впевнитися, що дана модель дійсно добре себе показує на будь-яких даних, а не тільки тих, на яких ми тестували і намагалися максимізувати результат на них.

Далі ми використовуємо токенізатор, що перетворює послідовності слів на числові значення, з якими наша модель може працювати. Для більшої ефективності навчання і уникнення недотренування, а також для того, щоб

зменшити об’єм використаної пам’яті, ми виділяємо з них лише 500 найпоширеніших слів, замінюючи решту спеціальним out-of-vocabulary токеном.

```
N_WORDS = 500
MAX_LEN = 512
tokenizer = Tokenizer(num_words = N_WORDS, oov_token="<OutOfVocabulary>")
tokenizer.fit_on_texts(x_train['text'])
word_index = tokenizer.word_index
```

Після цього залишається лише створити ланцюжки даних.

```
train_seqs = get_sequences(x_train['text'])
test_seqs =  get_sequences(x_test['text'])
val_seqs =  get_sequences(x_val['text'])
```

Ми також підготуємо альтернативну версію датасету, з якої видалені всі власні імена і замінені категорії (ім’я людини, локація і т.д.), виділені за допомогою Spacy. Це дозволяє кращу генералізацію текстів, оскільки власні імена часто унікальні і сильно відрізняються від статті до статті, на відміну від категорій власних імен, а також дозволяє уникнути упередження щодо конкретної тематики новин і зосередитися саме на стилістичних особливостях фейків.

```
tqdm.pandas()
df_no_proper = df.progress_apply(remove_proper_names, axis=1,
result_type='expand')
df_no_proper.to_csv('merged_dataset_proper_nouns_scrubbed.csv')
```

Датасет, що не містить власних імен, проходить аналогічний процес обробки та токенізації. На даному датасеті наївний класифікатор показує точність у 82%.

3.3 Продуктивність моделей на текстах з мінімальною передобробкою

Мала версія моделі SNN є достатньою, щоб показати достатньо високі результати -- на тестовому датасеті вона показує точність в 94%, що є достатнім для багатьох прикладних задач [4]. Однак дана модель дуже швидко досягає піка своїх можливостей, і після перших декількох епох виходить на плато своєї ефективності.

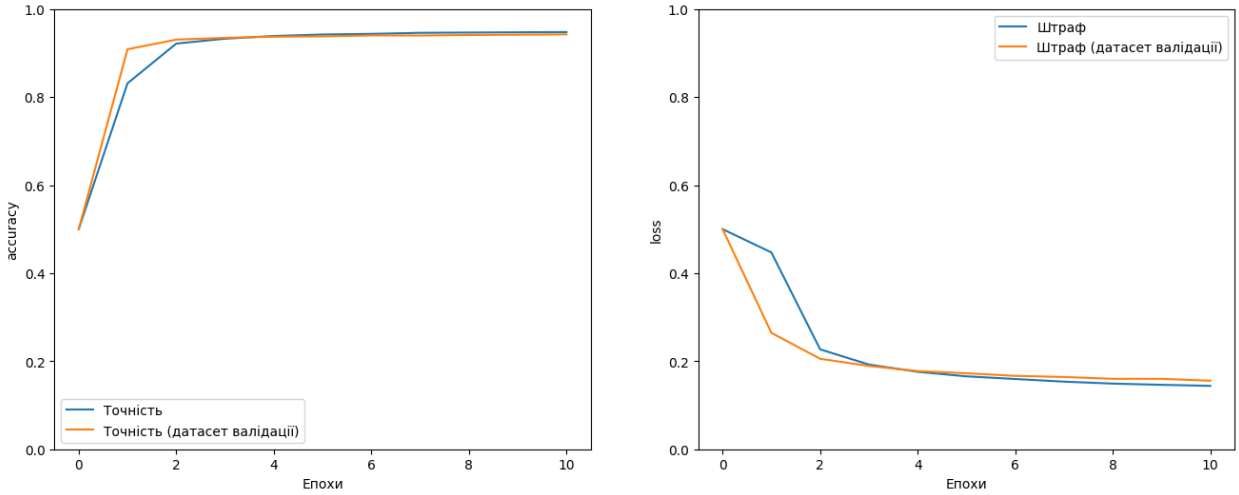

Рисунок 3.3.1 - Результати моделі SNN (зменшена версія) на кожній ітерації (epoch):

а) точність (accuracy)

б) штраф (loss)

Ми бачимо, що збільшення розміру моделі майже не впливає на результати - іншими словами, навіть менша версія модели досягає ліміту того, що можна виділити лише за допомогою виділення послідовностей лінійних залежностей, як-то з повнозв'язними шарами. Проблема швидкого насичення моделі залишається у силі.

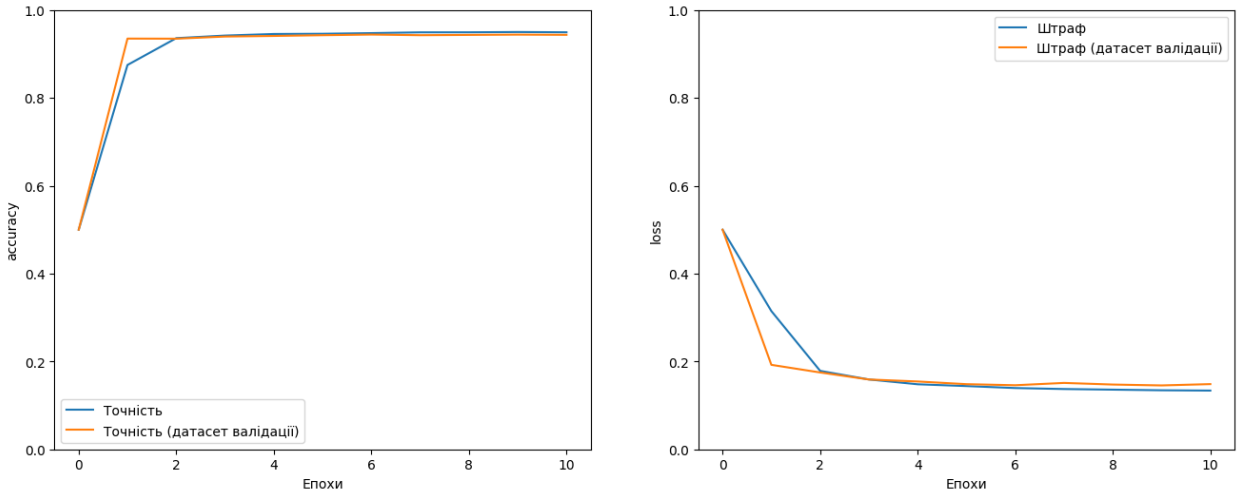

Рисунок 3.3.2 - Результати моделі SNN (збільшена версія) на кожній ітерації

Використання LSTM показує змішаний ефект. Так, у випадку моделі з близько 8к параметрів ми бачимо ефект перетренування - починаючи з деякого моменту, точність на валідаційному датасеті починає падати. Загальна точність на тестовому датасеті 90%.

Це може бути пояснено тим, що для складнішої архітектури, такої як LSTM, даного розміру моделі недостатньо для того, щоб повністю використати потенціал даного дизайну. У випадку врахування контексту ми зіткаємося з більшою кількістю можливих особливостей тексту та його варіацій, ніж модель здатна запам’ятати.

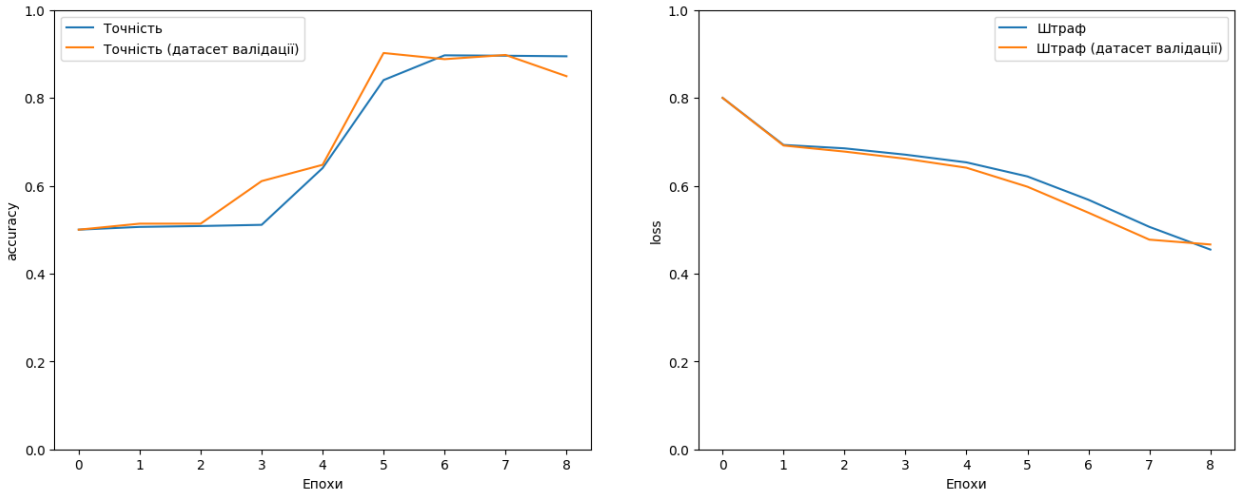

Рисунок 3.3.3 - Результати моделі LSTM (зменшена версія) на кожній ітерації

Але у випадку збільшення моделі і додавання Dropout для протистояння перетренуванню ми отримуємо найкращий з усіх отриманих результатів - точність на тестовому датасеті досягає 96%. Це показує здатність LSTM адаптуватися до контексту і враховувати його там, де послідовна модель на це нездатна.

Ми також бачимо, що незважаючи на значно вищу точність, модель також зберігає значно вищий loss. Це може мати кілька пояснень - наприклад, незважаючи на в цілому точні передбачення, модель має нижчу впевненість, або показує високу впевненість в небагатьох невірних відповідях.

Загалом це може показувати проблеми з розміткою датасету або нерепрезентативністю тренувальних даних, але у випадку перевіреного датасету і якісної рандомізації даних це, скоріше за все, є ознакою перехідного ефекту. В такому випадку, модель може показати кращі результати на подальшому дотренуванні.

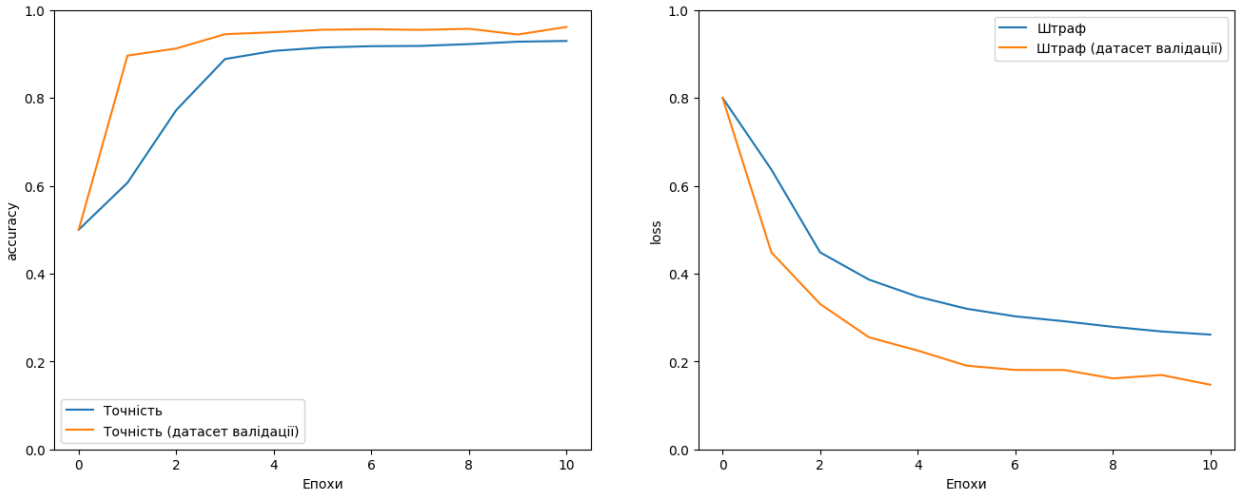

Рисунок 3.3.4 - Результати моделі LSTM (збільшена версія) на кожній ітерації

3.4 Продуктивність моделей на текстах, з яких були видалені власні імена

На етапі аналізу текстових даних ми бачили, що словниковий запас справжніх та фейкових новин є достатньо схожим - у тому числі це стосується і власних імен. Однак ми розглядали переважно лише найпопулярніші терміни з кожного тексту. Що ми побачимо у випадку, якщо ми видаляємо власні імена і змушуємо нейромережі брати до уваги виключно стилістичні та семантичні ознаки аналізуємих текстів?

Виявляєвляться, що загальний перформанс моделей падає, але незначно - в середньому, кожне з розглянутих рішень, що використовують нейронні мережі, досягає цілком задовільної точності.

Так, на малій SNN-моделі ми досягаємо такого ж самого резульату у 91%, з швидким насиченням перформансу і виходом на плато аналогічним тому, який ми спостерігали у попередній версії.

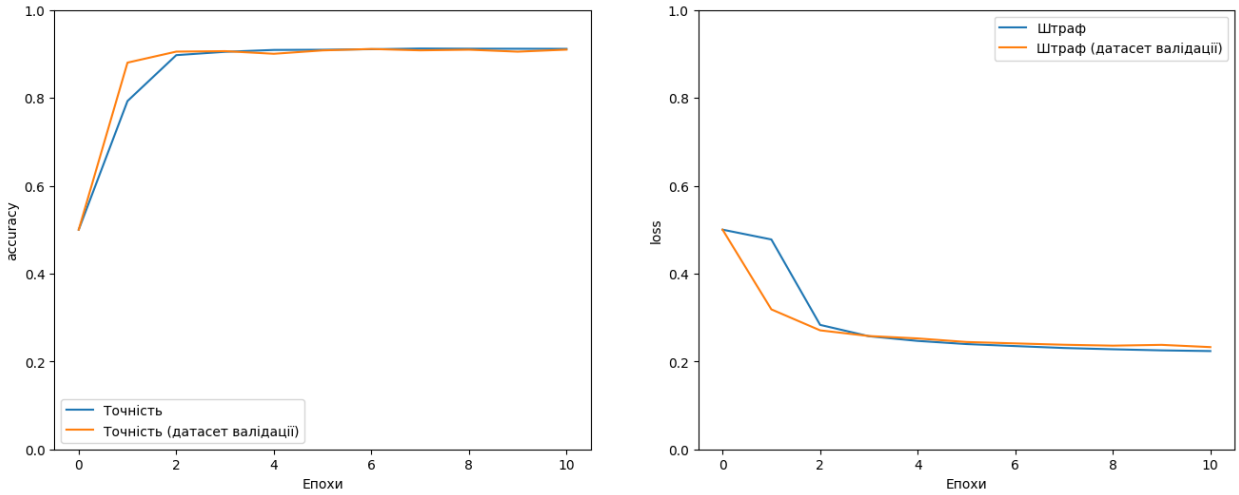

Рисунок 3.4.1 - Результати моделі SNN (зменшена версія) на кожній ітерації на зміненому датасеті

Збільшення моделі на цей раз не дає жодного покращення, залишаючись на планці у 91%.

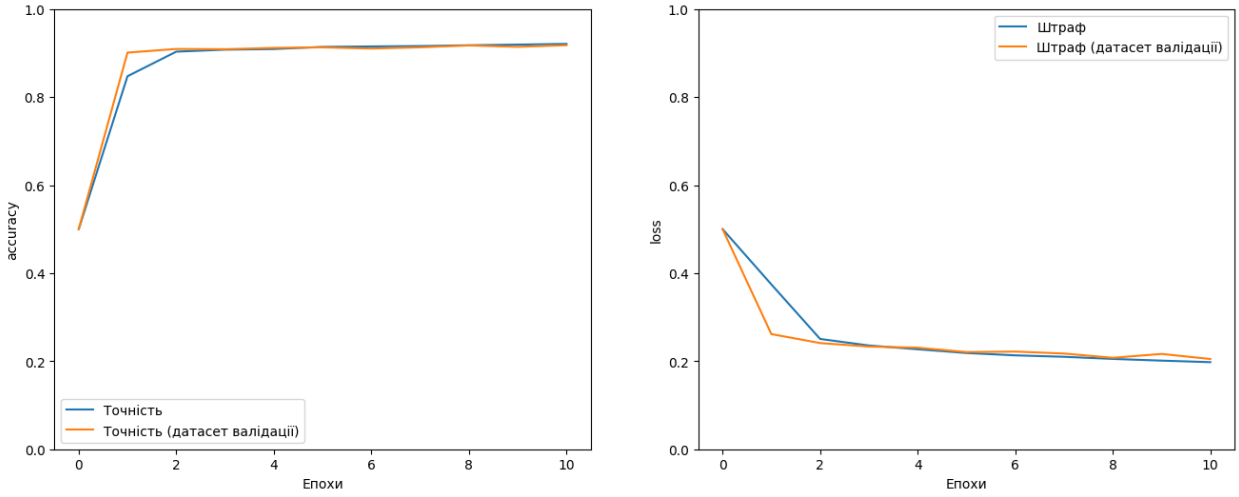

Рисунок 3.4.2 - Результати моделі SNN (збільшена версія) на кожній ітерації на зміненому датасеті

Схожі результати ми отримуємо від LSTM - на цей раз ми уникаємо проблеми перетренування завдяки меншій різноманітності датасету, досягаючи загальної точності у 92%.

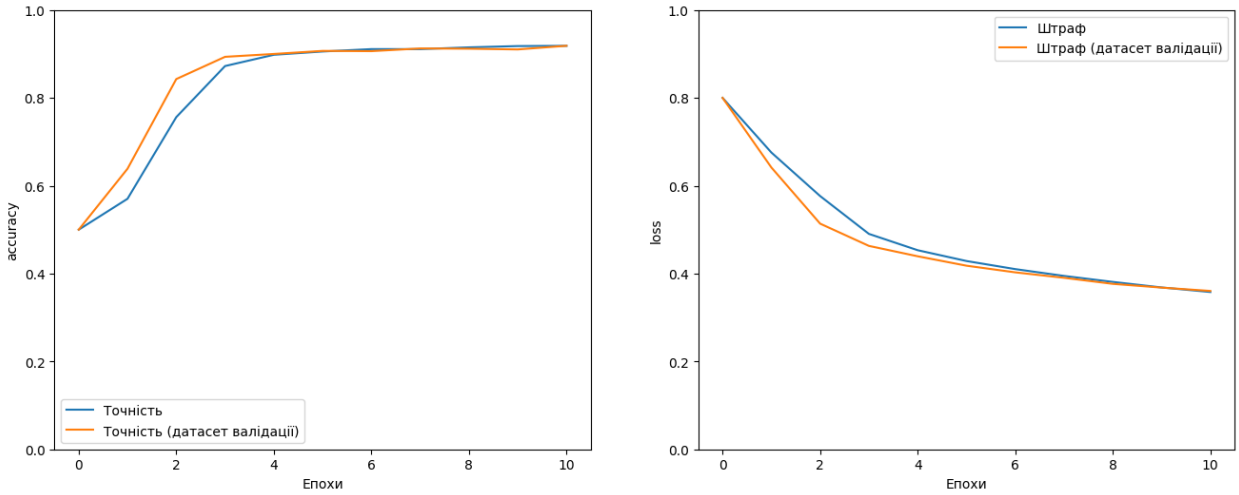

Рисунок 3.4.3 - Результати моделі LSTM (зменшена версія) на кожній ітерації на зміненому датасеті

Але подібна втрата нюансу має негативний ефект на більшій моделі, що досягає цього разу лише 94%. Зауважимо однак, що і цього разу дана модель показує найкращий результат з усіх, показуючи важливість врахування широкого контексту.

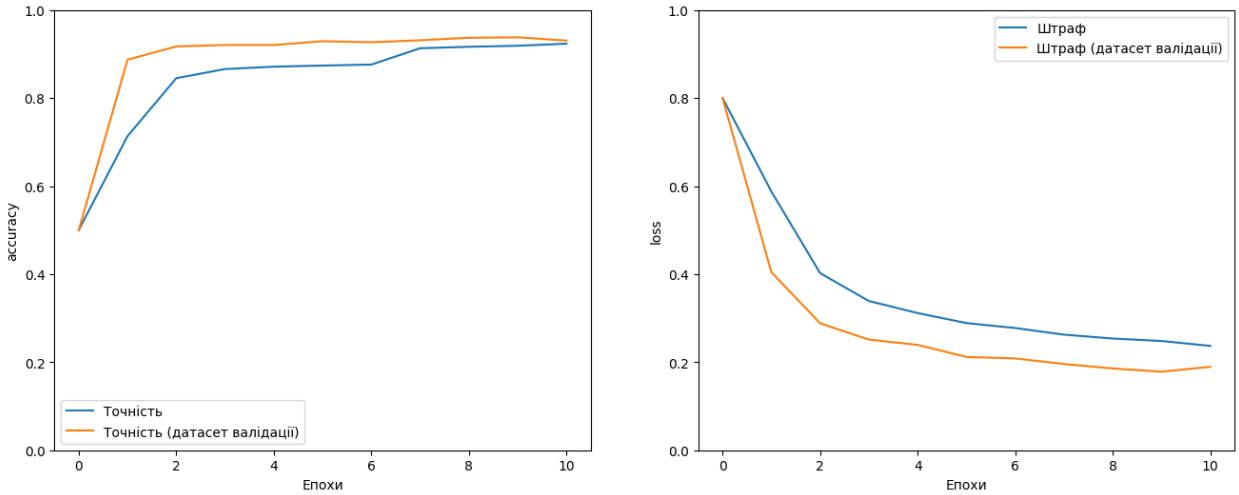

Рисунок 3.4.4 - Результати моделі LSTM (збільшена версія) на кожній ітерації на зміненому датасеті

3.5 Порівняння результатів

Таблиця 3.5.1 - Сумарні результати роботи кожного алгоритму

Модель	Датасет має власні імена		Датасет не має власних імен	
	Точність	Штраф	Точність	Штраф
Наївний Байєс	0.85	-	0.82	-
SNN - мала	0.91	0.236	0.91	0.23
SNN - велика	0.94	0.143	0.91	0.20
LSTM - мала	0.90	0.599	0.92	0.36
LSTM - велика	0.96	0.143	0.94	0.18

Сумаризуємо наші результати.
Модель SNN є достатньою для багатьох прикладних задач. Однак вона швидко досягає свого потенціалу і виходить на плато.

Використання LSTM має змішаний ефект, оскільки менші моделі показують ознаки перетренування. Збільшення розміру моделі та використання шару викидання допомагає запобігти перетренуванню і підвищити точність. Однак, ці моделі можуть мати вищий loss, що може бути пов'язано з недостатньою впевненістю моделі або проблемами розмітки датасету.

Видалення власних імен з текстів призводить до зниження загального перформансу моделей, але незначно. Використання SNN та LSTM показує точність в межах 91-94%. Збільшення розміру моделі не призводить до поліпшення результатів. LSTM-модель з урахуванням контексту показує найкращий результат з точністю 96%, що є високим теоретичним результатом [1] [4].

3.6 Потенціал для подальшого дослідження

Існує декілька можливих напрямків, які можуть потенційно допомогти перформансу розглянутих моделей.

Перш за все, на етапі аналізу тексту ми мали змогу спостерігати, що найпоширеніші слова та вирази переважно містять велику кількість стоп-слів. Видалення стоп-слів може допомогти зосередитися на більш суттєвих термінах і поліпшити якість класифікації.

По-друге, ми бачимо, що збільшена версія LSTM має потенціал для поліпшення. Довше тренування може дозволити даній моделі краще усвідомити та узагальнити особливості в текстах, що може призвести до поліпшення її точності.

До покращення може призвести і збільшення самої моделі, хоча варто зазначити, що це поліпшення буде виникати за рахунок швидкості і простоти створення та тренування моделі, що на сьогодні вимагає значних ресурсів.

Нарешті, різні моделі, що ми розглянули, мають різні "сліпі плями" і різні умови виникнення помилок. В такому випадку може бути слушним поєднати наші моделі у єдиний ансамбль з різних типів або конфігурацій моделей для досягнення кращої точності.

Кожен з цих напрямків розвитку може внести свій внесок у поліпшення точності і результатів моделі класифікації тексту, але для оцінки їхньої ефективності необхідне подальше експериментування.

ВИСНОВКИ

Для досягнення поставленої мети були зроблені такі кроки:

- Аналіз існуючих методів та моделей класифікації текстів, спрямованих на розпізнавання фейкових новин, зокрема застосування правилових систем, статистичних методів та глибокого навчання. Систематизація їхніх особливостей, переваг та недоліків з метою виокремлення найкращих підходів та практик.

- Аналіз доступних корпусів даних, які містять фейкові та достовірні новини, та визначення найбільш прийнятних для навчання та оцінки моделей розпізнавання. Розгляд різноманітних джерел даних, враховуючи їх розмаїтість та репрезентативність.

- Дослідження лінгвістичних особливостей фейкових новин, виявлення їх характерних ознак, які можуть служити важливими ознаками для класифікації.

- Експериментальна перевірка різних моделей та алгоритмів машинного навчання для класифікації текстів. Розробка рекомендації щодо підвищення ефективності моделей та їхнього застосування у реальних умовах. Визначення можливих шляхів удосконалення класифікації.

Загалом ми можемо зробити висновок, що машинне навчання має великий потенціал для класифікації текстів, зокрема для виявлення фейкових новин. Ефективність моделей може залежати від розміру та складності датасету, а також від загальної репрезентативності тренувальних даних. При цьому використання глибокого навчання, зокрема LSTM моделей, може бути

особливо корисним для аналізу текстових даних, оскільки вони здатні враховувати широкий контекст та семантичні залежності. Перехідні ефекти, такі як насичення моделі або зміна результатів при використанні різних оброблених даних, можуть спостерігатись та варто їх враховувати.

Продовження досліджень та розробок у цій галузі може призвести до подальшого вдосконалення моделей та покращення їх ефективності у реальних прикладних задачах. Порівняння та об'єднання різних підходів та методик може допомогти виявити найефективніші та надійніші рішення для класифікації текстів в різноманітних задач як теоретичного, так і прикладного характеру.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Sneha Singhania, Nigel Fernandez, Shrisha Rao. 3HAN: A Deep Neural Network for Fake News Detection, 2015. Режим доступу: https://link.springer.com/chapter/10.1007/978-3-319-70096-0_59
2. Pritika Bahad, Preeti Saxena, Raj Kamal. Fake News Detection using Bi-directional LSTM-Recurrent Neural Network, 2019. doi.org/10.1016/j.procs.2020.01.072
3. Смілянець Ф.А., Мажара О.О. Класифікація новин за достовірністю на основі методів машинного навчання, 2020 / Всеукраїнська науково-практична конференція молодих вчених та студентів «Інформаційні системи та технології управління – ІСТУ-2020» – Київ, 2020.с
4. Patil, D.R. Fake News Detection Using Majority Voting Technique, 2022. – 20 с. – (Препринт / ArXiv, abs/2203.09936)
5. François Chollet та ін. Keras. 2015. Режим доступу: https://keras.io
6. Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo та ін. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Режим доступу: https://www.tensorflow.org
7. McKinney, W. та ін. Data structures for statistical computing in python, 2007. In Proceedings of the 9th Python in Science Conference (Vol. 445, pp. 51–56). doi.org/10.25080/majora-92bf1922-00a
8. Ковбасюк О. М., Грицюк І. Ю. Виявлення фейкових новин методами машинного навчання, 2022. / Інформаційні технології в освіті та практиці. Матеріали науково-практичної конференції. – 80 с.
9. P Baldi, PJ Sadowski. Understanding Dropout. Advances in Neural Information Processing Systems 26 (NIPS 2013).

10. Fake News Classification. IEEE Transactions on Computational Social Systems: pp. 1-13 doi.org/10.1109/TCSS.2021.3068519
11. Ahmed H, Traore I, Saad S. “Detecting opinion spams and fake news using text classification”, Journal of Security and Privacy, Volume 1, Issue 1, Wiley, January/February 2018.
12. Ahmed H, Traore I, Saad S. “Detection of Online Fake News Using N-Gram Analysis and Machine Learning Techniques. In: Traore I., Woungang I., Awad A. (eds) Intelligent, Secure, and Dependable Systems in Distributed and Cloud Environments. ISDDC 2017. Lecture Notes in Computer Science, vol 10618. Springer, Cham (pp. 127-138).

ДОДАТОК А. ПРОГРАМНА РЕАЛІЗАЦІЯ

```
import tensorflow as tf
from tensorflow.keras.preprocessing.text import Tokenizer
import pandas as pd
import numpy as np
from tensorflow.keras.preprocessing.sequence import pad_sequences
import matplotlib.pyplot as plt
from keras.layers import Input, Dense, LSTM, Bidirectional, Dropout,
Embedding, GlobalAveragePooling1D
from keras.callbacks import EarlyStopping
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from tqdm import tqdm
from google.colab import drive
import csv
import spacy
import seaborn
```

```
drive.mount('/content/drive')

df1 = pd.read_csv('WELFake_Dataset.csv')
df2 = pd.read_csv('news_articles.csv')[['title', 'text','label']]
df2['label'] = df2['label'] == 'Real'
df = pd.concat([df1, df2])
df.dropna(inplace=True)
df.to_csv('dataset_merged.csv')
```

```
plt.figure(figsize = (20,6))
seaborn.histplot(df['text'].apply(lambda x: len(x.split())), bins =
range(1, 3000, 50), palette = 'Set1', alpha = 0.8)
plt.ylabel("кількість текстів")
plt.xlabel("довжина тексту")
plt.show()
```

```
all_tokens = []
for text in tqdm(df['text']):
   tokens = word_tokenize(text)
   all_tokens.extend(tokens)
stopwords = en.Defaults.stop_words
fake_tokens = [token for token, label in zip(all_tokens, df['label']) if
label == 0]
real_tokens = [token for token, label in zip(all_tokens, df['label']) if
label == 1]

fdist_fake = FreqDist(fake_tokens)
fdist_real = FreqDist(real_tokens)

top_fake_terms = fdist_fake.most_common(100)
print("Top 10 terms for fake news:")
for term, frequency in top_fake_terms:
   if term.lower() not in stopwords and any(c.isalpha() for c in term):
       print(term, ":", frequency)
top_real_terms = fdist_real.most_common(100)
print("Top 10 terms for real news:")
for term, frequency in top_real_terms:
   if term.lower() not in stopwords and any(c.isalpha() for c in term):
       print(term, ":", frequency)
```

```
nlp = spacy.load("en_core_web_sm")

def remove_proper_names(description):
   text = description["text"]
   if text and type(text) == str:
       doc = nlp(text)
       for ent in doc.ents:
           text = text.replace(ent.text, ent.label_)
       description["text"] = text

   title = description["title"]
   if title and type(title) == str:
       doc = nlp(title)
       for ent in doc.ents:
           title = title.replace(ent.text, ent.label_)
```

```
        description["title"] = title

    return description

tqdm.pandas()
df_no_proper = df.progress_apply(remove_proper_names, axis=1,
result_type='expand')
df_no_proper.to_csv('merged_dataset_proper_nouns_scrubbed.csv')
```

```
df = pd.read_csv('dataset_merged.csv')
```

```
TRAIN_SHARE, TEST_SHARE, VALIDATION_SHARE = 0.9, 0.05, 0.05
train_len = int(df.shape[0] * TRAIN_SHARE)
test_len = int(df.shape[0] * TEST_SHARE)
val_len = int(df.shape[0] * VALIDATION_SHARE)
```

```
x, x_test, y, y_test = train_test_split(df[['title', 'text']],
df['label'], test_size=0.1, train_size=0.9)
x_train, x_val, y_train, y_val = train_test_split(x, y, test_size = 0.5,
train_size =0.5)
```

```
N_WORDS = 500
MAX_LEN = 512
tokenizer = Tokenizer(num_words = N_WORDS, oov_token="<OutOfVocabulary>")
tokenizer.fit_on_texts(x_train['text'])
word_index = tokenizer.word_index
```

```
# Naive Bayesian bag-of-words classifier as a baseline
vectorizer = CountVectorizer(max_features=N_WORDS)
X = vectorizer.fit_transform(data['text']).toarray()

X_train, X_test, y_train, y_test = train_test_split(X, data['label'],
test_size=0.1, random_state=42)
```

```
naive_bayes = MultinomialNB()
naive_bayes.fit(X_train, y_train)
```

```
y_pred = naive_bayes.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

```
def get_sequences(texts):
 sequences = tokenizer.texts_to_sequences(texts)
 padded = pad_sequences(sequences, maxlen=MAX_LEN, padding='post',
truncating='post')
 return padded
train_seqs = get_sequences(x_train['text'])
test_seqs =  get_sequences(x_test['text'])
val_seqs =  get_sequences(x_val['text'])
```

```
model = tf.keras.Sequential([
   Embedding(N_WORDS, 16, input_length=MAX_LEN),
   GlobalAveragePooling1D(),
   Dense(6, activation='relu'),
   Dense(1, activation='sigmoid')
])
model.compile(loss='binary_crossentropy',optimizer='adam',metrics=['accura
cy'])
model.summary()
```

```
history = model.fit(train_seqs,
                    y_train.astype('float32'),
                    epochs=10,
                    validation_data=(val_seqs.astype('float32'),
y_val.astype('float32')))
```

```
def plot_history(history, metric):
   plt.plot([.5] + history.history[metric])
   plt.plot([.5] + history.history['val_'+metric], '')
   plt.xlabel("Епохи")
   plt.ylabel(metric)
   plt.ylim([0, 1])
   plt.legend([('Точність' if metric == 'accuracy' else 'Штраф'),
```

```
    ('Точність' if metric == 'accuracy' else 'Штраф') +' (датасет
валідації)']
plt.figure(figsize=(16, 6))
plt.subplot(1, 2, 1)
plot_history(history, 'accuracy')
plt.subplot(1, 2, 2)
plot_history(history, 'loss')
```

```
model.evaluate(test_seqs.astype('float32'), y_test.astype('float32'))
```

```
model.save('dense-16-6-1-larger-dataset-proper-excluded.h5')
```

```
model_dense_larger = tf.keras.Sequential([
   Embedding(N_WORDS, 64, input_length=MAX_LEN),
   GlobalAveragePooling1D(),
   Dense(16, activation='relu'),
   Dense(4, activation='relu'),
   Dense(1, activation='sigmoid')
])
model_dense_larger.compile(loss='binary_crossentropy',optimizer='adam',met
rics=['accuracy'])
model_dense_larger.summary()
```

```
history_dense = model_dense_larger.fit(train_seqs,
                    y_train.astype('float32'),
                    epochs=10,
                    validation_data=(val_seqs.astype('float32'),
y_val.astype('float32')))
```

```
plt.figure(figsize=(16, 6))
plt.subplot(1, 2, 1)
plot_history(history_dense, 'accuracy')
plt.subplot(1, 2, 2)
plot_history(history_dense, 'loss')
```

```
model_dense_larger.evaluate(test_seqs.astype('float32'),
y_test.astype('float32'))
```

```
model_dense_larger.save('dense-64-4-1-larger-dataset-proper-excluded.h5')
```

```
model_lstm = tf.keras.Sequential([
   Input(name='input',shape=[MAX_LEN]),
   Embedding(N_WORDS, 16),
   Bidirectional(tf.keras.layers.LSTM(4,  return_sequences=True)),
   Bidirectional(tf.keras.layers.LSTM(2)),
   Dense(2, activation='relu'),
   Dense(1, activation='sigmoid')
])
model_lstm.compile(loss=tf.keras.losses.BinaryCrossentropy(),
             optimizer=tf.keras.optimizers.Adam(1e-4),
             metrics=['accuracy'])
model_lstm.summary()
```

```
history_lstm = model_lstm.fit(train_seqs,
                   y_train.astype('float32'),
                   epochs=10,
                   validation_data=(val_seqs, y_val.astype('float32')),
                   batch_size = 64,

callbacks=[EarlyStopping(monitor='val_accuracy',mode='max', patience=3,
verbose=False,restore_best_weights=True)]
                   )
```

```
plt.figure(figsize=(16, 6))
plt.subplot(1, 2, 1)
plot_history(history_lstm, 'accuracy')
plt.subplot(1, 2, 2)
plot_history(history_lstm, 'loss')
```

```
model_lstm.evaluate(test_seqs, y_test.astype('float32'))
```

```
model_lstm.save('lstm-larger-dataset-proper-excluded.h5')
```

```
model_lstm_largel = tf.keras.Sequential([
   Input(name='input',shape=[MAX_LEN]),
   Embedding(N_WORDS, 32),
   Bidirectional(tf.keras.layers.LSTM(32,  return_sequences=True)),
   Bidirectional(tf.keras.layers.LSTM(4)),
   Dense(8, activation='relu'),
   Dropout(0.5),
   Dense(1, activation='sigmoid')
])

model_lstm_largel.compile(loss=tf.keras.losses.BinaryCrossentropy(),
             optimizer=tf.keras.optimizers.Adam(1e-4),
             metrics=['accuracy'])
model_lstm_largel.summary()
```

```
history_lstm_larger = model_lstm_largel.fit(train_seqs,
                  y_train.astype('float32'),
                  epochs=10,
                  validation_data=(val_seqs, y_val.astype('float32')),
                  batch_size = 64,

callbacks=[EarlyStopping(monitor='val_accuracy',mode='max', patience=3,
verbose=False,restore_best_weights=True)]
                  )
```

```
plt.figure(figsize=(16, 6))
plt.subplot(1, 2, 1)
plot_history(history_lstm_larger, 'accuracy')
plt.subplot(1, 2, 2)
plot_history(history_lstm_larger, 'loss')
```

```
model_lstm_largel.evaluate(test_seqs, y_test.astype('float32'))
model_lstm_largel.save('lstm-larger-larger-dataset-proper-excluded.h5')
```