BỘ GIÁO DỤC VÀ ĐÀO TẠO
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN
ĐẠI HỌC QUỐC GIA HỒ CHÍ MINH
-----o0o-----

BÁO CÁO KẾT THÚC HỌC PHẦN.

TÊN ĐỀ TÀI: DETECTING SENTIMENT IN TEXT.

NHÓM 20
21110294 – Trần Công Hiếu
21110287 – Lê Bá Hải

Thành phố Hồ Chí Minh, tháng 12 năm 2023

BỘ GIÁO DỤC VÀ ĐÀO TẠO
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN
ĐẠI HỌC QUỐC GIA HỒ CHÍ MINH

-----o0o-----

TÊN ĐỀ TÀI: DETECTING SENTIMENT IN TEXT.

HỌC PHẦN: PYTHON CHO KHOA HỌC DỮ LIỆU.
GIẢNG VIÊN: HÀ VĂN THẢO.

NHÓM 20
21110294 – Trần Công Hiếu
21110287 – Lê Bá Hải

Thành phố Hồ Chí Minh, tháng 12 năm 2024

Lời cảm ơn

Nhóm chúng em xin chân thành gửi lời cảm ơn từ tận đáy lòng và vô cùng sâu sắc của nhóm em đến giảng viên Hà Văn Thảo đã đồng hành với nhóm em, trong một khoảng thời gian chập chững, choáng ngợp trước những cách tiếp cận kiến thức mới lạ của kì học đầu tiên tại chuyên ngành, thầy đã xóa tan những khó khăn, cực nhọc bởi lượng kiến thức ấy thông qua những buổi học bổ ích, kiến thức sâu rộng không chỉ từ giáo trình mà đến cả những dự án thực tiễn, gần gũi nhất để mọi sinh viên theo học đều nắm bắt được vấn đề. Kèm theo đó là những kinh nghiệm quý báu về công việc sau này đến từ những trải nghiệm từ chính bản thân thầy. Và những phản hồi tích cực cho những thắc mắc của nhóm em về bài báo cáo. Luôn tạo điều kiện thuận lợi và tốt nhất cho lịch học từ trực tiếp lẫn trực tuyến, từ lớp lý thuyết lẫn thực hành để chúng em không bị bỏ lỡ bất kì bài học nào. Tất cả những điều đó là hành trang vững chắc và là nguồn cảm hứng để mà nhóm có thể hoàn thành một cách trọn vẹn nhất cho bài báo cáo này.

Đây là bài báo cáo dự án đầu tiên nên có thể vì vậy mà nhóm chúng em không tránh khỏi được những thiếu sót trong bài, rất mong được lắng nghe những phản hồi, đóng góp, sửa lỗi một cách thật lòng từ thầy để chúng em có được những kinh nghiệm rút ra cho chính bản thân nói riêng và những bài báo cáo khác nói chung sau này.

Sau cùng, hi vọng vào một thời điểm nào đó, có thể là môn học khác hoặc khóa luận tốt nghiệp, nhóm em có thể lại được gặp thầy tiếp tục dẫn dắt. Đó chắc chắn là niềm vinh dự lớn lao mà nhóm em có được trong quãng thời gian theo học tại môi trường đại học.

Và một lần nữa, nhóm em xin cảm ơn thầy vì tất cả những gì thầy đã truyền đạt cho chính nhóm em, các sinh viên khác và cũng như những thế hệ trẻ sau này.

MỤC LỤC

I. GIỚI THIỆU CHUNG.

Phát hiện cảm xúc trong văn bản (Detecting Sentiment in Text) là một vấn đề quan trọng trong xử lý ngôn ngữ tự nhiên (NLP). Phân tích cảm xúc trong văn bản có thể được sử dụng trong nhiều ứng dụng thực tế, chẳng hạn như Phân loại phản hồi của khách hàng, Theo dõi xu hướng xã hội, Phát hiện tin tức giả, Hỗ trợ khách hàng,…

Phân tích cảm xúc trong văn bản là một nhiệm vụ khó khăn vì có nhiều cách khác nhau để thể hiện cảm xúc trong văn bản. Ví dụ, một người có thể thể hiện cảm xúc tích cực bằng cách sử dụng các từ như "vui vẻ", "hạnh phúc", hoặc "mừng". Họ cũng có thể thể hiện cảm xúc tích cực bằng cách sử dụng các hình thức biểu cảm, chẳng hạn như ":)", ":D", hoặc ":^)".

Tính thực tiễn của đề tài: Phân tích cảm xúc trong văn bản có thể được sử dụng trong nhiều ứng dụng thực tế, chẳng hạn như phân loại phản hồi của khách hàng, theo dõi xu hướng xã hội, phát hiện tin tức giả, và hỗ trợ khách hàng.

Tính thách thức của đề tài: Phân tích cảm xúc trong văn bản là một nhiệm vụ khó khăn vì có nhiều cách khác nhau để thể hiện cảm xúc trong văn bản. Điều này khiến cho việc phát triển các phương pháp phân tích cảm xúc hiệu quả trở thành một thách thức thú vị.

Tiềm năng ứng dụng của đề tài: Đề tài này có tiềm năng ứng dụng thực tế lớn. Mong muốn được nghiên cứu đề tài này để góp phần phát triển các phương pháp phân tích cảm xúc hiệu quả hơn, phục vụ cho các ứng dụng thực tế.

Hơn nữa, trong bối cảnh thị trường cạnh tranh khốc liệt trong nước và ngoài nước. Mà trong đó khoa học, công nghệ phát triển mạnh mẽ, chuyển đổi số nhanh chóng, ngày càng trở thành nhân tố quyết định đến năng lực cạnh tranh của doanh nghiệp, dần dần trở thành điều kiện tiên quyết, là thách thức cho cả doanh nghiệp đang hoạt động và doanh nghiệp mới thành lập. Do đó, chủ đề “Phát hiện cảm xúc trong văn bản” là một đề tài rộng mở, đáng để quan tâm, phát triển.

II. ĐOẠN MÃ CHƯƠNG TRÌNH.

```
[1]: import matplotlib.pyplot as plt
     import os #operating system
     import re
     import shutil
     import string
     import tensorflow as tf

     from tensorflow.keras import layers
     from tensorflow.keras import losses
```

```
[2]: print(tf.__version__) #check tensorflow version
```

```
2.15.0
```

Đầu tiên là phần import các thư viện, module sau. Như là:

Module pyplot trong thư viện matplotlib.

“matplotlib.pyplot”: Thư viện Matplotlib là một trong những thư viện phổ biến nhất cho việc vẽ đồ thị và biểu đồ trong ngôn ngữ lập trình Python. Nó là một thư viện vẽ đồ thị rất mạnh mẽ hữu ích cho những người làm việc với Python và NumPy. Module được sử dụng nhiều nhất của Matplotib là Pyplot cung cấp giao diện như MATLAB nhưng thay vào đó, nó sử dụng Python và nó là nguồn mở. Và gọi tắt là “plt”, thuận tiện cho việc gọi các hàm, các lớp trong chương trình.

Và để import sử dụng trên các phần mềm khác thì trước tiên ta phải cài đặt thư viện này bằng câu lệnh pip install matplotlib hay trên Anaconda thì là “conda install matplotlib”.

“os”: là một module tiêu chuẩn trong Python dùng để tương tác với hệ điều hành. Module này cung cấp các hàm và phương thức để làm việc với các tác vụ liên quan đến hệ thống tệp và thư mục, quản lý môi trường, và thực hiện các thao tác hệ thống cơ bản.

- os.getcwd(): (cwd = current working directory) trả về thư mục làm việc hiện tại.
- os.listdir(): liệt kê tất cả các tệp, thư mục của đường dẫn truyền vào.

“re”: hay biểu thức chính quy (Regular Expression) là một chuỗi ký tự tạo thành một biểu mẫu tìm kiếm (search pattern). RegEx được sử dụng để kiểm tra xem một chuỗi có chưa mẫu tìm kiêms được chỉ định hay không.

“shutil”: là một thư viện tiêu chuẩn dùng để thực hiện các thao tác liên quan đến tệp và thư mục (file and directory operations). shutil cung cấp các hàm

để thực hiện các tác vụ như sao chép tệp, di chuyển tệp, xóa tệp và thư mục, và nhiều thao tác khác liên quan đến quản lý tệp và thư mục.

“tensorflow.keras” ta import sử dụng các lớp (layer) để tạo các mô hình mạng rơ-ron. Còn losses là lost-funtion.

1. Download and explore the IMDB dataset.

```
[3]: url = "https://ai.stanford.edu/~amaas/data/sentiment/aclImdb_v1.tar.gz"

     dataset = tf.keras.utils.get_file("aclImdb_v1", url,
                                       untar=True, cache_dir='.',
                                       cache_subdir='')

     dataset_dir = os.path.join(os.path.dirname(dataset), 'aclImdb')
```

```
Downloading data from
https://ai.stanford.edu/~amaas/data/sentiment/aclImdb_v1.tar.gz
84125825/84125825 [==============================] - 3s 0us/step
```

“ url = “…” ”: Gán đường dẫn đến nơi lấy dữ liệu.

“dataset = tf.keras.utils.get_file()”: Đây là một hàm từ TensorFlow để tải một tập dữ liệu từ một URL cụ thể và trả về đường dẫn đến tập dữ liệu đã tải. Với các tham số sau:

- "aclImdb_v1": Tên của tập dữ liệu sẽ được lưu trữ tại địa điểm cục bộ.
- “url”: URL của tập dữ liệu, ta đã gán ở dòng code trên.
- “untar = True”: Sau khi tải xong thì tập dữ liệu sẽ được nén hay không, với giá trị True nên tập dữ liệu sẽ được giải nén
- “cache_dir=’.’” : Đây là thư mục lưu trữ cache mặc định cho việc tải dữ liệu. Trong trường hợp này, tập dữ liệu sẽ được tải và giải nén trong thư mục hiện tại, được biểu thị bằng dấu chấm (".") - nghĩa là thư mục làm việc hiện tại của bạn.
- “cache_subdir=' ’”: Đây là một thư mục con tùy chọn trong thư mục cache. Trong trường hợp này, không có thư mục con cụ thể được chỉ định.

“os.path.dirname(dataset)”: Lấy đường dẫn của thư mục chứa tập dữ liệu đã tải và lưu vào biến dataset có được từ trước đó.

“os.path.join()”: Kết hợp đường dẫn của thư mục với tên thư mục 'aclImdb', tạo ra đường dẫn tới thư mục 'aclImdb' trong cùng thư mục với dataset.

Ví dụ: Nếu dataset là '/home/user/data', thì đoạn code sẽ tạo ra đường dẫn '/home/user/aclImdb' (nếu thư mục 'aclImdb' nằm trong thư mục '/home/user').

Ta có thể truy cập vào mục Files, và mở thư mục aclImdb ở khung bên trái. Hoặc ta cũng có thể xem các thư mục con trong nó bằng câu lệnh:

"os.listdir(dataset_dir)": Kết quả trả về các thư mục chứa trong thư mục từ đường dẫn dataset_dir.

```
[5]: import glob
     for i in glob.glob("aclImdb/*"):
       print(i.split("/")[-1])

     imdbEr.txt
     README
     train
     imdb.vocab
     test
```

Tương tự như os.listdir(), ta cũng có thể xem các mục đó thông qua thư viện glob. Ta sẽ import glob và chạy vòng lặp for.

"glob.glob("aclImdb/*")": Sẽ trả về một danh sách các đường dẫn tới tất cả các tệp tin và thư mục trong thư mục 'aclImdb'.

Vòng lặp for sau đó lặp qua từng đường dẫn trong danh sách này. i.split("/")[-1] sẽ tách đường dẫn thành các phần bằng dấu / và lấy phần tử cuối cùng trong danh sách sau khi tách (sử dụng [-1]). Điều này giúp lấy tên của tệp tin hoặc thư mục cuối cùng trong đường dẫn và in ra tên của từng tệp tin hoặc thư mục cuối cùng trong thư mục 'aclImdb'.

```
[6]: train_dir = os.path.join(dataset_dir, 'train')
     os.listdir(train_dir)

[6]: ['unsupBow.feat',
      'urls_pos.txt',
      'pos',
      'urls_unsup.txt',
      'unsup',
      'neg',
      'urls_neg.txt',
      'labeledBow.feat']
```

Tương tự như thao tác trên, ta có thể xem các thư mục chứa trong thư mục train bằng việc kết nối đường dẫn và sử dụng hàm listdir từ os để hiển thị.

“train_dir = os.path.join(dataset_dir, ‘train’)”: Kết hợp đường dẫn của thư mục với tên thư mục 'train', tạo ra đường dẫn tới thư mục 'train' trong cùng thư mục với dataset_dir.

```
[8]: sample_file = os.path.join(train_dir, 'pos/1181_9.txt')
     with open(sample_file) as f:
       print(f.read())
```

Bằng cách này ta có thể truy xuất đến bất kì thư mục nào và đọc thông tin bên trong. Kết quả thu được phản hồi từ khách hàng trong tập dữ liệu như sau:

```
Rachel Griffiths writes and directs this award winning short film. A
heartwarming story about coping with grief and cherishing the memory of those
we've loved and lost. Although, only 15 minutes long, Griffiths manages to
capture so much emotion and truth onto film in the short space of time. Bud
Tingwell gives a touching performance as Will, a widower struggling to cope with
his wife's death. Will is confronted by the harsh reality of loneliness and
helplessness as he proceeds to take care of Ruth's pet cow, Tulip. The film
displays the grief and responsibility one feels for those they have loved and
lost. Good cinematography, great direction, and superbly acted. It will bring
tears to all those who have lost a loved one, and survived.
```

2. Load the dataset.

Tiếp theo, ta sẽ tải dữ liệu ra khỏi đĩa và chuẩn bị nó thành đúng form phù hợp cho việc training. Để làm thế, ta sẽ sử dụng sự trợ giúp từ tiện ích text_dataset_from_directory, cái mà hữu ích với cấu trúc thư mục như sau:

```
main_directory/
...class_a/
......a_text_1.txt
......a_text_2.txt
...class_b/
......b_text_1.txt
......b_text_2.txt
```

Để chuẩn bị tập dữ liệu để phân loại nhị phân, ta sẽ cần hai thư mục trên đĩa, tương ứng với class_a và class_b. Đây sẽ là những đánh giá phim tích cực và tiêu cực, có thể tìm thấy trong aclImdb/train/pos và aclImdb/train/neg. Vì tập dữ liệu IMDB chứa các thư mục bổ sung nên ta sẽ xóa chúng trước khi sử dụng tiện ích này.

```
[9]: remove_dir = os.path.join(train_dir, 'unsup')
     shutil.rmtree(remove_dir)
```

Tiếp theo, ta sẽ sử dụng tiện ích text_dataset_from_directory để tạo một tf.data.Dataset có nhãn. “tf.data” là một tập hợp các công cụ mạnh mẽ để làm việc với dữ liệu.

Khi chạy thử nghiệm máy học, cách tốt nhất là chia tập dữ liệu thành ba phân chia: đào tạo, xác nhận và kiểm tra.

Bộ dữ liệu IMDB đã được chia thành đào tạo và kiểm tra, nhưng nó thiếu bộ xác thực. Hãy tạo bộ xác thực bằng cách sử dụng phân chia dữ liệu huấn luyện theo tỷ lệ 80:20 bằng cách sử dụng validation_split lập luận dưới đây.

```
[10]: batch_size = 32 # Amount of sample data in one training session
      seed = 42 #

      raw_train_ds = tf.keras.utils.text_dataset_from_directory(
          'aclImdb/train', # Path to the directory containing the training data
          batch_size=batch_size,
          validation_split=0.2, # The proportion of data that will be used for
      ↪validation during training
          subset='training',
          seed=seed)

      Found 25000 files belonging to 2 classes.
      Using 20000 files for training.

[11]: len(raw_train_ds.take(1))

[11]: 1
```

“batch_size = 32”: Lượng dữ liệu mẫu trong một phiên đào tạo. Dòng này xác định kích thước, số điểm dữ liệu được mô hình xử lý trong mỗi lần lặp đào tạo. Kích thước là 32 có nghĩa là mô hình sẽ học từ 32 mẫu huấn luyện trước khi cập nhật trọng số của nó.

“Seed = 42”: Dòng này xác định một hạt giống ngẫu nhiên. Việc đặt hạt giống đảm bảo rằng thứ tự dữ liệu giống nhau được sử dụng cho quá trình huấn luyện mỗi khi mã được chạy. Điều này mang lại sự nhất quán cho việc gỡ lỗi và so sánh các thử nghiệm khác nhau.

“raw_train_ds = tf.keras.utils.text_dataset_from_directory(…)”: Dòng này tạo đối tượng tập dữ liệu cho dữ liệu huấn luyện. Với các đối số sau:

- “'aclImdb/train'”: Phần này chỉ định đường dẫn đến thư mục chứa dữ liệu huấn luyện. Trong trường hợp này, nó giả định thư mục có tên là "aclImdb/train" và chứa các thư mục con dành cho các đánh giá tích cực và tiêu cực.
- "batch_size=batch_size”: Điều này chuyển batch_size được xác định trước đó cho hàm tạo tập dữ liệu.

- “validation_split=0.2”: Điều này dành 20% dữ liệu đào tạo để xác thực, được sử dụng để đánh giá hiệu suất của mô hình trong quá trình đào tạo như đã nói trước đó.
- “subset='training'”: Điều này yêu cầu hàm chỉ tải dữ liệu huấn luyện chứ không tải dữ liệu xác thực.
- seed=seed: Điều này chuyển hạt giống ngẫu nhiên để đảm bảo thứ tự dữ liệu có thể dự đoán được trong tập huấn luyện.

Như ta có thể thấy ở trên, có 25.000 ví dụ trong thư mục đào tạo, trong đó ta sẽ sử dụng 80% (hoặc 20.000) để đào tạo. Như ta sẽ thấy sau đây, ta có thể huấn luyện một mô hình bằng cách chuyển trực tiếp tập dữ liệu đến model.fit. Nếu mới sử dụng tf.data, ta cũng có thể lặp lại tập dữ liệu và in ra một vài ví dụ như sau.

[12]:
```
for text_batch, label_batch in raw_train_ds.take(1):
  for i in range(1):
    print(str(text_batch.numpy()[i]))
    # print("Label", label_batch.numpy()[i])
```

```
b'"Pandemonium" is a horror movie spoof that comes off more stupid than funny.
Believe me when I tell you, I love comedies. Especially comedy spoofs.
"Airplane", "The Naked Gun" trilogy, "Blazing Saddles", "High Anxiety", and
"Spaceballs" are some of my favorite comedies that spoof a particular genre.
"Pandemonium" is not up there with those films. Most of the scenes in this movie
had me sitting there in stunned silence because the movie wasn\'t all that
funny. There are a few laughs in the film, but when you watch a comedy, you
expect to laugh a lot more than a few times and that\'s all this film has going
for it. Geez, "Scream" had more laughs than this film and that was more of a
horror film. How bizarre is that?<br /><br />*1/2 (out of four)'
```

Lưu ý rằng các bài đánh giá chứa văn bản thô (có dấu câu và các thẻ HTML không thường xuyên như <br/>). Ta sẽ chỉ ra cách xử lý những điều này trong phần sau. Các nhãn là 0 hoặc 1. Để xem nhãn nào trong số này tương ứng với các đánh giá phim tích cực và tiêu cực, ta có thể kiểm tra thuộc tính class_names trên tập dữ liệu.

[13]:
```
print("Label 0 corresponds to", raw_train_ds.class_names[0])
print("Label 1 corresponds to", raw_train_ds.class_names[1])
```

```
Label 0 corresponds to neg
Label 1 corresponds to pos
```

Tiếp theo, ta sẽ tạo một tập dữ liệu xác thực và kiểm tra. Bạn sẽ sử dụng 5.000 đánh giá còn lại từ tập huấn luyện để xác nhận. Lưu ý: khi sử dụng các đối số validation_split và tập hợp con, hãy đảm bảo chỉ định một hạt giống ngẫu nhiên hoặc chuyển shuffle=False, để phần phân chia xác thực và đào tạo không bị chồng chéo.

[14]:
```
raw_val_ds = tf.keras.utils.text_dataset_from_directory(
    'aclImdb/train',
    batch_size=batch_size,
    validation_split=0.2,
    subset='validation',
    seed=seed)
```

```
Found 25000 files belonging to 2 classes.
Using 5000 files for validation.
```

[15]:
```
raw_test_ds = tf.keras.utils.text_dataset_from_directory(
    'aclImdb/test',
    batch_size=batch_size)
```

```
Found 25000 files belonging to 2 classes.
```

3. Prepare the data set for training.

Ở mục này, ta sẽ chuẩn hóa, mã hóa và vector hóa dữ liệu bằng cách sử dụng công cụ hữu ích Lớp tf.keras.layers.TextVectorization.

Tiêu chuẩn hóa đề cập đến việc xử lý trước văn bản, thường là loại bỏ các dấu câu hoặc các phần tử HTML để đơn giản hóa tập dữ liệu. Mã thông báo đề cập đến việc chia chuỗi thành các mã thông báo (ví dụ: tách một câu thành các từ riêng lẻ, bằng cách tách bằng khoảng trắng). Vector hóa đề cập đến việc chuyển đổi mã thông báo thành số để chúng có thể được đưa vào mạng lưới thần kinh. Tất cả những nhiệm vụ này có thể được thực hiện với lớp này.

Như đã thấy ở trên, các bài đánh giá chứa nhiều thẻ HTML khác nhau như <br />. Những thẻ này sẽ không bị xóa bởi bộ chuẩn hóa mặc định trong lớp TextVectorization (chuyển đổi văn bản thành chữ thường và loại bỏ dấu câu theo mặc định, nhưng không loại bỏ HTML). Do đó, ta sẽ viết một tùy chỉnh chức năng tiêu chuẩn hóa để loại bỏ HTML.

[16]:
```
def custom_standardization(input_data):
  lowercase = tf.strings.lower(input_data)
  stripped_html = tf.strings.regex_replace(lowercase, '<br />', ' ')
  return tf.strings.regex_replace(stripped_html,
                                  '[%s]' % re.escape(string.punctuation),
                                  '')
```

Định nghĩa hàm custom_standardization với lần lượt các câu lệnh sau:

“lowercase = tf.string.lower(input_data)”: Sử dụng hàm chuyển đổi tất cả ký tự trong input_data thành chữ viết thường và gán cho lowercase.

“stripped_html = tf.string.regex_replace()” thay thế mẫu “ ’
’ ” trong chuỗi lowercase trên bằng một khoảng trắng.

“tf.strings.regex_replace()”: Đây là một hàm thuộc thư viện TensorFlow, dùng để thay thế các mẫu ký tự trong một chuỗi bằng một chuỗi khác. Tham số đầu tiên của hàm này là chuỗi cần thay thế (stripped_html).

“ '%s' % re.escape(string.punctuation) ”: Phần này tạo ra một mẫu regex để xác định các ký tự cần thay thế.

“re.escape(string.punctuation)”: Dùng để thoát các ký tự metacharacter trong string.punctuation (ví dụ như dấu "."). Điều này đảm bảo chúng được tìm kiếm theo nghĩa đen chứ không phải theo chức năng regex.

“[%s]”: Đặt các ký tự đã thoát vào ngoặc vuông để tạo thành một tập hợp ký tự (character class). Điều này cho phép thay thế bất kỳ ký tự nào trong tập hợp này.

“ " ”: Chuỗi rỗng được dùng để thay thế các ký tự đã khớp với mẫu regex.

Do đó, kết quả của hàm trả về sẽ là chuỗi stripped_html nhưng loại bỏ tất cả các dấu câu.

Tiếp theo, bạn sẽ tạo một lớp TextVectorization. Bạn sẽ sử dụng lớp này để chuẩn hóa, mã hóa và vector hóa dữ liệu của chúng tôi. Bạn đặt out_mode thành int để tạo các chỉ số nguyên duy nhất cho mỗi mã thông báo.

Lưu ý rằng bạn đang sử dụng hàm phân tách mặc định và hàm tiêu chuẩn hóa tùy chỉnh mà bạn đã xác định ở trên. Bạn cũng sẽ xác định một số hằng số cho mô hình, chẳng hạn như Sequ_length tối đa rõ ràng, điều này sẽ khiến lớp đệm hoặc cắt bớt các chuỗi thành các giá trị Sequ_length chính xác.

[17]:
```
max_features = 10000
sequence_length = 250
```

```
vectorize_layer = layers.TextVectorization(
    standardize=custom_standardization,
    max_tokens=max_features,
    output_mode='int',
    output_sequence_length=sequence_length)
```

“max_features = 10000”: Biến max_features xác định số lượng từ vựng tối đa mà lớp vector hóa văn bản sẽ quản lý. Chỉ những từ xuất hiện thường xuyên nhất trong dữ liệu sẽ được giữ lại và được đánh số từ 1 đến max_features. Những từ nằm ngoài danh sách này sẽ được đánh một chỉ mục đặc biệt cho từ không xác định (UNK).

“sequence_length = 250”: Biến sequence_length xác định độ dài tối đa của mỗi chuỗi văn bản sau khi vector hóa. Nếu chuỗi dài hơn sequence_length, nó sẽ bị cắt bớt hoặc nếu ngắn hơn, thì sẽ được điền đầy (padding) để có độ dài là sequence_length.

Tiếp đó, ta sử dụng lớp TextVectorization – một lớp thuộc Tensorflow.Keras dùng để chuyển đổi văn bản thành vector số. Trong đó có các tham số sau:

- “standardize = custom_standardization”: Sử dụng hàm custom_standardization đã định nghĩa trên để chuẩn hóa văn bản trước khi mã hóa.
- “max_tokens = max_features”: Giới hạn số lượng từ vựng như đã nói ở trên.
- “ output_mode = 'int' ”: Xác định kiểu dữ liệu của vectơ đầu ra là số nguyên.
- “output_sequence_length=sequence_length”: Cắt bớt vectơ đầu ra sao cho tối đa chỉ còn 250 từ.

Kết quả trả về của hàm được lưu vào biến vectorize_layer.

```
[18]: # Make a text-only dataset (without labels), then call adapt
      train_text = raw_train_ds.map(lambda x, y: x)
      vectorize_layer.adapt(train_text)
```

“raw_train_ds” là tập dữ liệu (dataset) được sử dụng trong tensorflow, khi ta tạo ra raw_train_ds ở đoạn mã trên thì nó là object thuộc lớp Dataset. Trong lớp Dataset có phương thức map() có chức năng áp dụng 1 hàm lên mỗi phần tử trong bộ dữ liệu. Sử dụng phương thức map và đối số đầu tiên là một hàm lambda (lambda-function) nhận cặp giá trị (x,y) – cặp (văn bản, nhãn) và chỉ quan tâm đến x. Lưu kết quả vào tập dữ liệu mới train_text chỉ chứa các văn bản.

“vectorize_layer.adapt(train_text)”: Gọi phương thức adapt() của lớp vectorize_layer. Cung cấp tập dữ liệu train_text để lớp vector hóa học tập từ ngữ liệu. Quá trình này giúp lớp vector hóa xây dựng từ điển từ vựng và xác định cách biểu diễn các từ dưới dạng các vector số.

Hãy tạo một hàm để xem kết quả của việc sử dụng lớp này để xử lý trước một số dữ liệu.

```
[19]: def vectorize_text(text, label):
        text = tf.expand_dims(text, -1)
        return vectorize_layer(text), label
```

Định nghĩa hàm vectorize_text với hai tham số đầu vào.

"tf.expand_dims(text,-1)": Dòng này mở rộng một chiều mới có kích thước 1 vào cuối của tensor chứa văn bản (text). Điều này cần thiết để tương thích với lớp vector hóa phía sau (vectorize_layer). Tham số -1 mang ý nghĩa là mở rộng vào cuối tensor, còn 0 là mở rộng phần đầu.

"vectorize_layer(text)": Dòng này gọi đến một hàm tên vectorize_layer đã định nghĩa trước đó với đầu vào là văn bản đã được thêm chiều mới. Hàm có nhiệm vụ biến đổi văn bản thành một biểu diễn vector. Thực tế hoạt động của lớp này phụ thuộc vào cấu trúc và cách cài đặt của nó.

Trả về kết quả: Hàm trả về hai giá trị, giá trị đầu tiên là biểu diễn vector của văn bản được tạo bởi vectorize_layer và label một biến chứa nhãn của văn bản. Hàm trả về nhãn cùng với vector để thuận tiện cho các tác vụ xử lý dữ liệu sau này.

```
[20]: # retrieve a batch (of 32 reviews and labels) from the dataset
      text_batch, label_batch = next(iter(raw_train_ds))
      first_review, first_label = text_batch[0], label_batch[0]
      print("Review", first_review)
      print("Label", raw_train_ds.class_names[first_label])
      print("Vectorized review", vectorize_text(first_review, first_label))
```

```
Review tf.Tensor(b'Great movie - especially the music - Etta James - "At Last".
This speaks volumes when you have finally found that special someone.',
shape=(), dtype=string)
Label neg
Vectorized review (<tf.Tensor: shape=(1, 250), dtype=int64, numpy=
array([[  86,   17,  260,    2,  222,    1,  571,   31,  229,   11, 2418,
           1,   51,   22,   25,  404,  251,   12,  306,  282,    0,    0,
           0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
           0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
           0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
           0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
           0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
           0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
           0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
           0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
           0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
           0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
           0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
```

```
        0,      0,      0,      0,      0,      0,      0,      0,      0,      0,      0,
        0,      0,      0,      0,      0,      0,      0,      0,      0,      0,      0,
        0,      0,      0,      0,      0,      0,      0,      0,      0,      0,      0,
        0,      0,      0,      0,      0,      0,      0,      0,      0,      0,      0,
        0,      0,      0,      0,      0,      0,      0,      0,      0,      0,      0,
        0,      0,      0,      0,      0,      0,      0,      0,      0,      0,      0,
        0,      0,      0,      0,      0,      0,      0,      0,      0,      0,      0,
        0,      0,      0,      0,      0,      0,      0,      0,      0,      0,      0,
        0,      0,      0,      0,      0,      0,      0,      0,      0,      0,      0,
        0,      0,      0,      0,      0,      0,      0,      0]])>, <tf.Tensor:
shape=(), dtype=int32, numpy=0>)
```

“text_batch, label_batch = next(iter(raw_train_ds))”: Phần này tìm nạp một loạt 32 bài đánh giá và nhãn tương ứng của chúng từ tập dữ liệu có tên raw_train_ds. Cách hoạt động là từ hàm iter(raw_train_ds), chúng trả về 1 đối tượng là iterator có phương thức next(). Khi dùng next(iter(raw_train_ds)) thì đối tượng iterator được tạo đó sẽ lấy phần tử đầu tiên trong raw_train_ds, ở đây do tập chia theo batch nên next() sẽ lấy 1 batch chứa 32 cặp text_batch và label_batch.

text_batch: Danh sách chứa 32 bài đánh giá văn bản.

label_batch: Danh sách chứa các nhãn tương ứng, phân loại cho mỗi đánh giá.

Để xem thử cấu trúc dữ liệu đầu tiên thì ta lần lượt gán text_batch[0] cho first_review và label_batch[0] cho first_label. Sau đó, hiển thị các thông tin ta cần xem:

“print("Review", first_review)”: In nội dung văn bản đánh giá đầu tiên.

“print("Label", raw_train_ds.class_names[first_label])”: In nhãn phân loại tích cực hay tiêu cực của đánh giá đầu tiên. Label first_label chỉ chứa giá trị 0 hoặc 1, do đó để xem được phản hồi tính chất nào thì dùng class_names[] cho label đó để lấy nhãn từ thuộc tính class_names của tập dữ liệu.

“print("Vectorized review", vectorize_text(first_review, first_label))”: Gọi hàm có tên vectorize_text để chuyển đổi phần đánh giá văn bản thành dạng biểu diễn số (vector).

Có thể thấy ở trên, mỗi mã thông báo đã được thay thế bằng một số nguyên. Ta có thể tra cứu mã thông báo (chuỗi) mà mỗi số nguyên tương ứng bằng cách gọi .get_vocabulary() trên lớp.

```
[21]: print("1287 ---> ",vectorize_layer.get_vocabulary()[1287])
      print(" 313 ---> ",vectorize_layer.get_vocabulary()[313])
      print('Vocabulary size: {}'.format(len(vectorize_layer.get_vocabulary())))

      1287 --->  silent
       313 --->  night
      Vocabulary size: 10000
```

“print("1287 ---> ", vectorize_layer.get_vocabulary()[1287])”: Lấy từ điển từ vựng từ lớp vectorize_layer bằng phương thức get_vocabulary(). Truy cập từ có chỉ mục 1287 trong từ điển và in ra.

“print(" 313 ---> ", vectorize_layer.get_vocabulary()[313])”: Tương tự, in ra từ có chỉ mục 313 trong từ điển.

“print('Vocabulary size: {}'.format(len(vectorize_layer.get_vocabulary())))”: Lấy số lượng từ trong từ điển bằng hàm len() và in ra. Cụ thể, phương thức get_vocabulary() của lớp vectorize_layer sẽ trả về một từ điển từ vựng. Từ điển này chứa các từ trong tập dữ liệu đã được sử dụng để thích ứng lớp vector hóa. Tiếp theo, hàm len() sẽ được sử dụng để đếm số lượng từ trong từ điển. Cuối cùng, giá trị này sẽ được định dạng bằng hàm format() và in ra màn hình.

Gần như đã sẵn sàng để huấn luyện mô hình. Ở bước tiền xử lý cuối cùng, ta sẽ áp dụng. Lớp TextVectorization đã tạo trước đó cho tập dữ liệu huấn luyện, xác thực và kiểm tra.

```
[22]: train_ds = raw_train_ds.map(vectorize_text)
      val_ds = raw_val_ds.map(vectorize_text)
      test_ds = raw_test_ds.map(vectorize_text)
```

Lấy các tập dữ liệu thô raw_train_ds, raw_val_ds, và raw_test_ds là các tập dữ liệu ban đầu chứa văn bản thô và nhãn tương ứng.

Áp dụng hàm vector hóa “.map(vectorize_text)” là phương thức được gọi trên mỗi tập dữ liệu để áp dụng hàm vectorize_text cho từng văn bản trong tập dữ liệu đó.

Hàm vectorize_text chịu trách nhiệm chuyển đổi văn bản thô thành các vector số, có thể được sử dụng làm đầu vào cho các mô hình học máy. Lưu kết quả vào các tập dữ liệu mới là train_ds, val_ds, và test_ds là các tập dữ liệu mới được tạo ra sau khi vector hóa, chứa các vector văn bản và nhãn tương ứng.

Việc vector hóa văn bản là bước quan trọng trước khi đưa dữ liệu văn bản vào các mô hình học máy. Các mô hình học máy thường không thể xử lý trực tiếp văn bản thô mà cần các biểu diễn số học của văn bản. Bằng cách vector hóa

văn bản, chúng ta chuyển đổi văn bản thành các vector số mà các mô hình có thể hiểu và học hỏi được.

4. Configure the dataset for performance.

Có hai phương pháp quan trọng cần sử dụng khi tải dữ liệu để đảm bảo rằng việc nhập/xuất (I/O) không trở thành tắc nghẽn.

Phương thức .cache(): Lưu trữ dữ liệu trong bộ nhớ sau khi tải từ đĩa. Đảm bảo rằng tập dữ liệu không trở thành rào cản trong quá trình huấn luyện mô hình. Nếu tập dữ liệu quá lớn, không thể lưu trữ hoàn toàn trong bộ nhớ, phương thức này có thể được sử dụng để tạo bộ đệm hiệu quả trên đĩa, giúp việc đọc dữ liệu hiệu quả hơn so với đọc nhiều tệp nhỏ.

Phương thức .prefetch(): Cho phép xử lý dữ liệu và thực thi mô hình diễn ra đồng thời trong quá trình huấn luyện.

Để tìm hiểu thêm về cả hai phương thức này cũng như cách lưu trữ dữ liệu vào đĩa, vui lòng tham khảo hướng dẫn về hiệu suất dữ liệu (data performance guide).

[23]:
```
AUTOTUNE = tf.data.AUTOTUNE

train_ds = train_ds.cache().prefetch(buffer_size=AUTOTUNE)
val_ds = val_ds.cache().prefetch(buffer_size=AUTOTUNE)
test_ds = test_ds.cache().prefetch(buffer_size=AUTOTUNE)
```

“AUTOTUNE = tf.data.AUTOTUNE”: Gán giá trị tf.data.AUTOTUNE cho biến AUTOTUNE. Giá trị này cho phép TensorFlow tự động điều chỉnh kích thước bộ đệm prefetch dựa trên phần cứng hiện có, giúp tối ưu hóa hiệu suất.

“train_ds = train_ds.cache().prefetch(buffer_size=AUTOTUNE)”: Áp dụng phương thức cache() và prefetch() cho tập dữ liệu train_ds, trong đó:

- cache(): Lưu trữ dữ liệu đã được xử lý vào bộ nhớ đệm để tránh xử lý lại khi được truy cập nhiều lần. Điều này giúp tăng tốc độ đọc dữ liệu, đặc biệt khi có các thao tác xử lý tốn thời gian.
- prefetch(): Chủ động tải trước các phần tử dữ liệu vào bộ nhớ trong khi GPU đang xử lý các phần tử hiện tại. Điều này giúp giảm thời gian chờ đợi giữa các lần lặp và cải thiện hiệu suất huấn luyện.
- buffer_size=AUTOTUNE: Cho phép TensorFlow tự động chọn kích thước bộ đệm prefetch phù hợp.

Các dòng tương tự cho val_ds và test_ds: Áp dụng các phương thức cache() và prefetch() với kích thước bộ đệm tự động cho các tập dữ liệu xác thực và kiểm thử.

5. Create the model.

Sau khi chuẩn bị tiền xử lý dữ liệu xong, thì đây là lúc để tạo Mạng nơ-ron.

```
[24]: embedding_dim = 16
```

Trong trường hợp của embedding_dim = 16, điều này có nghĩa rằng dữ liệu đầu vào hoặc các đối tượng sẽ được biểu diễn bằng các vector có 16 chiều trong không gian nhúng. Kích thước này thường là một tham số có thể điều chỉnh trong quá trình đào tạo mô hình, và nó có thể ảnh hưởng đến khả năng của mô hình trong việc học và hiểu các đặc trưng của dữ liệu

```
[25]: model = tf.keras.Sequential([
        layers.Embedding(max_features + 1, embedding_dim),
        layers.Dropout(0.2),
        layers.GlobalAveragePooling1D(),
        layers.Dropout(0.2),
        layers.Dense(1)])

      model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, None, 16)          160016

 dropout (Dropout)           (None, None, 16)          0

 global_average_pooling1d (  (None, 16)                0
 GlobalAveragePooling1D)

 dropout_1 (Dropout)         (None, 16)                0

 dense (Dense)               (None, 1)                 17

=================================================================
Total params: 160033 (625.13 KB)
Trainable params: 160033 (625.13 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Để gọi tạo model được xây dựng mô hình mạng nơ ron một cách tuần tự các lớp layer thì ta dùng class Sequential trong thư viện tf.keras.

“model = tf.keras.Sequential([...])”: Tạo một model tuần tự trong TensorFlow Keras. Các lớp model được định nghĩa bên trong dấu ngoặc vuông.

Tiếp đó, thêm các layer cho model:

- “layers.Embedding(max_features + 1, embedding_dim)”: Biểu diễn các từ thành các vector số (embedding). Với “max_features” là số lượng từ vựng tối đa trong dữ liệu. “embedding_dim” là kích thước của vector embedding.
- “layers.Dropout(0.2)”: Ngẫu nhiên loại bỏ 20% kết nối trong quá trình huấn luyện để tránh overfitting. Tức tắt một số nút mạng theo đúng nghĩa đen trong quá trình huấn luyện để tránh học tủ (Over-fitting). Nếu 1 lớp fully connected có quá nhiều tham số và chiếm hầu hết tham số, các nút mạng trong lớp đó quá phụ thuộc lẫn nhau trong quá trình huấn luyện thì sẽ hạn chế sức mạnh của mỗi nút, dẫn đến việc kết hợp quá mức.
- “layers.GlobalAveragePooling1D()”: Gộp trung bình các giá trị embedding theo chiều dài câu.
- “layers.Dropout(0.2)”: Áp dụng dropout lần nữa để giảm overfitting.
- “layers.Dense(1)”: Tạo một lớp fully connected với 1 neuron, dùng để dự đoán đầu ra.

“model.summary()”: Hiển thị tóm tắt cấu trúc và số lượng tham số của model.

Bảng này hiển thị các tham số của một mô hình được sử dụng để tạo ra một chuỗi các sự kiện. Các tham số được liệt kê theo thứ tự tăng hoặc giảm, và chúng thường được sử dụng để dự đoán kết quả của các sự kiện. Cột đầu tiên của bảng cho biết tên của lớp mô hình. Trong trường hợp này, mô hình là một mô hình tuần tự. Mô hình tuần tự là một loại mô hình học máy được sử dụng để xử lý dữ liệu theo thứ tự. Các cột tiếp theo của bảng cho biết tên của lớp, hình dạng đầu ra của lớp và số lượng tham số của lớp.

Các lớp được xếp chồng lên nhau một cách tuần tự để xây dựng bộ phân loại:

1. Lớp đầu tiên là lớp Nhúng. Lớp này lấy các đánh giá được mã hóa số nguyên và tra cứu vectơ nhúng cho mỗi chỉ mục từ. Những vectơ này được học khi mô hình xe lửa. Các vectơ thêm một thứ nguyên vào mảng đầu ra. Kích thước kết quả là: (lô, trình tự, nhúng). Để tìm hiểu thêm về nội dung nhúng, hãy xem hướng dẫn về nội dung nhúng trong Word.
2. Tiếp theo, lớp GlobalAveragePooling1D trả về vectơ đầu ra có độ dài cố định cho mỗi ví dụ bằng cách lấy trung bình trên kích thước chuỗi. Điều này cho phép mô hình xử lý đầu vào có độ dài thay đổi theo cách đơn giản nhất có thể.

3. Lớp cuối cùng được kết nối chặt chẽ với một nút đầu ra duy nhất.

6. Loss function and optimizer.

Một mô hình cần có hàm mất mát và trình tối ưu hóa để đào tạo. Vì đây là vấn đề phân loại nhị phân và mô hình đưa ra xác suất (một lớp đơn vị có kích hoạt sigmoid), nên bạn sẽ sử dụng hàm mất mát losses.BinaryCrossentropy.

Bây giờ, hãy định cấu hình mô hình để sử dụng trình tối ưu hóa và hàm mất:

```
[26]: model.compile(loss=losses.BinaryCrossentropy(from_logits=True),
                    optimizer='adam',
                    metrics=tf.metrics.BinaryAccuracy(threshold=0.0))
```

“model.compile(...)”: Chuẩn bị model cho quá trình huấn luyện bằng cách thiết lập các thông số cần thiết.

“loss=losses.BinaryCrossentropy(from_logits=True)” : Đây là phần xác định hàm mất mát (loss function) được sử dụng trong quá trình huấn luyện mô hình. Trong trường hợp này, ta đang sử dụng hàm mất mát BinaryCrossentropy dành cho tác vụ phân loại nhị phân (binary classification). from_logits=True chỉ ra rằng mô hình của ta sẽ xuất ra các giá trị logits (trước khi áp dụng hàm kích hoạt sigmoid), và hàm mất mát sẽ tự động áp dụng hàm kích hoạt sigmoid lên giá trị đầu ra để tính toán mất mát. Hàm mất mát này thường được sử dụng trong các tác vụ phân loại nhị phân.

“ optimizer = ‘adam’ ”: Đây là phần xác định thuật toán tối ưu hóa được sử dụng trong quá trình huấn luyện mô hình. Trong trường hợp này, bạn đang sử dụng thuật toán tối ưu hóa Adam, một trong những thuật toán tối ưu hóa phổ biến cho việc cập nhật trọng số mạng neural trong quá trình học. Ngoài ra còn có sgd, adagrad, adadelta, nadam,…

“metrics=tf.metrics.BinaryAccuracy(threshold=0.0)”: Sử dụng BinaryAccuracy để đo lường độ chính xác của model trong việc phân loại nhị phân với threshold=0.0 thiết lập ngưỡng phân loại là 0.0, nghĩa là dự đoán trên 0.0 sẽ được xem là lớp 1, còn dưới 0.0 sẽ được xem là lớp 0.

7. Train the model.

```
[27]: epochs = 10
history = model.fit(
    train_ds,
    validation_data=val_ds,
    epochs=epochs)
```

"epochs = 10": Xác định số lần toàn bộ dữ liệu huấn luyện sẽ được sử dụng để huấn luyện model. Trong trường hợp này, model sẽ được huấn luyện 10 lần.

“history = model.fit(train_ds, validation_data=val_ds, epochs=epochs)”: Đoạn mã này chạy quá trình huấn luyện mô hình bằng cách sử dùng hàm fit(). Dưới đây là giải thích từng tham số:

- train_ds: Đây là dữ liệu đào tạo (training dataset) được sử dụng để huấn luyện mô hình.
- validation_data=val_ds: Đây là tập dữ liệu kiểm định (validation dataset) được sử dụng để đánh giá hiệu suất của mô hình trong quá trình huấn luyện. Dữ liệu này được sử dụng để kiểm tra mô hình và đánh giá mức độ overfitting.
- epochs=epochs: Đây là số lượng epoch (lần đi qua toàn bộ tập dữ liệu đào tạo) ta đã khởi gán trước đó là 10.

8. Evaluate the model.

Hãy xem mô hình hoạt động như thế nào. Hai giá trị sẽ được trả về. Mất mát (một con số thể hiện lỗi, giá trị càng thấp thì càng tốt) và độ chính xác.

```
[28]: loss, accuracy = model.evaluate(test_ds)

      print("Loss: ", loss)
      print("Accuracy: ", accuracy)
```

```
782/782 [==============================] - 5s 6ms/step - loss: 0.3099 -
binary_accuracy: 0.8736
Loss:  0.30986225605010986
Accuracy:  0.8735600113868713
```

“loss, accuracy = model.evaluate(test_ds)”: Đánh giá model trên tập dữ liệu test. Bằng phương thức evaluate của lớp Sequential. Với đối số là “test_ds”, tập dữ liệu test dùng để đánh giá model sau khi huấn luyện.

Kết quả đánh giá trả về loss hàm mất mát (sai số) của model trên tập test và accuracy độ chính xác của model trên tập test.

9. Create a plot of accuracy and loss over time.

model.fit() trả về một đối tượng History chứa từ điển với mọi thứ đã xảy ra trong quá trình đào tạo:

```
[29]: history_dict = history.history
      history_dict.keys()
```

```
[29]: dict_keys(['loss', 'binary_accuracy', 'val_loss', 'val_binary_accuracy'])
```

Từ history = model.fit(), ta có history là một đối tượng. Sau đó ta gọi thuộc tính của history bằng câu lệnh "history.history", nó chứa toàn bộ thông tin liên quan đến quá trình training. Và gọi ra bằng method .key().

Có bốn mục, một mục cho mỗi số liệu được theo dõi trong quá trình đào tạo và xác nhận. Ta có thể sử dụng những thông tin này để vẽ biểu đồ tổn thất đào tạo và xác thực để so sánh, cũng như độ chính xác trong quá trình đào tạo và xác thực:

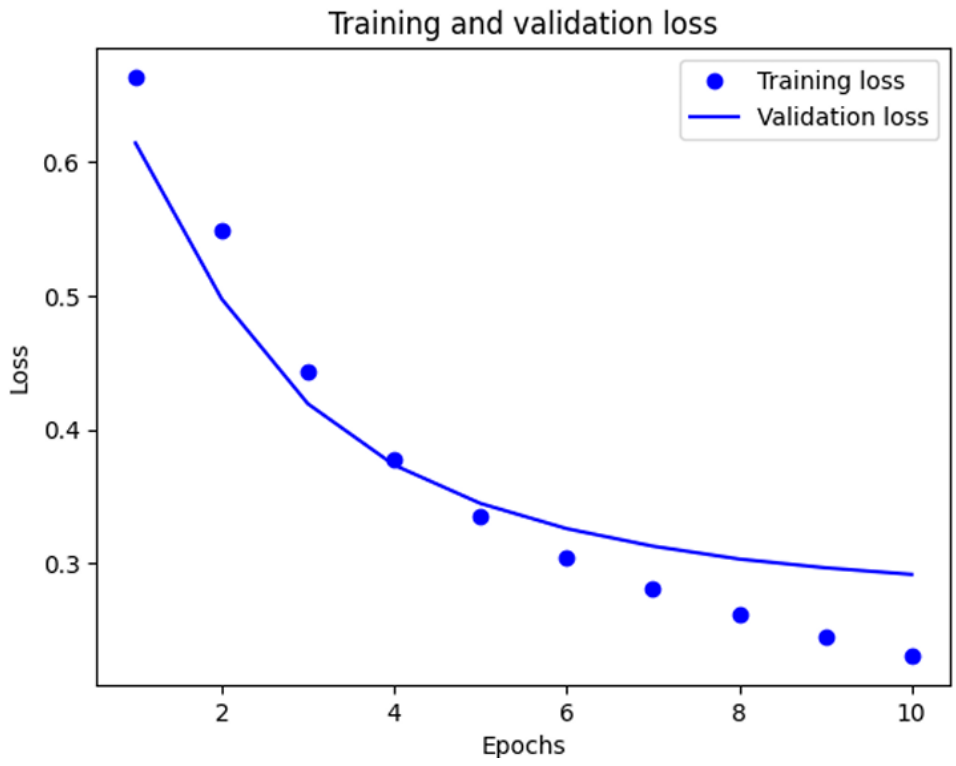
[30]:
```
acc = history_dict['binary_accuracy']
val_acc = history_dict['val_binary_accuracy']
loss = history_dict['loss']
val_loss = history_dict['val_loss']

epochs = range(1, len(acc) + 1)

# "bo" is for "blue dot"
plt.plot(epochs, loss, 'bo', label='Training loss')
# b is for "solid blue line"
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
```

```
plt.legend()

plt.show()
```



"acc = history_dict['binary_accuracy']": Lấy giá trị độ chính xác (accuracy) trên tập huấn luyện trong mỗi epoch.

“val_acc = history_dict['val_binary_accuracy']”: Lấy giá trị độ chính xác trên tập validation trong mỗi epoch.

“loss = history_dict['loss']”: Lấy giá trị hàm mất mát (loss) trên tập huấn luyện trong mỗi epoch.

“val_loss = history_dict['val_loss']”: Lấy giá trị hàm mất mát trên tập validation trong mỗi epoch.

“epochs = range(1, len(acc) + 1)”: Tạo một danh sách chứa các số từ 1 đến số epoch huấn luyện, dùng để làm trục hoành trong biểu đồ.

“plt.plot(epochs, loss, 'bo', label='Training loss')”: Vẽ đồ thị hàm mất mát trên tập huấn luyện bằng các điểm màu xanh dương, plt.plot() hàm được sử dụng để vẽ đồ thị. Các đối số gồm:

- epochs: Danh sách số epoch, dùng làm trục hoành trong biểu đồ.
- loss: Danh sách giá trị hàm mất mát trên tập huấn luyện, dùng làm trục tung trong biểu đồ.
- 'bo': Kiểu ký hiệu của các điểm trong đồ thị. Trong trường hợp này, các điểm sẽ là các chấm màu xanh dương.
- label='Training loss': Chú thích cho đường trong biểu đồ. Trong trường hợp này, đường biểu diễn hàm mất mát trên tập huấn luyện.

“plt.plot(epochs, val_loss, 'b', label='Validation loss')”: Vẽ đồ thị hàm mất mát trên tập validation bằng đường màu xanh dương, plt.plot() hàm được sử dụng để vẽ đồ thị. Các đối số gồm:

- plt.plot(): Hàm được sử dụng để vẽ đồ thị.
- epochs: Danh sách số epoch, dùng làm trục hoành trong biểu đồ.
- val_loss: Danh sách giá trị hàm mất mát trên tập validation, dùng làm trục tung trong biểu đồ.
- 'b': Kiểu ký hiệu của đường trong đồ thị. Trong trường hợp này, đường sẽ là đường màu xanh dương.
- label='Validation loss': Chú thích cho đường trong đồ thị. Trong trường hợp này, đường biểu diễn hàm mất mát trên tập validation.

“plt.title('Training and validation loss')”: Đặt tiêu đề cho biểu đồ là "Training and validation loss".

“plt.xlabel('Epochs')”: Đặt nhãn cho trục hoành là "Epochs".

“plt.ylabel('Loss')”: Đặt nhãn cho trục tung là "Loss".

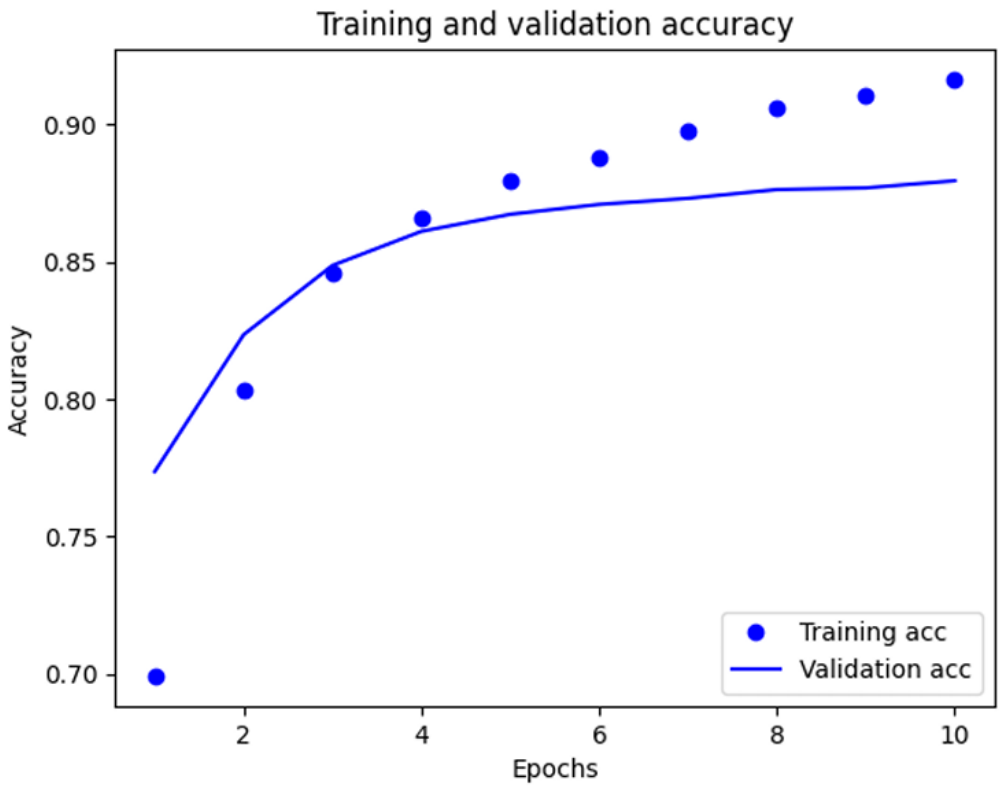
“plt.legend()”: Hiển thị chú thích cho các đường trong biểu đồ.

“plt.show()”: Hiển thị biểu đồ.

Tương tự, ta vẽ biểu đồ thể hiện mối quan hệ giữa độ chính xác trên bộ dữ liệu đào tạo và độ chính xác trên bộ dữ liệu thực trong quá trình phân loại.

[31]:
```
plt.plot(epochs, acc, 'bo', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend(loc='lower right')

plt.show()
```



Trong biểu đồ này, các dấu chấm thể hiện mất mát huấn luyện và độ chính xác huấn luyện, còn các đường liền nét thể hiện mất mát xác thực và độ chính xác xác thực.

Cần lưu ý rằng mất mát huấn luyện giảm dần theo từng epoch (vòng lặp) và độ chính xác huấn luyện tăng dần theo từng epoch. Đây là điều thường thấy khi sử dụng phương pháp tối ưu hóa gradient descent (hạ gradient) - phương pháp này sẽ tối thiểu hóa giá trị mong muốn trong mỗi lần lặp.

Tuy nhiên, điều này không xảy ra với mất mát xác thực và độ chính xác xác thực - chúng dường như đạt đỉnh trước cả độ chính xác huấn luyện. Đây là một ví dụ về hiện tượng overfitting (quá khớp): mô hình hoạt động tốt hơn trên

dữ liệu huấn luyện so với dữ liệu chưa từng thấy trước đây. Sau thời điểm này, mô hình được tối ưu hóa quá mức và học được những đặc trưng riêng của dữ liệu huấn luyện mà không thể khái quát hóa cho dữ liệu kiểm tra.

Trong trường hợp cụ thể này, bạn có thể ngăn chặn hiện tượng overfitting bằng cách đơn giản là dừng quá trình huấn luyện khi độ chính xác xác thực không còn tăng nữa. Một cách để thực hiện điều này là sử dụng callback tf.keras.callbacks.EarlyStopping.

10. Export the model.

Trong đoạn mã trên, ta đã áp dụng lớp TextVectorization cho tập dữ liệu trước khi đưa văn bản vào mô hình. Nếu ta muốn làm cho mô hình của mình có khả năng xử lý các chuỗi thô (ví dụ: để đơn giản hóa việc triển khai nó), ta có thể bao gồm lớp TextVectorization bên trong mô hình của mình. Để làm như vậy, ta có thể tạo một mô hình mới bằng cách sử dụng các trọng số bạn vừa huấn luyện.

[32]:
```
export_model = tf.keras.Sequential([
  vectorize_layer,
  model,
  layers.Activation('sigmoid')
])

export_model.compile(
    loss=losses.BinaryCrossentropy(from_logits=False), optimizer="adam",␣
  ↪metrics=['accuracy']
)

# Test it with `raw_test_ds`, which yields raw strings
loss, accuracy = export_model.evaluate(raw_test_ds)
print(accuracy)
```

```
782/782 [==============================] - 5s 6ms/step - loss: 0.3099 -
accuracy: 0.8736
0.8735600113868713
```

"export_model = tf.keras.Sequential(...)": Tạo một model mới có tên là export_model bằng cách sử dụng lớp Sequential trong TensorFlow Keras. Model này được thiết kế để xuất ra phục vụ cho việc sử dụng trong mục đích như đã nêu trên. Các đối số truyền vào như sau:

- "vectorize_layer": Lớp này có thể là một lớp Embedding hoặc một lớp tiền xử lý khác, dùng để biến đổi dữ liệu đầu vào thành dạng phù hợp cho model.
- "model": Đây là model đã được huấn luyện trước đó.

- “layers.Activation('sigmoid')”: Lớp này áp dụng hàm kích hoạt sigmoid lên đầu ra của model, chuyển đổi chúng thành xác suất nằm trong khoảng từ 0 đến 1.

Đây là bảng phân tích mã:

“import_model.compile(...)”: Điều này biên dịch mô hình, chuẩn bị cho việc đánh giá hoặc suy luận.

“loss=losses.BinaryCrossentropy(from_logits=False)”: Chỉ định hàm mất mát được sử dụng trong quá trình đánh giá. BinaryCrossentropy thích hợp cho các nhiệm vụ phân loại nhị phân. Đối số from_logits=False chỉ ra rằng đầu ra của mô hình đã ở dạng xác suất (do kích hoạt sigmoid), do đó không cần chia tỷ lệ thêm.

“Optimizer="adam"”: Đặt thuật toán tối ưu hóa cho Adam, đây là lựa chọn phổ biến để đào tạo mạng nơ-ron.

“metrics=['accuracy']”: Chỉ định rằng số liệu độ chính xác phải được tính toán trong quá trình đánh giá.

“loss, accuracy =export_model.evaluate(raw_test_ds)”: Đánh giá hiệu suất của mô hình trên tập dữ liệu raw_test_ds. Và trả về 2 giá trị loss (Giá trị tổn thất trung bình trên các mẫu thử nghiệm) và accuracy (Độ chính xác của mô hình trên tập kiểm tra).

“print(accuracy)”: In giá trị độ chính xác được tính toán ra bàn điều khiển.

11. Inference on new data.

Để nhận dự đoán cho các ví dụ mới, ta chỉ cần gọi model.predict().

```
[33]: examples = [
        "The movie was great!",
        "The movie was okay.",
        "The movie was terrible..."
      ]

      export_model.predict(examples)
```

```
1/1 [==============================] - 0s 121ms/step
```

```
[33]: array([[0.6360943 ],
             [0.45862636],
             [0.37422103]], dtype=float32)
```

Tạo mảng chứa 3 câu mẫu về đánh giá, phản hồi. Đưa qua model mới là export_model và thu được mảng accuracy ứng với từng mẫu đánh giá.

III. KẾT QUẢ.

Mô hình đã thực hiện được việc phân tích và phân loại cảm xúc từ phản hồi của khách hàng ở mức ổn đỉnh.

Ứng dụng được các thư viện trong Python để giải quyết các vấn đề trong môn học cũng như các vấn đề thực tế.

Mang lại kiến thức về việc thao tác, sử dụng cũng như hiểu rõ các thư viện trong Python để giải quyết một số vấn đề hiện hữu ngày nay.

IV. HƯỚNG PHÁT TRIỂN.

Chuẩn bị dữ liệu hay chuẩn hóa dữ liệu tốt hơn, bởi chúng là yếu tố quan trọng nhất trong việc phát triển các mô hình phân tích cảm xúc. Dữ liệu cần phải có độ chính xác cao và đa dạng về cảm xúc.

Lựa chọn mô hình cho giá trị accuracy cao hơn nhưng không bị Overfitting hoặc các vấn đề khác. Và xây dựng mô hình dựa trên từ vựng bằng việc sử dụng các từ vựng có liên quan đến các cảm xúc khác nhau để phân tích cảm xúc của văn bản hay dựa trên mô hình dựa trên ngữ pháp, sử dụng các cấu trúc ngữ pháp của văn bản để phân tích cảm xúc của văn bản. Cũng như, mô hình dựa trên học máy thông qua các thuật toán học máy để phân tích cảm xúc của văn bản.

Hướng đến việc phân tích cảm xúc trong văn bản tiếng Việt: Đây là một hướng phát triển tiềm năng, vì hiện nay chưa có nhiều nghiên cứu về phân tích cảm xúc trong văn bản tiếng Việt.

Phân tích cảm xúc trong văn bản đa ngôn ngữ: Đây là một hướng phát triển khó khăn hơn, vì cần phải phát triển các mô hình có thể phân tích cảm xúc trong nhiều ngôn ngữ khác nhau.

Phân tích cảm xúc trong văn bản theo ngữ cảnh: Đây là một hướng phát triển phức tạp hơn, vì cần phải xem xét ngữ cảnh của văn bản để phân tích cảm xúc chính xác hơn.

Phân tích cảm xúc trong văn bản chứa cả các icon, emoji hoặc ngữ cảnh nghĩa đen trắng, bóng gió. Thay vì phải loại bỏ chúng vì cho rằng đó là thừa.

TÀI LIỆU THAM KHẢO

[1]. TensorFlow v2.14.0, *tf.keras.layers.TextVectorization*, từ https://www.tensorflow.org/api_docs/python/tf/keras/layers/TextVectorization#adapt

[2]. Amazon, *what is Overfitting*, từ https://aws.amazon.com/vi/what-is/overfitting/

[3]. Viblo – Lê Quý Quyết, (03/06/2019), *Python RegEx*, từ https://viblo.asia/p/python-regex-Qbq5QMeE5D8

[4]. Viblo – Chung Pham Van, (23/05/2021), *Kỹ thuật Dropout (Bỏ học) trong Deep Learning*, từ https://viblo.asia/p/ky-thuat-dropout-bo-hoc-trong-deep-learning-XL6lAd8BZek

[5]. TabML, *Machine Learning cho dữ liệu dạng bảng, Embedding* từ https://machinelearningcoban.com/tabml_book/ch_embedding/embedding.html

[6]. Websitehcm.com, *Pooling Layers trong thư viện Keras*, từ https://websitehcm.com/pooling-layers-trong-thu-vien-keras/

[7]. VNCoder.vn – Ngô Đình Sơn, *Bài 11: Tổng hợp Model (Phần 1) – Keras cơ bản*, từ https://vncoder.vn/bai-hoc/tong-hop-model-phan-1-431