DSC650 Week 10 M ERSEVIM

10.1a Create a tokenize function that splits a sentence into words. Ensure that your tokenizer removes basic punctuation.

In [5]:
```
import string

class Vectorizer:
    def standardize(self, text):
        text = text.lower()
        return "".join(char for char in text if char not in string.punctuation)

    def tokenize(self, text):
        text = self.standardize(text)
        return text.split()

vectorizer = Vectorizer()

text = "This is a test sentence to put theough my python code"

vectorizer.tokenize(text)

result = tokenize(text)
print(result)
```

```
['this', 'is', 'a', 'test', 'sentence', 'to', 'put', 'theough', 'my', 'python', 'code']
```

In [ ]:

In [11]:
```
#### Assignment 10.1.b

# Implement an `ngram` function that splits tokens into N-grams.
# Just change the '3' to whatever you N to be...

from nltk import word_tokenize
from nltk.util import ngrams

text = ["This is a test sentence to put through my python code"]

for line in text:
    token = word_tokenize(line)
    ngram = list(ngrams(token, 3))

print(ngram)
```

```
[('This', 'is', 'a'), ('is', 'a', 'test'), ('a', 'test', 'sentence'), ('test', 'sentence', 'to'), ('sentence', 'to', 'put'),
('to', 'put', 'through'), ('put', 'through', 'my'), ('through', 'my', 'python'), ('my', 'python', 'code')]
```

In [ ]:

In [12]:
```
import numpy as np
samples = ["This is a test sentence to put through my python code"]
token_index = {}
for sample in samples:
    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(samples),
    max_length,
    max(token_index.values()) + 1))
for i, sample in enumerate(samples):
    for j, word in list(enumerate(sample.split()))[:max_length]:
        index = token_index.get(word)
        results[i, j, index] = 1

print(results)
```

```
[[[0. 1. 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. 1. 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. 1. 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. 1. 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. 1. 0. 0.]
  [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0.]]]
```

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.

In [25]:
```
from keras.layers import Embedding
#embedding_layer = Embedding(1000, 64)

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

In [28]:
```
from keras.datasets import imdb
from keras import preprocessing
from keras_preprocessing.sequence import pad_sequences
#from keras.utils import pad_sequences
max_features = 10000
maxlen = 20
(x_train, y_train), (x_test, y_test) = imdb.load_data(num_words=max_features)
x_train = preprocessing.sequence.pad_sequences(x_train, maxlen=maxlen)
x_test = preprocessing.sequence.pad_sequences(x_test, maxlen=maxlen)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_126872/2763516164.py in <module>
      6 maxlen = 20
      7 (x_train, y_train), (x_test, y_test) = imdb.load_data(num_words=max_features)
----> 8 x_train = preprocessing_sequence.pad_sequences(x_train, maxlen=maxlen)
      9 x_test = preprocessing_sequence.pad_sequences(x_test, maxlen=maxlen)

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

In [ ]:

In [29]:
```
import os
imdb_dir = 'C:\\Users\\Kate\\Documents\\GitHub\\dsc650\\data\\external\\imdb\\aclImdb'
train_dir = os.path.join(imdb_dir, 'train')
labels = []
texts = []
```

In [30]:
```
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
print('Found %s unique tokens.' % len(word_index))
data = pad_sequences(sequences, maxlen=maxlen)
labels = np.asarray(labels)
print('Shape of data tensor:', data.shape)
print('Shape of label tensor:', labels.shape)
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]
```

```
Found 0 unique tokens.
Shape of data tensor: (0, 100)
Shape of label tensor: (0,)
```

In [31]:
```
word_index = tokenizer.word_index
embedding_dim = 100
embedding_matrix = np.zeros((max_words, embedding_dim))
for word, i in word_index.items():
    if i < max_words:
        embedding_vector = embeddings_index.get(word)
        if embedding_vector is not None:
            embedding_matrix[i] = embedding_vector
```

```
In [32]: 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_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 embedding_2 (Embedding)     (None, 100, 100)          1000000   
                                                                 
 flatten_2 (Flatten)         (None, 10000)             0         
                                                                 
 dense_4 (Dense)             (None, 32)                320032    
                                                                 
 dense_5 (Dense)             (None, 1)                 33        
                                                                 
=================================================================
Total params: 1,320,065
Trainable params: 1,320,065
Non-trainable params: 0
_________________________________________________________________
Epoch 1/10

---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_126872/1713958331.py in <module>
     12     metrics=['acc'])
     13 
---> 14 history = model.fit(x_train, y_train,
     15     epochs=10,
     16     batch_size=32,

C:\ProgramData\Anaconda3\lib\site-packages\keras\utils\traceback_utils.py in error_handler(*args, **kwargs)
     65     except Exception as e:  # pylint: disable=broad-except
     66       filtered_tb = _process_traceback_frames(e.__traceback__)
---> 67       raise e.with_traceback(filtered_tb) from None
     68     finally:
     69       del filtered_tb

C:\ProgramData\Anaconda3\lib\site-packages\keras\engine\training.py in fit(self, x, y, batch_size, epochs, verbose, callbacks, validation_split, validation_data, shuffle, class_weight, sample_weight, initial_epoch, steps_per_epoch, validation_steps, validation_batch_size, validation_freq, max_queue_size, workers, use_multiprocessing)
   1418         logs = tf_utils.sync_to_numpy_or_python_type(logs)
   1419         if logs is None:
-> 1420           raise ValueError('Unexpected result of `train_function` '
   1421                            '(Empty logs). Please use '
   1422                            '`Model.compile(..., run_eagerly=True)`, or '

ValueError: Unexpected result of `train_function` (Empty logs). Please use `Model.compile(..., run_eagerly=True)`, or `tf.config.run_functions_eagerly(True)` for more information of where went wrong, or file a issue/bug to `tf.keras`.
```

In [33]:
```
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()
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_126872/16981932.py in <module>
      1 import matplotlib.pyplot as plt
----> 2 acc = history.history['acc']
      3 val_acc = history.history['val_acc']
      4 loss = history.history['loss']
      5 val_loss = history.history['val_loss']

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

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.

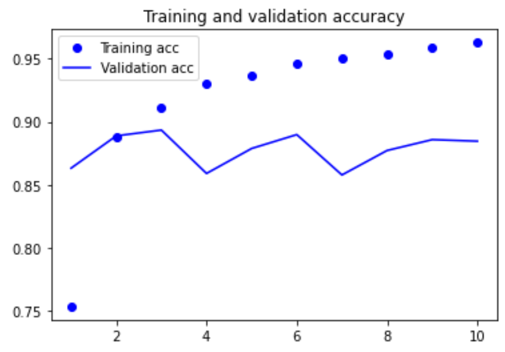
In [39]:
```
from keras.datasets import imdb
from keras_preprocessing import sequence
max_features = 10000
maxlen = 500
batch_size = 32
print('Loading data...')
(input_train, y_train), (input_test, y_test) = imdb.load_data(
num_words=max_features)
print(len(input_train), 'train sequences')
print(len(input_test), '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)
```

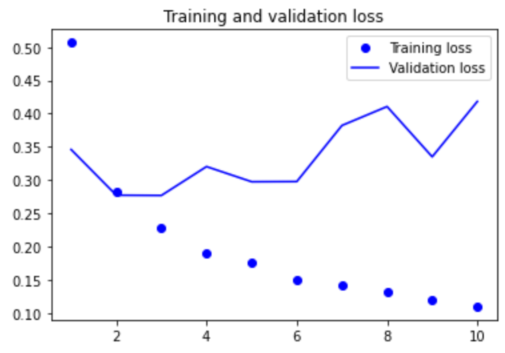
```
Loading data...
25000 train sequences
25000 test sequences
Pad sequences (samples x time)
input_train shape: (25000, 500)
input_test shape: (25000, 500)
```

In [40]:
```
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 [==============================] - 88s 533ms/step - loss: 0.5070 - acc: 0.7535 - val_loss: 0.3459 - val_acc: 0.8632
Epoch 2/10
157/157 [==============================] - 82s 523ms/step - loss: 0.2827 - acc: 0.8884 - val_loss: 0.2772 - val_acc: 0.8888
Epoch 3/10
157/157 [==============================] - 82s 524ms/step - loss: 0.2288 - acc: 0.9115 - val_loss: 0.2768 - val_acc: 0.8934
Epoch 4/10
157/157 [==============================] - 82s 523ms/step - loss: 0.1895 - acc: 0.9301 - val_loss: 0.3202 - val_acc: 0.8590
Epoch 5/10
157/157 [==============================] - 82s 522ms/step - loss: 0.1761 - acc: 0.9363 - val_loss: 0.2974 - val_acc: 0.8788
Epoch 6/10
157/157 [==============================] - 82s 523ms/step - loss: 0.1489 - acc: 0.9456 - val_loss: 0.2977 - val_acc: 0.8898
Epoch 7/10
157/157 [==============================] - 467s 3s/step - loss: 0.1417 - acc: 0.9506 - val_loss: 0.3821 - val_acc: 0.8578
Epoch 8/10
157/157 [==============================] - 81s 516ms/step - loss: 0.1324 - acc: 0.9535 - val_loss: 0.4106 - val_acc: 0.8772
Epoch 9/10
157/157 [==============================] - 34s 213ms/step - loss: 0.1186 - acc: 0.9592 - val_loss: 0.3350 - val_acc: 0.8858
Epoch 10/10
157/157 [==============================] - 30s 193ms/step - loss: 0.1096 - acc: 0.9628 - val_loss: 0.4183 - val_acc: 0.8846
```

In [41]:
```
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()
```





In [ ]:

```
```

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.

In [36]:

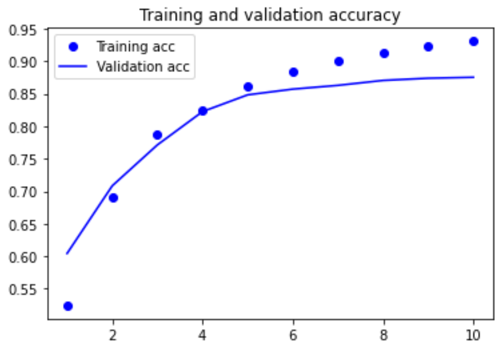
```
from keras.datasets import imdb
from keras_preprocessing import sequence
max_features = 10000
max_len = 500
print('Loading data...')
(x_train, y_train), (x_test, y_test) = imdb.load_data(num_words=max_features)
print(len(x_train), 'train sequences')
print(len(x_test), 'test sequences')
print('Pad 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)
print('x_test shape:', x_test.shape)
```

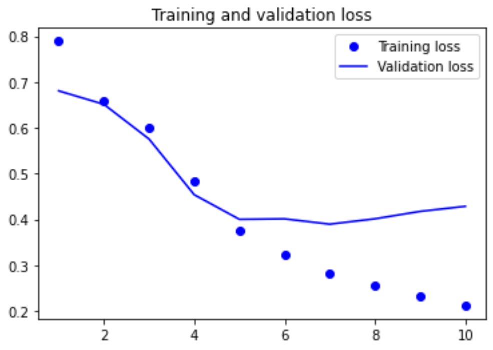
```
Loading data...
25000 train sequences
25000 test sequences
Pad sequences (samples x time)
x_train shape: (25000, 500)
x_test shape: (25000, 500)
```

In [37]:
```python
from keras.models import Sequential
from keras import layers
from 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_5"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_3 (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_6 (Dense)             (None, 1)                 33

=================================================================
Total params: 1,315,937
Trainable params: 1,315,937
Non-trainable params: 0
_________________________________________________________________
Epoch 1/10

C:\ProgramData\Anaconda3\lib\site-packages\keras\optimizers\optimizer_v2\rmsprop.py:135: UserWarning: The `lr` argument is depr
ecated, use `learning_rate` instead.
  super(RMSprop, self).__init__(name, **kwargs)

157/157 [==============================] - 38s 232ms/step - loss: 0.7892 - acc: 0.5243 - val_loss: 0.6806 - val_acc: 0.6044
Epoch 2/10
157/157 [==============================] - 36s 229ms/step - loss: 0.6598 - acc: 0.6909 - val_loss: 0.6513 - val_acc: 0.7084
Epoch 3/10
157/157 [==============================] - 36s 230ms/step - loss: 0.6014 - acc: 0.7869 - val_loss: 0.5755 - val_acc: 0.7714
Epoch 4/10
157/157 [==============================] - 36s 230ms/step - loss: 0.4837 - acc: 0.8240 - val_loss: 0.4539 - val_acc: 0.8226
Epoch 5/10
157/157 [==============================] - 36s 228ms/step - loss: 0.3758 - acc: 0.8615 - val_loss: 0.4002 - val_acc: 0.8484
Epoch 6/10
157/157 [==============================] - 36s 226ms/step - loss: 0.3220 - acc: 0.8840 - val_loss: 0.4012 - val_acc: 0.8572
Epoch 7/10
157/157 [==============================] - 35s 226ms/step - loss: 0.2828 - acc: 0.9018 - val_loss: 0.3896 - val_acc: 0.8630
Epoch 8/10
157/157 [==============================] - 36s 227ms/step - loss: 0.2553 - acc: 0.9136 - val_loss: 0.4013 - val_acc: 0.8706
Epoch 9/10
157/157 [==============================] - 36s 226ms/step - loss: 0.2338 - acc: 0.9228 - val_loss: 0.4175 - val_acc: 0.8740
Epoch 10/10
157/157 [==============================] - 35s 226ms/step - loss: 0.2122 - acc: 0.9311 - val_loss: 0.4286 - val_acc: 0.8754
```

In [38]:
```
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()
```





In [ ]: