Implementing Text-Preprocessing Functions in Python

May 19, 2022

Megan Holford

DSC 650

Spring 2022

For the first section of this exercise, 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.

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

[1]:
```
import keras
import pandas as pd
```

[2]:
```
def tokenize(sentence):
    tokens = keras.preprocessing.text.text_to_word_sequence(sentence)
    return(tokens)
```

[3]:
```
tokens = tokenize("When I was a young boy my father took me into the city to␣
↪see a marching band.")
tokens
```

[3]:
```
['when',
 'i',
 'was',
 'a',
 'young',
 'boy',
 'my',
 'father',
 'took',
 'me',
 'into',
```

```
 'the',
 'city',
 'to',
 'see',
 'a',
 'marching',
 'band']
```

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

[4]:
```
def ngram(tokens, n):
    ngrams = []
    for i in range(len(tokens)-n+1):
        ngram = ' '.join(word_list for word_list in tokens[i:i+n])
        ngrams.append(ngram)
    return(ngrams)
```

[5]:
```
ngram = ngram(tokens,4)
ngram
```

[5]:
```
['when i was a',
 'i was a young',
 'was a young boy',
 'a young boy my',
 'young boy my father',
 'boy my father took',
 'my father took me',
 'father took me into',
 'took me into the',
 'me into the city',
 'into the city to',
 'the city to see',
 'city to see a',
 'to see a marching',
 'see a marching band']
```

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

[6]:
```
def one_hot_encode(tokens):
    token_index = {}
    for token in tokens:
        if token in token_index:
            token_index[token] += 1
        else:
```

```
            token_index[token] = 1

    return([token_index.values()], token_index.keys())
```

[7]:
```
vals, cols = one_hot_encode(tokens)
df = pd.DataFrame(vals, columns=cols)

df
```

[7]:
```
   when  i  was  a  young  boy  my  father  took  me  into  the  city  to  \
0     1  1    1  2      1    1   1       1     1   1     1    1     1   1

   see  marching  band
0    1         1     1
```

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.

[8]:
```
import os
import numpy as np

imdb_dir = '/home/jovyan/dsc650/data/external/imdb/aclImdb'
train_dir = os.path.join(imdb_dir, 'train')

labels = []
texts = []

for label_type in ['neg', 'pos']:
    dir_name = os.path.join(train_dir, label_type)
    for fname in 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)
```

[9]:
```
max_words = 10000
embedding_dim = 100
maxlen = 100
training_samples = 200
validation_samples = 10000
```

[10]:
```
tokenizer = keras.preprocessing.text.Tokenizer(num_words=max_words)
tokenizer.fit_on_texts(texts)
sequences = tokenizer.texts_to_sequences(texts)

data = keras.preprocessing.sequence.pad_sequences(sequences,maxlen=maxlen)
labels = np.asarray(labels)
```

[11]:
```
indices = np.arange(data.shape[0])
np.random.shuffle(indices)

data = data[indices]
labels = labels[indices]
```

[12]:
```
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]
```

[13]:
```
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"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (None, 100, 100)          1000000
_________________________________________________________________
dense (Dense)                (None, 100, 32)           3232
_________________________________________________________________
dense_1 (Dense)              (None, 100, 1)            33
=================================================================
Total params: 1,003,265
```

```
Trainable params: 1,003,265
Non-trainable params: 0
_________________________________________________________________
Epoch 1/10
7/7 [==============================] - 1s 199ms/step - loss: 0.6911 - acc:
0.5317 - val_loss: 0.6923 - val_acc: 0.5169
Epoch 2/10
7/7 [==============================] - 1s 150ms/step - loss: 0.6819 - acc:
0.6249 - val_loss: 0.6921 - val_acc: 0.5200
Epoch 3/10
7/7 [==============================] - 1s 149ms/step - loss: 0.6741 - acc:
0.6458 - val_loss: 0.6932 - val_acc: 0.5180
Epoch 4/10
7/7 [==============================] - 1s 146ms/step - loss: 0.6661 - acc:
0.6522 - val_loss: 0.6939 - val_acc: 0.5198
Epoch 5/10
7/7 [==============================] - 1s 151ms/step - loss: 0.6573 - acc:
0.6612 - val_loss: 0.6960 - val_acc: 0.5204
Epoch 6/10
7/7 [==============================] - 1s 152ms/step - loss: 0.6475 - acc:
0.6643 - val_loss: 0.6980 - val_acc: 0.5231
Epoch 7/10
7/7 [==============================] - 1s 147ms/step - loss: 0.6374 - acc:
0.6652 - val_loss: 0.7003 - val_acc: 0.5241
Epoch 8/10
7/7 [==============================] - 1s 152ms/step - loss: 0.6282 - acc:
0.6668 - val_loss: 0.7033 - val_acc: 0.5276
Epoch 9/10
7/7 [==============================] - 1s 148ms/step - loss: 0.6194 - acc:
0.6650 - val_loss: 0.7083 - val_acc: 0.5259
Epoch 10/10
7/7 [==============================] - 1s 145ms/step - loss: 0.6109 - acc:
0.6668 - val_loss: 0.7120 - val_acc: 0.5254
```

[14]:
```python
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)
```

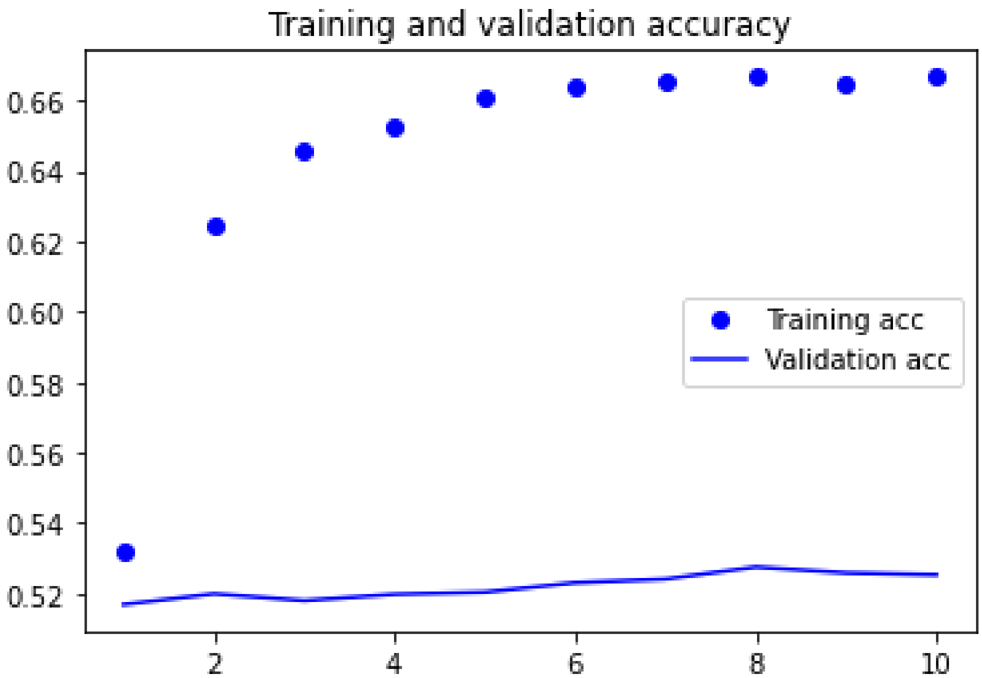
[15]:
```
sequences = tokenizer.texts_to_sequences(texts)
x_test = keras.preprocessing.sequence.pad_sequences(sequences, maxlen=maxlen)
y_test = np.asarray(labels)
```

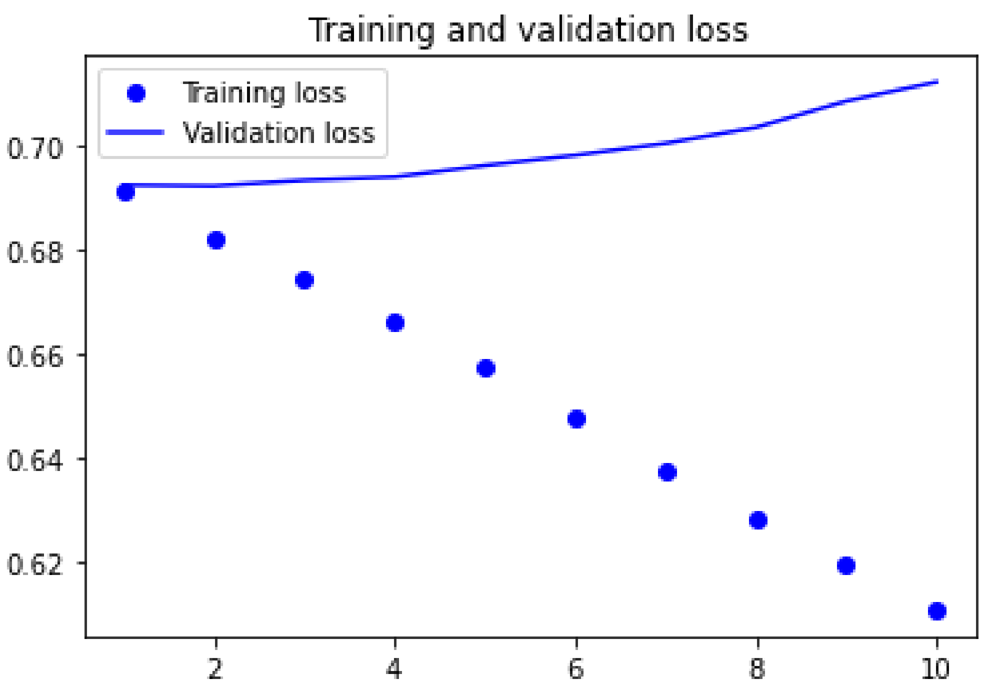
[16]:
```
model.evaluate(x_test,y_test)
```

```
782/782 [==============================] - 2s 3ms/step - loss: 0.7117 - acc:
0.5261
```

[16]: [0.711679995059967, 0.5261073708534241]

[17]:
```
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()
```





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.

[18]:
```
from keras.layers import LSTM

model = Sequential()
model.add(Embedding(max_words, embedding_dim, input_length=maxlen))
model.add(LSTM(32))
model.add(Dense(1, activation='sigmoid'))

model.compile(optimizer='rmsprop',
              loss='binary_crossentropy',
              metrics=['acc'])
history = model.fit(x_train, y_train,
                    epochs=10,
                    batch_size=128,
                    validation_split=0.2)
```

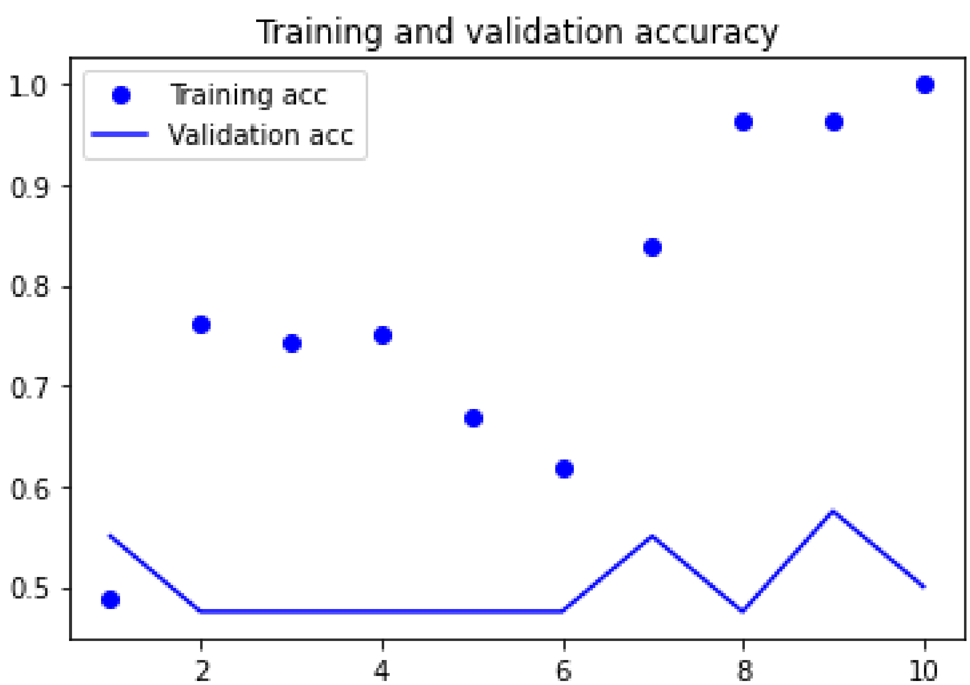
```
Epoch 1/10
2/2 [==============================] - 0s 225ms/step - loss: 0.6929 - acc:
0.4875 - val_loss: 0.6914 - val_acc: 0.5500
Epoch 2/10
2/2 [==============================] - 0s 80ms/step - loss: 0.6762 - acc: 0.7625
- val_loss: 0.6910 - val_acc: 0.4750
Epoch 3/10
2/2 [==============================] - 0s 74ms/step - loss: 0.6562 - acc: 0.7437
- val_loss: 0.6895 - val_acc: 0.4750
Epoch 4/10
2/2 [==============================] - 0s 75ms/step - loss: 0.6262 - acc: 0.7500
- val_loss: 0.6949 - val_acc: 0.4750
Epoch 5/10
2/2 [==============================] - 0s 80ms/step - loss: 0.5780 - acc: 0.6687
- val_loss: 0.7370 - val_acc: 0.4750
Epoch 6/10
2/2 [==============================] - 0s 83ms/step - loss: 0.5062 - acc: 0.6187
- val_loss: 0.6888 - val_acc: 0.4750
Epoch 7/10
2/2 [==============================] - 0s 79ms/step - loss: 0.4416 - acc: 0.8375
- val_loss: 0.6684 - val_acc: 0.5500
Epoch 8/10
2/2 [==============================] - 0s 71ms/step - loss: 0.4052 - acc: 0.9625
- val_loss: 0.7618 - val_acc: 0.4750
Epoch 9/10
2/2 [==============================] - 0s 79ms/step - loss: 0.2884 - acc: 0.9625
- val_loss: 0.6424 - val_acc: 0.5750
Epoch 10/10
2/2 [==============================] - 0s 74ms/step - loss: 0.3273 - acc: 1.0000
- val_loss: 0.8227 - val_acc: 0.5000
```

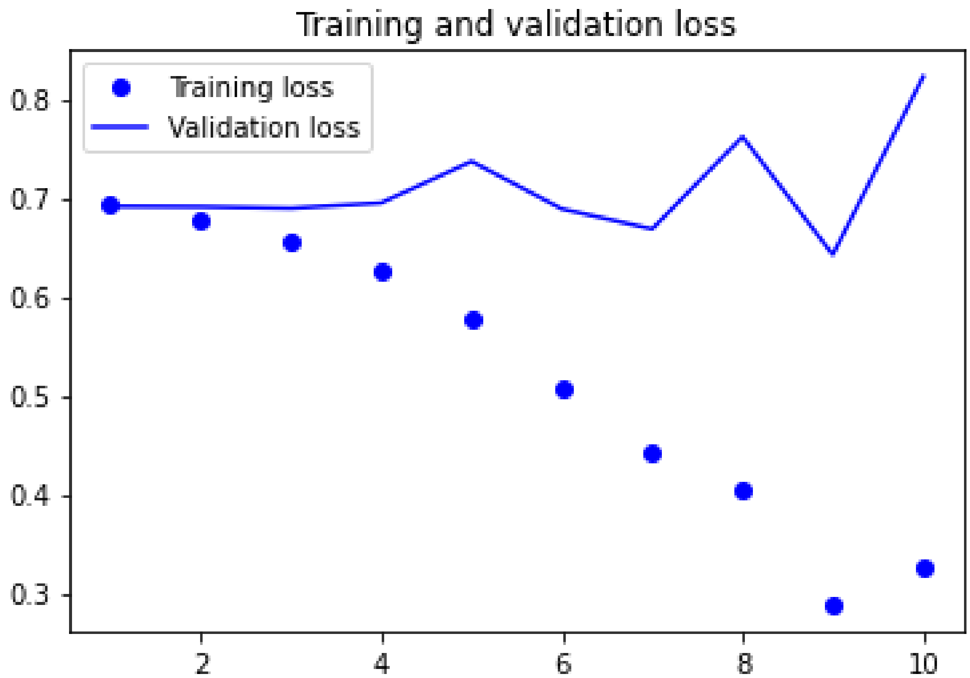
[19]:
```
model.evaluate(x_test,y_test)
```

```
782/782 [==============================] - 15s 19ms/step - loss: 0.8565 - acc:
0.5326
```

[19]: [0.8565438389778137, 0.5326399803161621]

[20]:
```
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()
```





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.

[21]:
```python
from keras.models import Sequential
from keras import layers
from keras.optimizers import RMSprop

max_features = 10000
max_len = 100

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.Flatten())
model.add(layers.Dense(1))

model.summary()
```

```python
model.compile(optimizer=RMSprop(lr=1e-4),
              loss='binary_crossentropy',
              metrics=['acc'])
history = model.fit(x_train, y_train,
                    epochs=100,
                    batch_size=128,
                    validation_split=0.2)
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_2 (Embedding)      (None, 100, 128)          1280000
_________________________________________________________________
conv1d (Conv1D)              (None, 94, 32)            28704
_________________________________________________________________
max_pooling1d (MaxPooling1D) (None, 18, 32)            0
_________________________________________________________________
conv1d_1 (Conv1D)            (None, 12, 32)            7200
_________________________________________________________________
flatten (Flatten)            (None, 384)               0
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 385
=================================================================
Total params: 1,316,289
Trainable params: 1,316,289
Non-trainable params: 0
_________________________________________________________________
Epoch 1/100
2/2 [==============================] - 0s 67ms/step - loss: 2.8999 - acc: 0.4062
- val_loss: 1.7159 - val_acc: 0.5250
Epoch 2/100
2/2 [==============================] - 0s 25ms/step - loss: 1.9735 - acc: 0.4062
- val_loss: 1.5745 - val_acc: 0.5250
Epoch 3/100
2/2 [==============================] - 0s 34ms/step - loss: 1.7920 - acc: 0.4062
- val_loss: 1.5206 - val_acc: 0.5250
Epoch 4/100
2/2 [==============================] - 0s 25ms/step - loss: 1.7233 - acc: 0.4062
- val_loss: 1.4770 - val_acc: 0.5250
Epoch 5/100
2/2 [==============================] - 0s 24ms/step - loss: 1.6685 - acc: 0.4062
- val_loss: 1.4387 - val_acc: 0.5250
Epoch 6/100
2/2 [==============================] - 0s 25ms/step - loss: 1.5603 - acc: 0.4062
- val_loss: 1.3958 - val_acc: 0.5250
Epoch 7/100
```

```
2/2 [==============================] - 0s 27ms/step - loss: 1.4967 - acc: 0.4062
- val_loss: 1.3625 - val_acc: 0.5250
Epoch 8/100
2/2 [==============================] - 0s 25ms/step - loss: 1.4548 - acc: 0.4062
- val_loss: 1.3281 - val_acc: 0.5250
Epoch 9/100
2/2 [==============================] - 0s 26ms/step - loss: 1.4133 - acc: 0.4062
- val_loss: 1.3005 - val_acc: 0.5250
Epoch 10/100
2/2 [==============================] - 0s 28ms/step - loss: 1.3244 - acc: 0.4062
- val_loss: 1.2547 - val_acc: 0.5250
Epoch 11/100
2/2 [==============================] - 0s 25ms/step - loss: 1.2575 - acc: 0.4062
- val_loss: 1.2336 - val_acc: 0.5250
Epoch 12/100
2/2 [==============================] - 0s 27ms/step - loss: 1.2290 - acc: 0.4062
- val_loss: 1.2103 - val_acc: 0.5250
Epoch 13/100
2/2 [==============================] - 0s 26ms/step - loss: 1.1985 - acc: 0.4062
- val_loss: 1.1829 - val_acc: 0.5250
Epoch 14/100
2/2 [==============================] - 0s 25ms/step - loss: 1.1652 - acc: 0.4062
- val_loss: 1.1563 - val_acc: 0.5250
Epoch 15/100
2/2 [==============================] - 0s 22ms/step - loss: 1.1329 - acc: 0.4062
- val_loss: 1.1312 - val_acc: 0.5250
Epoch 16/100
2/2 [==============================] - 0s 26ms/step - loss: 1.1020 - acc: 0.4062
- val_loss: 1.1062 - val_acc: 0.5250
Epoch 17/100
2/2 [==============================] - 0s 27ms/step - loss: 1.0717 - acc: 0.4062
- val_loss: 1.0833 - val_acc: 0.5250
Epoch 18/100
2/2 [==============================] - 0s 25ms/step - loss: 1.0441 - acc: 0.4062
- val_loss: 1.0625 - val_acc: 0.5250
Epoch 19/100
2/2 [==============================] - 0s 22ms/step - loss: 1.0179 - acc: 0.4062
- val_loss: 1.0402 - val_acc: 0.5250
Epoch 20/100
2/2 [==============================] - 0s 26ms/step - loss: 0.9903 - acc: 0.4062
- val_loss: 1.0169 - val_acc: 0.5250
Epoch 21/100
2/2 [==============================] - 0s 24ms/step - loss: 0.9619 - acc: 0.4062
- val_loss: 0.9954 - val_acc: 0.5250
Epoch 22/100
2/2 [==============================] - 0s 23ms/step - loss: 0.9354 - acc: 0.4062
- val_loss: 0.9769 - val_acc: 0.5250
Epoch 23/100
```

```
2/2 [==============================] - 0s 28ms/step - loss: 0.9108 - acc: 0.4062
- val_loss: 0.9557 - val_acc: 0.5250
Epoch 24/100
2/2 [==============================] - 0s 25ms/step - loss: 0.8841 - acc: 0.4062
- val_loss: 0.9341 - val_acc: 0.5250
Epoch 25/100
2/2 [==============================] - 0s 26ms/step - loss: 0.8570 - acc: 0.4062
- val_loss: 0.9148 - val_acc: 0.5250
Epoch 26/100
2/2 [==============================] - 0s 34ms/step - loss: 0.8318 - acc: 0.4062
- val_loss: 0.8960 - val_acc: 0.5250
Epoch 27/100
2/2 [==============================] - 0s 26ms/step - loss: 0.8075 - acc: 0.4062
- val_loss: 0.8797 - val_acc: 0.5250
Epoch 28/100
2/2 [==============================] - 0s 25ms/step - loss: 0.7852 - acc: 0.4062
- val_loss: 0.8644 - val_acc: 0.5250
Epoch 29/100
2/2 [==============================] - 0s 26ms/step - loss: 0.7632 - acc: 0.4062
- val_loss: 0.8474 - val_acc: 0.5250
Epoch 30/100
2/2 [==============================] - 0s 24ms/step - loss: 0.7398 - acc: 0.4062
- val_loss: 0.8311 - val_acc: 0.5250
Epoch 31/100
2/2 [==============================] - 0s 25ms/step - loss: 0.7179 - acc: 0.4062
- val_loss: 0.8195 - val_acc: 0.5250
Epoch 32/100
2/2 [==============================] - 0s 26ms/step - loss: 0.6987 - acc: 0.4062
- val_loss: 0.8044 - val_acc: 0.5250
Epoch 33/100
2/2 [==============================] - 0s 27ms/step - loss: 0.6764 - acc: 0.4062
- val_loss: 0.7910 - val_acc: 0.5250
Epoch 34/100
2/2 [==============================] - 0s 26ms/step - loss: 0.6559 - acc: 0.4062
- val_loss: 0.7799 - val_acc: 0.5250
Epoch 35/100
2/2 [==============================] - 0s 26ms/step - loss: 0.6366 - acc: 0.4062
- val_loss: 0.7666 - val_acc: 0.5250
Epoch 36/100
2/2 [==============================] - 0s 25ms/step - loss: 0.6166 - acc: 0.4187
- val_loss: 0.7581 - val_acc: 0.5250
Epoch 37/100
2/2 [==============================] - 0s 21ms/step - loss: 0.5993 - acc: 0.4187
- val_loss: 0.7474 - val_acc: 0.5250
Epoch 38/100
2/2 [==============================] - 0s 26ms/step - loss: 0.5805 - acc: 0.4500
- val_loss: 0.7379 - val_acc: 0.5250
Epoch 39/100
```

```
2/2 [==============================] - 0s 25ms/step - loss: 0.5626 - acc: 0.5063
- val_loss: 0.7301 - val_acc: 0.5250
Epoch 40/100
2/2 [==============================] - 0s 25ms/step - loss: 0.5461 - acc: 0.6000
- val_loss: 0.7234 - val_acc: 0.5250
Epoch 41/100
2/2 [==============================] - 0s 23ms/step - loss: 0.5305 - acc: 0.7000
- val_loss: 0.7173 - val_acc: 0.5250
Epoch 42/100
2/2 [==============================] - 0s 29ms/step - loss: 0.5153 - acc: 0.8125
- val_loss: 0.7120 - val_acc: 0.5250
Epoch 43/100
2/2 [==============================] - 0s 26ms/step - loss: 0.5010 - acc: 0.8562
- val_loss: 0.7077 - val_acc: 0.5250
Epoch 44/100
2/2 [==============================] - 0s 26ms/step - loss: 0.4869 - acc: 0.8938
- val_loss: 0.7028 - val_acc: 0.5250
Epoch 45/100
2/2 [==============================] - 0s 25ms/step - loss: 0.4718 - acc: 0.9375
- val_loss: 0.6978 - val_acc: 0.5250
Epoch 46/100
2/2 [==============================] - 0s 23ms/step - loss: 0.4563 - acc: 0.9500
- val_loss: 0.6933 - val_acc: 0.5250
Epoch 47/100
2/2 [==============================] - 0s 25ms/step - loss: 0.4408 - acc: 0.9812
- val_loss: 0.6905 - val_acc: 0.5250
Epoch 48/100
2/2 [==============================] - 0s 26ms/step - loss: 0.4263 - acc: 0.9875
- val_loss: 0.6880 - val_acc: 0.5250
Epoch 49/100
2/2 [==============================] - 0s 26ms/step - loss: 0.4118 - acc: 0.9875
- val_loss: 0.6866 - val_acc: 0.5000
Epoch 50/100
2/2 [==============================] - 0s 24ms/step - loss: 0.3984 - acc: 0.9937
- val_loss: 0.6863 - val_acc: 0.5500
Epoch 51/100
2/2 [==============================] - 0s 25ms/step - loss: 0.3869 - acc: 0.9937
- val_loss: 0.6861 - val_acc: 0.5750
Epoch 52/100
2/2 [==============================] - 0s 29ms/step - loss: 0.3752 - acc: 0.9937
- val_loss: 0.6867 - val_acc: 0.5500
Epoch 53/100
2/2 [==============================] - 0s 28ms/step - loss: 0.3621 - acc: 0.9937
- val_loss: 0.6880 - val_acc: 0.5250
Epoch 54/100
2/2 [==============================] - 0s 25ms/step - loss: 0.3499 - acc: 0.9937
- val_loss: 0.6886 - val_acc: 0.5250
Epoch 55/100
```

```
2/2 [==============================] - 0s 26ms/step - loss: 0.3391 - acc: 0.9937
- val_loss: 0.6895 - val_acc: 0.5500
Epoch 56/100
2/2 [==============================] - 0s 24ms/step - loss: 0.3280 - acc: 1.0000
- val_loss: 0.6927 - val_acc: 0.5750
Epoch 57/100
2/2 [==============================] - 0s 19ms/step - loss: 0.3157 - acc: 1.0000
- val_loss: 0.6953 - val_acc: 0.5500
Epoch 58/100
2/2 [==============================] - 0s 25ms/step - loss: 0.3042 - acc: 1.0000
- val_loss: 0.6974 - val_acc: 0.5250
Epoch 59/100
2/2 [==============================] - 0s 25ms/step - loss: 0.2929 - acc: 1.0000
- val_loss: 0.7015 - val_acc: 0.5000
Epoch 60/100
2/2 [==============================] - 0s 25ms/step - loss: 0.2814 - acc: 1.0000
- val_loss: 0.7034 - val_acc: 0.5000
Epoch 61/100
2/2 [==============================] - 0s 27ms/step - loss: 0.2704 - acc: 1.0000
- val_loss: 0.7103 - val_acc: 0.4750
Epoch 62/100
2/2 [==============================] - 0s 27ms/step - loss: 0.2592 - acc: 0.9937
- val_loss: 0.7189 - val_acc: 0.4750
Epoch 63/100
2/2 [==============================] - 0s 27ms/step - loss: 0.2481 - acc: 0.9812
- val_loss: 0.7262 - val_acc: 0.4750
Epoch 64/100
2/2 [==============================] - 0s 21ms/step - loss: 0.2375 - acc: 0.9812
- val_loss: 0.7207 - val_acc: 0.4750
Epoch 65/100
2/2 [==============================] - 0s 24ms/step - loss: 0.2273 - acc: 0.9812
- val_loss: 0.7213 - val_acc: 0.4750
Epoch 66/100
2/2 [==============================] - 0s 25ms/step - loss: 0.2172 - acc: 0.9875
- val_loss: 0.7262 - val_acc: 0.4750
Epoch 67/100
2/2 [==============================] - 0s 25ms/step - loss: 0.2076 - acc: 0.9875
- val_loss: 0.7187 - val_acc: 0.4750
Epoch 68/100
2/2 [==============================] - 0s 25ms/step - loss: 0.1977 - acc: 0.9875
- val_loss: 0.7124 - val_acc: 0.5000
Epoch 69/100
2/2 [==============================] - 0s 24ms/step - loss: 0.1882 - acc: 0.9875
- val_loss: 0.7113 - val_acc: 0.5000
Epoch 70/100
2/2 [==============================] - 0s 26ms/step - loss: 0.1783 - acc: 0.9875
- val_loss: 0.7166 - val_acc: 0.4750
Epoch 71/100
```

```
2/2 [==============================] - 0s 26ms/step - loss: 0.1694 - acc: 0.9875
- val_loss: 0.7080 - val_acc: 0.5000
Epoch 72/100
2/2 [==============================] - 0s 27ms/step - loss: 0.1598 - acc: 0.9937
- val_loss: 0.7095 - val_acc: 0.5000
Epoch 73/100
2/2 [==============================] - 0s 26ms/step - loss: 0.1507 - acc: 0.9875
- val_loss: 0.7117 - val_acc: 0.5000
Epoch 74/100
2/2 [==============================] - 0s 24ms/step - loss: 0.1422 - acc: 0.9875
- val_loss: 0.7043 - val_acc: 0.5250
Epoch 75/100
2/2 [==============================] - 0s 28ms/step - loss: 0.1335 - acc: 0.9875
- val_loss: 0.7016 - val_acc: 0.5500
Epoch 76/100
2/2 [==============================] - 0s 25ms/step - loss: 0.1240 - acc: 0.9875
- val_loss: 0.6983 - val_acc: 0.6000
Epoch 77/100
2/2 [==============================] - 0s 25ms/step - loss: 0.1145 - acc: 0.9937
- val_loss: 0.7020 - val_acc: 0.5500
Epoch 78/100
2/2 [==============================] - 0s 26ms/step - loss: 0.1058 - acc: 0.9875
- val_loss: 0.6943 - val_acc: 0.6250
Epoch 79/100
2/2 [==============================] - 0s 26ms/step - loss: 0.0994 - acc: 1.0000
- val_loss: 0.6933 - val_acc: 0.5500
Epoch 80/100
2/2 [==============================] - 0s 26ms/step - loss: 0.0897 - acc: 0.9875
- val_loss: 0.6925 - val_acc: 0.5250
Epoch 81/100
2/2 [==============================] - 0s 28ms/step - loss: 0.0815 - acc: 1.0000
- val_loss: 0.6946 - val_acc: 0.5750
Epoch 82/100
2/2 [==============================] - 0s 35ms/step - loss: 0.0752 - acc: 0.9937
- val_loss: 0.6935 - val_acc: 0.5250
Epoch 83/100
2/2 [==============================] - 0s 21ms/step - loss: 0.0681 - acc: 0.9937
- val_loss: 0.6917 - val_acc: 0.5250
Epoch 84/100
2/2 [==============================] - 0s 26ms/step - loss: 0.0618 - acc: 0.9937
- val_loss: 0.6941 - val_acc: 0.5750
Epoch 85/100
2/2 [==============================] - 0s 28ms/step - loss: 0.0575 - acc: 0.9937
- val_loss: 0.6914 - val_acc: 0.5250
Epoch 86/100
2/2 [==============================] - 0s 27ms/step - loss: 0.0508 - acc: 1.0000
- val_loss: 0.6904 - val_acc: 0.5000
Epoch 87/100
```

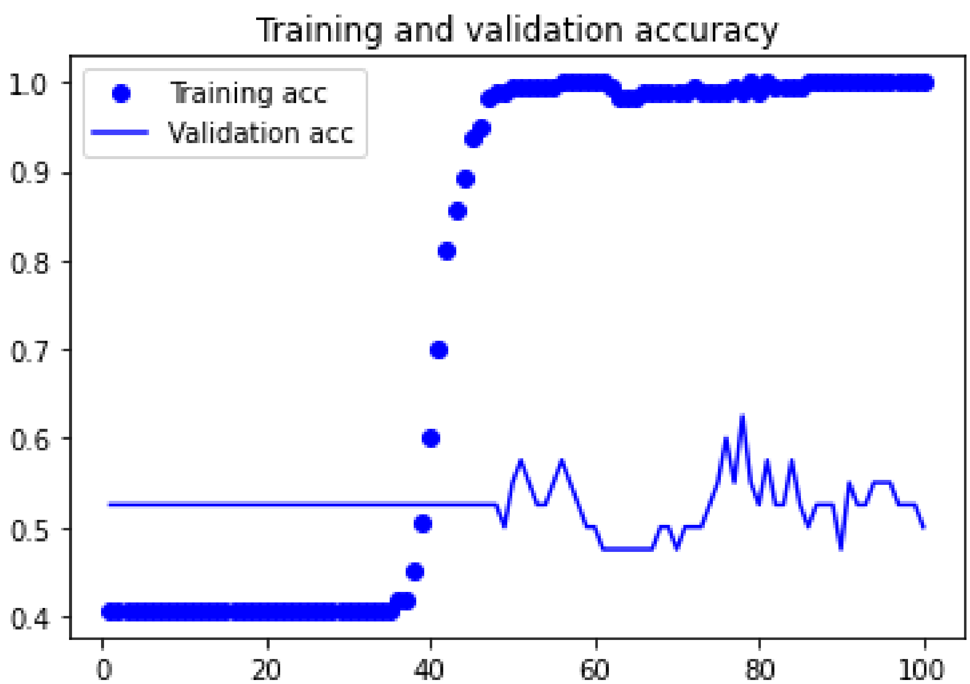
```
2/2 [==============================] - 0s 25ms/step - loss: 0.0459 - acc: 1.0000
- val_loss: 0.6911 - val_acc: 0.5250
Epoch 88/100
2/2 [==============================] - 0s 27ms/step - loss: 0.0411 - acc: 1.0000
- val_loss: 0.6860 - val_acc: 0.5250
Epoch 89/100
2/2 [==============================] - 0s 29ms/step - loss: 0.0359 - acc: 1.0000
- val_loss: 0.6869 - val_acc: 0.5250
Epoch 90/100
2/2 [==============================] - 0s 33ms/step - loss: 0.0302 - acc: 1.0000
- val_loss: 0.6854 - val_acc: 0.4750
Epoch 91/100
2/2 [==============================] - 0s 29ms/step - loss: 0.0272 - acc: 1.0000
- val_loss: 0.6860 - val_acc: 0.5500
Epoch 92/100
2/2 [==============================] - 0s 152ms/step - loss: 0.0237 - acc:
1.0000 - val_loss: 0.6868 - val_acc: 0.5250
Epoch 93/100
2/2 [==============================] - 0s 28ms/step - loss: 0.0209 - acc: 1.0000
- val_loss: 0.6867 - val_acc: 0.5250
Epoch 94/100
2/2 [==============================] - 0s 31ms/step - loss: 0.0184 - acc: 1.0000
- val_loss: 0.6852 - val_acc: 0.5500
Epoch 95/100
2/2 [==============================] - 0s 31ms/step - loss: 0.0162 - acc: 1.0000
- val_loss: 0.6854 - val_acc: 0.5500
Epoch 96/100
2/2 [==============================] - 0s 29ms/step - loss: 0.0147 - acc: 1.0000
- val_loss: 0.6851 - val_acc: 0.5500
Epoch 97/100
2/2 [==============================] - 0s 29ms/step - loss: 0.0131 - acc: 1.0000
- val_loss: 0.6857 - val_acc: 0.5250
Epoch 98/100
2/2 [==============================] - 0s 26ms/step - loss: 0.0124 - acc: 1.0000
- val_loss: 0.6846 - val_acc: 0.5250
Epoch 99/100
2/2 [==============================] - 0s 27ms/step - loss: 0.0107 - acc: 1.0000
- val_loss: 0.6840 - val_acc: 0.5250
Epoch 100/100
2/2 [==============================] - 0s 24ms/step - loss: 0.0099 - acc: 1.0000
- val_loss: 0.6852 - val_acc: 0.5000
```

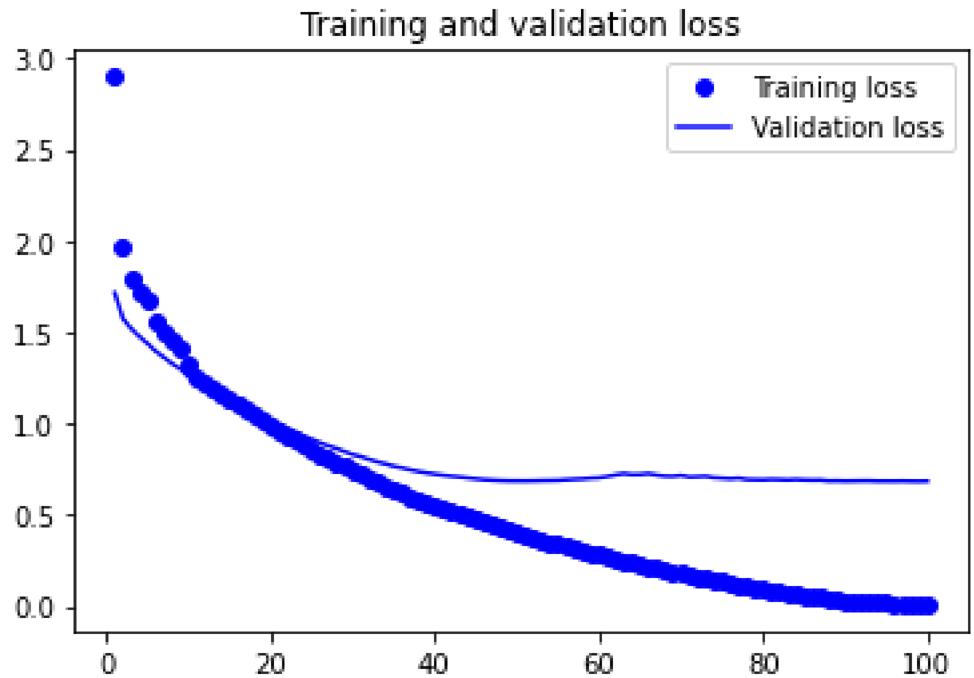
[22]:
```
model.evaluate(x_test,y_test)
```

```
782/782 [==============================] - 3s 3ms/step - loss: 0.6850 - acc:
0.5606
```

[22]: [0.6850090026855469, 0.5605999827384949]

[23]:
```
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()
```




Training and validation loss
Training loss
Validation loss
3.0
2.5
2.0
1.5
1.0
0.5
0.0
0
20
40
60
80
100