Assignment 10

DSCT650

Taniya Adhikari

Assignment 10.1a

In [1]:

```
import re
from string import punctuation
```

In [2]:

```
def tokenize(sentence, split = " "):

    # removes punctuation
    pattern = r'[^A-Za-z ]'
    regex = re.compile(pattern)
    sentence = regex.sub(' ', sentence)
    sentence = sentence.replace("\n", "")


    # Convert string to lower case
    sentence = sentence.lower()

    #removes duplicate spaces
    sentence = re.sub(' +', ' ', sentence)


    # remove whitespace
    sentence = sentence.strip()

    # Split by spaces
    tokens = sentence.split(split)
    return tokens
```

In [3]:

```
s = """
She stared at it, and rubbed her eyes, and stared at it again.

"Well! I never!" she said at last. "And me thinking it was a pot of gold!
I must have been dreaming. But this is luck! Silver is far less trouble—easier
to mind, and not so easy stolen. Them gold pieces would have been the death o'
me,
and with this great lump of silver—"

"""
```

In [4]:

```
tokens = tokenize(s)
print(tokens)
```

```
['she', 'stared', 'at', 'it', 'and', 'rubbed', 'her', 'eyes', 'and', 'stare
d', 'at', 'it', 'again', 'well', 'i', 'never', 'she', 'said', 'at', 'last',
'and', 'me', 'thinking', 'it', 'was', 'a', 'pot', 'of', 'gold', 'i', 'must',
'have', 'been', 'dreaming', 'but', 'this', 'is', 'luck', 'silver', 'is', 'fa
r', 'less', 'trouble', 'easier', 'to', 'mind', 'and', 'not', 'so', 'easy', 's
tolen', 'them', 'gold', 'pieces', 'would', 'have', 'been', 'the', 'death',
'o', 'me', 'and', 'with', 'this', 'great', 'lump', 'of', 'silver']
```

Assignment 10.1b

In [5]:

```
def ngram(tokens, n):
    ngrams = []

    for num in range(0, len(tokens)):
        ngram = ' '.join(tokens[num:num + n])
        ngrams.append(ngram)

    return ngrams

print(ngram(tokens, 2))
```

```
['she stared', 'stared at', 'at it', 'it and', 'and rubbed', 'rubbed her', 'h
er eyes', 'eyes and', 'and stared', 'stared at', 'at it', 'it again', 'again
well', 'well i', 'i never', 'never she', 'she said', 'said at', 'at last', 'l
ast and', 'and me', 'me thinking', 'thinking it', 'it was', 'was a', 'a pot',
'pot of', 'of gold', 'gold i', 'i must', 'must have', 'have been', 'been drea
ming', 'dreaming but', 'but this', 'this is', 'is luck', 'luck silver', 'silv
er is', 'is far', 'far less', 'less trouble', 'trouble easier', 'easier to',
'to mind', 'mind and', 'and not', 'not so', 'so easy', 'easy stolen', 'stolen
them', 'them gold', 'gold pieces', 'pieces would', 'would have', 'have been',
'been the', 'the death', 'death o', 'o me', 'me and', 'and with', 'with thi
s', 'this great', 'great lump', 'lump of', 'of silver', 'silver']
```

Assignment 10.1c

In [6]:

```
import numpy as np
def one_hot_encode(tokens, num_words):
    token_index = {}
    for word in tokens:
        if word not in token_index:
            token_index[word] = len(token_index) + 1

    results = np.zeros(shape=(num_words,max(token_index.values()) + 1))
    for j, word in enumerate(tokens):
        index = token_index.get(word)
        results[j, index] = 1.
    return results
```

In [7]:
```
one_hot_encode(tokens, len(tokens))
```

Out[7]:
```
array([[0., 1., 0., ..., 0., 0., 0.],
       [0., 0., 1., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       ...,
       [0., 0., 0., ..., 0., 0., 1.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.]])
```

Assignment 10.2

In [8]:
```
import pandas as pd
import matplotlib.pyplot as plt
import tensorflow

from tensorflow import keras
from keras import models, layers, losses, optimizers
from keras.models import Sequential
from keras.layers import Embedding, Flatten, Dense
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
```

```
Using TensorFlow backend.
```

In [9]:
```
import os
from pathlib import Path
import shutil, random
```

In [10]:
```
current_dir = Path(os.getcwd()).absolute()
current_dir = Path(current_dir).parents[2]
current_dir
```

Out[10]:
```
WindowsPath('C:/Users/bibek/Documents/GitHub/dsc650')
```

In [11]:
```
data_dir = current_dir.joinpath('data')
external_dir = data_dir.joinpath('external')
imdb_dir = external_dir.joinpath('imdb')
base_dir = imdb_dir.joinpath("aclImdb")
base_dir
```

Out[11]:
```
WindowsPath('C:/Users/bibek/Documents/GitHub/dsc650/data/external/imdb/aclImd
b')
```

In [12]:
```
train_dir = base_dir.joinpath("train")
test_dir = base_dir.joinpath("test")
```

In [13]:

```
def data_set(directory):
    labels = []
    texts = []
    for label_type in ['neg', 'pos']:
        dir_name = os.path.join(directory, label_type)
        for root, dirs, files in os.walk(dir_name):
            for file in files:
                current_path = Path(root).joinpath(file)
                with open(current_path, encoding="utf8") as f:
                    review = f.read()
                    texts.append(review)
                    f.close()
                    if label_type == 'neg':
                        labels.append(0)
                    else:
                        labels.append(1)
    return texts, labels
```

In [14]:

```
X_train, Y_train = data_set(train_dir)
X_test, Y_test = data_set(test_dir)
```

In [15]:

```
X_train[0], Y_train[0]
```

Out[15]:

```
("Story of a man who has unnatural feelings for a pig. Starts out with a open
ing scene that is a terrific example of absurd comedy. A formal orchestra aud
ience is turned into an insane, violent mob by the crazy chantings of it's si
ngers. Unfortunately it stays absurd the WHOLE time with no general narrative
eventually making it just too off putting. Even those from the era should be
turned off. The cryptic dialogue would make Shakespeare seem easy to a third
grader. On a technical level it's better than you might think with some good
cinematography by future great Vilmos Zsigmond. Future stars Sally Kirkland a
nd Frederic Forrest can be seen briefly.",
 0)
```

In [16]:

```
X_test[0], Y_test[0]
```

Out[16]:

```
("Once again Mr. Costner has dragged out a movie for far longer than necessar
y. Aside from the terrific sea rescue sequences, of which there are very few
I just did not care about any of the characters. Most of us have ghosts in th
e closet, and Costner's character are realized early on, and then forgotten u
ntil much later, by which time I did not care. The character we should really
care about is a very cocky, overconfident Ashton Kutcher. The problem is he c
omes off as kid who thinks he's better than anyone else around him and shows
no signs of a cluttered closet. His only obstacle appears to be winning over
Costner. Finally when we are well past the half way point of this stinker, Co
stner tells us all about Kutcher's ghosts. We are told why Kutcher is driven
to be the best with no prior inkling or foreshadowing. No magic here, it was
all I could do to keep from turning it off an hour in.",
 0)
```

In [17]:

```python
def preprocess_data(review):
    # removes punctuation
    pattern = r'[^A-Za-z ]'
    regex = re.compile(pattern)
    review = regex.sub(' ', review)
    review = review.replace("\n", "")

    # Convert string to lower case
    review = review.lower()
    #removes duplicate spaces
    review = re.sub(' +', ' ', review)
    # remove whitespace
    review = review.strip()

    return review
```

In [18]:

```python
X_train = [preprocess_data(review) for review in X_train]
X_test = [preprocess_data(review) for review in X_test]
```

In [19]:

```python
len(X_train)
```

Out[19]:

```
25000
```

Tokenizing

In [20]:

```python
max_words = 10000
tokenizer = Tokenizer(num_words = max_words)
tokenizer.fit_on_texts(X_train)
sequences = tokenizer.texts_to_sequences(X_train)
word_index = tokenizer.word_index
print('Found %s unique tokens.' % len(word_index))
```

```
Found 73272 unique tokens.
```

In [21]:

```python
max_len = max([len(x) for x in sequences])
data = pad_sequences(sequences, maxlen=max_len)
labels = np.asarray(Y_train)

print('Shape of data tensor:', data.shape)
print('Shape of label tensor:', labels.shape)
```

```
Shape of data tensor: (25000, 2234)
Shape of label tensor: (25000,)
```

In [22]:

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

In [23]:

```
# Split the data into training and validation sets
X_train = data[500:]
Y_train = labels[500:]

X_val = data[:500]
Y_val = labels[:500]
```

Training the Model

In [24]:

```
embedding_dim = 100

model = Sequential()
model.add(Embedding(max_words,
                    embedding_dim,
                    input_length=max_len))
model.add(Flatten())
model.add(Dense(32, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
embedding_1 (Embedding)      (None, 2234, 100)         1000000   
_________________________________________________________________
flatten_1 (Flatten)          (None, 223400)            0         
_________________________________________________________________
dense_1 (Dense)              (None, 32)                7148832   
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 33        
=================================================================
Total params: 8,148,865
Trainable params: 8,148,865
Non-trainable params: 0
_________________________________________________________________
```

In [25]:
```
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))
```

```
C:\Users\bibek\anaconda3\envs\dsc650\lib\site-packages\tensorflow_core\python
\framework\indexed_slices.py:433: UserWarning: Converting sparse IndexedSlice
s to a dense Tensor of unknown shape. This may consume a large amount of memo
ry.
  "Converting sparse IndexedSlices to a dense Tensor of unknown shape. "

Train on 24500 samples, validate on 500 samples
Epoch 1/10
24500/24500 [==============================] - 51s 2ms/step - loss: 0.4495 -
acc: 0.7946 - val_loss: 0.3443 - val_acc: 0.8600
Epoch 2/10
24500/24500 [==============================] - 53s 2ms/step - loss: 0.1559 -
acc: 0.9409 - val_loss: 0.3847 - val_acc: 0.8720
Epoch 3/10
24500/24500 [==============================] - 55s 2ms/step - loss: 0.0419 -
acc: 0.9859 - val_loss: 0.5166 - val_acc: 0.8620
Epoch 4/10
24500/24500 [==============================] - 54s 2ms/step - loss: 0.0125 -
acc: 0.9959 - val_loss: 0.7283 - val_acc: 0.8760
Epoch 5/10
24500/24500 [==============================] - 52s 2ms/step - loss: 0.0169 -
acc: 0.9962 - val_loss: 0.9751 - val_acc: 0.8400
Epoch 6/10
24500/24500 [==============================] - 52s 2ms/step - loss: 0.0090 -
acc: 0.9985 - val_loss: 1.1550 - val_acc: 0.8600
Epoch 7/10
24500/24500 [==============================] - 52s 2ms/step - loss: 0.0104 -
acc: 0.9980 - val_loss: 1.2676 - val_acc: 0.8500
Epoch 8/10
24500/24500 [==============================] - 52s 2ms/step - loss: 0.0036 -
acc: 0.9991 - val_loss: 1.3762 - val_acc: 0.8600
Epoch 9/10
24500/24500 [==============================] - 52s 2ms/step - loss: 0.0114 -
acc: 0.9986 - val_loss: 1.5495 - val_acc: 0.8500
Epoch 10/10
24500/24500 [==============================] - 52s 2ms/step - loss: 0.0016 -
acc: 0.9996 - val_loss: 1.7535 - val_acc: 0.8280
```

Evaluating the Model

In [26]:
```
X_test = pad_sequences(sequences, maxlen=max_len)
Y_test = np.asarray(labels)
```

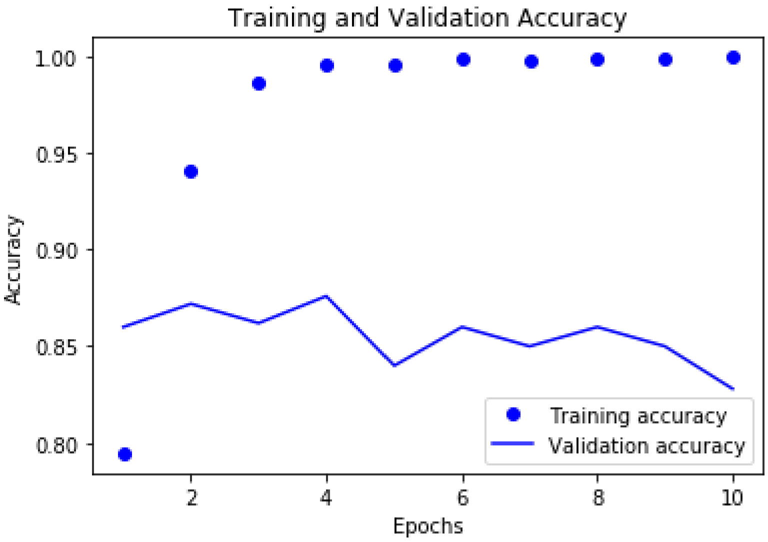
In [27]:
```
model.evaluate(X_test,Y_test)
```

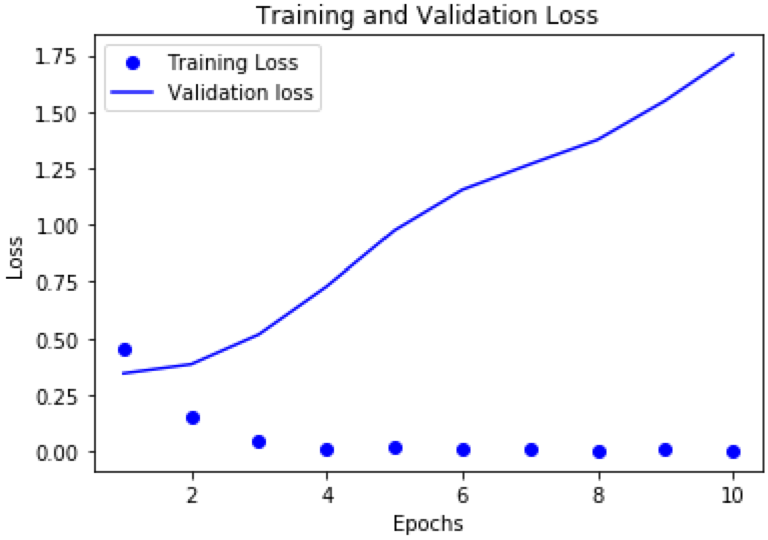
```
25000/25000 [==============================] - 7s 279us/step
```

Out[27]:
```
[13.71816395401001, 0.4968000054359436]
```

In [28]:

```
history_dict = history.history

acc = history_dict['acc']
val_acc = history_dict['val_acc']
loss_values = history_dict['loss']
val_loss_values = history_dict['val_loss']
epochs = range(1,len(acc) + 1)

# Plotting metrics
plt.plot(epochs, acc,  'bo', label = 'Training accuracy')
plt.plot(epochs, val_acc, 'b', label = 'Validation accuracy')
plt.title('Training and Validation Accuracy')
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.legend()

plt.figure()

plt.plot(epochs, loss_values,  'bo', label = 'Training Loss')
plt.plot(epochs, val_loss_values, 'b', label = 'Validation loss')
plt.title('Training and Validation Loss')
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend()

plt.show()
```


Training and Validation Accuracy


Training and Validation Loss

Assignment 10.3

In [29]:
```
from keras.layers import LSTM
```

In [30]:
```
max_features = 10000
model = Sequential()
model.add(Embedding(max_words,
                    embedding_dim,
                    input_length=max_len))
model.add(LSTM(32))
model.add(Dense(1, activation='sigmoid'))
```

In [31]:
```
model.compile(optimizer='rmsprop',
              loss='binary_crossentropy',
              metrics=['acc'])
history = model.fit(X_train, Y_train,
                    epochs=10,
                    batch_size=128,
                    validation_data=(X_val, Y_val))
```

```
C:\Users\bibek\anaconda3\envs\dsc650\lib\site-packages\tensorflow_core\python
\framework\indexed_slices.py:433: UserWarning: Converting sparse IndexedSlice
s to a dense Tensor of unknown shape. This may consume a large amount of memo
ry.
  "Converting sparse IndexedSlices to a dense Tensor of unknown shape. "

Train on 24500 samples, validate on 500 samples
Epoch 1/10
24500/24500 [==============================] - 495s 20ms/step - loss: 0.4544
- acc: 0.7927 - val_loss: 0.3596 - val_acc: 0.8620
Epoch 2/10
24500/24500 [==============================] - 466s 19ms/step - loss: 0.2797
- acc: 0.8907 - val_loss: 0.4469 - val_acc: 0.8580
Epoch 3/10
24500/24500 [==============================] - 470s 19ms/step - loss: 0.2318
- acc: 0.9129 - val_loss: 0.3020 - val_acc: 0.8900
Epoch 4/10
24500/24500 [==============================] - 449s 18ms/step - loss: 0.2031
- acc: 0.9251 - val_loss: 0.3877 - val_acc: 0.8680
Epoch 5/10
24500/24500 [==============================] - 474s 19ms/step - loss: 0.1830
- acc: 0.9338 - val_loss: 0.3534 - val_acc: 0.8840
Epoch 6/10
24500/24500 [==============================] - 474s 19ms/step - loss: 0.1638
- acc: 0.9401 - val_loss: 0.4127 - val_acc: 0.8740
Epoch 7/10
24500/24500 [==============================] - 505s 21ms/step - loss: 0.1491
- acc: 0.9473 - val_loss: 0.4494 - val_acc: 0.8680
Epoch 8/10
24500/24500 [==============================] - 506s 21ms/step - loss: 0.1447
- acc: 0.9498 - val_loss: 0.4349 - val_acc: 0.8740
Epoch 9/10
24500/24500 [==============================] - 467s 19ms/step - loss: 0.1198
- acc: 0.9573 - val_loss: 0.4533 - val_acc: 0.8640
Epoch 10/10
24500/24500 [==============================] - 514s 21ms/step - loss: 0.1140
- acc: 0.9622 - val_loss: 0.3792 - val_acc: 0.8820
```

Model Evaluation

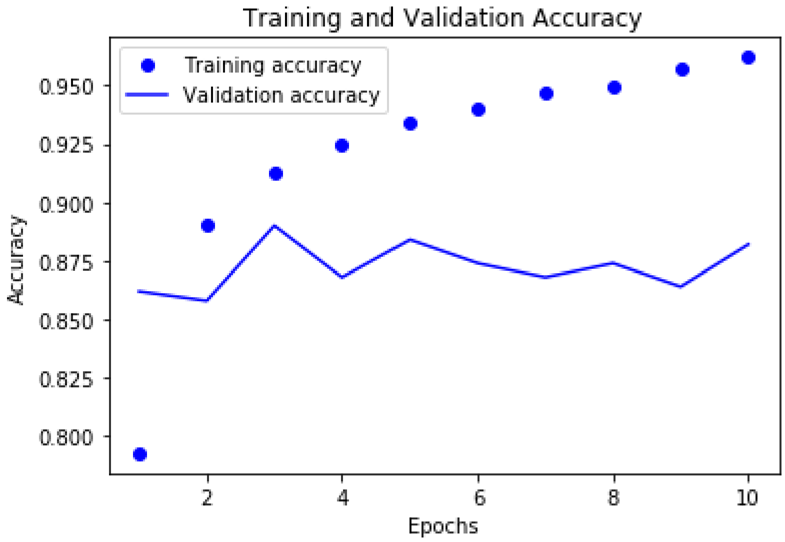
In [32]:
```
X_test = pad_sequences(sequences, maxlen=max_len)
Y_test = np.asarray(labels)

model.evaluate(X_test,Y_test)
```

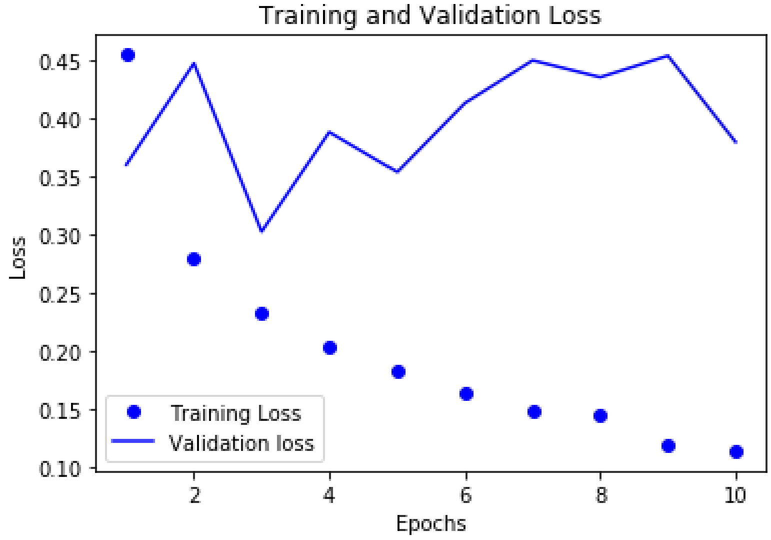
```
25000/25000 [==============================] - 139s 6ms/step
```

Out[32]:
```
[2.317019919424057, 0.49656000733337555]
```

In [33]:

```
history_dict = history.history

acc = history_dict['acc']
val_acc = history_dict['val_acc']
loss_values = history_dict['loss']
val_loss_values = history_dict['val_loss']
epochs = range(1,len(acc) + 1)

# Plotting metrics
plt.plot(epochs, acc,  'bo', label = 'Training accuracy')
plt.plot(epochs, val_acc, 'b', label = 'Validation accuracy')
plt.title('Training and Validation Accuracy')
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.legend()

plt.figure()

plt.plot(epochs, loss_values,  'bo', label = 'Training Loss')
plt.plot(epochs, val_loss_values, 'b', label = 'Validation loss')
plt.title('Training and Validation Loss')
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend()

plt.show()
```





Assignment 10.4

In [34]:
```
from keras.models import Sequential
from keras import layers
from keras.optimizers import RMSprop
```

In [35]:

```
model = Sequential()
model.add(Embedding(max_words,
                    embedding_dim,
                    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, activation='sigmoid'))
model.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
embedding_3 (Embedding)      (None, 2234, 100)         1000000   
_________________________________________________________________
conv1d_1 (Conv1D)            (None, 2228, 32)          22432     
_________________________________________________________________
max_pooling1d_1 (MaxPooling1 (None, 445, 32)           0         
_________________________________________________________________
conv1d_2 (Conv1D)            (None, 439, 32)           7200      
_________________________________________________________________
global_max_pooling1d_1 (Glob (None, 32)                0         
_________________________________________________________________
dense_4 (Dense)              (None, 1)                 33        
=================================================================
Total params: 1,029,665
Trainable params: 1,029,665
Non-trainable params: 0
_________________________________________________________________
```

In [36]:

```
model.compile(optimizer=RMSprop(learning_rate=1e-4),
              loss='binary_crossentropy',
              metrics=['acc'])
history = model.fit(X_train, Y_train,
                    epochs=10,
                    batch_size=128,
                    validation_data=(X_val, Y_val))
```

```
C:\Users\bibek\anaconda3\envs\dsc650\lib\site-packages\tensorflow_core\python
\framework\indexed_slices.py:433: UserWarning: Converting sparse IndexedSlice
s to a dense Tensor of unknown shape. This may consume a large amount of memo
ry.
  "Converting sparse IndexedSlices to a dense Tensor of unknown shape. "

Train on 24500 samples, validate on 500 samples
Epoch 1/10
24500/24500 [==============================] - 92s 4ms/step - loss: 0.6922 -
acc: 0.5345 - val_loss: 0.6905 - val_acc: 0.5860
Epoch 2/10
24500/24500 [==============================] - 92s 4ms/step - loss: 0.6838 -
acc: 0.6572 - val_loss: 0.6776 - val_acc: 0.7220
Epoch 3/10
24500/24500 [==============================] - 96s 4ms/step - loss: 0.6540 -
acc: 0.7644 - val_loss: 0.6225 - val_acc: 0.7580
Epoch 4/10
24500/24500 [==============================] - 91s 4ms/step - loss: 0.5540 -
acc: 0.8018 - val_loss: 0.5019 - val_acc: 0.7980
Epoch 5/10
24500/24500 [==============================] - 95s 4ms/step - loss: 0.4177 -
acc: 0.8419 - val_loss: 0.4100 - val_acc: 0.8180
Epoch 6/10
24500/24500 [==============================] - 92s 4ms/step - loss: 0.3329 -
acc: 0.8698 - val_loss: 0.3870 - val_acc: 0.8320
Epoch 7/10
24500/24500 [==============================] - 95s 4ms/step - loss: 0.2885 -
acc: 0.8853 - val_loss: 0.3501 - val_acc: 0.8620
Epoch 8/10
24500/24500 [==============================] - 94s 4ms/step - loss: 0.2599 -
acc: 0.8979 - val_loss: 0.3345 - val_acc: 0.8600
Epoch 9/10
24500/24500 [==============================] - 96s 4ms/step - loss: 0.2382 -
acc: 0.9073 - val_loss: 0.3457 - val_acc: 0.8620
Epoch 10/10
24500/24500 [==============================] - 100s 4ms/step - loss: 0.2208 -
acc: 0.9152 - val_loss: 0.3218 - val_acc: 0.8700
```

Model Evaluation

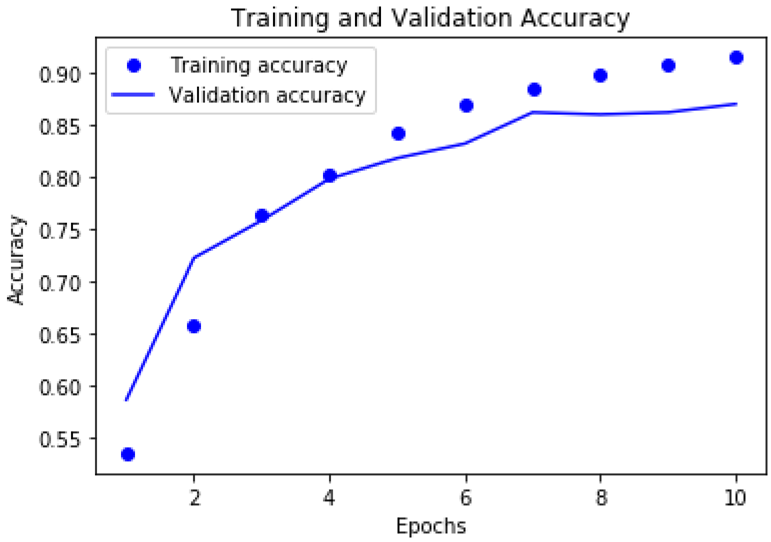
In [37]:
```
X_test = pad_sequences(sequences, maxlen=max_len)
Y_test = np.asarray(labels)

model.evaluate(X_test,Y_test)
```

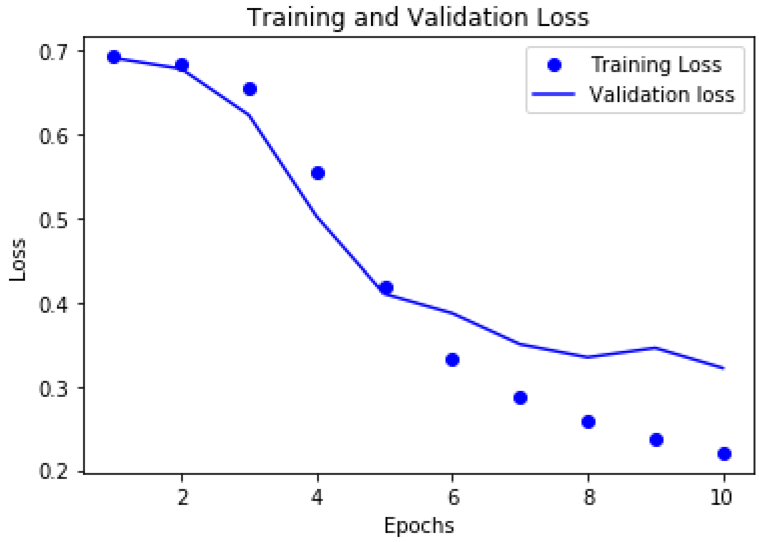
```
25000/25000 [==============================] - 19s 746us/step
```

Out[37]:
```
[1.7728079881000518, 0.49623998899959564]
```

In [38]:

```
history_dict = history.history

acc = history_dict['acc']
val_acc = history_dict['val_acc']
loss_values = history_dict['loss']
val_loss_values = history_dict['val_loss']
epochs = range(1,len(acc) + 1)

# Plotting metrics
plt.plot(epochs, acc,  'bo', label = 'Training accuracy')
plt.plot(epochs, val_acc, 'b', label = 'Validation accuracy')
plt.title('Training and Validation Accuracy')
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.legend()

plt.figure()

plt.plot(epochs, loss_values,  'bo', label = 'Training Loss')
plt.plot(epochs, val_loss_values, 'b', label = 'Validation loss')
plt.title('Training and Validation Loss')
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend()

plt.show()
```


Training and Validation Accuracy
● Training accuracy
Validation accuracy
Accuracy
0.90
0.85
0.80
0.75
0.70
0.65
0.60
0.55
2
4
6
8
10
Epochs


Training and Validation Loss
● Training Loss
Validation loss
Loss
0.7
0.6
0.5
0.4
0.3
0.2
2
4
6
8
10
Epochs

```
In [ ]:
```