Міністерство освіти і науки України Національний університет

«Одеська політехніка»

Кафедра прикладної математики

ЗВІТ З ПРЕДДИПЛОМНОЇ ПРАКТИКИ НА ТЕМУ

«Програмне забезпечення аналізу настрою тексту за допомогою нейронної мережі глибокого навчання»

Виконала:
студентка 4 курсу
групи НАВ-191
Матиченко А.Д.

Прийняла:
доктор техн. наук. доцент.
Полякова М.В.

Одеса 2023

СПИСОК СКОРОЧЕНЬ

NLP (Natural Language Processing) - Обробка Природної Мови

MLP (Multi-Layer Perceptron) - Багатошаровий Персептрон

ANN (Deep Artificial Neural Networks) - Глибокі Штучні Нейронні Мережі

DNN (Deep Neural Network) - Глибока Нейронна Мережа

RNN (Recurrent Neural Network) – Рекурентна Нейрона Мережа

LSTM (Long Short Term Memory Network) - Мережа Довгострокової Пам'яті

ML (Machine Learning) - Машинне Навчання

ЗМІСТ

ВСТУП

Ключові слова: обробка природної мови, аналіз настроїв, текстова аналітика, глибоке навчання, нейроні мережі, python.

Мета проекту: розробка програмного забезпечення (нейронних мереж) для реалізації аналізу настроїв на основі текстової аналітики.

Цей проект слугує для самостійного вивчення елементів глибокого навчання і поглибити знання.

У цій роботі ми розглянемо аспекти, що стосуються обробки природної мови (NLP), текстової аналітики та глибокого навчання. Ми зосередимося на конкретних реальних проблемах і сценаріях, а також на тому, як ми можемо використовувати глибоке навчання для їх вирішення.

Наша проблема полягає в аналізі настроїв або аналізі думок, коли ми хочемо проаналізувати деякі текстові документи та передбачити їхні настрої чи думку на основі вмісту цих документів.

Аналіз настроїв також відомий як аналіз думок або аналіз думок. Ключова ідея полягає в тому, щоб використовувати методи аналізу тексту, NLP, глубокого навчання та лінгвістики для вилучення важливої інформації або даних із неструктурованого тексту. Це, у свою чергу, може допомогти нам отримати якісні результати, як-от загальний настрій на позитивній, нейтральній або негативній шкалі, і кількісні результати, як-от полярність настроїв, пропорції суб’єктивності та об’єктивності.

Ми зосереджуємося на спробах проаналізувати велику кількість рецензій на фільми та визначити настрої. Ми розглядаємо спектр методів аналізу настроїв, серед яких:

- новіші керовані моделі глибокого навчання
- розширені керовані моделі глибокого навчання

1. Етапи і терміни написання нейронних мереж

Загальний термін написання нейронної мережі складе один місяць або 31 день. Період написання: до 03.03.2023. Робота буде складатися з наступних етапів:

- пошук відповідної літератури;
- Вивчення бібліотек
- Вибір інструментів та середовищ для розвитку нейронних мереж
- Вивчення основних розділів, пов'язаних з нейронними мережами;

2. Технологічні вимоги до нейронних мереж

2.1 Система повинна виконувати наступні функції:

-Імпортувати дані
-Обробка даних
-Нормалізація даних
- Налаштування параметрів нейронної мережі
-Навчання нейронних мереж
-Візуалізація даних
-Збереження навченої нейронної мережі
-Експортувати дані

2.2 Базові дані:

- Випадковий шум
- Статистика
- Параметри нормалізації
- Параметри навчання нейронних мереж

2.3 Новіші моделі керованого глибокого навчання

Для цієї моделі ми будемо використовувати повністю пов’язану чотиришарову глибоку нейронну мережу (багатошаровий персептрон (MLPs) або глибоку (ANN)).

2.4 Розширені керовані моделі глибокого навчання

Для цієї моделі ми будемо користуватися новим і цікавим підходом до керованого глибокого навчання є використання рекурентних нейронних мереж (RNN) і мереж довготривалої короткочасної пам’яті (LSTM), які також враховують послідовність даних (слів, подій тощо).

3. Структура і опис реалізації

3.1 Попередня обробка та нормалізація тексту

Один із ключових кроків перед тим, як зануритися в процес розробки функцій і моделювання, включає очищення, попередню обробку та нормалізацію тексту, щоб привести текстові компоненти, як-от фрази та слова, до певного стандартного формату. Це забезпечує стандартизацію всього корпусу документів, що допомагає створювати значущі функції та допомагає зменшити шум, який може виникати через багато факторів, як-от нерелевантні символи, спеціальні символи, теги XML і HTML тощо.

Для попередньої обробки та нормалізації тексту були прописані наступні функції:

Очищення тексту: наш текст часто містить непотрібний вміст, як-от теги HTML, які не додають великої цінності під час аналізу настроїв. Тому нам потрібно переконатися, що ми видалили їх перед тим, як видобувати функції. Бібліотека BeautifulSoup виконує чудову роботу, надаючи необхідні для цього функції. Функція strip_html_tags(...) дозволяє очищати та видаляти HTML-код.

```
from bs4 import BeautifulSoup

def strip_html_tags(text):
    soup = BeautifulSoup(text, "html.parser")
    stripped_text = soup.get_text()
    return stripped_text
```

Видалення символів із наголосами: у нашому наборі даних ми маємо справу з оглядами англійською мовою, тому нам потрібно переконатися, що

символи будь-якого іншого формату, особливо символи з наголосами, перетворюються та стандартизуються на символи ASCII. Простим прикладом може бути перетворення é на e. Функція remove_accented_ chars(...) допомагає нам у цьому.

```
import unicodedata

def remove_accented_chars(text):
    text = unicodedata.normalize('NFKD', text).encode('ascii',
'ignore').decode('utf-8', 'ignore')
    return text
```

Розширені скорочення: скорочення - це в основному скорочені версії слів або складів. Ці скорочені версії існуючих слів або фраз створюються шляхом видалення певних літер і звуків. Найчастіше зі слів видаляють голосні.

```
def expand_contractions(text, contraction_mapping=CONTRACTION_MAP):

    contractions_pattern =
re.compile('({})'.format('|'.join(contraction_mapping.keys())),
                                    flags=re.IGNORECASE|re.DOTALL)
    def expand_match(contraction):
        match = contraction.group(0)
        first_char = match[0]
        expanded_contraction = contraction_mapping.get(match) \
                                if contraction_mapping.get(match) \
                                 else contraction_mapping.get(match.lower())
        expanded_contraction = first_char+expanded_contraction[1:]
        return expanded_contraction

    expanded_text = contractions_pattern.sub(expand_match, text)
    expanded_text = re.sub("'", "", expanded_text)
    return expanded_text
```

Видалення спеціальних символів: ще одним важливим завданням очищення та нормалізації тексту є видалення спеціальних символів і символів, які часто створюють додатковий шум у неструктурованому тексті. Для цього можна використовувати прості регулярні вирази.

```
import re

def remove_special_characters(text):
    text = re.sub('[^a-zA-Z0-9\s]', '', text)
    return text
```

Створення основи та лемматизація: основи слів, як правило, є основною формою можливих слів, які можна створити шляхом приєднання до основи афіксів, таких як префікси та суфікси, для створення нових слів. Це відомо як перегин. Зворотний процес отримання основної форми слова відомий як коріння. Простим прикладом є слова ДИВИТЬСЯ, ДИВИТИСЯ та ДИВИВСЯ.

```
def lemmatize_text(text):
    text = nlp(text)
    text = ' '.join([word.lemma_ if word.lemma_ != '-PRON-' else word.text
for word in text])
    return text
```

Видалення стоп-слів: слова, які мають невелике значення або взагалі не мають значення, особливо при побудові значущих функцій із тексту, також відомі як стоп-слова або стоп-слова. Зазвичай це слова, які мають максимальну частоту, якщо ви використовуєте простий термін або частоту слів у корпусі документів.

```
def remove_stopwords(text, is_lower_case=False):
    tokens = tokenizer.tokenize(text)
    tokens = [token.strip() for token in tokens]
    if is_lower_case:
        filtered_tokens = [token for token in tokens if token not in
stopword_list]
    else:
        filtered_tokens = [token for token in tokens if token.lower() not in
stopword_list]
    filtered_text = ' '.join(filtered_tokens)
    return filtered_text
```

Ми використовуємо всі ці компоненти та об’єднуємо їх разом у наступній функції під назвою normalize_corpus(...), яку можна використати для отримання корпусу документа як вхідних даних і повернення того самого корпусу за допомогою очищених та нормалізованих текстових документів.

```
def normalize_corpus(corpus, html_stripping=True, contraction_expansion=True,
                     accented_char_removal=True, text_lower_case=True,
                     text_lemmatization=True, special_char_removal=True,
                     stopword_removal=True):

    normalized_corpus = []
    for doc in corpus:
        if html_stripping:
            doc = strip_html_tags(doc)
        if accented_char_removal:
            doc = remove_accented_chars(doc)
        if contraction_expansion:
            doc = expand_contractions(doc)
        if text_lower_case:
            doc = doc.lower()
        doc = re.sub(r'[\r|\n|\r\n]+', ' ',doc)
        special_char_pattern = re.compile(r'([{.(-)!}])')
        doc = special_char_pattern.sub(" \\1 ", doc)
        if text_lemmatization:
            doc = lemmatize_text(doc)
        if special_char_removal:
            doc = remove_special_characters(doc)
        doc = re.sub(' +', ' ', doc)
        if stopword_removal:
            doc = remove_stopwords(doc, is_lower_case=text_lower_case)
        normalized_corpus.append(doc)
    return normalized_corpus
```

3.2 Новіші моделі керованого глибокого навчання

ANN базується на сукупності з’єднаних одиниць або вузлів, які називаються штучними нейронами, які приблизно моделюють нейрони біологічного мозку. Кожне з’єднання, як синапси в біологічному мозку, може передавати сигнал до інших нейронів. Штучний нейрон отримує сигнали, потім обробляє їх і може сигналізувати підключеним до нього нейронам. «Сигнал» у з’єднанні є дійсним числом , а вихід кожного нейрона обчислюється за допомогою деякої нелінійної функції суми його входів.

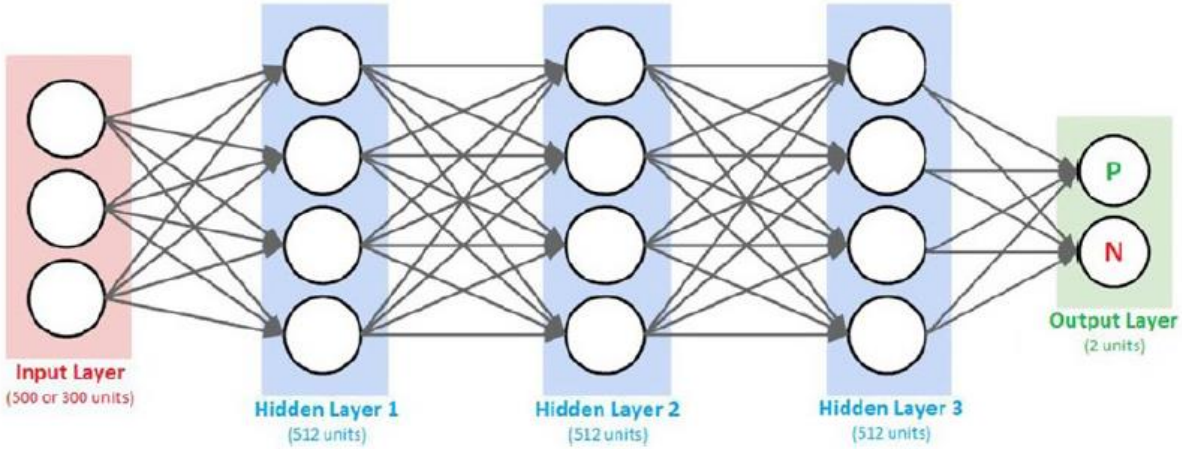

Рис 1. Повністю пов’язана модель глибокої нейронної мережі для класифікації настроїв

3.2.1 Використані бібліотеки

Для успішного написання нейронної мережі на мові Python в розширені Jupiter в середовищі Visual Studio Code і перепідготовки даних нам знадобляться наступні бібліотеки:

Pandas — це пакет Python, який забезпечує швидкі, гнучкі та виразні структури даних, розроблені для того, щоб зробити роботу з «реляційними» даними. Він має на меті стати основним будівельним блоком високого рівня для практичного аналізу реальних даних , а саме: обчислює статистичні дані та відповідайте на запитання щодо даних; очищає дані; зберігає очищені, трансформовані дані назад у CSV, інший файл або базу даних;

NumPy — це модуль Python з відкритим вихідним кодом, який надає вам високопродуктивний об’єкт багатовимірного масиву та широкий вибір функцій для роботи з масивами.

Scikit-learn — це безкоштовна бібліотека ML для Python, яка містить різні алгоритми класифікації, регресії та кластеризації. Ви можете використовувати Scikit-learn разом із бібліотеками NumPy і SciPy.

model_evaluation_utils — оцінка моделі — це процес використання різних показників оцінювання для розуміння ефективності моделі машинного навчання, а також її сильних і слабких сторін.

Keras — це бібліотека глибокого навчання з відкритим кодом, написана на Python. Він був ініційований Франсуа Шолле і вперше опублікований 28 березня 2015 року. Keras забезпечує уніфікований інтерфейс для різних серверних програм, зокрема TensorFlow, Microsoft Cognitive Toolkit і Theano.

Gensim — це бібліотека з відкритим вихідним кодом для неконтрольованого тематичного моделювання, індексування документів,

пошуку схожості та інших функцій обробки природної мови за допомогою сучасного статистичного машинного навчання.

IPython — це інтерпретатор командного рядка для інтерактивної роботи з мовою програмування Python. Це не просто розширення вбудованої оболонки Python, а набір програм для розробки та запуску програм Python

3.2.2 Завантаження та нормалізація даних

Для нашої задачі ми візьмемо базу даних відгуків з сайту kaggle. Розіб'ємо нашу базу даних на дві вибірки і проведемо нормалізацію. Нормалізація даних - це процес стандартизації діапазону значень даних. Тексти — це ще один клас необроблених даних, які потребують особливої уваги та обробки, перш ніж наші алгоритми зможуть зрозуміти

Текстові дані, що представляють природну мову, є дуже шумними та вимагають власного набору кроків для суперечок. Рядкові дані зазвичай проходять такі етапи суперечок, як токенізація (розбиття рядкових даних на складові одиниці), лематизація, видалення стоп-слова, тому під час нормалізації наших даних скористуємося функцією, яку прописали раніше:

```
dataset = pd.read_csv(r'movie_reviews.csv')

reviews = np.array(dataset['review'])
sentiments = np.array(dataset['sentiment'])

# build train and test datasets
train_reviews = reviews[:35000]
train_sentiments = sentiments[:35000]
test_reviews = reviews[35000:]
test_sentiments = sentiments[35000:]

# normalize datasets
norm_train_reviews = tn.normalize_corpus(train_reviews)
norm_test_reviews = tn.normalize_corpus(test_reviews)
```

3.2.3 Кодування міток класу передбачення

Оскільки кодування міток у Python є частиною попередньої обробки даних, ми скористаємося модулем попередньої обробки з пакета sklearn та імпортуємо клас LabelEncoder, як показано нижче:

```
le = LabelEncoder()
num_classes=2
# tokenize train reviews & encode train labels
tokenized_train = [tn.tokenizer.tokenize(text)
                   for text in norm_train_reviews]
y_tr = le.fit_transform(train_sentiments)
y_train = keras.utils.to_categorical(y_tr, num_classes)
# tokenize test reviews & encode test labels
tokenized_test = [tn.tokenizer.tokenize(text)
                   for text in norm_test_reviews]
y_ts = le.fit_transform(test_sentiments)
y_test = keras.utils.to_categorical(y_ts, num_classes)
```

3.2.4 Розробка функцій із вбудованими словами

Існує кілька просунутих моделей векторизації слів, які нещодавно набули великої популярності. Вбудовування слів можна використовувати для виділення ознак і моделювання мови. Це представлення намагається зіставити кожне слово чи фразу в повний числовий вектор таким чином, щоб семантично схожі слова чи терміни мали тенденцію зустрічатися ближче одне до одного, і їх можна було кількісно визначити за допомогою цих вбудовань. Модель word2vec є, мабуть, однією з найпопулярніших імовірнісних мовних моделей на основі нейронної мережі, і її можна використовувати для вивчення

векторів розподіленого представлення слів. Вбудовування слів, створене за допомогою word2vec, передбачає взяття корпусу текстових документів, що представляють слова у великому великому векторному просторі так, що кожне слово має відповідний вектор у цьому просторі, а подібні слова (навіть семантично) розташовані близько одне до одного, аналогічно те, що ми спостерігали в подібності документів раніше:

```
w2v_num_features = 500
w2v_model = gensim.models.Word2Vec(tokenized_train, size=w2v_num_features,
window=150,
                                   min_count=10, sample=1e-3)
avg_wv_train_features = averaged_word2vec_vectorizer(corpus=tokenized_train,
model=w2v_model,
                                                     num_features=500)
avg_wv_test_features = averaged_word2vec_vectorizer(corpus=tokenized_test,
model=w2v_model,
                                                    num_features=500)
```

3.2.5 Побудова архітектури глибокої нейронної мережі

Ми будемо використовувати повністю пов’язану глибоку нейронну мережу (DNN), оскільки нейрони або одиниці в кожній парі суміжних шарів повністю попарно з’єднані. Наступна функція використовує keras поверх tensorflow для створення потрібної моделі DNN.

```
def construct_deepnn_architecture(num_input_features):
    dnn_model = Sequential()
    dnn_model.add(Dense(512, activation='relu',
input_shape=(num_input_features,)))
    dnn_model.add(Dropout(0.2))
    dnn_model.add(Dense(512, activation='relu'))
    dnn_model.add(Dropout(0.2))
    dnn_model.add(Dense(512, activation='relu'))
    dnn_model.add(Dropout(0.2))
    dnn_model.add(Dense(2))
    dnn_model.add(Activation('softmax'))
```

```
    dnn_model.compile(loss='categorical_crossentropy', optimizer='adam',
                      metrics=['accuracy'])
    return dnn_model
w2v_dnn = construct_deepnn_architecture(num_input_features=500)
```

Кінцевий вихідний рівень складається з двох блоків із функцією активації softmax. Функція softmax в основному є узагальненням логістичної функції, яку ми бачили раніше, яку можна використовувати для представлення розподілу ймовірностей на n можливих результатів класу. У нашому випадку n = 2, де клас може бути додатним або від'ємним, а ймовірності softmax допоможуть нам визначити те саме. Двійковий класифікатор softmax також взаємозамінно відомий як функція бінарної логістичної регресії.

3.2.6 Візуалізація прикладу роботи глибокої архітектури

Візуалізуємо архітектуру моделі DNN за допомогою keras, подібно до того, що ми зробили, використовуючи наступний код.

```
from IPython.display import SVG
from keras.utils.vis_utils import model_to_dot

SVG(model_to_dot(w2v_dnn, show_shapes=True, show_layer_names=False,
                 rankdir='TB').create(prog='dot', format='svg'))
```

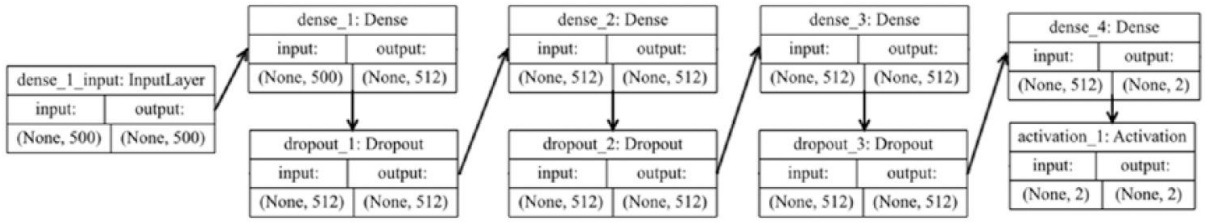

Рис 2. Візуалізація архітектури глибокої нейронної мережі.

3.2.7 Навчання моделі, прогнозування та оцінка продуктивності

Для навчання нашої глибокої нейронної мережі зробимо п’ять епох з розміром вибірки 100.

```
batch_size = 100
w2v_dnn.fit(avg_wv_train_features, y_train, epochs=5, batch_size=batch_size,
            shuffle=True, validation_split=0.1, verbose=1)
y_pred = w2v_dnn.predict_classes(avg_wv_test_features)
predictions = le.inverse_transform(y_pred)
meu.display_model_performance_metrics(true_labels=test_sentiments,
predicted_labels=predictions,
                                      classes=['positive', 'negative'])
```

```
Model Performance metrics:
------------------------------
Accuracy: 0.88
Precision: 0.88
Recall: 0.88
F1 Score: 0.88

Model Classification report:
------------------------------
             precision    recall  f1-score   support

   positive       0.88      0.89      0.88      7510
   negative       0.89      0.87      0.88      7490

avg / total       0.88      0.88      0.88     15000


Prediction Confusion Matrix:
------------------------------
                  Predicted:
                    positive negative
Actual: positive        6711      799
        negative         952     6538
```

Ми отримуємо точність перевірки близько 88% .

3.3 Розширені керовані моделі глибокого навчання

Іншим новим і цікавим підходом до керованого глибокого навчання є використання рекурентних нейронних мереж (RNN) і мереж довготривалої короткочасної пам’яті (LSTM), які також враховують послідовність даних (слів, подій тощо). Це більш просунуті моделі, ніж ваші звичайні повністю підключені глибокі мережі, і зазвичай для навчання потрібно більше часу. Ми будемо використовувати keras поверх tensorflow і спробуємо побудувати тут модель класифікації на основі LSTM і використовувати вбудовування слів як наші функції.

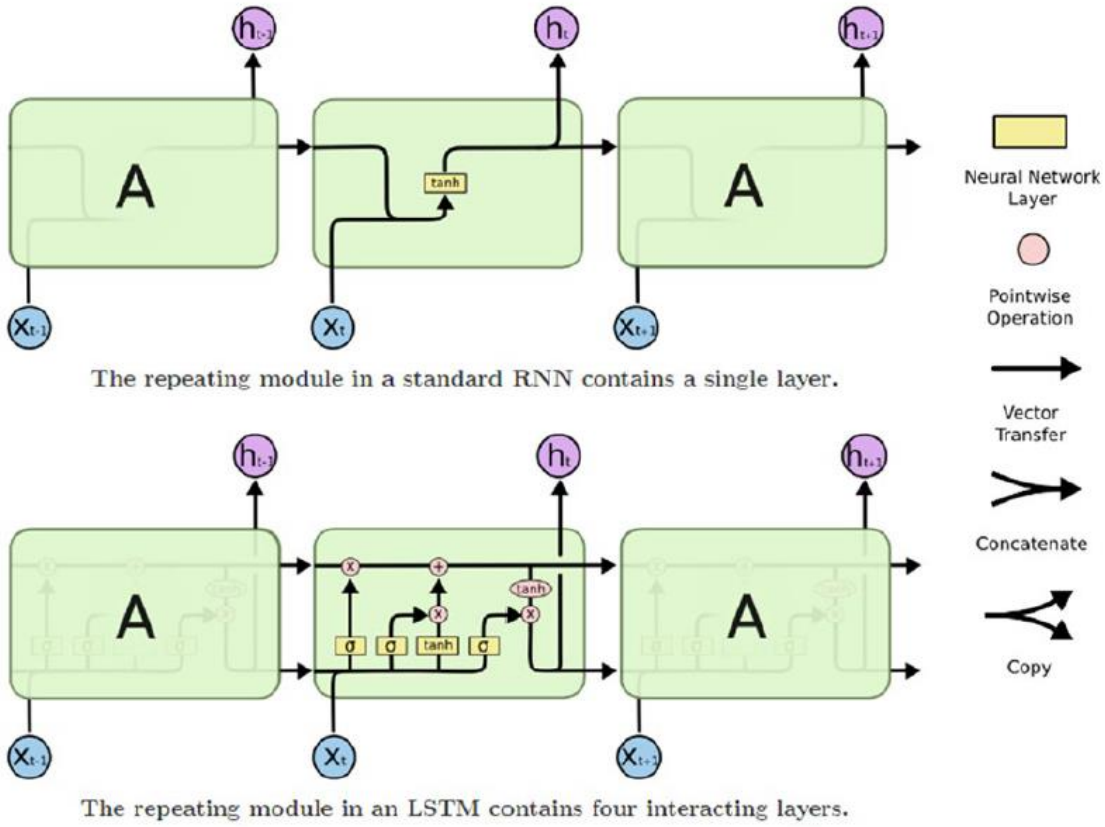

Рис 3. Візуалізація рекурентної нейронної мережі і мереж довготривалої короткочасної пам’яті.

3.3.1 Розбиття тексту на блоки для тренування і навчання

Ми працюватимемо над нашими нормалізованими та попередньо обробленими наборами даних огляду навчання та тестування, norm_train_reviews і norm_test_reviews, які ми створили під час нашого попереднього аналізу. Токенізуватимемо ці набори даних, щоб кожен текстовий огляд розкладався на відповідні токени.

```
tokenized_train =[tn.tokenizer.tokenize(text) for text in norm_train_reviews]
tokenized_test = [tn.tokenizer.tokenize(text) for text in norm_test_reviews]
```

3.3.2 Створення картографічного словника

Вбудовування Word має тенденцію векторизувати текстові документи у вектори фіксованого розміру, щоб ці вектори намагалися захопити контекстну та семантичну інформацію. Для генерації вбудовування ми будемо використовувати рівень вбудовування від keras, який вимагає представлення документів у вигляді токенізованих і числових векторів. Ми вже маємо вектори токенізованого тексту в наших змінних tokenized_train і tokenized_text. Однак нам потрібно було б перетворити їх у числові представлення. Окрім цього, нам також знадобиться, щоб вектори були однакового розміру, навіть незважаючи на те, що огляди токенізованого тексту матимуть різну довжину через різницю в кількості маркерів у кожному огляді. Нам потрібно створити слово для індексації словникового запасу для представлення кожного лексемованого текстового огляду в числовій формі:

```
token_counter = Counter([token for review in tokenized_train for token in
review])
vocab_map = {item[0]: index+1 for index, item in
enumerate(dict(token_counter).items())}
max_index = np.max(list(vocab_map.values()))
vocab_map['PAD_INDEX'] = 0
vocab_map['NOT_FOUND_INDEX'] = max_index+1
vocab_size = len(vocab_map)
```

3.3.3 Кодування та додавання наборів даних і кодування міток класу передбачення

Ми можело легко відфільтрувати та використати тут більш релевантні терміни (залежно від їх частоти), використовуючи функцію most_common(count) із Counter і взявши перші терміни підрахунку зі списку унікальних терміни в навчальному корпусі. Тепер ми будемо кодувати токенізовані текстові огляди на основі попередньої vocab_map.

```
from keras.preprocessing import sequence
from sklearn.preprocessing import LabelEncoder

# get max length of train corpus and initialize label encoder
le = LabelEncoder()
num_classes=2 # positive -> 1, negative -> 0
max_len = np.max([len(review) for review in tokenized_train])

## Train reviews data corpus
# Convert tokenized text reviews to numeric vectors
train_X = [[vocab_map[token] for token in tokenized_review] for
tokenized_review in tokenized_train]
train_X = sequence.pad_sequences(train_X, maxlen=max_len) # pad
## Train prediction class labels
# Convert text sentiment labels (negative\positive) to binary encodings (0/1)
train_y = le.fit_transform(train_sentiments)

## Test reviews data corpus
# Convert tokenized text reviews to numeric vectors
test_X = [[vocab_map[token] if vocab_map.get(token) else
vocab_map['NOT_FOUND_INDEX']
           for token in tokenized_review]
              for tokenized_review in tokenized_test]
test_X = sequence.pad_sequences(test_X, maxlen=max_len)
## Test prediction class labels
# Convert text sentiment labels (negative\positive) to binary encodings (0/1)
```

```
test_y = le.transform(test_sentiments)
```

3.3.4 Створення архітектуру моделі LSTM

Проблема рекурентних нейронних мереж полягає в тому, що вони просто зберігають попередні дані у своїй «короткочасній пам’яті». Як тільки пам'ять в ньому закінчується, інформація, яка зберігалася найдовше, просто видаляється і замінюється новими даними. Модель LSTM намагається уникнути цієї проблеми, зберігаючи лише вибрану інформацію в короткочасній пам’яті. Ця короткочасна пам'ять зберігається в так званому стані клітини. Побудуємо нашу нейромережу:

```
from keras.models import Sequential
from keras.layers import Dense, Embedding, Dropout, SpatialDropout1D
from keras.layers import LSTM

EMBEDDING_DIM = 128 # dimension for dense embeddings for each token
LSTM_DIM = 64 # total LSTM units

model = Sequential()
model.add(Embedding(input_dim=vocab_size, output_dim=EMBEDDING_DIM,
input_length=max_len))
model.add(SpatialDropout1D(0.2))
model.add(LSTM(LSTM_DIM, dropout=0.2, recurrent_dropout=0.2))
model.add(Dense(1, activation="sigmoid"))

model.compile(loss="binary_crossentropy", optimizer="adam",
              metrics=["accuracy"])
```

3.3.5 Візуалізація архітектури моделі

```
from IPython.display import SVG
from keras.utils.vis_utils import model_to_dot

SVG(model_to_dot(model, show_shapes=True, show_layer_names=False,
                 rankdir='LR').create(prog='dot', format='svg'))
```

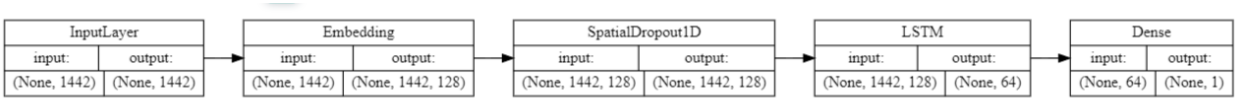

Рис 4. Візуалізація архітектури моделі

3.3.6 Прогнозування та оцінювання продуктивність моделі

Результати показують, що ми отримали точність моделі та міру F1 88%, що є хорошим результатом! Маючи більше якісних даних, можно очікувати ще кращих результатів.

```
pred_test = model.predict_classes(test_X)
predictions = le.inverse_transform(pred_test.flatten())
15000/15000 [==============================] - 352s
meu.display_model_performance_metrics(true_labels=test_sentiments,
predicted_labels=predictions,
                                      classes=['positive', 'negative'])
Model Performance metrics:
------------------------------
Accuracy: 0.88
Precision: 0.88
Recall: 0.88
F1 Score: 0.88

Model Classification report:
------------------------------
             precision    recall  f1-score   support

   positive       0.87      0.88      0.88      7510
   negative       0.88      0.87      0.88      7490

avg / total       0.88      0.88      0.88     15000


Prediction Confusion Matrix:
------------------------------
                 Predicted:
                   positive negative
Actual: positive       6633      877
        negative        972     6518
```

ВИСНОВКИ

В результаті роботи над проектом було реалізовано дві нейромережі за допомогою різних допоміжних пакетів Python. В ході роботи, ми зосереджуємося на спробах проаналізувати велику кількість рецензій на фільми та визначити настрої за допомогою новітніх керованих моделей глибокого навчання та розширених керованих моделей глибокого навчання. Також ми ознайомились з різними методами обробки та нормалізації даних.

СПИСОК ДЖЕРЕЛ

1. https://www.kaggle.com/datasets/nltkdata/movie-review
2. Dipanjan Sarkar, Raghav Bali, Tushar Sharma “Practical Machine Learning with Python: A Problem-Solver’s Guide to Building Real-World Intelligent Systems”, Apress Berkeley, CA, 2018
3. François Chollet “Deep Learning with Python, Second Edition”, 2021