from keras.datasets import imdb

%matplotlib inline

import numpy as np

import pandas as pd

from matplotlib import cm

import matplotlib.pyplot as plt

import seaborn as sns

import os

import time

from keras.preprocessing import sequence

from keras.models import Sequential

from keras.layers import Dense, Dropout, Activation

from keras.layers import Embedding

from keras.layers import Conv1D, GlobalMaxPooling1D

from keras.callbacks import EarlyStopping

from keras import models

(X\_train, y\_train), (X\_test, y\_test) = imdb.load\_data()

X = np.concatenate((X\_train, X\_test), axis=0)

y = np.concatenate((y\_train, y\_test), axis=0)

##training data shape review

print("Training data: ")

print(X.shape)

print(y.shape)

print("Classes: ")

print(np.unique(y))

print("Number of words: ")

print(len(np.unique(np.hstack(X))))

print("Review length: ")

result = [len(x) for x in X]

print("Mean %.2f words (%f)" % (np.mean(result), np.std(result)))

# plot review length

plt.boxplot(result)

plt.show()

(train\_data, train\_labels), (test\_data, test\_labels) = imdb.load\_data(num\_words=5000)

def vectorize\_sequences(sequences, dimension=5000):

# Create an all-zero matrix of shape (len(sequences), dimension)

results = np.zeros((len(sequences), dimension))

for i, sequence in enumerate(sequences):

results[i, sequence] = 1. # set specific indices of results[i] to 1s

return results

# Our vectorized training data

x\_train = vectorize\_sequences(train\_data)

# Our vectorized test data

x\_test = vectorize\_sequences(test\_data)

# Our vectorized labels one-hot encoder

y\_train = np.asarray(train\_labels).astype('float32')

y\_test = np.asarray(test\_labels).astype('float32')

from keras import layers

from keras import models

model = models.Sequential()

model.add(layers.Dense(32, activation='relu', input\_shape=(5000,)))

model.add(layers.Dense(32, activation='relu',))

model.add(layers.Dense(1, activation='sigmoid'))

#Set validation set aside

x\_val = x\_train[:10000]

partial\_x\_train = x\_train[10000:]

y\_val = y\_train[:10000]

partial\_y\_train = y\_train[10000:]

model.compile(optimizer='adam',

loss='binary\_crossentropy',

metrics=['acc'])

start\_time\_m1 = time.time()

history = model.fit(partial\_x\_train,

partial\_y\_train,

epochs=20,

batch\_size=512,

validation\_data=(x\_val, y\_val))

total\_time\_m1 = time.time() - start\_time\_m1

print("The Dense Convolutional Neural Network 1 layer took %.4f seconds to train." % (total\_time\_m1))

history\_dict = history.history

history\_dict.keys()

import matplotlib.pyplot as plt

%matplotlib inline

acc = history.history['acc']

val\_acc = history.history['val\_acc']

loss = history.history['loss']

val\_loss = history.history['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()

plt.clf() # clear figure

acc\_values = history\_dict['acc']

val\_acc\_values = history\_dict['val\_acc']

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('Loss')

plt.legend()

plt.show()

model.summary()

from sklearn.metrics import confusion\_matrix, accuracy\_score, auc

#predictions

pred = model.predict(x\_test)

classes\_x=np.argmax(pred,axis=1)

#accuracy

accuracy\_score(y\_test,classes\_x)

#Confusion Matrix

conf\_mat = confusion\_matrix(y\_test, classes\_x)

print(conf\_mat)

conf\_mat\_normalized = conf\_mat.astype('float') / conf\_mat.sum(axis=1)[:, np.newaxis]

sns.heatmap(conf\_mat\_normalized)

plt.ylabel('True label')

plt.xlabel('Predicted label')

#Dense with Two Layer

model2 = models.Sequential()

model2.add(layers.Dense(32, activation='relu', input\_shape=(5000,)))

model2.add(layers.Dense(32, activation='relu'))

model2.add(layers.Dense(32, activation='relu'))

model2.add(layers.Dense(1, activation='sigmoid'))

model2.compile(optimizer='adam',

loss='binary\_crossentropy',

metrics=['acc'])

start\_time\_m2 = time.time()

history= model2.fit(partial\_x\_train,

partial\_y\_train,

epochs=20,

batch\_size=512,

validation\_data=(x\_val, y\_val))

total\_time\_m2 = time.time() - start\_time\_m2

print("The Dense Convolutional Neural Network 2 layers took %.4f seconds to train." % (total\_time\_m2))

acc = history.history['acc']

val\_acc = history.history['val\_acc']

loss = history.history['loss']

val\_loss = history.history['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('DNN 2 layer Training and validation loss')

plt.xlabel('Epochs')

plt.ylabel('Loss')

plt.legend()

plt.show()

plt.clf() # clear figure

acc\_values = history\_dict['acc']

val\_acc\_values = history\_dict['val\_acc']

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

plt.plot(epochs, val\_acc, 'b', label='Validation acc')

plt.title('DNN 2 layer Training and validation accuracy')

plt.xlabel('Epochs')

plt.ylabel('Loss')

plt.legend()

plt.show()

model2.summary()

from numpy.ma.core import argmax

pred = model2.predict(x\_test)

classes\_x=argmax(pred,axis=-1)

#accuracy

accuracy\_score(y\_test,classes\_x)