УДК 004.8, 004.67
DOI: 10.18523/2617-3808.2025.8.15-27

Глибовець А. М., Дубовик А. В., Афонін А. О.

ПРОГРАМНА СИСТЕМА КЛАСИФІКАЦІЇ ТЕКСТІВ НА ОСНОВІ МАШИННОГО НАВЧАННЯ ТА РЕКУРЕНТНОЇ НЕЙРОННОЇ МЕРЕЖІ

У цій роботі описано побудову та результати тестування програмної системи автоматичної класифікації текстів, яка полягає в розподілі текстів за певними категоріями, зокрема текстів українською мовою. Наш застосунок побудований на використанні трьох моделей — Naive Bayes, Support Vector Machine, LSTM — архітектури рекурентної нейронної мережі Recurrent Neural Network (RNN) та їх комбінації. Він дає змогу доволі швидко і точно класифікувати тексти, надавати користувачу можливість зручним способом натренувати систему на власних даних і досить просто налаштувати параметри для оптимальних результатів.

Для ефективного опрацювання вхідних даних і реалізації алгоритму класифікації ми вибрали мову програмування Python. Основними бібліотеками реалізації функціоналу застосунку стали TensorFlow, scikit-learn (для надання простого та зрозумілого інтерфейсу), Natural Language Toolkit (nltk), NumPy, Pandas. Matplotlib і seaborn застосовували для візуалізації даних і побудови графіків. Розроблений графічний застосунок здатен розпізнавати тексти (англійською або українською мовою) чотирьох категорій (World, Sports, Science / Technology, Business) з точністю близько 92 %.

Для навчання моделей ми застосували AG News Classification Dataset із kaggle.com.

Тестування застосунку підтвердило припущення, що спеціалізовані моделі, крім того, що є значно ефективнішими в плані використання ресурсів, також можуть демонструвати кращий результат у класифікації текстів, ніж LLM. Система також може бути швидко адаптована й до задачі фільтрації спаму. За декілька секунд можна отримати SVM модель, яка зможе розпізнавати типові спам-повідомлення з точністю близько 99 %. Так само були протестовані можливості системи при розпізнаванні емоційної забарвленості тексту. Вдалося досягти точності 87,75 %.

Ключові слова: автоматична класифікація текстів, Naive Bayes, Support Vector Machine, LSTM, Recurrent Neural Network, машинне навчання, Python, TensorFlow, AG News Classification Dataset.

Вступ

Нині, з розвитком інформаційних технологій і зростанням обсягу доступної інформації, стає надзвичайно важливим ефективне управління аналізом даних. Одним із ключових завдань у цьому контексті є автоматична класифікація текстів, яка полягає в розподілі текстів за певними категоріями. Основні виклики, пов'язані з автоматичною класифікацією текстів, — це різноманітність форматів, структур текстів і семантична складність мови. Для вирішення цих проблем потрібні ефективні алгоритми та методи оброблення природної мови.

Програмні системи (ПС), що використовують технології машинного навчання (МН), такі як нейронні мережі та методи оброблення природної мови (NLP), демонструють високу ефективність у розв'язанні цієї задачі завдяки своїй здатності адаптуватися до нових даних і швидкості оброблення [1; 2; 4; 6].

У контексті оброблення текстів класифікація полягає у призначенні кожному текстовому документу або фрагменту відповідної категорії або класу, що допомагає в організації, розумінні та аналізі великих обсягів інформації [4]. Технології штучного інтелекту, зокрема методи МН, дали можливість значно покращити ефективність і точність класифікації текстів. Автоматична класифікація тексту є одним із фундаментальних завдань оброблення природної мови. Системи автоматичної класифікації текстів (САКТ) можна умовно поділити на такі три групи: системи на основі правил, системи на основі машинного навчання (Naive Bayes, Support Vector Machine (SVM), Recurrent Neural Network (RNN), Bidirectional Encoder Representations from Transformers (BERT), Large Language Models (LLM)), гібридні системи. Коротку їхню характеристику наведено у роботах [13; 15; 18].

Гібридні системи класифікації текстів можуть поєднувати переваги різних підходів і методів для створення більш точних та ефективних моделей [3]. Вони становлять потужний інструмент, який може бути налаштований та оптимізований для конкретних потреб. Такі моделі можуть досягати високої точності та ефективності у різних сценаріях та сферах застосування. Проте один із основних недоліків гібридних систем полягає в їхній складності. Поєднання різних методів і підходів може призвести до значного збільшення складності самої системи, зокрема щодо розробки, розуміння, підтримки та масштабування.

Тому ми вирішили розробити систему, яка зможе доволі швидко і точно класифікувати тексти та надавати користувачу можливість зручним способом натренувати систему на власних даних і налаштувати параметри для оптимальних результатів. У цій статті опишемо цю розробку.

Для ефективного опрацювання вхідних даних і реалізації алгоритму класифікації ми вибрали мову програмування Python, оскільки для цієї мови існує сформована екосистема бібліотек МН, оброблення даних, матричної математики тощо. Великий вибір бібліотек, простота і зручність використання, а також усі інші згадані фактори роблять Python стандартним вибором для проєктів, що стосуються МН і нейронних мереж.

Основними бібліотеками реалізації функціоналу нашої програмної системи класифікації текстів (ПСАКТ) за певними категоріями стали: TensorFlow — безкоштовна бібліотека для МН із відкритим кодом [16] (відома своєю масштабованістю та високою продуктивністю); scikit-learn — надає простий і зрозумілий інтерфейс для реалізації класичних алгоритмів МН [12]; Natural Language Toolkit (nltk) — для реалізації задач опрацювання природної мови [8]; NumPy [9] — забезпечує зручність доступу до великих і багатовимірних масивів, а також функцій високого рівня для роботи з такими масивами; Pandas [10] — для доступу до структури даних DataFrame, яка є швидкою, гнучкою та інтуїтивно зрозумілою; Matplotlib [7] і seaborn [14] — для візуалізації даних та побудови графіків.

1. Аналіз вхідних даних

Для навчання моделей ми застосували AG News Classification Dataset із kaggle.com [5]. Цей набір даних містить 120 тисяч коротких текстів, зібраних із різних джерел новин. Усі тексти рівномірно (по 30 тисяч) розподілені між такими чотирма категоріями:

- World (Світ) — тексти про події та новини з різних країн світу, міжнародні відносини, а також глобальні проблеми, як-от зміна клімату, міграція та інші;
- Sports (Спорт) — тексти про спортивні події, змагання, турніри, команди та виступи спортсменів у різних дисциплінах, таких як футбол, баскетбол, теніс тощо;
- Business (Бізнес) — тексти про компанії, фінансові новини, аналіз ринків, бізнес-стратегії, економічні тенденції та інші аспекти бізнесу, зокрема корпоративні злиття і поглиблений огляд економічних подій різного роду;
- Science / Technology (Наука / Технології) — тексти про нові відкриття в науці, технологічні досягнення, інновації у технологічній сфері, зокрема роботи над штучним інтелектом, космічними відкриттями, медичними технологіями тощо.

Щоб робота з даними була більш ефективною, ми вирішили розробити модуль, який виконував би попереднє оброблення текстових даних шляхом побудови функції, яка отримує на вхід текст і виконує такі дії: токенізація тексту, видалення стоп-слів, лематизація або стемінг токенів (допомагає значно зменшити розмір словника та покращити точність аналізу шляхом зведення різних форм одного слова до єдиного кореня). Всі ці операції мають допомогти зменшити розмірність текстових даних, виокремити важливі ключові слова та поліпшити якість аналізу.

Поділ набору даних також є дуже вагомим етапом для ефективного навчання та коректної оцінки продуктивності моделі. У більшості випадків достатньо дотримуватися рівномірних пропорцій класів. Тому ми розділили всі наявні екземпляри на три категорії: навчальний набір даних, який використовують для безпосереднього тренування й підбору оптимальних ваг та коефіцієнтів моделі; валідаційний набір даних, який використовують для перевірки моделі після кожного циклу тренування і який може допомогти підібрати оптимальні гіперпараметри для моделі; тестувальний набір даних, який необхідний для неупередженої оцінки кінцевих результатів моделі.

Тому використаний набір даних розбивали на категорії так: 96 тис. текстів (80 % загальної кількості) використано для безпосереднього тренування моделі; для валідації — 10 % загальної кількості текстів, тобто 12 тис.; решту текстів виділено для тестування фінальних результатів.

Щоби перевірити, наскільки ефективно моделі можуть навчатись, маючи досить обмежений обсяг даних для тренування, ми розбивали набір даних також на три категорії: 0,1 % (120 екземплярів) — для тренування; 0,1 % (120 екземплярів) — для валідації; 99,8 % (117 600 екземплярів) — для тестування.

Також ми обмежилися використанням комбінації трьох моделей: Naive Bayes, SVM і RNN.

Naive Bayes є швидким алгоритмом, який потребує найменше ресурсів серед усіх трьох згаданих. Якщо припущення про незалежність між ознаками справджується, то може демонструвати доволі ефективні результати, що робить його гарним вибором для таких задач, як, наприклад, фільтрація спаму. Бібліотека scikit-learn має клас MultinominalNB, який реалізує наївний баєсів класифікатор для багатьох номіальних моделей.

SVM також порівняно економний у плані використання ресурсів комп'ютера, але не настільки, як наївний баєсів класифікатор. Використання його може демонструвати доволі ефективні результати навіть після тренування на досить невеликому обсягу даних. Бібліотека scikit-learn також надає реалізацію опорно-векторної машини у формі класу LinearSVC.

RNN серед трьох вибраних алгоритмів найбільш затратний у плані ресурсів, проте в загальному випадку після тренування на достатній кількості даних демонструє найкращі результати. Підходить для завдань, де є важливим контекст, як-от аналіз настроїв тощо. Для того щоб уникнути проблеми затухання градієнта, буде використано LSTM.

BERT і LLM ми не використовували, оскільки вони потребують більш потужних обчислювальних ресурсів та великої кількості часу для тренування. Відкинули й підхід на основі правил, адже ми не зможемо адаптувати старі правила для нових категорій у випадку, якщо користувач натренує модель на власних текстах. Загалом згаданих трьох методів має бути достатньо, щоб переваги одного методу могли компенсувати недоліки іншого, не використовуючи при цьому надмірну кількість ресурсів комп'ютера.

Згідно з нашою задачею, ми передбачили можливість використовувати будь-яку комбінацію моделей для тренування на користувацьких текстах. Наприклад, для швидкого результату можна навчити тільки Naive Bayes, а для більш точних результатів натренувати всі три моделі. Так само й для класифікації має бути можливість використовувати будь-яку комбінацію натренованих моделей. Для цього отримували відсоткові передбачення категорії з кожного класифікатора, а фінальний результат за замовчуванням є середнім арифметичним усіх значень. Але у користувача повинна бути можливість зменшити вплив на результат класифікації тих класифікаторів, які, на його думку, показують гірший результат, і навпаки — збільшити для тих, які показують кращий результат. Тому остаточний результат класифікації тексту визначався такою формулою: $x = a \cdot w_c + b \cdot w_b + c \cdot w_c$ де: x — результат системи класифікації, a — результат класифікації з використанням Naive Bayes, w_c — «вага» результату класифікації Naive Bayes, ціле число, більше ніж 0, що визначається користувачем, b — результат класифікації з використанням SVM, w_b — «вага» результату класифікації SVM, c – результат класифікації з використанням RNN, w_c — «вага» результату класифікації RNN. Використання трьох коефіцієнтів дало можливість коригувати систему класифікації для більш оптимальних результатів.

2. Реалізація ПС класифікації

ПС реалізовано у вигляді застосунку.

2.1. Структура застосунку

Представлена на рис. 1 UML діаграма класів схематично демонструє основні класи, що реалізовані в застосунку, їхню взаємодію та функції, які вони зможуть виконувати.

Клас TextPreprocessor дає можливість вибрати мову та опції оброблення (стемінг і лематизація), а також надає метод preprocess, який прийматиме на вхід необроблений текст, а повертатиме вже оброблений.

TextClassifierNB — класифікатор, що використовує Naive Bayes. Має доступ до обробника тексту й містить список назв категорій, які здатен класифікувати. Метод train приймає на вхід або шлях до директорії, з якої будуть діставатись і оброблюватись текстові файли для тренування або ж уже оброблений набір даних у формі DataFrame. Після виконання тренування повертає прогнозоване

значення точності. Метод save виконує збереження моделі до вказаної директорії, а load — завантаження. Метод predict приймає на вхід необроблений текст і повертає результат класифікації.

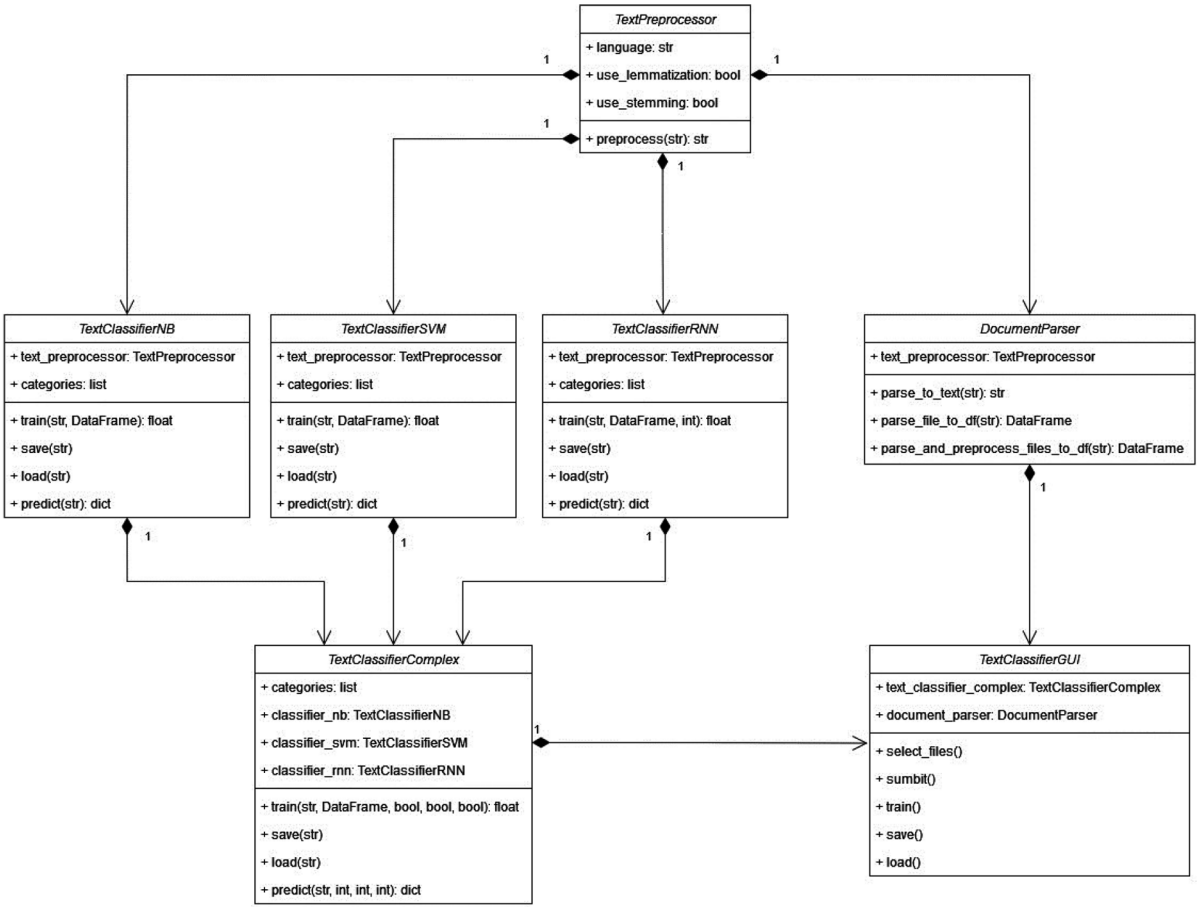

Рис. 1. Діаграма класів застосунку

TextClassifierSVM — клас, що працює схожим чином із попереднім класом, але використовує SVM. TextClassifierRNN — класифікатор, що використовує RNN модель. На відміну від інших класифікаторів, тут при тренуванні можна зазначити кількість епох.

DocumentParser — клас, що спрощує роботу з документами. Містить набір методів, які витягуються текст із txt, docx чи pdf файлів у потрібному форматі.

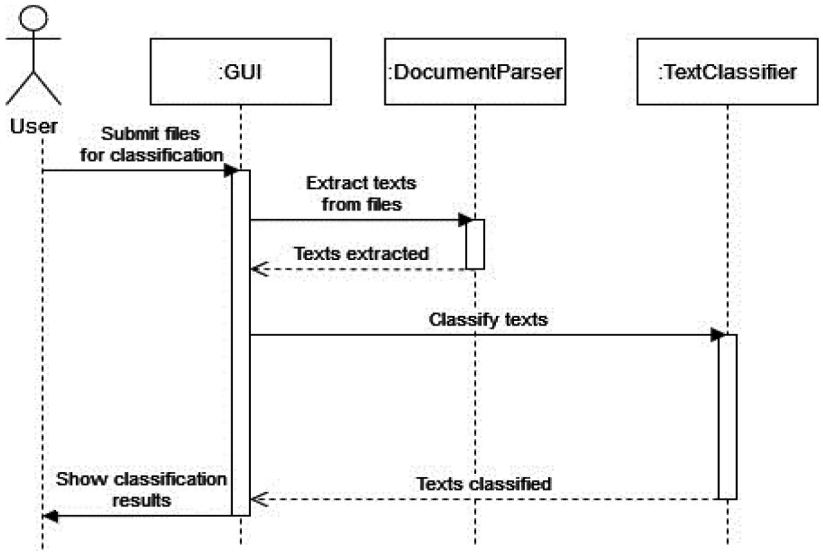

Рис. 2. Діаграма послідовності для процесу класифікації

TextClassifierComplex — має доступ до всіх трьох класифікаторів. При тренуванні є можливість обрати, які з моделей потрібно використовувати. Метод predict використовує коефіцієнти для кожного класифікатора при обчисленні результатів.

TextClassifierGUI — клас, який управляє графічним інтерфейсом. Працює з DocumentParser для отримання вмісту документів та з TextClassifierComplex для його класифікації.

Процес взаємодії графічного інтерфейсу з іншими сутностями описує UML діаграма послідовності (рис. 2), яка відображає простий випадок використання системи для класифікації файлів користувача.

Користувач обирає файли через GUI й запускає процес класифікації. DocumentParser витягує з файлів текстову інформацію, після чого вона передається в класифікатор тексту для визначення категорії. Результати класифікації повертаються й демонструються користувачеві у форматі, що відповідає обраним в графічному інтерфейсі опціям.

Рисунок 3 демонструє процес тренування моделі на файлах користувача та її збереження.

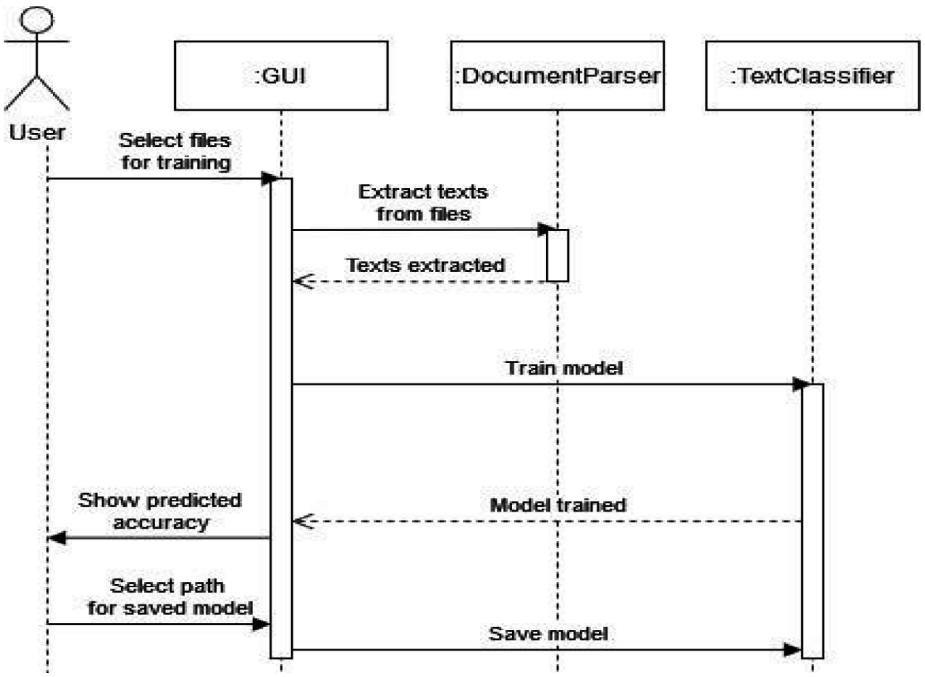

Рис. 3. Діаграма послідовності для процесу тренування моделі

Користувач вибирає файли для тренування (файли мають бути відсортовані за піддиректоріями відповідно до назв класів чи категорій). Текст із файлів витягується та передається до класифікатора для тренування. Після тренування користувач зможе побачити значення прогнозованої точності класифікації. Є можливість зберегти натреновану модель в обрану директорію для того, щоб була можливість завантажити її пізніше.

2.2. Оброблення тексту

Клас TextPreprocessor розроблено з використанням базових функцій бібліотеки NLTK. При ініціалізації є можливість обрати мову (доступна англійська та експериментальна реалізація для української), а також увімкнути опції стемінгу чи лематизації (рис. 4).

```
class TextPreprocessor:
    Andrii Dubovyk *
    def __init__(self, language='english', use_lemmatization=False, use_stemming=True):
        self.language = language
        self.use_stemming = use_stemming
        self.use_lemmatization = use_lemmatization
        if language == 'ukrainian':
            self.__setup_ukrainian_language()
        else:
            self.__setup_default_language()
```

Рис. 4. Ініціалізація класу TextPreprocssor

Для англійської мови застосовують PorterStemmer, WorldNetLemmatizer та список стоп-слів, який надає бібліотека NLTK. Для оброблення ж текстів української мови (рис. 5) використовують відповідний список стоп-слів із текстового файлу. Як стемер використано бібліотеку UkStemmer [17], а для лематизації — pymorhy2 [11].

```
def __setup_ukrainian_language(self):
    stopwords_ua = pandas.read_csv( filepath_or_buffer: "resources/stopwords_ua.txt", header=None, names=['stopwords'])
    self.stop_words = list(stopwords_ua.stopwords)
    if self.use_stemming:
        self.stemmer = uk_stemmer.UkStemmer()
        self.stem = self.stemmer.stem_word
    if self.use_lemmatization:
        self.lemmatizer = pymorphy2.MorphAnalyzer(lang='uk')
        self.lemmatize = lambda word: self.lemmatizer.parse(word)[0].normal_form
```

Рис. 5. Фрагмент коду, що демонструє налаштування для української мови

Для полегшення роботи з файлами створено модуль DocumentParser, який може одразу діставати текстову інформацію з файлів формату txt, pdf та docx, які відсортовано за директоріями відповідно до її класів (рис. 6). Текст оброблюється й повертається в зручному форматі (яким у цьому випадку є DataFrame з бібліотеки pandas).

```
@staticmethod
def parse_files_to_df(directory):
    classes = []
    for root, dirs, files in os.walk(directory):
        for d in dirs:
            classes.append(d)
    data = []
    for category in classes:
        category_dir = os.path.join(directory, category)
        for filename in os.listdir(category_dir):
            file_path = os.path.join(category_dir, filename)
            text = DocumentParser.parse_to_text(file_path)
            data.append({'label': category, 'text': text})
    return pd.DataFrame(data)

1 usage   Andrii Dubovyk
@staticmethod
def parse_and_preprocess_files_to_df(directory):
    df = DocumentParser.parse_files_to_df(directory)
    text_preprocessor = TextPreprocessor(language='english', use_stemming=False, use_lemmatization=True)
    df['text'] = df['text'].apply(lambda txt: text_preprocessor.preprocess(txt))
    return df
```

Рис. 6. Фрагмент коду, що демонструє процес завантаження та обробки текстових файлів

2.3. Реалізація алгоритмів класифікації

Для реалізації класифікатора Naive Bayes використано клас MutlinominalNB з бібліотеки scikit-learn (рис. 7).

```
# Define model
self.model = Pipeline([
    ('tfidf', TfidfVectorizer()),
    ('nb', MultinomialNB()),
])

# Train model
self.model.fit(x_train['text'], x_train['label'])
```

Рис. 7. Визначення моделі Naive Bayes та її тренування

Схожим чином для SVM використовується функція LinearSVC з бібліотеки scikit-learn. Але звичайна реалізація LinearSVC всього лише може видавати результат у вигляді однієї найбільш вірогід-

ної категорії. Щоб отримувати відсоткове значення для кожної з категорій, ми робимо обгортку (рис. 8) за допомогою CallibratedClassifierCV.

```
# Define model
self.model = Pipeline([
    ('tfidf', TfidfVectorizer()),
    ('svc', CalibratedClassifierCV(LinearSVC())),
])

# Train model
self.model.fit(x_train['text'], x_train['label'])
```

Рис. 8. Визначення моделі SVM та її тренування

Методи класифікації Naive Bayes та SVM доволі просто реалізовані за допомогою відповідних функцій бібліотеки scikit-learn, однак реалізація RNN за допомогою TensorFlow надає нам більше можливостей для модифікації та налаштувань задля покращення ефективності моделі.

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 text_vectorization (TextVec  (None, None)            0
 torization)

 embedding (Embedding)       (None, None, 64)          640000

 bidirectional (Bidirectiona  (None, None, 128)        66048
 l)

 global_max_pooling1d (Globa  (None, 128)              0
 lMaxPooling1D)

 dense (Dense)               (None, 64)                8256

 batch_normalization (BatchN  (None, 64)               256
 ormalization)

 dropout (Dropout)           (None, 64)                0

 dense_1 (Dense)             (None, 4)                 260

=================================================================
Total params: 714,820
Trainable params: 714,692
Non-trainable params: 128
```

Рис. 9. Архітектура RNN моделі з використанням LSTM

Можемо детально розглянути архітектуру розробленої RNN (LSTM) моделі (рис. 9). Всього вона складається з 8 шарів.

TextVectorization перетворює оброблений текст на послідовність індексів токенів. Використовує словник розміром 10 000, що є більш ніж достатньо з урахуванням того, що текст проходить оброблення, яке передбачає лематизацію. Шар убудування Embedding зберігає один вектор на слово. При виклику він перетворює послідовності індексів слів на послідовності векторів. Ці вектори можна тренувати. Після навчання на достатній кількості даних слова зі схожими лексичними значеннями часто мають подібні вектори. Має параметр mask_zero=True, що дає змогу ефективно працювати з текстами різних розмірів. Bidirectional є обгорткою для RNN, що обробляє послідовний вхід у двох напрямках. У цьому випадку використовує LSTM шар із 64 нейронами для збереження контексту.

Рівень об'єднання GlovalMaxPooling1D зменшує чутливість до особливостей, створюючи більш узагальнені дані для кращих результатів тренування. Повнозв'язний шар Dense використовує функцію активації relu та містить 64 нейрони.

BatchNormalization використовують для підвищення швидкості та стабільності навчання штучних нейронних мереж. Це досягається шляхом нормалізації вхідних даних шарів через повторне центрування та масштабування. Dropout відкидає випадковим чином деякі нейрони, може допомогти частково уникнути надмірного тренування, яке відбувається, коли модель надто добре або занадто довго тренувалась на заданому наборі даних і їй не вдається демонструвати хороші результати при застосуванні до нових даних. У цьому випадку використовується значення rate 0,2, що означає, що 20 % випадкових нейронів будуть ігноруватись. Також варто зауважити, що цей шар використовується лише при тренуванні, тому під час справжніх передбачень моделі нейрони не будуть ігноруватись. Повнозв'язний шар Dense містить 4 нейрони, що відповідає кількості категорій у нашому наборі даних. Використовує функцію активації softmax та повертає розподіл ймовірностей визначених категорій.

Ця модель компілюється з такими параметрами: sparse_categorical_crossentropy, Adam, accuracy. Перший застосовується для обчислення втрат між прогнозованими та справжніми мітками. Це доцільний вибір для задачі класифікації з кількома класами, де мітки не кодуються як one-hot вектори, але представлені цілими числами. Дає можливість отримати список найбільш імовірних категорій. Оптимізатор Adam (зі швидкістю навчання 1e-4) ефективно пристосовує швидкість навчання для кожного параметра, зокрема на основі оцінок першого та другого моментів градієнта. Метрика accuracy використовується для оцінки точності моделі під час навчання. Вона вимірює відсоток правильно класифікованих зразків з усієї кількості екземплярів.

Використання функції зворотного виклику EarlyStopping покликано зупиняти тренування, коли точність класифікації моделі для валідаційного набору даних перестає зростати. В цьому випадку ця функція використовується з такими параметрами: val_accuracy, 2, 0.001. Перший параметр позначає значення, за яким потрібно робити нагляд, тут точність моделі на валідаційному наборі даних. Другий — визначає кількість епох без покращення, після яких навчання буде припинено. Третій — мінімальна зміна значення, що контролюється, яка буде вважатися покращенням (тобто абсолютна зміна менше ніж min_delta вважатиметься відсутністю покращення). Коли ця функція спрацьовуватиме, відповідне повідомлення виводиться на консоль. Після спрацювання функції буде відновлено той стан моделі, у якому вона мала найкраще значення точності. Ця функція в багатьох випадках відбирає в нас потребу підбирати оптимальну (для уникнення недостатнього або надмірного тренування) кількість епох вручну, дозволяючи просто запустити тренування на великій кількості епох і дочекатись, доки точність перестане зростати.

У підсумку після тренування можемо побачити графіки (рис. 10) зміни точності і втрат (синім кольором показані результати для тренувальних даних, оранжевим — для валідаційних). Тренування відбувалось впродовж п'яти епох, доки точність не перестала зростати, після чого було відновлено стан моделі, в якому вона демонструвала найкращу точність на валідаційних даних (у цьому випадку це третя епоха).

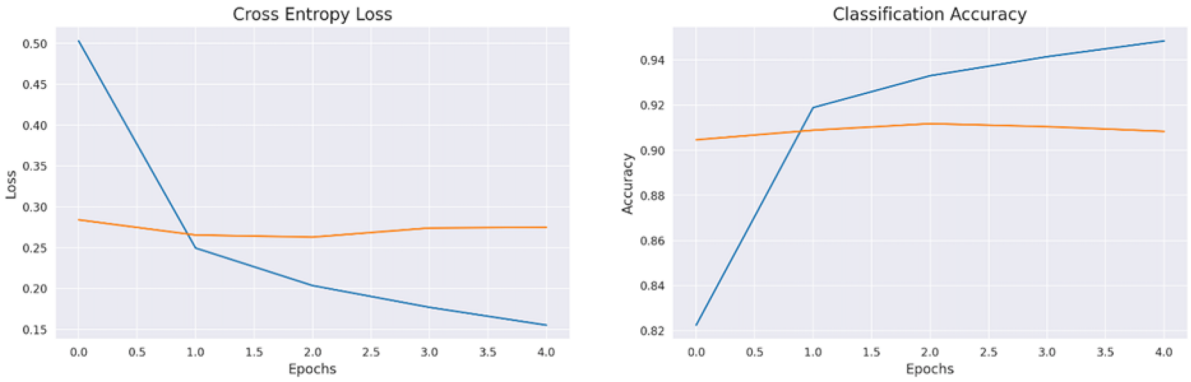

Рис. 10. Втрати та точність LSTM моделі впродовж тренування

Також були протестовані деякі інші часто використовувані оптимізатори, як-от Adagard, Adadelta і RMSprop, із метою визначити їхній вплив на ефективність натренованої моделі. З іншими оптимі-

заторами, використовуючи стандартну швидкість навчання зі значенням 0,001, після тренування максимально вдалось досягти таких значень: Adagard — 90,22 % точності та значення втрат 0,2933; Adadelta — 90,71 % точності та значення втрат 0,2764; RMSprop — 91,93 % точності та значення втрат 0,2524.

Тільки з оптимізатором RMSprop вдалося досягнути дещо кращого результату, ніж з Adam, натомість оптимізатори Adagard і Adadelta не змогли показати значення точності вище ніж 91 %, тому було вирішено далі використовувати саме цей оптимізатор. Також тестувалось тренування моделі з іншою (0,01 та 0,0001) швидкістю навчання, відмінною від стандартної 0,001, але якихось суттєвих змін в результатах помічено не було, тому швидкість навчання зі стандартним значенням 0.001 була залишена як оптимальна.

2.4. Система класифікації з використанням комбінації алгоритмів

Кожну реалізацію алгоритму класифікації з попереднього підрозділу було додано до окремих відповідних класів: TextClassifierNB, TextClassifierSVM, TextClassifierRNN. Ці класи мають такі методи, необхідні для роботи:

– train — приймає на вхід або директорію з текстами або DataFrame з обробленими текстами, для RNN моделі може також приймати кількість епох для навчання, тренує модель і повертає значення точності класифікації у відсотках;
– save — приймає на вхід директорію, до якої зберігає навчену модель (NB та SVM моделі зберігаються у форматі pkl, RNN — у форматі keras, окремо в json файл зберігаються назви категорій);
– load — приймає на вхід директорію, з якої завантажує попередньо збережену модель;
– predict — приймає на вхід текст користувача, для якого визначає категорії, результат повертає в форматі відсортованого Python словника, де ключі це назви категорій, а значення — ймовірність належності до категорії від нуля до одиниці.

Аналогічно розроблено клас TextClassifierComplex, що використовує всі вищезгадані класи та містить такі методи: train — схожий на методи окремих класів, за допомогою булевих параметрів use_nb, use_svm, use_rnn є можливість визначати, які моделі будуть тренуватись; save — зберігає всі натреновані моделі в обрану директорію; load — завантажує всі моделі, які збережені в обраній директорії; predict — аналогічний до методу окремих класів, має параметри nb_weight, svm_weight, rnn_weight, значення яких коригує вплив передбачення відповідної моделі на загальний результат.

2.5. Графічний інтерфейс

Було розроблено графічний інтерфейс, який дає змогу користувачу не тільки ефективно використовувати систему з натренованими моделями, які навчені на AG News Classification Dataset, а й натренувати систему на власних файлах, зберегти результати навчання та використовувати її надалі. Одразу при запуску застосунку завантажуються вже заздалегідь навчені моделі.

Вгорі інтерфейсного вікна містяться елементи, які стосуються тренування, збереження та завантаження моделі користувача. Прапорці Train NB, Train SVM, Train RNN відповідають за те, які моделі будуть тренуватись. Кнопка Train відкриває діалогове вікно, що дозволяє обрати директорію з текстовими документами (txt, pdf або docx) для тренування, потім запускається сам процес тренування обраних моделей. Після тренування в діалоговому вікні буде показано приблизне значення точності класифікації (середнє арифметичне для всіх використаних моделей на тренувальному наборі даних). Варто зазначити, що текстові документи в обраній директорії мають бути розміщені належним чином: кожен документ має бути в піддиректорії, назва якої відповідає визначеній категорії тексту. Кнопка Save дозволяє зберегти модель в одну з директорій комп'ютера, а кнопка Load — завантажити попередньо збережену модель. Текст Categories: [перелік категорій] демонструє, які категорії здатна розпізнавати модель, яка зараз використовується.

Далі розміщена група елементів графічного інтерфейсу, що стосуються завантаження документів для класифікації. Є можливість вибрати мову наданих текстів, що необхідно для правильної класифікації. За замовчуванням використовується англійська мова, але є можливість вибрати українську, в такому разі всі обрані тексти користувача будуть автоматично перекладатись через Google Translate на англійську мову перед тим, як модель буде робити передбачення їхньої категорії. Це дає змогу не тренувати модель заново для іншої мови, а використовувати ту саму модель для розпізнавання

текстів мов, відмінних від англійської. Документи для класифікації обирають у діалоговому вікні, що з'являється після натискання Select Files. Повний шлях до обраних файлів демонструється у відповідному текстовому полі.

Остання група елементів інтерфейсу відповідає за саму класифікацію і параметри системи, що при цьому використовуються. Поля NB Weight, SVM Weight, RNN Weight дозволяють коригувати вплив кожної окремої моделі на загальний результат. Прапорець Detailed Results впливає на формат відображення результатів; якщо він відмічений, то для кожного документа буде вказано відсоткове значення належності до категорій, якщо ні — лише одна найбільш вірогідна категорія (без значення відсотків). Кнопка Submit запускає процес визначення категорії, а Copy Results дозволяє швидко та зручно скопіювати результати в буфер обміну.

2.6. Оцінка результатів

На тестах класифікації новин найкращі результати для використаного набору даних демонструвала саме RNN модель, тому в цьому випадку детальніше розглянемо саме її результати (рис. 11). На тестових даних розроблена модель у остаточному її варіанті показала такі результати: точність — 91,92500114440918 %, втрати — 0,26782718300819397.

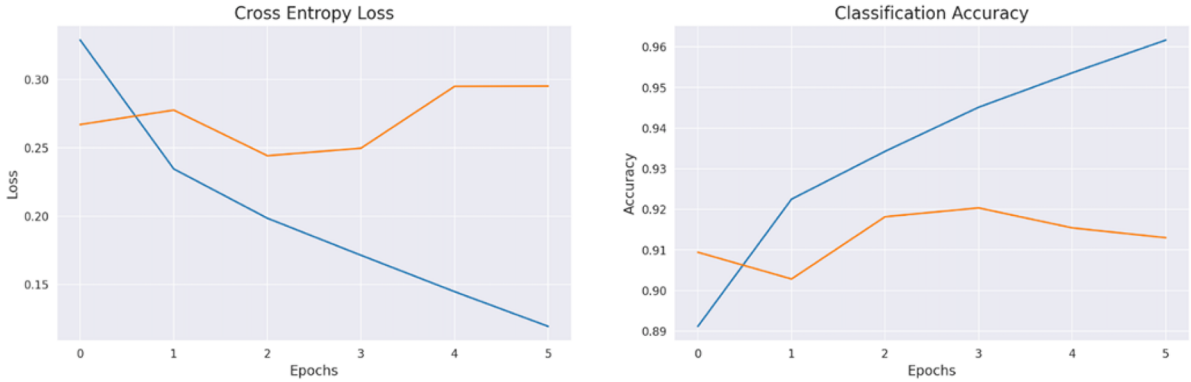

Рис. 11. Графіки втрат і точності RNN моделі в її остаточному вигляді

Також у звіті про класифікацію (рис. 12) можна побачити деякі показники для кожної з чотирьох категорій окремо: Precision — кількість справжніх позитивних результатів, поділена на кількість прогнозованих позитивних результатів; Recall — кількість справжніх позитивних результатів, поділена на загальну кількість фактичних позитивних результатів; F1-score — середнє гармонічне між precision та recall.

	precision	recall	f1-score	support
World	0.89	0.89	0.89	3000
Sports	0.90	0.89	0.89	3000
Business	0.96	0.98	0.97	3000
Science/Technology	0.93	0.91	0.92	3000
accuracy			0.92	12000
macro avg	0.92	0.92	0.92	12000
weighted avg	0.92	0.92	0.92	12000

Рис. 12. Звіт про класифікацію

У застосунку передбачено побудову і виведення матриці невідповідностей, яка демонструє, скільки екземплярів однієї категорії було визначено іншою, невідповідною категорією під час передбачень категорій. Тести показали, що найчастіше модель плутає між собою категорії World і Sports. Також порівняно часто плутає Science / Technology з World і Sports із Science / Techonlogy. Категорія ж Business визначається найбільш впевнено і плутається з усіма іншими доволі рідко.

Для порівняння ефективності навченої моделі категорії тексту для 200 екземплярів з цього ж набору даних (AG News Classification Dataset) було визначено через ChatGPT версії 3.5. В результаті правильно розпізнано було лише 176 категорій, що показує точність зі значенням близько 88 %. Реа-

лізована ж у цій роботі RNN (LSTM) модель демонструє майже 92 %. Отже, можна зробити припущення, що спеціалізовані моделі, крім того, що є значно ефективнішими в плані використання ресурсів, також можуть демонструвати кращий результат у класифікації текстів, ніж LLM.

Із другим поділом набору даних, який містить обмежений обсяг даних для тренування, моделі показали такий результат: Naive Bayes сягнула значення точності в 68,34 %, SVM досягла 70,25 % точності, LSTM модель продемонструвала точність передбачення всього лише 24,98 %. Це демонструє перевагу моделей SVM та Naive Bayes в ситуації з недостатньою кількістю даних для тренування.

Також було перевірено, наскільки ефективно система може навчатись на інших текстах. З kaggle.com із цією метою було завантажено набір даних із назвою (10)Dataset Text Document Classification. Цей набір даних містить по сотні текстів кожної з десяти категорій: «бізнес», «розваги», «їжа», «графіка», «історія», «медицина», «політика», «космос», «спорт» і «технології». На завантаження та оброблення текстів із цих файлів знадобилося 4,78 секунди. Моделі впорались із цією задачею так: Naive Bayes — демонструє 97 % точності, витративши 0,2 секунди на навчання, SVM — демонструє 98 % точності, витративши 0,59 секунди на навчання, RNN — демонструє 91 % точності, витративши 974 секунди на навчання, комбінація всіх моделей (з однаковими вагами) має точність 94,5 %.

Щоб перевірити, як система виконує таку задачу, як фільтрація спаму, було використано Spam Mails Dataset із kaggle.com, який містить 5171 зразок тексту з визначеними спам-повідомленнями. Моделі справляються з цією задачею з такою точністю та витраченим часом на навчання (без урахування 9,53 секунди на завантаження файлів і оброблення тексту): Naive Bayes — 92,07 % та 0,41 секунди, SVM — 99,23 % і 0,54 секунди, RNN — 98,07 % і 1153 секунди, комбінація всіх моделей (з однаковими вагами) має точність 99,13 %.

Отже, система може бути швидко адаптована й до такої задачі. Майже за 10 секунд можна отримати SVM модель, яка зможе розпізнавати типові спам-повідомлення з точністю близько 99 %.

Так само були протестовані можливості системи при розпізнаванні емоційної забарвленості тексту. Для цього використано Emotions dataset for NLP з kaggle.com, який містить 16 тисяч зразків тексту з визначеними емоціями (сум, злість, страх, радість, здивування, любов). Текст із файлів завантажується та оброблюється за 2,8 секунди. Моделі демонструють таку точність і час навчання: Naive Bayes — 56,99 % і 0,081 секунди, SVM — 83,09 % і 0,83 секунди, RNN — 85,56 % і 13,35 секунди, комбінація всіх моделей (з однаковими вагами) має точність 59,31 %. Але якщо проекспериментувати з коефіцієнтами, то можна отримати кращі результати, ніж при використанні кожної моделі окремо. Наприклад, вдалося досягти точності 87,75 %, використавши такі значення: 1 для Naive Bayes, 5 для SVM та 13 для RNN.

У результаті навіть якщо один або два алгоритми не можуть показати хороший результат для певної задачі, то майже завжди як мінімум одна модель може ефективно класифікувати тексти з точністю понад 85 %. Отже, система може застосуватись майже для всіх загальних випадків класифікації.

Висновки

У цій роботі описано побудову та результати тестування ПС автоматичної класифікації текстів, яка полягає в розподілі текстів за певними категоріями, зокрема текстів українською мовою. Наш застосунок побудований на використанні трьох моделей — Naive Bayes, Support Vector Machine (SVM), LSTM архітектури рекурентної нейронної мережі Recurrent Neural Network (RNN) та їх комбінації. Він дає можливість доволі швидко і точно класифікувати тексти, зручним способом натренувати систему на власних даних і досить просто налаштувати параметри для оптимальних результатів.

Для ефективного опрацювання вхідних даних і реалізації алгоритму класифікації ми обрали мову програмування Python. Основними бібліотеками реалізації функціоналу нашого застосунку стали: TensorFlow (за гарну масштабованість і високу продуктивність); scikit-learn (для надання простого та зрозумілого інтерфейсу); Natural Language Toolkit (nltk, для реалізації задач опрацювання природної мови); NumPy (що забезпечує зручність доступу до великих та багатовимірних масивів); Pandas (для простого доступу до структури даних DataFrame). Matplotlib та seaborn застосовувалися для візуалізації даних та побудови графіків.

Для навчання моделей ми застосували AG News Classification Dataset із kaggle.com. Розроблений графічний застосунок здатен розпізнавати тексти (англійською або українською мовою) чотирьох категорій (World, Sports, Science / Technology, Business) із точністю близько 92 %.

Тестування застосунку підтвердило припущення, що спеціалізовані моделі, крім того, що є значно ефективнішими в плані використання ресурсів, також можуть демонструвати кращий результат у класифікації текстів, ніж LLM. Система також може бути швидко адаптована й до задачі фільтрації спаму. За декілька секунд можна отримати SVM модель, яка зможе розпізнавати типові спам-повідомлення з точністю близько 99 %. Так само були протестовані можливості системи при розпізнаванні емоційної забарвленості тексту. Вдалось досягти точності 87,75 %.

Продовження досліджень та експериментів у цій сфері можуть призвести до підвищення ефективності створених моделей у розпізнаванні використаних у цій роботі або будь-яких інших категорій текстів. Також подальший розвиток може бути спрямований на покращення зручності та функціональності розробленого графічного застосунку.

Ця робота була підтримана грантом Фонду Саймонса (SFI-PD-Ukraine-00014577; T.S.)

Список літератури

1. Глибовець А. М. Програмна система перевірки на плагіат українських текстів [Електронний ресурс] / А. М. Глибовець, М. О. Бікчентаєв // Наукові записки НаУКМА. Комп'ютерні науки. — 2022. — Том 5. — С. 16–25. — Режим доступу: https://doi.org/10.18523/2617-3808.2022.5.16-25.
2. A Generative Learning Model for Heterogeneous Text Classification Based on Collaborative Partial Classifications [Electronic resource] / Zie Eya Ekolle, Ryuji Kohno // Applied Sciences. — 2023. — Vol. 13 (14). — P. 8211. — Mode of access: https://www.mdpi.com/2076-3417/13/14/8211.
3. A Hybrid Text Classification Approach for Analysis of Student Essays [Electronic resource] / C. P. Rosé, A. Roque, D. Bhembe, K. Vanlehn // Proceedings of the HLT-NAACL 03 Workshop on Building Educational Applications Using Natural Language Processing. — Pp. 79–86. — Mode of access: https://aclanthology.org/W03-0210.pdf.
4. A Survey on Text Classification Algorithms: From Text to Predictions [Electronic resource] / A. Gasparetto, M. Marcuzzo, A. Zangari, A. Albarelli // Information. — 2022. — Vol. 13 (2). — P. 3. — Mode of access: https://www.mdpi.com/2078-2489/13/2/83.
5. Kaggle [Electronic resource]. — Mode of access: https://www.kaggle.com.
6. Machine Learning Glossary [Electronic resource]. — Mode of access: https://developers.google.com/machine-learning/glossary.
7. Matplotlib [Electronic resource]. — Mode of access: https://matplotlib.org.
8. Natural Language Toolkit [Electronic resource]. — Mode of access: https://www.nltk.org.
9. NumPy [Electronic resource]. — Mode of access: https://numpy.org.
10. Pandas [Electronic resource]. — Mode of access: https://pandas.pydata.org.
11. Pymorphy2 [Electronic resource]. — Mode of access: https://github.com/pymorphy2/pymorphy2.
12. Scikit-learn [Electronic resource]. — Mode of access: https://scikit-learn.org/stable.
13. Scikit-learn: Naive Bayes [Electronic resource]. — Mode of access: https://scikit-learn.org/stable/modules/naive_bayes.html.
14. Seaborn [Electronic resource]. — Mode of access: https://seaborn.pydata.org.
15. Support vector machine [Electronic resource]. — Mode of access: https://en.wikipedia.org/wiki/Support_vector_machine.
16. TensorFlow [Electronic resource]. — Mode of access: https://www.tensorflow.org.
17. UkStemmer [Electronic resource]. — Mode of access: https://github.com/Desklop/Uk_Stemmer
18. Understanding Text Classification in Python [Electronic resource]. — Mode of access: https://www.datacamp.com/tutorial/text-classification-pythonpython.

References

Datacamp. (n. d.). *Understanding text classification in Python*. https://www.datacamp.com/tutorial/text-classification-pythonpython.

Ekolle, Z. E., & Kohno, R. (2023). A generative learning model for heterogeneous text classification based on collaborative partial classifications. *Applied Sciences*, *13* (14), 8211. https://www.mdpi.com/2076-3417/13/14/8211.

Gasparetto, A., Marcuzzo, M., Zangari, A., & Albarelli, A. (2022). A survey on text classification algorithms: From text to predictions. *Information*, *13* (2), 83. https://www.mdpi.com/2078-2489/13/2/83.

Google Developers. (n. d.). *Machine learning glossary*. https://developers.google.com/machine-learning/glossary.

Hlybovets, A. M., & Bikchentayev, M. (2022). Prohramna systema perevirky na plahiat ukrainskykh tekstiv. *NaUKMA Research Papers. Computer Science*, *5*, 16–25. https://doi.org/10.18523/2617-3808.2022.5.16-25 [in Ukrainian].

Kaggle. (n. d.). https://www.kaggle.com.

Matplotlib. (n. d.). https://matplotlib.org.

Natural Language Toolkit (nltk). (n. d.). https://www.nltk.org.

NumPy. (n. d.). https://numpy.org.

Pandas. (n. d.). https://pandas.pydata.org.

Pymorphy2. (n. d.). https://github.com/pymorphy2/pymorphy2.

Rosé, C. P., Roque, A., Bhembe, D., & VanLehn, K. (2003). A hybrid text classification approach for analysis of student essays. In *Proceedings of the HLT-NAACL 03 Workshop on Building Educational Applications Using Natural Language Processing* (pp. 79–86). https://aclanthology.org/W03-0210.pdf.

Scikit-learn. (n. d.). https://scikit-learn.org/stable.

Scikit-learn. (n. d.). *Naive Bayes*. https://scikit-learn.org/stable/modules/naive_bayes.html.

Seaborn. (n. d.). https://seaborn.pydata.org.

Support vector machine. (n. d.). *Wikipedia*. https://en.wikipedia.org/wiki/Support_vector_machine.

TensorFlow. (n. d.). https://www.tensorflow.org.

UkStemmer. (n. d.). https://github.com/Desklop/Uk_Stemmer.

A. Hlybovets, A. Dubovyk

TEXT CLASSIFICATION SOFTWARE SYSTEM BASED ON MACHINE LEARNING AND RECURRENT NEURAL NETWORK

In the context of rapid advancements in information technology and the exponential growth of accessible data, efficient data analysis management has become increasingly critical. One of the key challenges in this domain is automatic text classification — the process of assigning texts to specific categories. This task is complicated by the diversity of formats, structural variability, and the inherent semantic complexity of natural language. Addressing these challenges requires robust algorithms and effective natural language processing (NLP) techniques.

This paper describes the development and testing of a software system for automatic text classification, which involves categorizing texts into predefined classes, including texts written in Ukrainian. Our application is based on the use of three models: Naive Bayes, Support Vector Machine, and the LSTM-based architecture of Recurrent Neural Networks (RNN), as well as their combinations. It allows for fast and accurate text classification and provides users with a convenient way to train the system on their own datasets and easily configure parameters for optimal results.

To efficiently process input data and implement the classification algorithm, we chose the Python programming language. The core libraries used for the application's functionality include TensorFlow, scikit-learn (for a simple and intuitive interface), Natural Language Toolkit (nltk), NumPy, and Pandas. Matplotlib and Seaborn were used for data visualization and plotting. The developed graphical application is capable of recognizing texts (in English or Ukrainian) in four categories ("World", "Sports", "Science/Technology", "Business") with an accuracy of approximately 92%. For model training, we used the "AG News Classification Dataset" from Kaggle.com.

Testing confirmed the hypothesis that specialized models, in addition to being significantly more resource-efficient, can also outperform large language models (LLMs) in text classification tasks. The system can also be quickly adapted for spam filtering tasks. Within seconds, it is possible to obtain an SVM model capable of identifying typical spam messages with about 99 % accuracy. We also tested the system's capabilities in detecting emotional tone in texts, achieving an accuracy of 87.75 %.

This work was supported by a grant from the Simons Foundation (SFI-PD-Ukraine-00014577; T.S.).

Keywords: automatic text classification, Naive Bayes, Support Vector Machine, LSTM, Recurrent Neural Network, machine learning, Python, TensorFlow, AG News Classification Dataset.

Матеріал надійшов 18.06.2025