Assignment 10

author: Rachel Nelson

class: DSC650

Assignment 10.1a

In the first part of the assignment, you will implement basic text-preprocessing functions in Python. These functions do not need to scale to large text documents and will only need to handle small inputs.

272
```
from nltk.tokenize import word_tokenize
import string
import re

# Assignment 10.1.a
def tokenize(sentence):
# load data
    # split into words
    from nltk.tokenize import word_tokenize
    tokens = word_tokenize(text)
    # convert to lower case
    tokens = [w.lower() for w in tokens]
    # remove punctuation from each word
    import string
    table = str.maketrans('', '', string.punctuation)
    stripped = [w.translate(table) for w in tokens]
    # remove remaining tokens that are not alphabetic
    words = [word for word in stripped if word.isalpha()]
    print(words)
    return tokens
#https://machinelearningmastery.com/clean-text-machine-learning-python/
```

273
```
text = "This is the song that never ends. It just goes on and on, my friend. Some people started singing
           "not knowing what it was. And, they'll continue singing it forever, just because" \
           "This is the song that never ends…"
tokens = tokenize(text)
```

```
['this', 'is', 'the', 'song', 'that', 'never', 'ends', 'it', 'just', 'goes', 'on', 'and', 'on', 'my', 'fr
```

Assignment 10.1b

Implement an `ngram` function that splits tokens into N-grams.

```
274 import nltk
    def ngram(tokens, n):
        stripped = [w.translate(tokens) for w in tokens]
        ngrams = nltk.ngrams(stripped, n)
        return ngrams
    ngram(tokens, 3)
    # External References
    # https://www.analyticsvidhya.com/blog/2021/09/what-are-n-grams-and-how-to-implement-them-in-python/#:~:t
    # https://stackoverflow.com/questions/13423919/computing-n-grams-using-python
```

```
274 <zip at 0x1796e930108>
```

Assignment 10.1c

Implement a one_hot_encode function to create a vector from a numerical vector from a list of tokens.

```
275 import numpy as np
    def one_hot_encode(tokens, num_words):
        token_index = {}
        for sample in tokens:
            for word in sample.split():
                if word not in token_index:
                    token_index[word] = len(token_index) + 1
        max_length = 10
        results = np.zeros(shape=(len(tokens),
                                  max_length,
                                  max(token_index.values()) + 1))
        return results
    one_hot_encode(tokens,5)
    # Listing 6.1 Word-level one-hot encoding (toy example using token words)
```

```
275 array([[[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.]]])
```

Assignment 10.2

Using listings 6.16, 6.17, and 6.18 in Deep Learning with Python as a guide, train a sequential model with embeddings on the IMDB data found in data/external/imdb/. Produce the model performance metrics and training and validation accuracy curves within the Jupyter notebook.

276
```
# Listing 6.8 Processing the labels of the raw IMDB Data
import os
imdb_dir = 'D:/College/DSC650/dsc650/data/external/imdb/aclImdb/'
train_dir = os.path.join(imdb_dir, 'train')
test_dir = os.path.join(imdb_dir, 'test')
labels = []
texts = []
for label_type in ['neg', 'pos']:
    dir_name = os.path.join(test_dir, label_type)
    for fname in sorted(os.listdir(dir_name)):
        if fname[-4:] == '.txt':
            f = open(os.path.join(dir_name, fname), encoding="utf8")
            texts.append(f.read())
            f.close()
            if label_type == 'neg':
                labels.append(0)
            else:
                labels.append(1)
```

277
```
# Listing 6.9 Tokenizing the text of the raw IMDB data
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
import numpy as np

maxlen = 100
training_samples = 200
validation_samples = 10000
```

```
max_words = 10000

tokenizer = Tokenizer(num_words=max_words)
tokenizer.fit_on_texts(texts)
sequences = tokenizer.texts_to_sequences(texts)

word_index = tokenizer.word_index
data = pad_sequences(sequences, maxlen=maxlen)
labels = np.asarray(labels)

indices = np.arange(data.shape[0])
np.random.shuffle(indices)
data = data[indices]
labels = labels[indices]

x_train = data[:training_samples]
y_train = labels[:training_samples]
x_val = data[training_samples: training_samples + validation_samples]
y_val = labels[training_samples: training_samples + validation_samples]
```

278
```
# listing 6.12 model definition
from keras.models import Sequential
from keras.layers import Embedding, Flatten, Dense
model = Sequential()
model.add(Embedding(max_words, embedding_dim, input_length=maxlen))
model.add(Flatten())
model.add(Dense(32,activation='relu'))
model.add(Dense(1, activation='sigmoid'))
model.summary()
```

```
Model: "sequential_14"
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 embedding_12 (Embedding)     (None, 100, 100)          1000000

 flatten_12 (Flatten)         (None, 10000)             0

 dense_24 (Dense)             (None, 32)                320032

 dense_25 (Dense)             (None, 1)                 33

=================================================================
Total params: 1,320,065
Trainable params: 1,320,065
Non-trainable params: 0
_________________________________________________________________
```

279
```
# Listing 6.14 Training and evaluation
model.compile(optimizer='rmsprop',
              loss='binary_crossentropy',
              metrics=['acc'])
history = model.fit(x_train, y_train,
                    epochs=10,
                    batch_size=32,
                    validation_data=(x_val, y_val))
model.save_weights('pre_trained_glove_model.h5')
```

```
Epoch 1/10
7/7 [==============================] - 1s 70ms/step - loss: 0.6979 - acc: 0.4550 - val_loss: 0.6919 - val
Epoch 2/10
7/7 [==============================] - 0s 57ms/step - loss: 0.5059 - acc: 0.9750 - val_loss: 0.6937 - val
Epoch 3/10
```

```
7/7 [==============================] - 0s 62ms/step - loss: 0.2984 - acc: 0.9950 - val_loss: 0.7030 - val
Epoch 4/10
7/7 [==============================] - 0s 62ms/step - loss: 0.1298 - acc: 1.0000 - val_loss: 0.6991 - val
Epoch 5/10
7/7 [==============================] - 0s 60ms/step - loss: 0.0575 - acc: 1.0000 - val_loss: 0.7000 - val
Epoch 6/10
7/7 [==============================] - 0s 59ms/step - loss: 0.0301 - acc: 1.0000 - val_loss: 0.7037 - val
Epoch 7/10
7/7 [==============================] - 0s 61ms/step - loss: 0.0161 - acc: 1.0000 - val_loss: 0.7158 - val
Epoch 8/10
7/7 [==============================] - 0s 59ms/step - loss: 0.0093 - acc: 1.0000 - val_loss: 0.7125 - val
Epoch 9/10
7/7 [==============================] - 0s 62ms/step - loss: 0.0056 - acc: 1.0000 - val_loss: 0.7180 - val
Epoch 10/10
7/7 [==============================] - 0s 60ms/step - loss: 0.0034 - acc: 1.0000 - val_loss: 0.7236 - val
```

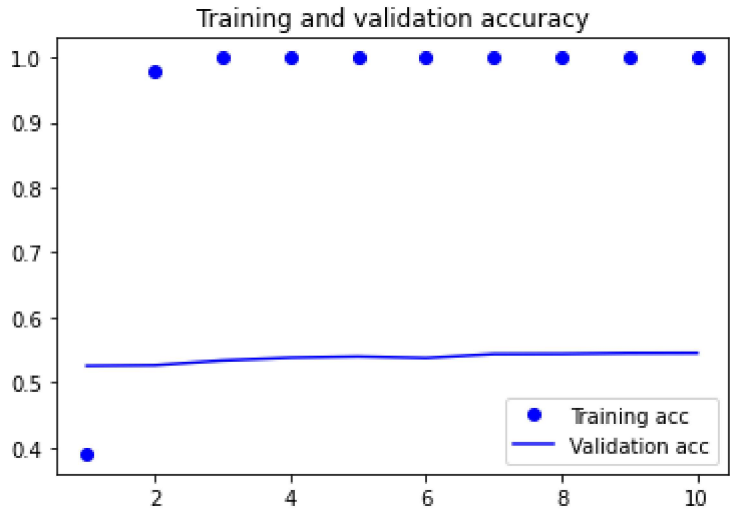
288

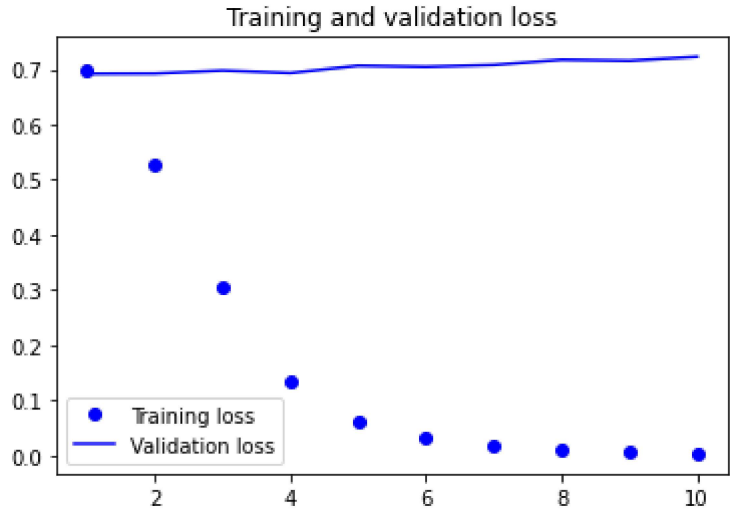
```
# Listing 6.16
from keras.models import Sequential
from keras.layers import Embedding, Flatten, Dense
model = Sequential()
model.add(Embedding(max_words, embedding_dim, input_length=maxlen))
model.add(Flatten())
model.add(Dense(32, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
model.summary()
model.compile(optimizer='rmsprop',
              loss='binary_crossentropy',
              metrics=['acc'])
history = model.fit(x_train, y_train,
                    epochs=10,
                    batch_size=32,
                    validation_data=(x_val, y_val))
```

```
Model: "sequential_17"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_15 (Embedding)    (None, 100, 100)          1000000

 flatten_15 (Flatten)        (None, 10000)             0

 dense_30 (Dense)            (None, 32)                320032

 dense_31 (Dense)            (None, 1)                 33

=================================================================
Total params: 1,320,065
Trainable params: 1,320,065
Non-trainable params: 0
_________________________________________________________________
Epoch 1/10
7/7 [==============================] - 1s 86ms/step - loss: 0.6917 - acc: 0.5150 - val_loss: 0.6927 - val
Epoch 2/10
7/7 [==============================] - 0s 71ms/step - loss: 0.4860 - acc: 0.9700 - val_loss: 0.6957 - val
Epoch 3/10
7/7 [==============================] - 0s 64ms/step - loss: 0.2720 - acc: 1.0000 - val_loss: 0.6997 - val
Epoch 4/10
7/7 [==============================] - 0s 61ms/step - loss: 0.1226 - acc: 1.0000 - val_loss: 0.7038 - val
Epoch 5/10
7/7 [==============================] - 0s 65ms/step - loss: 0.0595 - acc: 1.0000 - val_loss: 0.7033 - val
Epoch 6/10
7/7 [==============================] - 0s 64ms/step - loss: 0.0312 - acc: 1.0000 - val_loss: 0.7086 - val
```

```
Epoch 7/10
7/7 [==============================] - 0s 64ms/step - loss: 0.0175 - acc: 1.0000 - val_loss: 0.7278 - val
Epoch 8/10
7/7 [==============================] - 0s 68ms/step - loss: 0.0101 - acc: 1.0000 - val_loss: 0.7307 - val
Epoch 9/10
7/7 [==============================] - 0s 65ms/step - loss: 0.0059 - acc: 1.0000 - val_loss: 0.7243 - val
Epoch 10/10
7/7 [==============================] - 0s 65ms/step - loss: 0.0036 - acc: 1.0000 - val_loss: 0.7330 - val
```

282
```
# Plotting Results of 6.16
import matplotlib.pyplot as plt
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)
plt.plot(epochs, acc, 'bo', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.legend()
plt.figure()
plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()
plt.show()
```





```
289 # Listing 6.17
    test_dir = os.path.join(imdb_dir, 'test')

    labels = []
    texts = []

    for label_type in ['neg', 'pos']:
        dir_name = os.path.join(test_dir, label_type)
        for fname in sorted(os.listdir(dir_name)):
            if fname[-4:] == '.txt':
                f = open(os.path.join(dir_name, fname))
                texts.append(f.read())
                f.close()
                if label_type == 'neg':
                    labels.append(0)
                else:
                    labels.append(1)

    sequences = tokenizer.texts_to_sequences(texts)
    x_test = pad_sequences(sequences, maxlen=maxlen)
    y_test = np.asarray(labels)
```

```
---------------------------------------------------------------------------
UnicodeDecodeError                        Traceback (most recent call last)

~\AppData\Local\Temp/ipykernel_17376/1823448156.py in <module>
     10         if fname[-4:] == '.txt':
     11             f = open(os.path.join(dir_name, fname))
---> 12             texts.append(f.read())
     13             f.close()
     14             if label_type == 'neg':

~\AppData\Local\Programs\Python\Python37\lib\encodings\cp1252.py in decode(self, input, final)
     21 class IncrementalDecoder(codecs.IncrementalDecoder):
     22     def decode(self, input, final=False):
---> 23         return codecs.charmap_decode(input,self.errors,decoding_table)[0]
     24
     25 class StreamWriter(Codec,codecs.StreamWriter):

UnicodeDecodeError: 'charmap' codec can't decode byte 0x9d in position 1757: character maps to <undefined
```

```
290 # Listing 6.18 Evaluating the model on the test set
    model.load_weights('pre_trained_glove_model.h5')
    model.evaluate(x_test, y_test)
```

```
    782/782 [==============================] - 1s 1ms/step - loss: 0.7185 - acc: 0.5459□□□□□□□□□□□□□□□□□□□□□□□□□□□□

290 [0.7185297608375549, 0.5458800196647644]
```

Assignment 10.3

Using listing 6.27 in Deep Learning with Python as a guide, fit the same data with an LSTM layer. Produce the model performance metrics and training and validation accuracy curves within the Jupyter notebook.

295

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)

~\AppData\Local\Temp/ipykernel_17376/1061558647.py in <module>
      3
      4 max_features = 10000
----> 5 input_train = sequence.pad_sequences(input_train, maxlen=maxlen)
      6 from keras.layers import LSTM
      7 model = Sequential()

NameError: name 'input_train' is not defined
```

296

```
# Listing 6.22 Preparing the IMDB Datga
from keras.datasets import imdb
from keras.preprocessing import sequence
max_features = 10000
maxlen = 500
batch_size = 32
print('Loading data...')
(input_train, y_train), (input_test, y_test) = imdb.load_data(
num_words=max_features)
print(len(input_train), 'train sequences')
print(len(input_test), 'test sequences')
print('Pad sequences (samples x time)')
input_train = sequence.pad_sequences(input_train, maxlen=maxlen)
input_test = sequence.pad_sequences(input_test, maxlen=maxlen)
print('input_train shape:', input_train.shape)
print('input_test shape:', input_test.shape)
```

```
Loading data...
25000 train sequences
25000 test sequences
Pad sequences (samples x time)
input_train shape: (25000, 500)
input_test shape: (25000, 500)
```

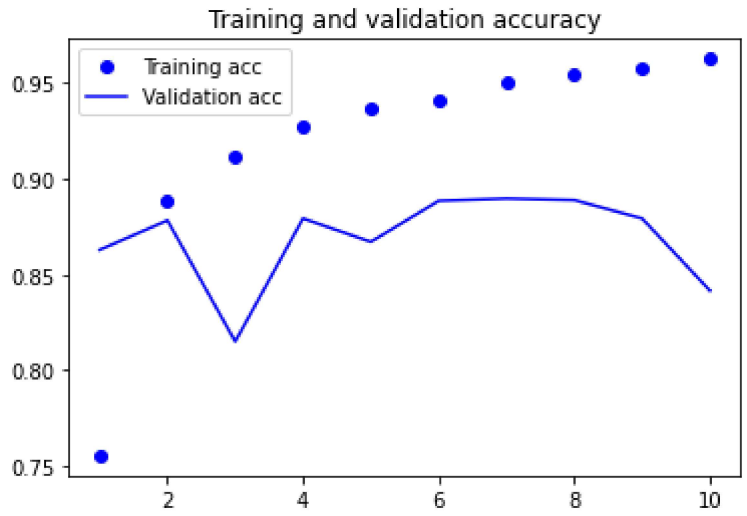
297

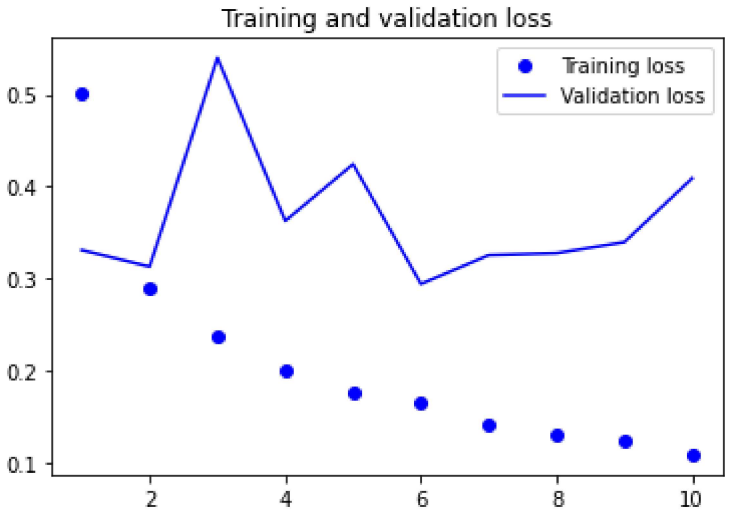
```
# Listing 6.27
from keras.preprocessing import sequence

max_features = 10000
input_train = sequence.pad_sequences(input_train, maxlen=maxlen)
from keras.layers import LSTM
model = Sequential()
model.add(Embedding(max_features, 32))
model.add(LSTM(32))
model.add(Dense(1, activation='sigmoid'))
model.compile(optimizer='rmsprop',
              loss='binary_crossentropy',
              metrics=['acc'])
history = model.fit(input_train, y_train,
                    epochs=10,
                    batch_size=128,
                    validation_split=0.2)
```

```
Epoch 1/10
157/157 [==============================] - 28s 166ms/step - loss: 0.5020 - acc: 0.7556 - val_loss: 0.3310
Epoch 2/10
157/157 [==============================] - 26s 164ms/step - loss: 0.2896 - acc: 0.8888 - val_loss: 0.3131
Epoch 3/10
157/157 [==============================] - 26s 166ms/step - loss: 0.2366 - acc: 0.9117 - val_loss: 0.5399
```

```
Epoch 4/10
157/157 [==============================] - 25s 162ms/step - loss: 0.2008 - acc: 0.9275 - val_loss: 0.3627
Epoch 5/10
157/157 [==============================] - 26s 163ms/step - loss: 0.1769 - acc: 0.9363 - val_loss: 0.4243
Epoch 6/10
157/157 [==============================] - 26s 164ms/step - loss: 0.1646 - acc: 0.9406 - val_loss: 0.2942
Epoch 7/10
157/157 [==============================] - 26s 164ms/step - loss: 0.1421 - acc: 0.9504 - val_loss: 0.3255
Epoch 8/10
157/157 [==============================] - 26s 164ms/step - loss: 0.1314 - acc: 0.9547 - val_loss: 0.3276
Epoch 9/10
157/157 [==============================] - 26s 164ms/step - loss: 0.1236 - acc: 0.9570 - val_loss: 0.3396
Epoch 10/10
157/157 [==============================] - 27s 170ms/step - loss: 0.1089 - acc: 0.9625 - val_loss: 0.4090
```

```
300 # Plotting Results
    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)

    plt.plot(epochs, acc, 'bo', label='Training acc')
    plt.plot(epochs, val_acc, 'b', label='Validation acc')
    plt.title('Training and validation accuracy')
    plt.legend()

    plt.figure()

    plt.plot(epochs, loss, 'bo', label='Training loss')
    plt.plot(epochs, val_loss, 'b', label='Validation loss')
    plt.title('Training and validation loss')
    plt.legend()

    plt.show()
```





Assignment 10.4

Using listing 6.46 in Deep Learning with Python as a guide, fit the same data with a simple 1D convnet. Produce the model performance metrics and training and validation accuracy curves within the Jupyter notebook.

303
```
# Listing 6.45 Preparing the IMDB Data
from keras.datasets import imdb
from keras.preprocessing import sequence
max_features = 10000
max_len = 500
print('Loading data...')
(x_train, y_train), (x_test, y_test) = imdb.load_data(num_words=max_features)
print(len(x_train), 'train sequences')
print(len(x_test), 'test sequences')
print('Pad sequences (samples x time)')
x_train = sequence.pad_sequences(x_train, maxlen=max_len)
x_test = sequence.pad_sequences(x_test, maxlen=max_len)
print('x_train shape:', x_train.shape)
```

```
Loading data...
25000 train sequences
25000 test sequences
Pad sequences (samples x time)
x_train shape: (25000, 500)
```

304
```
from keras.models import Sequential
from keras import layers
from tensorflow.keras.optimizers import RMSprop

model = Sequential()
model.add(layers.Embedding(max_features, 128, input_length=max_len))
model.add(layers.Conv1D(32, 7, activation='relu'))
model.add(layers.MaxPooling1D(5))
model.add(layers.Conv1D(32, 7, activation='relu'))
model.add(layers.GlobalMaxPooling1D())
model.add(layers.Dense(1))

model.summary()
```

```
model.compile(optimizer=RMSprop(lr=1e-4),
              loss='binary_crossentropy',
              metrics=['acc'])
history = model.fit(x_train, y_train,
                    epochs=10,
                    batch_size=128,
                    validation_split=0.2)
```

```
Model: "sequential_22"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_18 (Embedding)    (None, 500, 128)          1280000

 conv1d (Conv1D)             (None, 494, 32)           28704

 max_pooling1d (MaxPooling1D  (None, 98, 32)           0
 )

 conv1d_1 (Conv1D)           (None, 92, 32)            7200

 global_max_pooling1d (Globa  (None, 32)               0
 lMaxPooling1D)

 dense_34 (Dense)            (None, 1)                 33

=================================================================
Total params: 1,315,937
Trainable params: 1,315,937
Non-trainable params: 0
_________________________________________________________________
Epoch 1/10
D:\College\venv\lib\site-packages\keras\optimizer_v2\rmsprop.py:130: UserWarning: The `lr` argument is de
  super(RMSprop, self).__init__(name, **kwargs)

157/157 [==============================] - 21s 128ms/step - loss: 0.8993 - acc: 0.5055 - val_loss: 0.6877
Epoch 2/10
157/157 [==============================] - 19s 118ms/step - loss: 0.6726 - acc: 0.6413 - val_loss: 0.6696
Epoch 3/10
157/157 [==============================] - 18s 117ms/step - loss: 0.6329 - acc: 0.7433 - val_loss: 0.6287
Epoch 4/10
157/157 [==============================] - 19s 122ms/step - loss: 0.5502 - acc: 0.8013 - val_loss: 0.5132
Epoch 5/10
157/157 [==============================] - 18s 115ms/step - loss: 0.4281 - acc: 0.8396 - val_loss: 0.4221
Epoch 6/10
157/157 [==============================] - 21s 136ms/step - loss: 0.3511 - acc: 0.8722 - val_loss: 0.4019
Epoch 7/10
157/157 [==============================] - 18s 116ms/step - loss: 0.3037 - acc: 0.8913 - val_loss: 0.3966
Epoch 8/10
157/157 [==============================] - 19s 124ms/step - loss: 0.2702 - acc: 0.9043 - val_loss: 0.4081
Epoch 9/10
157/157 [==============================] - 21s 131ms/step - loss: 0.2441 - acc: 0.9163 - val_loss: 0.4235
Epoch 10/10
157/157 [==============================] - 21s 135ms/step - loss: 0.2187 - acc: 0.9258 - val_loss: 0.4397
```

```
# Plotting Results
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)

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

```
plt.plot(epochs, val_acc, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.legend()

plt.figure()

plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()

plt.show()
```