Submitted by: Md. Saifur Rahman
Email: mdsaifur.rahman.1@und.edu
Date: 02/15/2021

HOMEWORK #2: Supervised/Unsupervised Learning

CSCI 543: Assignment 2

Problem #1

Introduction to Machine Learning using TensorFlow

Report:

Sentiment Analysis using Basic Text Classification (NLP)

Abstract

I am implementing and run a tutorial about Natural Language Processing (NLP). More specifically, I perform Sentiment Analysis on an IMDB movie review dataset by text calcification. This IMDB dataset contains plain text files stored on disk which is unstructured data. I am performing this implementation of text analysis using python programming language. Besides, I am using Keras, TensorFlow etc. for data preprocessing, text vectorization and building the ML model for sentiment analysis on a large movie review dataset.

TensorFlow tutorial link: https://www.tensorflow.org/tutorials/keras/text_classification

My code in google drive (Colab): https://colab.research.google.com/drive/1CEvh5aGVM1_zpMli_M0mfCUFB6ZPjOeL?usp=sharing

Tools: Python programming language, Keras, TensorFlow, Google Colab, Google drive.

Dataset Details

I am using the IMBD movie review dataset from Stanford University AI Lab (https://ai.stanford.edu/~amaas/data/sentiment/aclImdb_v1.tar.gz). This is unstructured data with plain text. This Large Movie Review Dataset contains 50,000 movie review from the Internet Movie Database. These datasets are split into 25,000 reviews for training the model and remaining 25,000 reviews for testing purposes. Machine learning model usually required three types of dataset to perform any experimental analysis, (i) 'train dataset' to train the model, (ii) 'validation dataset' for evaluation the quality of the model, (iii) 'test dataset' to test the model after the model has gone through the validation process. This IMDB dataset has two type of data; train data, test data but doesn't has validation dataset. So, I divided the train dataset using 80:20 split which makes 20,000 reviews for training purpose and 5000 data for validation purpose.

Submitted by: Md. Saifur Rahman
Email: mdsaifur.rahman.1@und.edu
Date: 02/15/2021

The Approach

I am performing Sentiment Analysis using binary classifier that is a well-known Machine Learning (ML) approach in the domain of NLP. I am training the Sentiment Analysis model to classify the data into positive movie reviews and negative movie reviews. At the end, this sentiment analysis ML model will analyze new reviews form the users(outsider) and will check & predict status of the given review (the review is positive or negative). This is a four-layer Neural Network ML model. These four layer sequentially build the classifier. These are given below.

i. **Input/Embedding Layer:** This layer takes encoded reviews in integer form and lock up a vector for each word.
ii. **GlobalAveragePooling1D Layer:** This return fixed-length output vector for each example review.
iii. **Dense Layer:** Output from the previous layer piped through 16 hidden nodes of NN.
iv. **Output Layer:** Single output node.

The complete procedure of this ML approach for Sentiment Analysis is shown below.

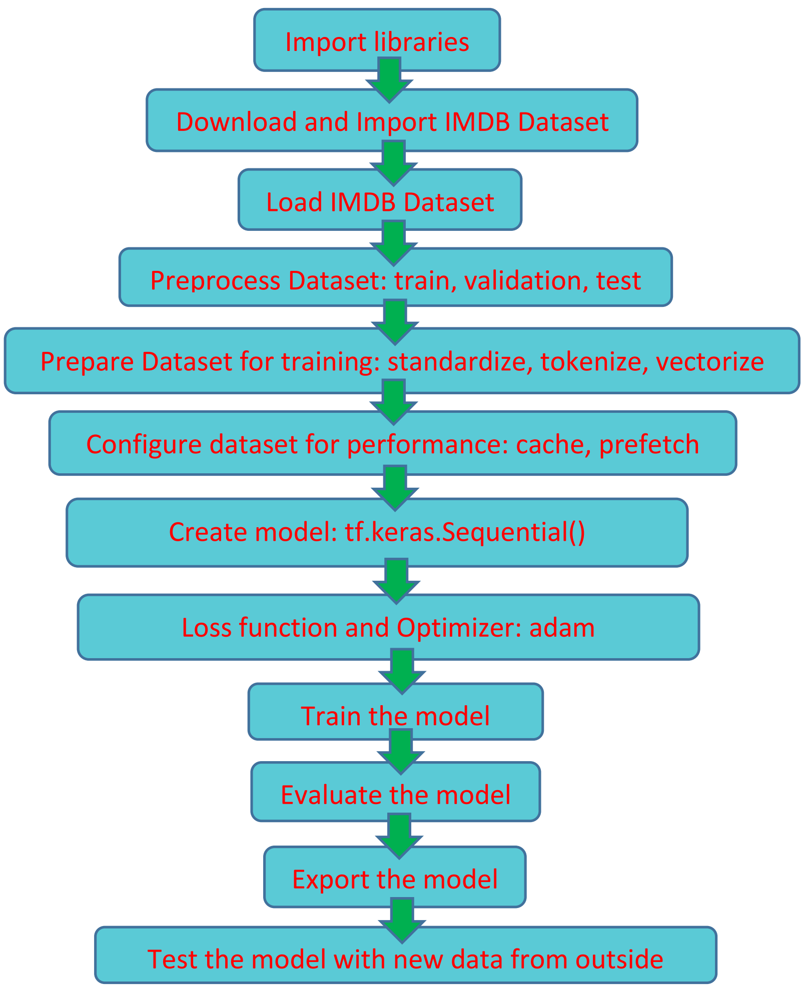

fig: Complete procedure of this ML approach for Text Analysis

Submitted by: Md. Saifur Rahman
Email: mdsaifur.rahman.1@und.edu
Date: 02/15/2021

ML Model Performance

To evaluate the performance of this model, I perform loss analysis and evaluate the accuracy too. To check these I call loss and accuracy function from model's evaluate function like the following.

```
loss, accuracy = model.evaluate(test_ds)

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

This print the following.

```
Loss:  0.30993229150772095
Accuracy:  0.873960018157959
```

So, total loss is 30% and accuracy is more than 87% which means this model works good and will predict pretty accurate result.

Output (Inference on New Data)

```
# To get predictions for new examples, I simply call model.predict()
new_examples = [
"Exciting movie. Super!",
"I don't like the ending scence. Bad story.",
"I was expecting more.",
"Excellent movie",
"I like this horror movie!",
"I am sorry to say but I don’t like this movie."]

export_model.predict(new_examples)
```

Printed output in an array of vectors is like the following.

```
array([[0.5624368 ],
       [0.36314815],
       [0.58180296],
       [0.67614913],
       [0.5000946 ],
       [0.38121626]], dtype=float32)
```

Studying these values (vectors)I can say,

i. Review 1 is positive (vector > 0.5)
ii. Review 2 is negative (vector < 0.5)
iii. Review 3 is positive (maybe this is a wrong inference!) (vector > 0.5)
iv. Review 4 is positive (vector > 0.5)

Submitted by: Md. Saifur Rahman
Email: mdsaifur.rahman.1@und.edu
Date: 02/15/2021

v. Review 5 is positive (vector > 0.5)
vi. Review 6 is negative (vector < 0.5)

Codes with Comments:

I am giving explanation of each line segment from my implementation of this neural network to perform sentiment analysis on IMDB reviews.

Colab link for the code: https://colab.research.google.com/drive/1CEvh5aGVM1_zpMli_M0mfCUFB6ZPjOeL?usp=sharing

```
"""SentimentAnalysis.ipynb

Author: Md. Saifur Rahman

Original file is located at

    https://colab.research.google.com/drive/1CEvh5aGVM1_zpMli_M0mfCUFB6ZPjOeL

"""


# importing libraries and packages to perform this text analysis

import matplotlib.pyplot as plt

import os

import re

import shutil

import string

import tensorflow as tf

from tensorflow.keras import layers

from tensorflow.keras import losses

from tensorflow.keras import preprocessing

from tensorflow.keras.layers.experimental.preprocessing import TextVectorization


# Check tensorflow version

print(tf.version.VERSION)

print(tf.__version__)


# Download IMDB dataset on disk and explore the dataset
```

Submitted by: Md. Saifur Rahman
Email: mdsaifur.rahman.1@und.edu
Date: 02/15/2021

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


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

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


# Check what I have in the IMDB dataset directory

print(os.listdir(dataset_dir))

os.listdir(dataset_dir)


# Set this directory as the training dataset directory and check the file list in this directory

train_dir = os.path.join(dataset_dir, 'train')

os.listdir(train_dir)


# Take a look at a text file from the directory. Read the text from the file and print this

sample_file = os.path.join(train_dir,'pos/0_9.txt')

#file = open(sample_file).read()

#file = open(sample_file).readline()

#print(file)

with open(sample_file) as f:

  print(f.read())


# Remove unnecessary file from the directory

remove_dir = os.path.join(train_dir, 'unsup')

shutil.rmtree(remove_dir)


# Running machine learning experiment to the labeled data and check if I have three types of dataset (train, validation, and test)

# Here, I am creating training dataset for the model

batch_size = 32

seed = 42

raw_train_ds = tf.keras.preprocessing.text_dataset_from_directory( 'aclImdb/train', batch_size = batch_size, validation_split = 0.2, subset ='training', seed = seed)
```

Submitted by: Md. Saifur Rahman
Email: mdsaifur.rahman.1@und.edu
Date: 02/15/2021

```
# Print example to view label of the data. I get 0 and 1 as label of data.
for text_batch, label_batch in raw_train_ds.take(1):
  for i in range(3):
    print("Review",text_batch.numpy()[i])
    print("Label",label_batch.numpy()[i])


# Print what these labels means?
print("Label 0 corresponds to", raw_train_ds.class_names[0])
print("Label 1 corresponds to", raw_train_ds.class_names[1])


# Creat validation dataset
raw_val_ds = tf.keras.preprocessing.text_dataset_from_directory('aclImdb/train', batch_size= batch_size,
validation_split=0.2,subset='validation',seed=seed)


# Create test dataset
raw_test_ds = tf.keras.preprocessing.text_dataset_from_directory('aclImdb/test',batch_size=batch_size)


# Print the class of these three dataset
print(raw_test_ds.class_names)
print(raw_val_ds.class_names)
print(raw_train_ds.class_names)


# Prepare the dataset
# Create a user defined function to remove HTML tag '<br />' for the dataset
def custom_standerdization(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),'')


# Create a TextVectorization layer to standardize, tokenize and vectorize my dataset
max_features = 10000  # maximun size of the vocabulary for this task
```

Submitted by: Md. Saifur Rahman
Email: mdsaifur.rahman.1@und.edu
Date: 02/15/2021

```
sequence_length = 250 # pad limit


vectorize_layer = TextVectorization( standardize=custom_standerdization, max_tokens= max_features,
output_mode='int',output_sequence_length=sequence_length)


# Make a text only dataset without labels

train_text = raw_train_ds.map(lambda x, y:x) # (x,y) means (Review,label)

#print(train_text)

# Call the adapt() function fit the state of the preprocessing layer to the dataset. This will build an index of strings to
integers

vectorize_layer.adapt(train_text)


# Create a user defined function see the result of preprocessing stage (i.e a list of integers converted from string
data)

def vectorize_text(text,label):

  text = tf.expand_dims(text, -1)

  return vectorize_layer(text) , label


# Print the converted intergers from the string data

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))


# Check and print what integer represents which string/word.

print("10:", vectorize_layer.get_vocabulary()[10])

print("344:", vectorize_layer.get_vocabulary()[344])

print("5188:", vectorize_layer.get_vocabulary()[5188])

print("6:", vectorize_layer.get_vocabulary()[6])

print("0:", vectorize_layer.get_vocabulary()[0])

print("Vocabulary size: ", len(vectorize_layer.get_vocabulary()))
```

Submitted by: Md. Saifur Rahman
Email: mdsaifur.rahman.1@und.edu
Date: 02/15/2021

```
# Apply TextVectorization layer to the train, validation and test dataset before fed into the model

train_ds = raw_train_ds.map(vectorize_text)

val_ds = raw_val_ds.map(vectorize_text)

test_ds = raw_test_ds.map(vectorize_text)


# Configure dataset for performance by using cache() and prefetch()

AUTOTUNE = tf.data.AUTOTUNE

# The number of elements to prefetch should be equal to (or possibly greater than) the number of batches consumed by a single training step. You could either manually tune this value, or set it to tf.data.AUTOTUNE, which will prompt the tf.data runtime to tune the value dynamically at runtime.

# Prefetching overlaps the preprocessing and model execution of a training step. While the model is executing training step s, the input pipeline is reading the data for step s+1. Doing so reduces the step time.

# The tf.data.Dataset.cache transformation can cache a dataset, either in memory or on local storage. This will save some operations (like file opening and data reading) from being executed during each epoch.

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)


# Create the Neural Network (model)

embedding_dim = 16

# input_dim: Integer. Size of the vocabulary, i.e. maximum integer index + 1.

# output_dim: Integer. Dimension of the dense embedding


# Define and set the layer of this Neural Network. I am setting four layer for this NN. The detail is on my report.

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()


# Configure the model to use a loss function and an optimizer
```

Submitted by: Md. Saifur Rahman
Email: mdsaifur.rahman.1@und.edu
Date: 02/15/2021

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


# Train this NN model

epochs = 10

#Each trail to learn from the input dataset is called an epoch

history = model.fit( train_ds, validation_data=val_ds,epochs=epochs)


# Check loss and accuracy to evaluate the performance of this model

# Gradient Descent is an optimization algorithm for finding a local minimum of a differentiable function. Gradient descent is simply used to find the values of a function's parameters (coefficients) that minimize a cost function as far as possible.

loss, accuracy = model.evaluate(test_ds)


print("Loss",loss)

print("Accuracy",accuracy)


# create a plot to better understand

# model.fit() returns a History object that contains a dictionary with everything that happened during training

history_dict = history.history

history_dict.keys()


# plot to visualize loss

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" for blue dot

plt.plot(epochs, loss, 'bo', label = 'Training Loss')

# "b" for solid bule line

plt.plot(epochs, val_loss, 'b', label = 'Validation Loss')
```

Submitted by: Md. Saifur Rahman
Email: mdsaifur.rahman.1@und.edu
Date: 02/15/2021

```
plt.xlabel("Ephocs")

plt.ylabel("Loss")

plt.legend()

plt.show()


# plot to visualize accuracy

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()


# In the code above, I applied the TextVectorization layer to the dataset before feeding text to the model. If I want to make my model capable of processing raw strings (for example, to simplify deploying it), I can include the TextVectorization layer inside the model.

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)
```

Submitted by: Md. Saifur Rahman
Email: mdsaifur.rahman.1@und.edu
Date: 02/15/2021

To get predictions for new examples, I simply call model.predict()

new_examples = ["Exciting movie. Super!", "I don't like the ending scence. Bad story.", "I was expecting more.", "Excellent movie", "I like this horror movie!", "I am sorry to say but Idont like this movie."]

Print vectors for new example

export_model.predict(new_examples)