КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ТАРАСА ШЕВЧЕНКА
Факультет комп'ютерних наук та кібернетики
Кафедра теоретичної кібернетики

Кваліфікаційна робота

на здобуття ступеня бакалавра

за спеціальністю 122 Комп'ютерні науки

на тему:

Знаходження ключових фраз у тексті за допомогою методів машинного навчання

Виконав студент 4-го курсу
Денис БІШИР _____
(підпис)

Науковий керівник:
Кандидат технічних наук
Сергій КОНДРАТЮК _____
(підпис)

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

Роботу розглянуто й допущено до захисту на засіданні кафедри теоретичної кібернетики

«____»_____2023 р., протокол № ____

Завідувач кафедри

Юрій КРАК _____

(підпис)

Київ - 2023

РЕФЕРАТ

Обсяг роботи 40 сторінок, 12 ілюстрацій, 5 джерел посилань, 2 додатки.

ВИЗНАЧЕННЯ КЛЮЧОВИХ СЛІВ, МЕТОДИ ОБРОБКИ ТЕКСТУ, НЕЙРОННІ МЕРЕЖІ, МАШИННЕ НАВЧАННЯ, NLTK, TENSORFLOW, SKLEARN, PYTHON, PYCHARM, F1-MIPA.

Об’єктом роботи є вирішення задачі знаходження ключових фраз у тексті за допомогою методів машинного навчання. Предметом є програмний застосунок для знаходження ключових фраз у тексті.

Метою роботи є створення програмного засобу, який тренує нейронну мережу і використовує її в подальшому для знаходження ключових фраз у тексті. Інструментами розробки є безкоштовне середовище PyCharm та мова програмування Python.

Результатом роботи є проведення аналізу методів обробки тексту, методів тренування нейронних мереж, знайомство з середовищем розробки та мовою програмування Python, включаючи її сторонні бібліотеки, такі як TensorFlow, NLTK та Sklearn. Було створено програмне забезпечення, що тренує штучний інтелект і використовує його для знаходження ключових фраз у тексті.

Результатом роботи програми можуть бути поняття, терміни, імена, фрази, які найкраще описують зміст тексту або є важливими для його розуміння. Ці ключові фрази можуть бути використані для подальшого аналізу, категоризації, пошуку або візуалізації текстових даних.

Також подібні підходи можуть биту застосовані при розробці пошукових систем, рекомендаційних систем, для моніторингу брендів, класифікації документів або сумаризації тексту.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАЧЕННЯ 5
ВСТУП 6
1 ОБЛАСТЬ ДОСЛІДЖЕННЯ ТА ЗАСОБИ РЕАЛІЗАЦІЇ 9
1.1 **Методи обробки тексту** **9**
1.2 **Визначення поняття та можливі підходи вирішення задачі** **11**
1.3 **Використання машинного навчання** **13**
1.4 **Засоби аналізу точності роботи методів машинного навчання** **14**
1.5 **Мова програмування Python та середовище PyCharm** **16**
1.6 **Модулі TensorFlow, NLTK та Sklearn** **17**
2 МОДЕЛЬ ТА ЇЇ ТРЕНУВАННЯ 20
2.1 **Критерії тренування моделі** **20**
2.2 **Виявлення проблем та підбір параметрів** **21**
2.3 **Вибір моделі** **21**
2.4 **Підготовка даних для тренування** **22**
2.5 **Підготовка вхідного файлу** **23**
2.6 **Алгоритм тренування моделі** **25**
3 ПРОГРАМНА РЕАЛІЗАЦІЯ ТА РЕЗУЛЬТАТИ 27
3.1 **Алгоритм роботи програми** **27**
3.2 **Демонстрація виконання програми** **28**
3.3 **Аналіз етапів навчання мережі** **29**
3.4 **Обчислення F1-міри** **33**
ВИСНОВКИ 35
ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ 37

ДОДАТОК А 38
ДОДАТОК Б 40

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАЧЕННЯ

IDE – Integrated Design Environment, інтегроване середовище розробки;

NLP – Natural Language Processing, обробка природної мови;

NLTK – Natural Language Toolkit, набор інструментів природної мови;

ROC-крива – Receiver Operating Characteristic curve, крива робочих характеристик приймача;

TF – Term Frequency, частота вживання слова;

TF-IDF – Term Frequency-Inverse Document Frequency, частота вживання слова-інверсна частота в документі;

SVM – Support Vector Machine, опорна машина векторів.

ВСТУП

Оцінка сучасного об'єкта розробки. Знаходження ключових фраз у тексті за допомогою методів машинного навчання є актуальним напрямком досліджень у галузі обробки природної мови. Завдяки прогресу в глибинному навчанні та доступності великих обсягів даних, моделі для виявлення ключових фраз стають все потужнішими і точнішими.

Використання методів машинного навчання дозволяє автоматично виявляти і виділяти ключові фрази в тексті, що має велике значення для багатьох сфер, включаючи обробку тексту, пошукові системи, розуміння природної мови, автоматичну категоризацію та сумаризацію текстів, моніторинг соціальних мереж та багато іншого.

Актуальність роботи та підстави для виконання. Знаходження ключових фраз у тексті є важливою задачею, оскільки ці фрази відображають основну суть та тематику тексту.

Дослідження в цій області мають такі актуальність і підстави для виконання:

- Управління інформацією: Знаходження ключових фраз допомагає в структуруванні та організації текстової інформації, що дозволяє краще розуміти та ефективно використовувати текстові дані.
- Пошукові системи: Використання ключових фраз підвищує ефективність пошуку, оскільки дозволяє точніше визначити сутність тексту та забезпечити кращу відповідність між запитами користувачів та документами.
- Автоматична обробка тексту: Знаходження ключових фраз є важливим етапом для багатьох задач обробки тексту, таких як автоматична категоризація, сумаризація, аналіз настрою тощо.
- Аналітика соціальних медіа: Виявлення ключових фраз допомагає в

моніторингу та аналізі великих обсягів даних з соціальних медіа, що дозволяє виявити тематичні тенденції, сентимент та інші корисні відомості.

Загалом, робота з знаходженням ключових фраз за допомогою методів машинного навчання має широкі застосування та актуальність в різних галузях, де важлива аналітика та обробка текстової інформації.

Мета й завдання роботи. Описана робота передбачає створення програми для знаходження ключових слів у тексті за допомогою методів машинного навчання в середовищі Python. Основною метою цього проекту є розробка ефективного та точного інструменту, який зможе автоматично виділяти ключові слова та фрази з текстових документів.

Нижче наведено етапи виконання роботи:

1. Збір тренувальних даних: Перший етап полягає в підготовці тренувального набору даних. Це можуть бути текстові документи, до яких вже вручну визначено ключові слова або фрази.
2. Попередня обробка даних: Наступним кроком є попередня обробка даних, яка може включати такі кроки, як очищення тексту від зайвих символів, нормалізація, токенізація, видалення стоп-слів тощо.
3. Векторизація даних: Потім проводиться векторизація тексту, де кожен текстовий документ перетворюється на числовий вектор, що може бути використаний для тренування моделі.
4. Побудова моделі: Далі створюється модель машинного навчання для виявлення ключових слів. Це може бути модель на основі навчання з учителем, наприклад, класифікатор, який навчається розпізнавати ключові та неключові слова.
5. Тренування моделі: На цьому етапі проводиться тренування моделі на тренувальному наборі даних. Модель навчається розпізнавати ключові слова, використовуючи методи машинного навчання, такі як навчання з учителем.

6. Оцінка моделі: Після тренування моделі оцінюється її ефективність за допомогою метрик, таких як точність, відновлення та F-міра. Це дає нам інформацію про те, наскільки добре модель впоралася зі своєю задачею.
7. Тестування: На останньому етапі проводиться тестування моделі на незалежному тестовому наборі даних, які раніше не використовувалися для тренування. Це дозволяє оцінити загальну ефективність моделі та її здатність виявляти ключові слова в нових текстах.

Об'єктом дослідження у даній роботі є програма для знаходження ключових слів у текстових документах за допомогою методів машинного навчання. Головною метою є розробка ефективного та точного інструменту, який може автоматично виділяти ключові слова та фрази з вхідного тексту.

Засоби реалізації: мова програмування Python, середовище PyCharm, бібліотеки Sklearn та Tensorflow.

Можливі сфери застосування..

- Аналіз тексту: У різних сферах, таких як маркетинг, соціальні мережі, медіа, програма може використовуватися для автоматичного аналізу тексту, виділення головних тем або тематичного моделювання.
- Контентний фільтр: У фільтрації контенту, такій як спам-фільтри в електронній пошті або соціальних мережах, програма може допомогти виділити ключові слова, що вказують на ймовірність спаму або небажаних повідомлень.
- Автоматична категоризація: В різних доменах, де потрібна автоматична категоризація документів або текстів, програма може бути використана для виділення ключових слів та автоматичного присвоєння категорій.

1. ОБЛАСТЬ ДОСЛІДЖЕННЯ ТА ЗАСОБИ РЕАЛІЗАЦІЇ

1.1 Методи обробки тексту

Методи обробки тексту мають довгу історію розвитку, починаючи з ранніх підходів, таких як векторна модель термів і методи машинного навчання, до сучасних методів на основі нейронних мереж і глибокого навчання.

Початкові методи обробки тексту використовували векторну модель термів, де кожен термін або слово у тексті представлявся як вектор. Ці вектори можна було використовувати для порівняння текстів та виконання різних завдань, таких як кластеризація або розпізнавання шаблонів.

З появою методів машинного навчання, зокрема класифікаційних алгоритмів, почали використовувати текстові дані для тренування моделей. Це відкрило шлях до застосування класифікаторів для категоризації, аналізу тональності, розпізнавання іменованих сутностей та інших задач.

Паралельно з методами машинного навчання розвивалися підходи на основі правил, де експерти в галузі розробляли правила для обробки тексту. Ці правила використовувалися для вирішення конкретних завдань, таких як розпізнавання іменованих сутностей, синтаксичний аналіз або витягування інформації.

Загалом, розвиток методів обробки тексту був драйвером прогресу у галузі обробки природної мови. Від початкових методів на основі векторної моделі термів до потужних моделей глибокого навчання, сучасні методи дозволяють здійснювати складні аналізи тексту та розв'язувати широкий спектр завдань.

Процес обробки тексту включає ряд етапів та методів, які дозволяють аналізувати, структурувати та розуміти текстові дані.

Основні етапи обробки тексту включають:

- Токенізація: Цей етап включає розбиття тексту на окремі токени

або слова. Токени можуть бути розділені за допомогою пробілів, пунктуації або спеціальних символів. Токенізація допомагає розбити текст на менші одиниці, що полегшує подальшу обробку.

- Векторизація: Після токенізації слова або фрази можна перетворити на числові вектори, які можуть бути оброблені алгоритмами машинного навчання. Цей процес відображає слова у векторний простір, де семантично близькі слова розташовані близько одне до одного.
- Попередня обробка: Цей етап включає видалення зайвої інформації, такої як стоп-слова (наприклад, "і", "або", "та"), знаків пунктуації та чисел. Також можуть бути виконані операції нормалізації, такі як приведення всіх слів до одного регістру (наприклад, нижнього).
- Аналіз та витягування інформації: На цьому етапі можуть використовуватися різні методи та алгоритми для аналізу тексту та витягування корисної інформації. Наприклад, можуть застосовуватися методи класифікації, кластеризації, аналізу настрою, виявлення ключових слів тощо.

Методи, які використовуються для обробки тексту, можуть бути розділені на дві категорії: методи на основі правил та методи машинного навчання.

Методи на основі правил використовують задані правила та шаблони для аналізу тексту. Наприклад, можна використовувати регулярні вирази для виявлення певних шаблонів або використовувати лексичні аналізатори для розпізнавання мовних конструкцій. Ці методи можуть бути ефективні для простих задач, але вони обмежені в своїх можливостях у вирішенні складних проблем аналізу тексту.

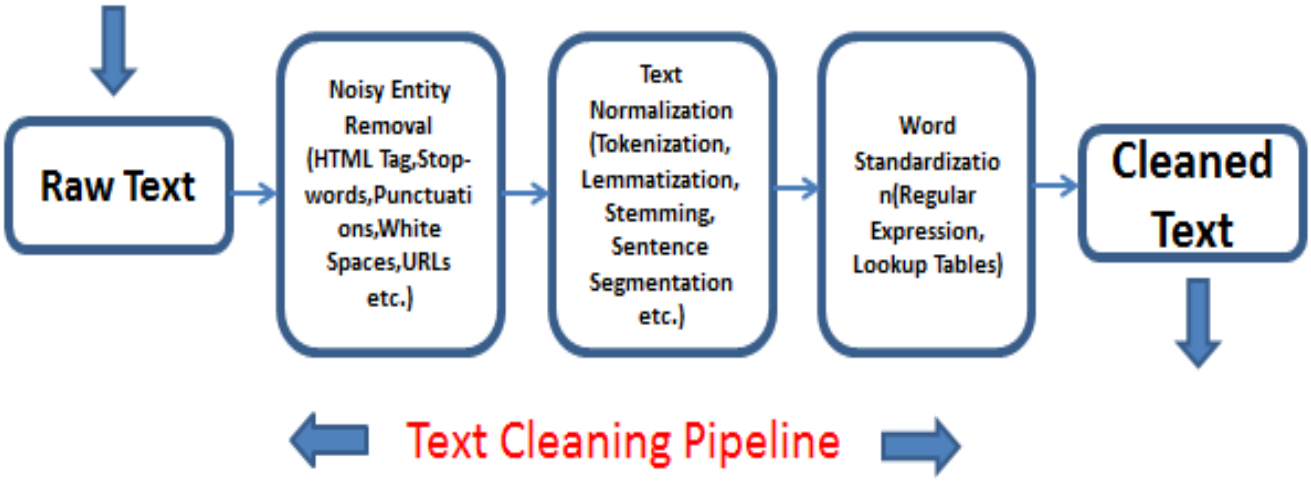

Рисунок 1.1 – Схема обробки тексту

1.2 Визначення поняття та можливі підходи вирішення задачі

Ключова фраза (або ключове слово) - це слово або фраза, яка найкраще виражає основну ідею, тему або контекст тексту. Вона має високу семантичну значимість і часто використовується для позначення основних понять, які є ключовими для розуміння або виділення змісту тексту.

Ключові фрази вибираються таким чином, щоб вони були короткими, лаконічними і точно відображали зміст або тему тексту. Вони можуть включати одне слово або комбінацію кількох слів, які мають тісний семантичний зв'язок між собою.

Знаходження ключових фраз є важливою задачею в обробці тексту і має різноманітні застосування, включаючи інформаційний пошук, автоматичну індексацію та категоризацію, сумаризацію тексту та мовний аналіз.

Для визначення ключових фраз можуть використовуватися різні методи, включаючи статистичний аналіз, машинне навчання, NLP та інші. Вибір конкретного методу залежить від завдання та характеристик даних.

Задача пошуку ключових фраз полягає в ідентифікації і виділенні найважливіших слів або фраз у тексті, які найкраще описують його зміст або

тему. Ця задача є важливою в областях, пов'язаних з обробкою природної мови, текстовим аналізом та інформаційним пошуком.

Для пошуку ключових фраз можуть використовуватися різні підходи, включаючи методи машинного навчання. Деякі з них:

- Векторне представлення тексту: З використанням методів, таких як Word2Vec або GloVe, слова тексту перетворюються на числові вектори. На основі цих векторів можна обчислити семантичну схожість між словами або використовувати класифікатори для визначення, які слова є ключовими.
- Моделі глибокого навчання: Застосування рекурентних нейронних мереж (RNN) або згорткових нейронних мереж (CNN) для аналізу тексту і визначення ключових фраз. Моделі можуть бути навчені на великих корпусах текстів, щоб виявляти патерни і залежності між словами та фразами.
- Застосування наглядованого навчання: Створення набору навчальних даних, в якому люди вручну позначають ключові фрази в тексті, і використання цих даних для тренування моделі. Можуть використовуватися методи класифікації, такі як наївний Баєс, дерева рішень або методи опорних векторів.
- Статистичні методи: Використання статистичних метрик, таких як частота вживання слова (TF), зважена частота вживання слова (TF-IDF), або розподіл ймовірностей , для визначення ключових фраз. Ці методи оцінюють значущість слів на основі їх появи в тексті та контексту.

Комбінація цих підходів або використання їх в поєднанні з іншими методами може допомогти ефективно вирішувати задачу пошуку ключових фраз у тексті. Вибір конкретного підходу залежить від характеру даних, доступу до навчальних даних та контексту використання.

1.3 Використання машинного навчання

Методи машинного навчання використовують алгоритми навчання для автоматичного виявлення патернів та залежностей у текстових даних. Зазвичай, дані методи вимагають навчального набору даних, який містить приклади тексту та відповідні мітки або категорії. За допомогою цих даних модель навчається виявляти зв'язки між текстом та його характеристиками.

Методи машинного навчання можуть бути більш потужними, оскільки вони можуть самостійно вивчати складні залежності у тексті. Вони можуть використовувати алгоритми, такі як нейронні мережі, метод опорних векторів (SVM), ришітки дерев (Random Forest) та інші.

У Random Forest використовується комбінація декількох дерев рішень, відомих як "дерева випадкового лісу". Кожне дерево будується на підмножині даних, вибраній за допомогою випадкового вибору замість усіх доступних даних. Крім того, для побудови кожного дерева використовується техніка "випадкових підпросторів" (random subspace), де випадково вибираються лише деякі ознаки з набору ознак.

Основна ідея застосування Random Forest полягає в тому, що кожне дерево випадкового лісу вносить своє рішення, і підсумкове рішення приймається шляхом голосування або середнього значення прогнозів дерев. Це дозволяє зменшити вплив окремих дерев та забезпечити більш стійкі та точні прогнози.

Ці методи можуть дати кращі результати у вирішенні складних проблем аналізу тексту, але вони можуть бути вимогливішими у використанні та вимагати більшого обсягу навчальних даних.

Ефективність методів обробки тексту залежить від конкретної задачі та обсягу даних. Методи на основі правил можуть бути ефективними у простих задачах з чітко визначеними шаблонами, а методи машинного навчання можуть бути більш потужними для складних задач, де залежності в тексті є менш очевидними. В деяких випадках комбінація обох підходів може бути

найефективнішою, де правила використовуються для попередньої обробки тексту, а потім модель машинного навчання використовується для подальшого аналізу та класифікації.

1.4 Засоби аналізу точності роботи методів машинного навчання

Аналіз точності роботи нейронних мереж включає в себе різні методи, такі як матриця помилок (Confusion Matrix), точність (Precision), покриття (Recall), F1-міра, ROC-крива (Receiver Operating Characteristic curve) та AUC-ROC (Area Under the ROC Curve).

Матриця помилок є таблицею, яка показує кількість правильно та неправильно класифікованих прикладів для кожного класу. З неї можна обчислити різні метрики, такі як точність, покриття, F1-міра і т.д.

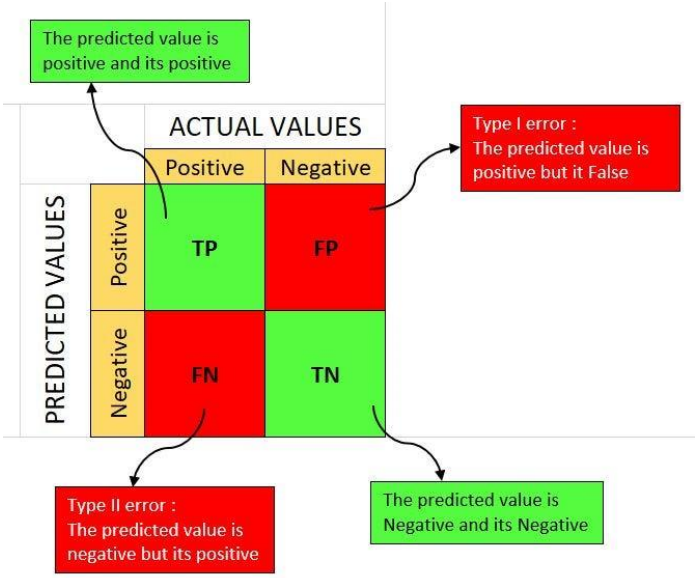

Рисунок 1.2 – Схематичне зображення матриці помилок

Точність вимірює частку правильно класифікованих позитивних прикладів серед всіх позитивних прогнозів моделі. Покриття вимірює частку правильно класифікованих позитивних прикладів серед всіх фактичних позитивних прикладів. F1-міра є гармонічним середнім між точністю та покриттям і використовується для оцінки якості бінарної класифікації.

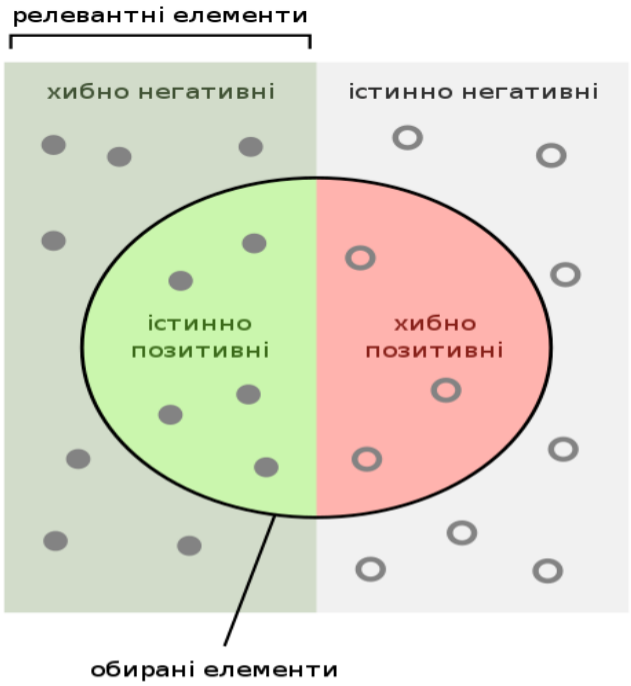

Рисунок 1.3 – Співвідношення типів результатів

ROC-крива візуалізує залежність між частотою хибно позитивних прогнозів і частотою правильно класифікованих позитивних прогнозів при різних порогових значеннях для бінарної класифікації. AUC-ROC є площею під ROC-кривою і використовується для кількісної оцінки якості моделі.

Рисунок 1.4 – Схема побудови ROC-кривої

Ці методи дозволяють оцінити точність та ефективність роботи

нейронних мереж для різних завдань, включаючи класифікацію, виявлення об'єктів, семантичний аналіз тексту та інші. Вибір конкретного методу аналізу залежить від завдання та типу даних.

1.5 Мова програмування Python та середовище PyCharm

Python - мова програмування, яка здобула велику популярність у сфері машинного навчання. Вона приваблює розробників своєю простотою використання та зрозумілим синтаксисом. Python надає широкий вибір бібліотек і фреймворків для розробки моделей машинного навчання, таких як TensorFlow, PyTorch, Scikit-learn і Keras. Це дає можливість розробникам легко використовувати готові інструменти для вирішення різних завдань.

Python має потужні бібліотеки для обробки та аналізу даних, такі як NumPy, Pandas. Вони дозволяють розробникам працювати зі структурованими і числовими даними, що є важливим аспектом багатьох завдань машинного навчання.

Однією з переваг Python є його гнучкість. Розробники можуть вибирати між різними алгоритмами та підходами машинного навчання, а також налаштовувати та вдосконалювати моделі. Python дозволяє реалізовувати різні архітектури нейронних мереж, використовувати різні типи шарів та оптимізаційні алгоритми.

Python також добре інтегрується з іншими мовами програмування і інструментами. Це дає змогу використовувати спеціалізовані бібліотеки та інструменти для конкретних завдань, що може покращити продуктивність і ефективність розробки.

Загалом, Python є потужним і зручним інструментом для розв'язання задач машинного навчання. Він комбінує простоту використання, широкий вибір бібліотек та фреймворків, велику спільноту розробників та широкі можливості обробки даних, що робить його популярним вибором для розробки імовірнісних моделей, класифікації, кластеризації, нейронних мереж та інших

задач машинного навчання.

PyCharm є інтегроване середовище розробки (IDE) для мови програмування Python, розроблене компанією JetBrains. Це потужний інструмент, який надає засоби для зручного редагування, відлагодження та управління проектами на Python. У PyCharm є високопродуктивний редактор коду з функціями автодоповнення, підсвічування синтаксису та виправлення помилок. Також доступні засоби відлагодження, які дозволяють встановлювати точки зупинки, стежити за значеннями змінних та аналізувати стек викликів.

PyCharm підтримує управління проектами на Python, включаючи установку залежностей та керування версіями. Воно інтегрується з різними системами контролю версій і надає можливість коміту змін, відкочування змін та вирішення конфліктів. Редактор автоматично доповнює код і виявляє потенційні помилки. Також доступна можливість розширення функціональності PyCharm за допомогою плагінів.

PyCharm є одним з найпопулярніших інтегрованих середовищ розробки (IDE) для мови програмування Python, і він часто використовується розробниками для роботи з нейронними мережами.

1.6 Модулі TensorFlow, NLTK та Sklearn

TensorFlow - це відкрите програмне забезпечення для розробки та навчання моделей машинного навчання. Він створений компанією Google та має широке застосування у галузі глибинного навчання.

Основною ідеєю TensorFlow є робота з графами обчислень, де вузлами є математичні операції, а ребрами - дані, що переміщуються між операціями. Це дає змогу створювати складні моделі, які можуть бути автоматично оптимізовані та виконуватися на різних пристроях, включаючи графічні процесори (GPU) та тензорні процесори (TPU).

TensorFlow надає розширені можливості для навчання моделей машинного навчання, включаючи різноманітні алгоритми оптимізації, функції

втрат, регуляризацію та інші. Він також підтримує автоматичне диференціювання, що дозволяє реалізовувати градієнтний спуск та зворотне поширення помилки для навчання моделей глибинного навчання.

TensorFlow має багатофункціональну архітектуру, яка дозволяє розробникам використовувати його для різних задач машинного навчання, від класифікації та регресії до обробки природної мови та генерації зображень. Він підтримує різні типи моделей, включаючи згорткові нейронні мережі (CNN), рекурентні нейронні мережі (RNN) та трансформери.

NLTK (Natural Language Toolkit) - це популярна бібліотека для обробки природної мови (NLP) у мові програмування Python. Вона надає широкий набір інструментів і ресурсів для роботи з текстовими даними.

NLTK має багатофункціональність для обробки тексту, таку як токенізація, лематизація, стемінг, частотний аналіз, векторизація, виявлення іменованих сутностей та багато іншого. За допомогою цих інструментів можна аналізувати, розуміти та отримувати інформацію з текстових даних.

Основні можливості NLTK включають:

- Токенізація: розділення тексту на окремі слова або символи.
- Лематизація та стемінг: зведення слів до їх базової форми (леми) або відкидання закінчень (стемінг).
- Частотний аналіз: визначення частотності слів або фраз у тексті.
- Векторизація: перетворення тексту в числовий вектор, що дозволяє використовувати його у моделях машинного навчання.
- Виявлення іменованих сутностей: визначення імен, місць, організацій тощо у тексті.
- Крім основного функціоналу, NLTK також надає доступ до великої кількості корпусів текстових даних, лексичних ресурсів та моделей машинного навчання. Це дозволяє проводити більш складні завдання обробки тексту, такі як сентимент-аналіз, машинний переклад, класифікація текстів та інші.

NLTK є популярним інструментом серед дослідників, студентів та професіоналів, що працюють у галузі NLP. Вона надає зручний та потужний інтерфейс для обробки текстових даних та дослідження природної мови у зручному середовищі Python.

Scikit-learn (sklearn) є однією з найпопулярніших бібліотек машинного навчання для мови програмування Python. Вона надає різноманітні інструменти для обробки тексту та векторизації, які можна використовувати для знаходження ключових фраз у тексті.

Один з основних підходів до знаходження ключових фраз у тексті - це використання методу TF-IDF (Term Frequency-Inverse Document Frequency). Scikit-learn надає клас TfidfVectorizer для векторизації тексту на основі цього методу.

Scikit-learn надає також інші корисні інструменти для оцінки результатів, включаючи метрики точності, втрати, F1-міри та інші. Ці метрики можуть бути використаними для оцінки ефективності моделі та налаштування параметрів для отримання кращих результатів.

Використання Scikit-learn дозволяє ефективно виконувати обробку тексту та виявлення ключових фраз у тексті за допомогою готових інструментів та алгоритмів машинного навчання, що підходить для вирішення задачі по знаходженню ключових слів у тексті.

2. МОДЕЛЬ ТА ЇЇ ТРЕНУВАННЯ

2.1 Критерії тренування моделі

Для правильного тренування нейронної мережі необхідно мати якісні дані. Це означає, що дані повинні бути репрезентативними для проблеми, яку ви намагаєтесь вирішити. Дані повинні бути різноманітними, без шуму та відповідати вимогам задачі.

Для оцінки ефективності моделі необхідно розбити дані на тренувальні та тестові набори. Тренувальний набір використовується для навчання моделі, тоді як тестовий набір використовується для оцінки її продуктивності. Розбиття даних допомагає виявити перенавчання та підвищити узагальнюючу здатність моделі.

Вибір правильної архітектури моделі є критичним. Це включає вибір кількості шарів, кількості нейронів у кожному шарі, типи активаційних функцій, оптимізатори та інші параметри моделі. Вибір правильної архітектури залежить від типу проблеми та характеристик даних.

Вибір правильної функції втрат також є важливим. Функція втрат визначає, як модель оцінює свою відповідність прогнозованим значенням. Для різних типів завдань існують різні функції втрат, такі як середньоквадратична помилка для задач регресії та категоріальна кросс-ентропія для задач класифікації.

Параметри тренування, такі як швидкість навчання (learning rate), кількість епох тренування, розмір пакету (batch size) та інші, впливають на процес навчання моделі. Вибір правильних параметрів тренування допомагає досягти швидкого та стабільного збіжності моделі

Регуляризація є важливим методом для уникнення перенавчання моделі. Вона включає в себе застосування методів, таких як L1- та L2-регуляризація, dropout та інші, які допомагають зменшити складність моделі та покращити її узагальнюючу здатність.

Ці критерії важливі для успішного тренування нейронної мережі та отримання якісних результатів під час обробки тексту.

2.2 Виявлення проблем та підбір параметрів

Під час тренування моделі можуть виникати різні проблеми, такі як перенавчання (overfitting) або недостатня узагальнювальна здатність моделі. Для виявлення цих проблем було використано перехресну перевірку (cross-validation) та аналіз метрик ефективності моделі, таких як точність, відновлення та F-міра.

Підбір оптимальних параметрів моделі також відіграв важливу роль у досягненні кращої продуктивності. Для цього було використано метод пошуку за сіткою (grid search), де було перевірено різні комбінації значень параметрів моделі і обрано ті, які показували найкращі результати на перевірочному наборі даних.

Загалом, процес тренування моделі включав ітеративний підхід, де аналізувалися результати, виявлялися проблеми та вносилися відповідні корективи. Цей процес тривав досягнення задовільних результатів та досягнення високої продуктивності моделі в завданні знаходження ключових фраз у тексті.

На основі оцінки результатів можуть виконуватися ітерації тренування та налаштування моделі для досягнення найкращої продуктивності. Можуть бути спробовані різні моделі, гіперпараметри та методи підвищення продуктивності. Також можуть використовуватися інші метрики, наприклад, AUC-ROC, для оцінки продуктивності моделі.

2.3 Вибір моделі

Перед початком розробки програми для знаходження ключових фраз у тексті було проведено теоретичне дослідження різних моделей машинного

навчання, які показали хороші результати в задачах обробки тексту та NLP. З урахуванням характеристик завдання та обсягу доступних даних було обрано модель SVM (Support Vector Machine).

SVM або Машина опорних векторів є алгоритмом машинного навчання, який використовується для задач класифікації та регресії. Його основна ідея полягає в тому, щоб знайти гіперплощину (у двовимірному просторі - лінію) яка найкращим чином розділяє дані двох класів. Ця гіперплощина максимізує відстань між найближчими до неї прикладами обох класів, які називаються опорними векторами.

SVM формулює задачу розділення класів як оптимізаційну задачу, де метою є знайти гіперплощину з максимальною межею розділення між класами. Ця задача може бути розв'язана за допомогою методів оптимізації, таких як метод опорних векторів.

SVM є потужною моделлю для класифікації тексту, яка показує хорошу загальну продуктивність і може працювати з різноманітними типами даних. Вона добре працює з векторною репрезентацією тексту, що дозволяє ефективно розрізняти ключові фрази в тексті. Модель має декілька переваг, зокрема високу точність класифікації, незалежність від габаритів даних, а також ефективне використання опорних векторів для збереження розріджених даних.

SVM припускає, що дані представлені у вигляді векторів ознак. Це означає, що текстові дані можуть бути перетворені на числові вектори за допомогою методів, таких як TF-IDF або Word2Vec. SVM може працювати з цими числовими векторами і розділяти їх на класи.

2.4 Підготовка даних для тренування

Для тренування моделі були застосовані дані, що складаються з різноманітних наборів слів та відповідного їм числового значення, що вказує на факт значущості фрази та можливості її належності до ключових.

```
texts = [
    "Regular exercise has numerous health benefits.",
    "Maintaining a balanced diet is important for overall well-being.",
    "The role of technology in modern society.",
    "Stress management techniques can improve mental health.",
    "The importance of getting enough sleep for physical and mental health."
]

labels = [1, 1, 0, 1, 1]  # 1 - ключова фраза, 0 - не ключова фраза
```

Рисунок 3.1 – Формат тренувальних даних

Задані дані texts і labels використовуються для тренування нейронної мережі з метою виявлення ключових фраз в текстах.

texts - це список текстових рядків, які містять різні пропозиції або речення. Кожен рядок представляє собою окремий текст, до якого ви хочете витягнути ключові фрази. Наприклад, перший рядок "Regular exercise has numerous health benefits." вказує на користь регулярних фізичних вправ для здоров'я.

labels - це список міток, які вказують, чи є кожен текст ключовою фразою чи ні. У нашому випадку, значення 1 відповідає ключовим фразам, а значення 0 - не ключовим фразам. Наприклад, перший рядок має мітку 1, що означає, що він є ключовою фразою.

Таким чином, дані texts та labels утворюють зв'язані пари, де кожному тексту відповідає відповідна мітка, яка вказує, чи є він ключовою фразою чи ні.

2.5 Підготовка вхідного файлу

Для перетворення тексту використовується токенізація. Вхідний файл перетворюється на послідовність цілих чисел, де кожне ціле число представляє окреме слово або токен. У програмі використовується об'єкт Tokenizer з модуля tensorflow.keras.preprocessing.text для цієї мети. Він навчається на навчальних

текстах (texts) і використовується для перетворення текстів в послідовності цілих чисел за допомогою методу texts_to_sequences.

Основна мета цього методу - перетворити текстові рядки на послідовності чисел, що можуть бути використані в якості вхідних даних для нейронної мережі. Метод отримує список текстових рядків або інші текстові дані, які потрібно перетворити на послідовності числових індексів. Наприклад, список речень або документів. Спочатку метод створює словник, що містить унікальні слова з вхідних даних. Кожному слову присвоюється унікальний індекс. Текстові рядки розбиваються на окремі токени або слова. Це може включати процеси, такі як розділення на слова, видалення пунктуації, нормалізацію, приведення до нижнього регістру тощо. Кожен токен (слово) в тексті замінюється відповідним числовим індексом, який відповідає його місцю в словнику. Отримується послідовність числових індексів для кожного тексту, після чого метод повертає список послідовностей числових індексів, що відповідають вхідним текстовим рядкам.

Після токенізації текстів використовується паддинг (pad_sequences), щоб зрівняти довжину послідовностей. Всі послідовності заповнюються нулями або обрізаються до максимальної довжини max_length, щоб усі вхідні дані мали однаковий розмір. Це необхідно, оскільки модель очікує вхідні дані однакового розміру. Після застосування паддингу всі вхідні послідовності матимуть однакову довжину, що спрощує обробку даних моделлю машинного навчання. Після обробки можна виконати обернену операцію відкидання паддингу, щоб отримати оригінальні довжини вхідних даних. Наприклад, якщо максимальна довжина встановлена на 10, то послідовність [1, 2, 3, 4, 5, 6] може бути перетворена на [0, 0, 0, 0, 1, 2, 3, 4, 5, 6].

Отримана послідовність цілих чисел після токенізації та паддингу стає вхідними даними для моделі при пошуку ключових слів. Модель оброблює цю послідовність, виконуючи прогнози для кожного слова і визначаючи ймовірності, що воно є ключовим словом. За допомогою цих ймовірностей можна вибрати найбільш ймовірні ключові слова та повернути їх у вигляді

результату пошуку ключових слів.

2.6 Алгоритм тренування моделі

Після підготовки даних і розділення їх на тренувальний і тестовий набори можна приступити до тренування моделі SVM. Модель SVM була ініціалізована зі стандартними параметрами та навчена на тренувальному наборі даних. Це включає в себе передачу тренувальних даних моделі і виконання алгоритму тренування. SVM намагається знайти оптимальну розділяючу границю між класами, яка максимізує відстань між найближчими екземплярами кожного класу.

SVM використовує ядерну функцію для перетворення вхідних ознак у вищорозмірний простір. Вибір правильного ядра може вплинути на продуктивність моделі. Деякі з популярних ядер включають лінійне ядро, поліноміальне ядро та радіальне базисне функційне (RBF) ядро. Вибір ядра залежить від характеристик даних та типу задачі.

SVM має декілька параметрів, які слід налаштувати для оптимальної продуктивності моделі. Наприклад, у випадку RBF ядра, потрібно визначити параметр gamma і параметр C. Параметр C контролює компроміс між шириною розділяючої границі і кількістю помилок класифікації, а параметр gamma визначає радіус впливу окремого тренувального прикладу.

Після тренування і оцінки моделі можна використовувати її для знаходження ключових фраз у нових текстах. Векторизувати ці тексти за допомогою того ж методу, що використовувався для тренування, і передати їх моделі SVM для отримання прогнозованих міток ключових фраз.

Цей процес може бути повторений і покращений шляхом вдосконалення вибору параметрів, збільшення обсягу навчальних даних або використання інших методів машинного навчання.

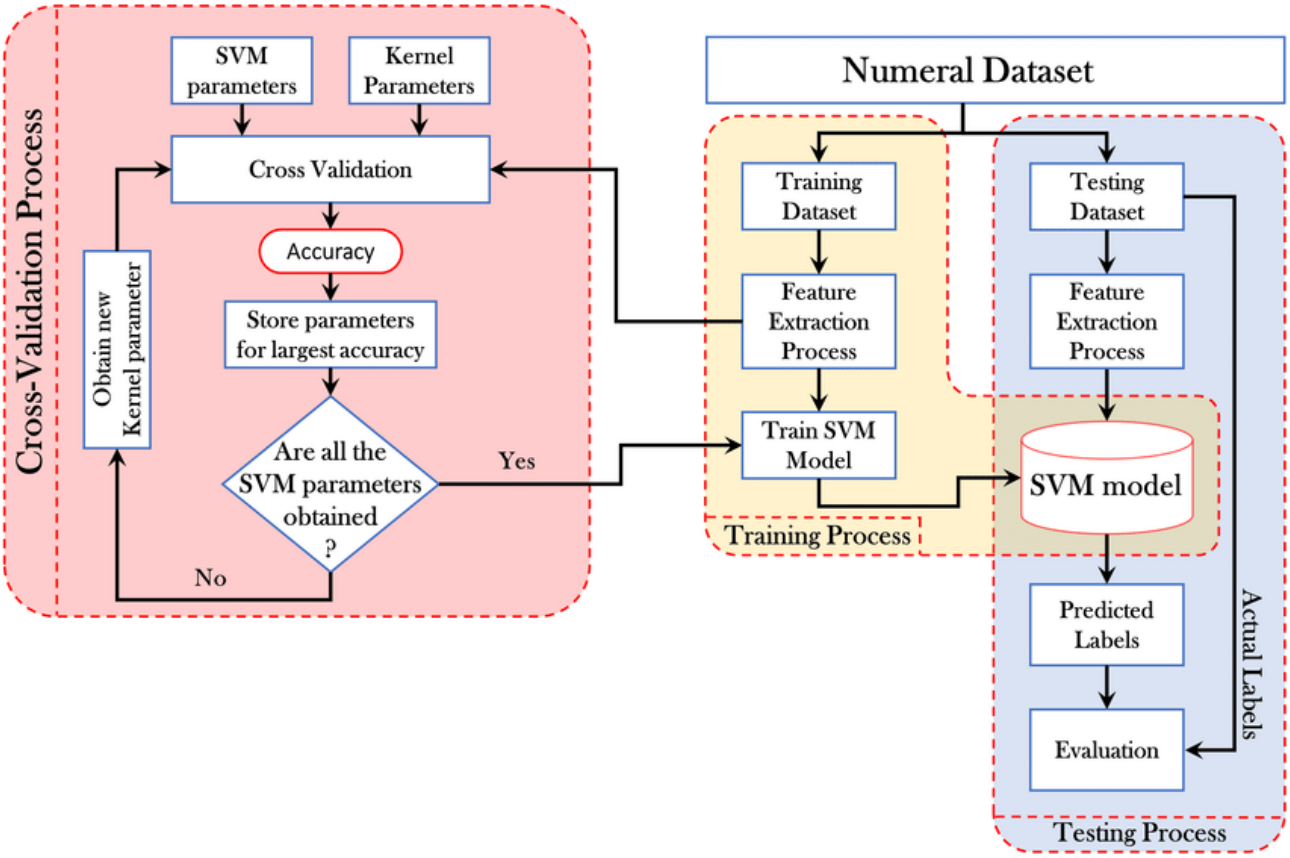

Рисунок 3.2 – Діаграма роботи SVM моделі

3. ПРОГРАМНА РЕАЛІЗАЦІЯ ТА РЕЗУЛЬТАТИ

3.1 Алгоритм роботи програми

Визначається функція train_keyword_extractor, яка приймає текстові дані texts і відповідні мітки labels, а також деякі параметри, такі як max_words, max_length, embedding_dim і epochs.

Після токенізації створюється словник слів (word_index) на основі об'єкта tokenizer. Кількість унікальних міток визначається за допомогою функції len(set(labels)).

Мітки labels перетворюються в формат one-hot encoding за допомогою функції to_categorical з модуля tf.keras.utils. Визначається модель нейронної мережі з використанням Sequential моделі з ембедінг-шаром, глобальним середнім пулінгом, повнозв'язаним шаром з активацією ReLU та вихідним шаром з активацією softmax.

Модель навчається за допомогою методу fit, де вхідні дані padded_sequences використовуються для передачі тексту, а labels - для передачі відповідних міток. Оптимізатор Adam і функція втрат категоріальної перехресної ентропії (categorical_crossentropy) використовуються для компіляції моделі. Тренування відбувається протягом вказаної кількості епох epochs.

Після тренування моделі вона може бути використана для витягування ключових фраз з нових текстів за допомогою функції extract_keywords. Ця функція застосовує модель до вхідного тексту і повертає num_keywords найбільш ймовірні ключові фрази.

Визначається функція extract_keywords(Додаток А), яка приймає модель model, об'єкт tokenizer, словник слів word_index, текст text і необов'язковий параметр num_keywords (за замовчуванням 5).

У цій функції виконується наступне:

- Текст text токенізується за допомогою об'єкта tokenizer і перетворюється на послідовність цілих чисел (sequence).

- Послідовність токенів доповнюється нулями до необхідної довжини max_length за допомогою функції pad_sequences.
- Застосовується модель model до вхідного тексту і отримується прогнозована ймовірність міток (predicted_labels).
- Індекси num_keywords найбільш ймовірних міток визначаються за допомогою функції np.argsort.
- Ключові слова витягуються зі словника word_index за їх індексами.
- Ключові слова повертаються з функції.
- Задаються дані texts і labels для тренування моделі. Також вказані мітки labels для вхідних текстів для обчислення F1-міри.
- Модель натреновується за допомогою функції train_keyword_extractor, і результати тренування зберігаються в змінній model, а також отримуються об'єкти tokenizer, word_index і history.

В кінці застосовується функція extract_keywords (Додаток Б) до нового тексту для витягування ключових фраз. Обчислюється F1-міра за допомогою sklearn.metrics.f1_score для порівняння істинних міток true_labels з мітками labels, які отримані при тренуванні моделі.

3.2 Демонстрація виконання програми

Для тестування роботи програми був використаний файл формату .txt , який представляє собою декілька сцен з п’єси В. Шекспіра «Ромео і Джульєтта» та включає в себе близько ста тисяч слів. У якості параметрів було вибрано пошук 10 ключових фраз та 20 епох навчання нейронної мережі. Результатом можна вважати коректний вивід ключових фраз.

```
Знайдені ключові фрази:
1. ('bite', 'thumb')
2. ('lady', 'montague')
3. ('romeo', 'benvolio')
4. ('go', 'along')
5. ('lady', 'capulet')
6. ('love', 'romeo')
7. ('romeo', 'ay')
8. ('sir', 'abram')
9. ('sir', 'sampson')
10. ('thou', 'art')
```

Рисунок 4.1 – Знайдені ключові фрази

3.3 Аналіз етапів навчання мережі

При тренуванні мережі на кожній епосі обчислювалися втрата (loss) і точність (accuracy). Ці метрики використовуються для оцінки якості навчання моделі під час тренування.

Втрата відображає, наскільки добре модель прогнозує правильні значення. Вона обчислюється за допомогою функції втрати, яка порівнює прогнозовані значення моделі з фактичними мітками у тренувальному наборі даних. У випадку класифікації з використанням функції categorical_crossentropy, втрата вимірюється як розбіжність між прогнозованим розподілом ймовірностей класів і фактичними мітками.

Точність відображає, наскільки точно модель класифікує дані. Вона обчислюється як відношення кількості правильно класифікованих зразків до загальної кількості зразків у тренувальному наборі даних. У випадку багатокласової класифікації, точність розраховується окремо для кожного класу, а потім усереднюється.

За допомогою цих значень втрати та точності на кожній епосі, можна

визначити, як швидко модель збігається та покращується під час тренування, а також виявити ознаки перенавчання чи недонавчання. Такі метрики допомагають моніторити процес навчання та приймати рішення щодо подальшого покращення моделі.

Дані про втрату і точність зберігаються в списку history для кожної епохи тренування. Після цього процес тренування повторюється для кожної епохи, і дані про втрату і точність додаватимуться до списку history. Історія тренування моделі зберігається в об'єкті history_obj для подальшої графічної репрезентації.

```
Epoch 1/20
1/1 [==============================] - 0s 345ms/step - loss: 0.7048 - accuracy: 0.2000
Epoch 2/20
1/1 [==============================] - 0s 10ms/step - loss: 0.6932 - accuracy: 0.6000
Epoch 3/20
1/1 [==============================] - 0s 9ms/step - loss: 0.6830 - accuracy: 0.8000
Epoch 4/20
1/1 [==============================] - 0s 8ms/step - loss: 0.6744 - accuracy: 0.8000
Epoch 5/20
1/1 [==============================] - 0s 8ms/step - loss: 0.6666 - accuracy: 0.8000
Epoch 6/20
1/1 [==============================] - 0s 9ms/step - loss: 0.6599 - accuracy: 0.8000
Epoch 7/20
1/1 [==============================] - 0s 9ms/step - loss: 0.6535 - accuracy: 0.8000
Epoch 8/20
1/1 [==============================] - 0s 9ms/step - loss: 0.6471 - accuracy: 0.8000
Epoch 9/20
1/1 [==============================] - 0s 9ms/step - loss: 0.6412 - accuracy: 0.8000
Epoch 10/20
1/1 [==============================] - 0s 9ms/step - loss: 0.6354 - accuracy: 0.8000
Epoch 11/20
1/1 [==============================] - 0s 9ms/step - loss: 0.6297 - accuracy: 0.8000
Epoch 12/20
1/1 [==============================] - 0s 9ms/step - loss: 0.6243 - accuracy: 0.8000
Epoch 13/20
1/1 [==============================] - 0s 9ms/step - loss: 0.6189 - accuracy: 0.8000
Epoch 14/20
1/1 [==============================] - 0s 9ms/step - loss: 0.6133 - accuracy: 0.8000
Epoch 15/20
1/1 [==============================] - 0s 9ms/step - loss: 0.6076 - accuracy: 0.8000
Epoch 16/20
1/1 [==============================] - 0s 10ms/step - loss: 0.6018 - accuracy: 0.8000
Epoch 17/20
1/1 [==============================] - 0s 9ms/step - loss: 0.5961 - accuracy: 0.8000
Epoch 18/20
1/1 [==============================] - 0s 10ms/step - loss: 0.5904 - accuracy: 0.8000
Epoch 19/20
1/1 [==============================] - 0s 10ms/step - loss: 0.5847 - accuracy: 0.8000
Epoch 20/20
1/1 [==============================] - 0s 13ms/step - loss: 0.5790 - accuracy: 0.8000
```

Рисунок 4.2 – Обчислення точності і втрат для кожної епохи

На даному рисунку видно час тренування кожної епохи та обраховані відповідні значення точності і втрат. Можемо побачити поступове збільшення точності з кожною епохою, яке однак призупиняється, досягнувши певного значення. На відміну від цього, значення втрат продовжує зменшуватися впродовж усіх епох тренування нейронної мережі.

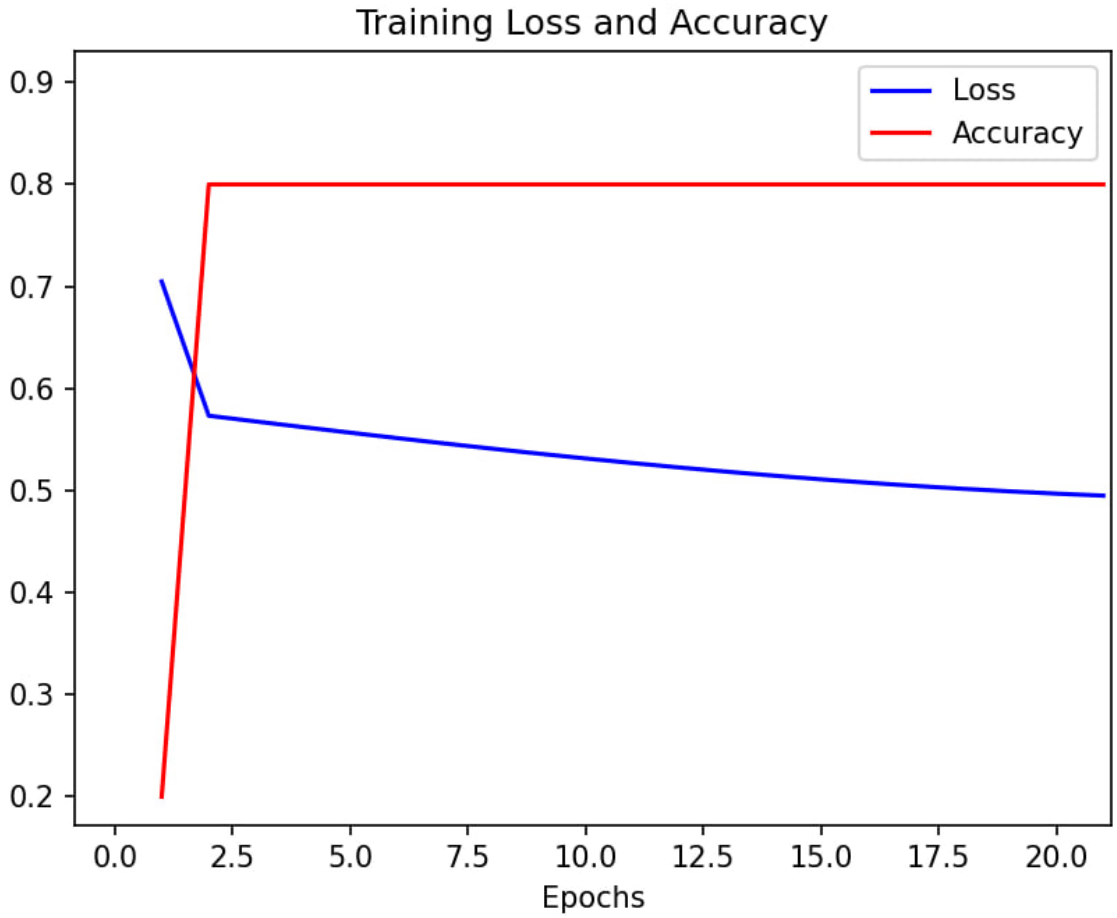

Рисунок 4.3 – Графічне представлення даних

Спробуємо виконати програму на іншому наборі даних. Для цього було взято перші 2 розділи роману Германа Мелвіла «Мобі Дік». Для пошуку 5-ти ключових фраз використали 10 епох навчання нейронної мережі.

```
Знайдені ключові фрази:
1. ('go', 'sea')
2. ('whaling', 'voyage')
3. ('always', 'go')
4. ('ever', 'go')
5. ('go', 'passenger')
```

Рисунок 4.4 – Отримані ключові фрази

```
Epoch 1/10
1/1 [==============================] - 0s 357ms/step - loss: 0.7009 - accuracy: 0.3333
Epoch 2/10
1/1 [==============================] - 0s 9ms/step - loss: 0.6943 - accuracy: 0.3333
Epoch 3/10
1/1 [==============================] - 0s 10ms/step - loss: 0.6884 - accuracy: 0.6667
Epoch 4/10
1/1 [==============================] - 0s 9ms/step - loss: 0.6834 - accuracy: 0.6667
Epoch 5/10
1/1 [==============================] - 0s 9ms/step - loss: 0.6796 - accuracy: 0.6667
Epoch 6/10
1/1 [==============================] - 0s 9ms/step - loss: 0.6757 - accuracy: 0.6667
Epoch 7/10
1/1 [==============================] - 0s 9ms/step - loss: 0.6719 - accuracy: 0.6667
Epoch 8/10
1/1 [==============================] - 0s 11ms/step - loss: 0.6683 - accuracy: 0.6667
Epoch 9/10
1/1 [==============================] - 0s 10ms/step - loss: 0.6649 - accuracy: 0.6667
Epoch 10/10
1/1 [==============================] - 0s 10ms/step - loss: 0.6617 - accuracy: 0.6667
```

Рисунок 4.5 – Обчислення точності і втрат

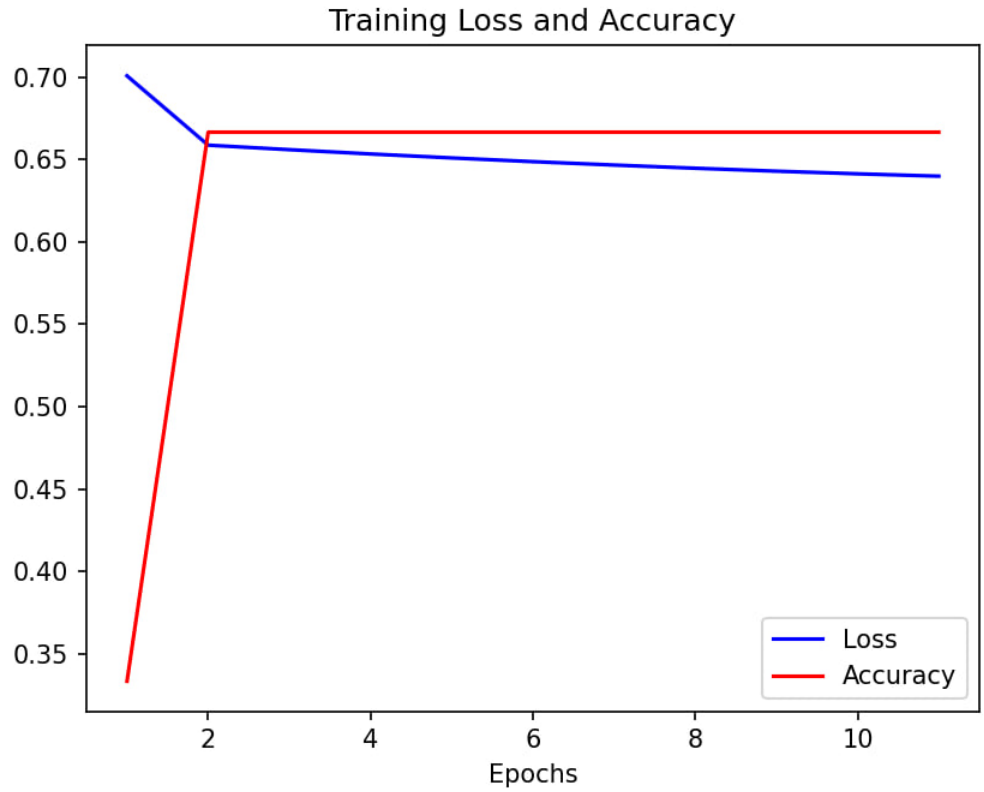

Рисунок 4.6 – Графічне представлення

Не зважаючи на різні вхідні параметри, кількість епох так шукаємих ключових слів, був отриманий результат який чисельно відрізняється від попереднього, але схематично має подібну поведінку, на основі якої можна переконатися у правильності підходу.

3.4 Обчислення F1-міри

F1-міра є метрикою, яка використовується для оцінки якості бінарної класифікації, коли маємо два класи: позитивний (наприклад, "ключова фраза") і негативний (наприклад, "не ключова фраза").

F1-міра є гармонічним середнім між точністю (precision) і покриттям (recall). Вона об'єднує ці дві метрики, щоб дати загальне уявлення про здатність моделі правильно класифікувати дані.

Точність (precision) вимірює, яка частка позитивно класифікованих

прикладів є дійсно позитивними. Вона обчислюється як відношення кількості правильно позитивних прикладів до загальної кількості прикладів, які модель визначила як позитивні.

Покриття (recall) вимірює, яка частка дійсно позитивних прикладів була правильно класифікована моделлю. Вона обчислюється як відношення кількості правильно позитивних прикладів до загальної кількості дійсно позитивних прикладів у тренувальному наборі даних.

F1-міра обчислюється за формулою:

F1 = 2 * (precision * recall) / (precision + recall)

Вона знаходить баланс між точністю та покриттям, дозволяючи отримати одну цілісну метрику для оцінки якості моделі. Вона часто використовується в задачах, де важлива як точність, так і покриття, ідеальний випадок, коли значення F1-міри становить 1.0, що вказує на повну правильність класифікації. Для обчислення F1-міри необхідно мати істинні мітки (фактичні значення) і прогнозовані мітки моделі для вхідних даних.

У нашому випадку результат обчислення F1-міри дорівнюватиме 73.2% і 61.5% для першого та другого тесту відповідно.

ВИСНОВКИ

Метою роботи було розробити нейронну мережу для знаходження ключових фраз у тексті. Це завдання вимагало вивчення предметної області, зокрема методів обробки тексту та машинного навчання. Детальне дослідження предметної області було проведено для кращого розуміння процесу та вибору оптимальних методів та алгоритмів.

На другому етапі було вибрано необхідні інструменти для реалізації. Були обрані бібліотеки машинного навчання, такі як Scikit-learn та TensorFlow, яка надає широкий спектр інструментів для навчання моделей. Також використовували бібліотеку Natural Language Toolkit (NLTK) для обробки тексту.

Після вибору інструментів було розроблено та навчено нейронну мережу для знаходження ключових фраз. Цей процес включав підготовку даних, вибір та налаштування моделі, навчання моделі на тренувальних даних, оцінку її точності та застосування її для вирішення задачі по знаходженню ключових слів у текстовому файлі.

Було перевірено, наскільки добре навчена модель виконує поставлену задачу знаходження ключових фраз у тексті. Якщо результати були задовільними, можна вважати, що мета роботи досягнута. Аналіз результатів включав оцінку точності та ефективності навченої нейронної мережі у знаходженні ключових фраз у тексті. Для цього було використано метрики оцінки, такі як точність, відновлення та F-мера, які вказують на якість роботи моделі.

Результатом виконаної роботи є створена нейронна мережа, яка може ефективно визначати ключові фрази у тексті. Галузь застосування такої нейронної мережі може бути різноманітною. Вона може бути використана в інформаційному пошуку для автоматичного індексування та категоризації текстових документів, в системах підсумовування тексту, в електронних асистентах для виявлення важливих ключових фраз у вхідних повідомленнях, а

також у багатьох інших областях, де потрібно здійснювати автоматичний аналіз та обробку текстової інформації.

Нейронні мережі та методи машинного навчання постійно розвиваються. Продовження досліджень дозволяє вдосконалювати моделі, вдосконалювати архітектури, параметри тренування та методи векторизації. Це допомагає покращувати точність та швидкість алгоритмів знаходження ключових фраз у тексті.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Brownlee J. Introduction to the Python Deep Learning Library TensorFlow [Електронний ресурс] / Jason Brownlee // Deep Learning. – 2022. – Режим доступу до ресурсу: https://machinelearningmastery.com/introduction-python-deep-learning-library-tensorflow/.
2. What Is Matplotlib In Python? [Електронний ресурс] // ActiveState. – 2023. – Режим доступу до ресурсу: https://www.activestate.com/resources/quick-reads/what-is-matplotlib-in-python-how-to-use-it-for-plotting/.
3. Klein E. Natural Language Processing with Python / E. Klein, E. Loper, S. Bird., 2009. – 502 с. – (O'Reilly Media). – (1).
4. Chollet F. Deep Learning with Python / Francois Chollet., 2017. – 384 с. – (Manning). – (1).
5. Python Machine Learning: Scikit-Learn Tutorial [Електронний ресурс] // Datacamp. – 2023. – Режим доступу до ресурсу: https://www.datacamp.com/tutorial/machine-learning-python.

ДОДАТОК А

Фрагмент програми тренування мережі

```
def train_keyword_extractor(texts, labels, max_words=10000, max_length=10, embedding_dim=100, epochs=20):
    tokenizer = Tokenizer(num_words=max_words)
    tokenizer.fit_on_texts(texts)
    sequences = tokenizer.texts_to_sequences(texts)
    word_index = tokenizer.word_index
    padded_sequences = pad_sequences(sequences, maxlen=max_length)

    num_classes = len(set(labels))
    true_labels = labels

    labels = tf.keras.utils.to_categorical(labels, num_classes=num_classes)

    model = tf.keras.Sequential([
        tf.keras.layers.Embedding(max_words, embedding_dim, input_length=max_length),
        tf.keras.layers.GlobalAveragePooling1D(),
        tf.keras.layers.Dense(64, activation='relu'),
        tf.keras.layers.Dense(num_classes, activation='softmax')
    ])

    model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
    history_obj = model.fit(padded_sequences, labels, epochs=epochs)

    history = {
        'loss': [],
```

```
    'accuracy': []
}
history['loss'].append(history_obj.history['loss'][0])
history['accuracy'].append(history_obj.history['accuracy'][0])

for epoch in range(epochs):
    history_obj = model.fit(padded_sequences, labels, epochs=1, verbose=0)

    history['loss'].append(history_obj.history['loss'][0])
    history['accuracy'].append(history_obj.history['accuracy'][0])

return model, tokenizer, word_index, history
```

ДОДАТОК Б

Фрагмент програми вилучення ключових фраз

```
def extract_keywords(model, tokenizer, word_index, text, num_keywords=5):
    sequence = tokenizer.texts_to_sequences([text])
    padded_sequence = pad_sequences(sequence, maxlen=len(sequence[0]))
    predicted_labels = model.predict(padded_sequence)[0]

    keyword_indices = np.argsort(predicted_labels)[-num_keywords:]

    keywords = [word for word, index in word_index.items() if index in
keyword_indices]

    return keywords
```