Sentiment Analysis using LSTM model

Introduction (Method)

To conduct sentiment analysis using an LSTM model, the Amazon review dataset from Kaggle (available from Kaggle) was used. The dataset has already been adapted into ratings (0: negative, 1: positive). After importing training, validation, and test data, new training datasets of sizes 10%, 25%, and 50% were created to evaluate the impact of data volume. These new training datasets, along with a full training dataset, were applied to the LSTM model.!

Additionally, encoders with 100-word, 250-word, and 500-word vocabularies were created to investigate whether the number of stored frequent words impacts LSTM models.

Firstly, the impact of dataset size on performance was investigated. LSTM models were trained using 10%, 25%, and 50% of the training data. For each model, the following training hyperparameters were used:

- Use 4 LSTM modules
- Use a 100-word encoder
- Train for 15 epochs

Secondly, the effect of the number of LSTM modules on model performance was examined. Models with 8 LSTM units and 12 LSTM units were created and trained. For each model, the following training parameters were utilised:

- Create and use a 250-word encoder
- Use 25% of the training data
- Train for 15 epochs

Thirdly, the investigation focused on how the size of the encoder's vocabulary impacts model performance. Models using encoders with 250-word and 500-word vocabularies were created. For each model, the following training parameters were used:

- Utilise a model with 4 LSTM modules
- Use 25% of the training data
- Train for 15 epochs Finally, the best model was considered based on experiments regarding dataset size, the number of LSTM modules, and the size of the encoder's vocabulary. Hyperparameters were set based on previous experiments.

Importing the required libraries

In [1]:
```
import numpy as np
import tensorflow as tf
import json

import matplotlib.pyplot as plt
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Embedding, LSTM
%matplotlib inline
```

Creating some constants

In [2]:
```
EMBEDDING_LEN = 32  # The length of the word embedding vector
BATCH_SIZE = 1024
```

In [3]:
```
METRICS = [
      lambda : tf.keras.metrics.TruePositives(name='tp'),
      lambda : tf.keras.metrics.FalsePositives(name='fp'),
      lambda : tf.keras.metrics.TrueNegatives(name='tn'),
      lambda : tf.keras.metrics.FalseNegatives(name='fn'),

      lambda : tf.keras.metrics.BinaryAccuracy(name='accuracy'),
      lambda : tf.keras.metrics.Precision(name='precision'),
      lambda : tf.keras.metrics.Recall(name='recall'),
]

def fresh_metrics():
    return [metric() for metric in METRICS]
```

Loading the data: see note about dataset sizes above

In [4]:
```
train_filename = '/datasets/amazon-reviews/train.csv'
val_filename = '/datasets/amazon-reviews/validation.csv'
test_filename = '/datasets/amazon-reviews/test.csv'
```

In [5]:
```
train_data = tf.data.experimental.make_csv_dataset(train_filename, batch_
train_data = train_data.map(lambda d: (d['Review'], d['Rating']))
```

In [6]:
```
# take 10% of the training data
train_data10 = train_data.shard(10, 0)

train_data10 = train_data10.cache()
train_data10 = train_data10.shuffle(50000)
train_data10 = train_data10.prefetch(buffer_size=tf.data.AUTOTUNE)
```

In [7]:
```
# take 25% of the training data
train_data25 = train_data.shard(4, 0)

train_data25 = train_data25.cache()
train_data25 = train_data25.shuffle(50000)
train_data25 = train_data25.prefetch(buffer_size=tf.data.AUTOTUNE)
```

In [8]:
```
# take 50% of the training data
train_data50 = train_data.shard(2, 0)
```

```
train_data50 = train_data50.cache()
train_data50 = train_data50.shuffle(50000)
train_data50 = train_data50.prefetch(buffer_size=tf.data.AUTOTUNE)
```

In [9]:
```
# use all the training data
train_data = train_data.cache()
train_data = train_data.shuffle(50000)
train_data = train_data.prefetch(buffer_size=tf.data.AUTOTUNE)
```

In [10]:
```
validation_data = tf.data.experimental.make_csv_dataset(val_filename, bat
validation_data = validation_data.map(lambda d: (d['Review'], d['Rating']
validation_data = validation_data.cache()
validation_data = validation_data.prefetch(buffer_size=tf.data.AUTOTUNE)
```

In [11]:
```
test_data = tf.data.experimental.make_csv_dataset(test_filename, batch_si
test_data = test_data.map(lambda d: (d['Review'], d['Rating']))
test_data = test_data.cache()
test_data = test_data.prefetch(buffer_size=tf.data.AUTOTUNE)
```

Creating encoders

The following code creates and saves text encoders for different vocabulary sizes. Running this code will take some time, but this saves the files for use later.

After your first run of this notebook, comment out the cells that adapt and save the encoders.

In [12]:
```
!mkdir encoder100
!mkdir encoder250
!mkdir encoder500
```

```
mkdir: cannot create directory ‘encoder100’: File exists
mkdir: cannot create directory ‘encoder250’: File exists
mkdir: cannot create directory ‘encoder500’: File exists
```

Create, adapt, and save an encoder

You will only need to do this once. You may like to comment out the code in these cells after the first run.

In [13]:
```
encoder = tf.keras.layers.TextVectorization(max_tokens=100)
encoder.adapt(train_data.map(lambda text, _label: text))
encoder.save_assets('encoder100')
```

In [14]:
```
encoder = tf.keras.layers.TextVectorization(max_tokens=250)
encoder.adapt(train_data.map(lambda text, _label: text))
encoder.save_assets('encoder250')
```

In [15]:
```
encoder = tf.keras.layers.TextVectorization(max_tokens=500)
encoder.adapt(train_data.map(lambda text, _label: text))
encoder.save_assets('encoder500')
```

Create an encoder and load its assets from file

In [16]:
```
encoder100 = tf.keras.layers.TextVectorization(max_tokens=100)
encoder100.load_assets('encoder100')
```

In [17]:
```
encoder250 = tf.keras.layers.TextVectorization(max_tokens=250)
encoder250.load_assets('encoder250')
```

In [18]:
```
encoder500 = tf.keras.layers.TextVectorization(max_tokens=500)
encoder500.load_assets('encoder500')
```

Trial before LSTM model analysis

The following is an example LSTM model, using a vocabulary size of 100 words, 4 LSTM units, trained over 5 epochs with 10% of the training dataset.

The model and training records are saved as `example_model`

In [30]:
```
# Build a model
example_model = Sequential([
    encoder,
    Embedding(input_dim=len(encoder.get_vocabulary()), output_dim=EMBEDDI
    LSTM(4),
    Dense(1, activation='sigmoid')
])
```

In [31]:
```
# Compile the model
example_model.compile(loss='binary_crossentropy',
              optimizer='adam',
              metrics='accuracy')

example_history = example_model.fit(
    train_data10,
    validation_data=validation_data,
    epochs=5,
    verbose=1)

example_model.save('example_review.keras')
with open('example_review_history.json', 'w') as f:
    json.dump(example_history.history, f)
```

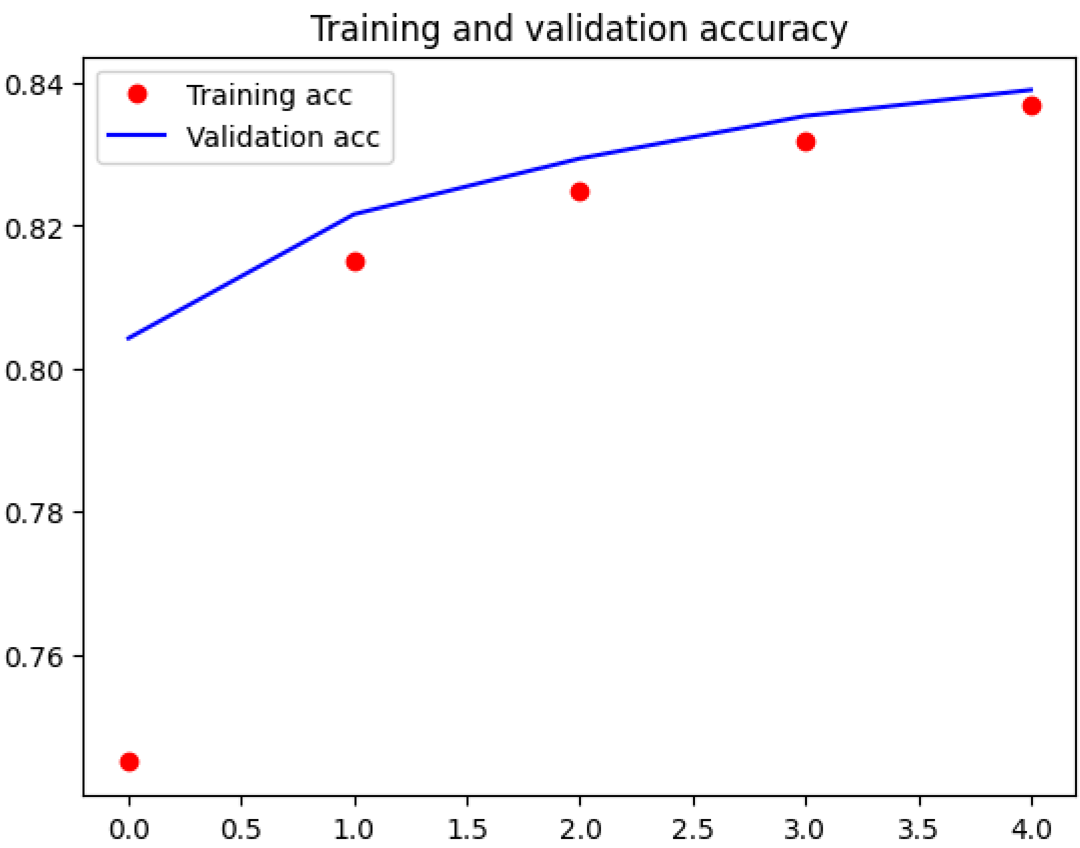
```
Epoch 1/5
352/352 [==============================] - 45s 38ms/step - loss: 0.5363 -
accuracy: 0.7449 - val_loss: 0.4413 - val_accuracy: 0.8042
Epoch 2/5
352/352 [==============================] - 12s 33ms/step - loss: 0.4143 -
accuracy: 0.8150 - val_loss: 0.3979 - val_accuracy: 0.8216
Epoch 3/5
352/352 [==============================] - 12s 34ms/step - loss: 0.3888 -
accuracy: 0.8250 - val_loss: 0.3807 - val_accuracy: 0.8294
Epoch 4/5
352/352 [==============================] - 12s 33ms/step - loss: 0.3755 -
accuracy: 0.8318 - val_loss: 0.3705 - val_accuracy: 0.8354
Epoch 5/5
352/352 [==============================] - 12s 33ms/step - loss: 0.3654 -
accuracy: 0.8370 - val_loss: 0.3649 - val_accuracy: 0.8390
```

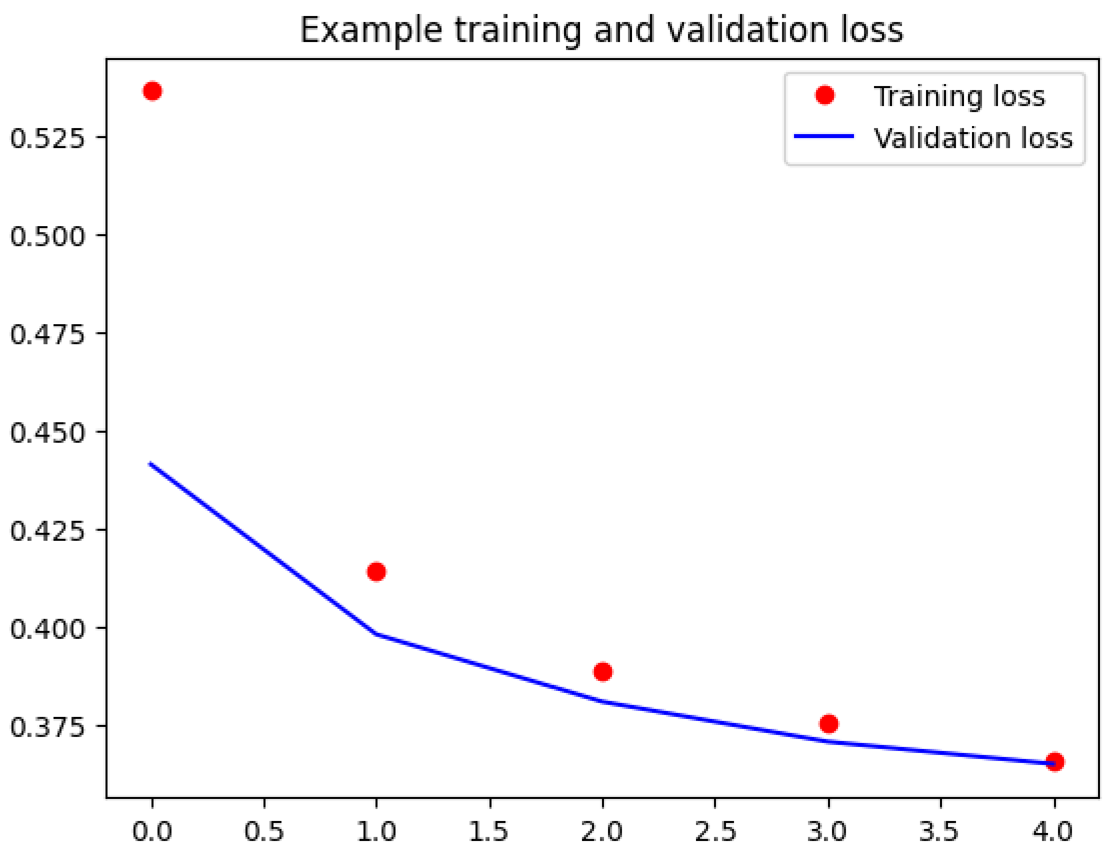
Reload the model and training history

In [32]:
```
example_model = tf.keras.models.load_model('example_review.keras')
with open('example_review_history.json') as f:
    example_history = json.load(f)
```

Plot the training history

In [33]:
```
acc = example_history['accuracy']
val_acc = example_history['val_accuracy']
loss = example_history['loss']
val_loss = example_history['val_loss']

epochs = range(len(acc))

plt.plot(epochs, acc, 'ro', 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, 'ro', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Example training and validation loss')
plt.legend()

plt.show()
```





Evaluate the model

In [34]:
```
example_model.compile(metrics=fresh_metrics())
example_model_results = example_model.evaluate(validation_data, return_di
example_model_results
```

```
79/79 [==============================] - 3s 22ms/step - loss: 0.0000e+00 -
tp: 34235.0000 - fp: 7181.0000 - tn: 32887.0000 - fn: 5697.0000 - accurac
y: 0.8390 - precision: 0.8266 - recall: 0.8573
```

Out[34]:
```
{'loss': 0.0,
 'tp': 34235.0,
 'fp': 7181.0,
 'tn': 32887.0,
 'fn': 5697.0,
 'accuracy': 0.8390250205993652,
 'precision': 0.8266128897666931,
 'recall': 0.8573324680328369}
```

Investigating the size of the dataset

The first investigation is how the number of LSTM modules impacts the performance of the model.

For your models you should:

- use 4 LSTM modules
- use a 100-word encoder
- train for 15 epochs

Training 10% of the dataset

In [35]:
```
# put your code here
# Build the model
model_10 = Sequential([
    encoder100,
    Embedding(input_dim=len(encoder100.get_vocabulary()), output_dim=EMBE
    LSTM(4),
    Dense(1, activation='sigmoid')
])
```

In [36]:
```
# Compile the model
model_10.compile(loss='binary_crossentropy',
              optimizer='adam',
              metrics=fresh_metrics())
```

In [37]:
```
# Train the model
history_10 = model_10.fit(
    train_data10,
    validation_data=validation_data,
    epochs=15,
    verbose=1)
```

```
Epoch 1/15
352/352 [==============================] - 19s 40ms/step - loss: 0.6065 -
tp: 114260.0000 - fp: 51616.0000 - tn: 129262.0000 - fn: 65310.0000 - accu
racy: 0.6756 - precision: 0.6888 - recall: 0.6363 - val_loss: 0.5593 - val
_tp: 28031.0000 - val_fp: 10960.0000 - val_tn: 29108.0000 - val_fn: 11901.
0000 - val_accuracy: 0.7142 - val_precision: 0.7189 - val_recall: 0.7020
Epoch 2/15
352/352 [==============================] - 12s 35ms/step - loss: 0.5524 -
tp: 126805.0000 - fp: 49248.0000 - tn: 131630.0000 - fn: 52765.0000 - accu
racy: 0.7170 - precision: 0.7203 - recall: 0.7062 - val_loss: 0.5448 - val
_tp: 28826.0000 - val_fp: 11241.0000 - val_tn: 28827.0000 - val_fn: 11106.
0000 - val_accuracy: 0.7207 - val_precision: 0.7194 - val_recall: 0.7219
Epoch 3/15
352/352 [==============================] - 12s 34ms/step - loss: 0.5442 -
tp: 125441.0000 - fp: 46733.0000 - tn: 134145.0000 - fn: 54129.0000 - accu
racy: 0.7202 - precision: 0.7286 - recall: 0.6986 - val_loss: 0.5410 - val
_tp: 29407.0000 - val_fp: 11677.0000 - val_tn: 28391.0000 - val_fn: 10525.
0000 - val_accuracy: 0.7225 - val_precision: 0.7158 - val_recall: 0.7364
Epoch 4/15
352/352 [==============================] - 12s 34ms/step - loss: 0.5397 -
tp: 125481.0000 - fp: 45735.0000 - tn: 135143.0000 - fn: 54089.0000 - accu
racy: 0.7231 - precision: 0.7329 - recall: 0.6988 - val_loss: 0.5361 - val
_tp: 28290.0000 - val_fp: 10230.0000 - val_tn: 29838.0000 - val_fn: 11642.
0000 - val_accuracy: 0.7266 - val_precision: 0.7344 - val_recall: 0.7085
Epoch 5/15
352/352 [==============================] - 12s 34ms/step - loss: 0.5353 -
tp: 126161.0000 - fp: 45156.0000 - tn: 135722.0000 - fn: 53409.0000 - accu
racy: 0.7265 - precision: 0.7364 - recall: 0.7026 - val_loss: 0.5319 - val
_tp: 28343.0000 - val_fp: 9958.0000 - val_tn: 30110.0000 - val_fn: 11589.0
000 - val_accuracy: 0.7307 - val_precision: 0.7400 - val_recall: 0.7098
Epoch 6/15
352/352 [==============================] - 12s 34ms/step - loss: 0.5313 -
tp: 127697.0000 - fp: 45415.0000 - tn: 135463.0000 - fn: 51873.0000 - accu
racy: 0.7301 - precision: 0.7377 - recall: 0.7111 - val_loss: 0.5278 - val
_tp: 29029.0000 - val_fp: 10404.0000 - val_tn: 29664.0000 - val_fn: 10903.
0000 - val_accuracy: 0.7337 - val_precision: 0.7362 - val_recall: 0.7270
Epoch 7/15
352/352 [==============================] - 12s 34ms/step - loss: 0.5268 -
tp: 128979.0000 - fp: 45239.0000 - tn: 135639.0000 - fn: 50591.0000 - accu
racy: 0.7341 - precision: 0.7403 - recall: 0.7183 - val_loss: 0.5221 - val
_tp: 28751.0000 - val_fp: 9736.0000 - val_tn: 30332.0000 - val_fn: 11181.0
000 - val_accuracy: 0.7385 - val_precision: 0.7470 - val_recall: 0.7200
Epoch 8/15
352/352 [==============================] - 12s 34ms/step - loss: 0.5220 -
tp: 130360.0000 - fp: 45080.0000 - tn: 135798.0000 - fn: 49210.0000 - accu
racy: 0.7384 - precision: 0.7430 - recall: 0.7260 - val_loss: 0.5209 - val
_tp: 27080.0000 - val_fp: 7983.0000 - val_tn: 32085.0000 - val_fn: 12852.0
000 - val_accuracy: 0.7396 - val_precision: 0.7723 - val_recall: 0.6782
Epoch 9/15
352/352 [==============================] - 12s 34ms/step - loss: 0.5169 -
tp: 131127.0000 - fp: 44497.0000 - tn: 136381.0000 - fn: 48443.0000 - accu
racy: 0.7422 - precision: 0.7466 - recall: 0.7302 - val_loss: 0.5138 - val
_tp: 29323.0000 - val_fp: 9934.0000 - val_tn: 30134.0000 - val_fn: 10609.0
000 - val_accuracy: 0.7432 - val_precision: 0.7469 - val_recall: 0.7343
Epoch 10/15
352/352 [==============================] - 12s 34ms/step - loss: 0.5127 -
tp: 131543.0000 - fp: 43951.0000 - tn: 136927.0000 - fn: 48027.0000 - accu
racy: 0.7448 - precision: 0.7496 - recall: 0.7325 - val_loss: 0.5149 - val
_tp: 26866.0000 - val_fp: 7494.0000 - val_tn: 32574.0000 - val_fn: 13066.0
000 - val_accuracy: 0.7430 - val_precision: 0.7819 - val_recall: 0.6728
```

```
Epoch 11/15
352/352 [==============================] - 12s 34ms/step - loss: 0.5098 -
tp: 131870.0000 - fp: 43550.0000 - tn: 137328.0000 - fn: 47700.0000 - accu
racy: 0.7468 - precision: 0.7517 - recall: 0.7344 - val_loss: 0.5109 - val
_tp: 27512.0000 - val_fp: 7898.0000 - val_tn: 32170.0000 - val_fn: 12420.0
000 - val_accuracy: 0.7460 - val_precision: 0.7770 - val_recall: 0.6890
Epoch 12/15
352/352 [==============================] - 12s 34ms/step - loss: 0.5069 -
tp: 132382.0000 - fp: 43466.0000 - tn: 137412.0000 - fn: 47188.0000 - accu
racy: 0.7485 - precision: 0.7528 - recall: 0.7372 - val_loss: 0.5054 - val
_tp: 29787.0000 - val_fp: 9894.0000 - val_tn: 30174.0000 - val_fn: 10145.0
000 - val_accuracy: 0.7495 - val_precision: 0.7507 - val_recall: 0.7459
Epoch 13/15
352/352 [==============================] - 12s 34ms/step - loss: 0.5045 -
tp: 132757.0000 - fp: 43146.0000 - tn: 137732.0000 - fn: 46813.0000 - accu
racy: 0.7504 - precision: 0.7547 - recall: 0.7393 - val_loss: 0.5035 - val
_tp: 30405.0000 - val_fp: 10421.0000 - val_tn: 29647.0000 - val_fn: 9527.0
000 - val_accuracy: 0.7506 - val_precision: 0.7447 - val_recall: 0.7614
Epoch 14/15
352/352 [==============================] - 12s 34ms/step - loss: 0.5019 -
tp: 133118.0000 - fp: 42826.0000 - tn: 138052.0000 - fn: 46452.0000 - accu
racy: 0.7523 - precision: 0.7566 - recall: 0.7413 - val_loss: 0.5017 - val
_tp: 30392.0000 - val_fp: 10287.0000 - val_tn: 29781.0000 - val_fn: 9540.0
000 - val_accuracy: 0.7522 - val_precision: 0.7471 - val_recall: 0.7611
Epoch 15/15
352/352 [==============================] - 12s 34ms/step - loss: 0.5001 -
tp: 133266.0000 - fp: 42767.0000 - tn: 138111.0000 - fn: 46304.0000 - accu
racy: 0.7529 - precision: 0.7571 - recall: 0.7421 - val_loss: 0.4996 - val
_tp: 30268.0000 - val_fp: 10089.0000 - val_tn: 29979.0000 - val_fn: 9664.0
000 - val_accuracy: 0.7531 - val_precision: 0.7500 - val_recall: 0.7580
```

In [38]:

```python
# Save the model and training history
model_10.save('model_10.keras')
with open('history_10.json', 'w') as f:
    json.dump(history_10.history, f)
```

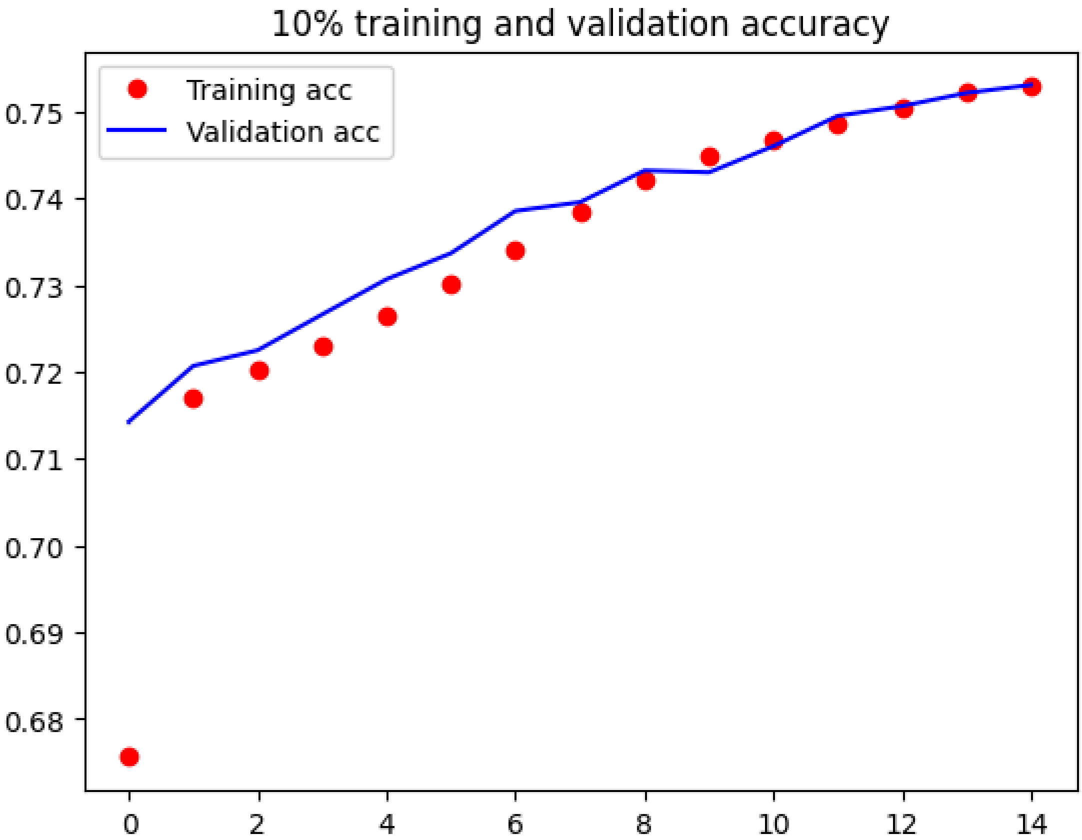
In [34]:

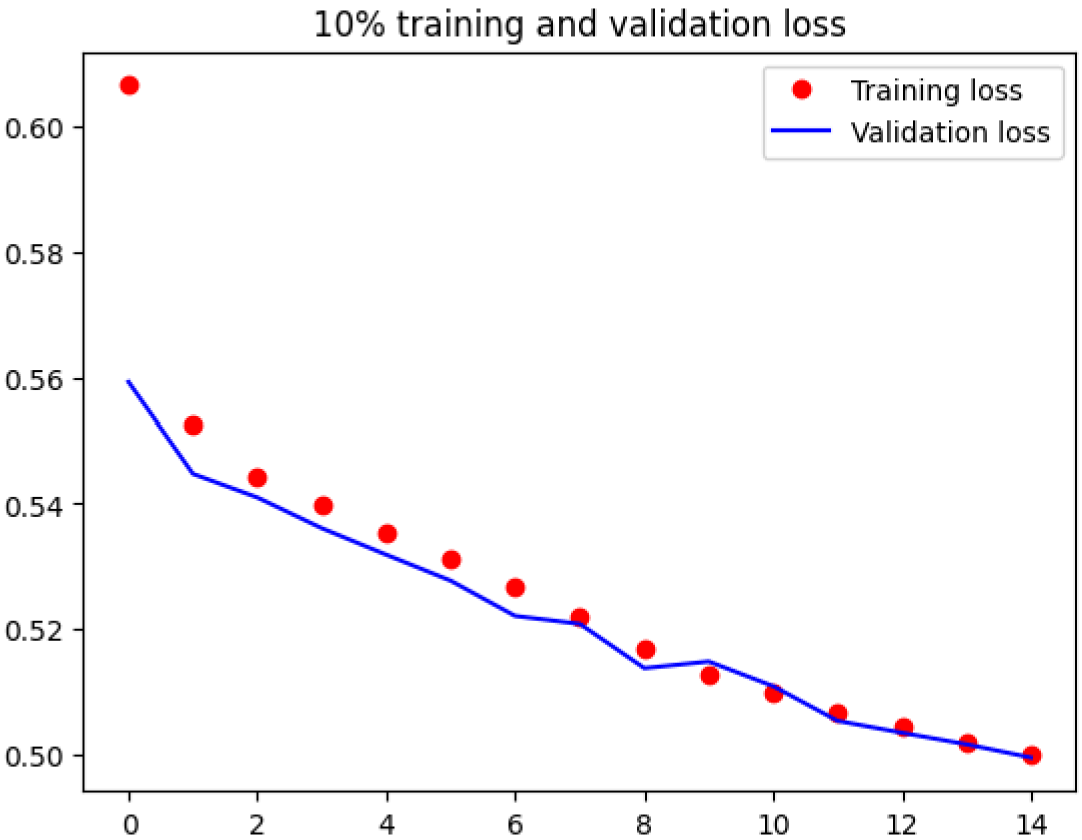
```python
# Reload
model_10 = tf.keras.models.load_model('model_10.keras')
with open('history_10.json') as f:
    history_10 = json.load(f)
```

In [35]:

```python
# Create the plot
acc_10 = history_10['accuracy']
val_acc_10 = history_10['val_accuracy']
loss_10 = history_10['loss']
val_loss_10 = history_10['val_loss']

epochs = range(len(acc_10))

plt.plot(epochs, acc_10, 'ro', label='Training acc')
plt.plot(epochs, val_acc_10, 'b', label='Validation acc')
plt.title('10% training and validation accuracy')
plt.legend()

plt.figure()

plt.plot(epochs, loss_10, 'ro', label='Training loss')
plt.plot(epochs, val_loss_10, 'b', label='Validation loss')
plt.title('10% training and validation loss')
```

```
plt.legend()

plt.show()
```





Comment:

Regarding accuracy, validation started around 0.71, and both training and validation followed a similar trend, gradually increasing, forming a straight line, and reaching 0.75. Concerning loss, validation started around 0.56. Subsequently, both training and validation displayed a similar trend, gradually decreasing, forming a straight line, and reaching around 0.50.

In [44]:
```
model_10.compile(metrics=fresh_metrics())
# Evaluate Model with 10% of the Training Data
model_10_results = model_10.evaluate(test_data, return_dict=True)
model_10_results
```

```
313/313 [==============================] - 8s 22ms/step - loss: 0.0000e+00
- tp: 121237.0000 - fp: 40105.0000 - tn: 119827.0000 - fn: 38831.0000 - ac
curacy: 0.7533 - precision: 0.7514 - recall: 0.7574
```

Out[44]:
```
{'loss': 0.0,
 'tp': 121237.0,
 'fp': 40105.0,
 'tn': 119827.0,
 'fn': 38831.0,
 'accuracy': 0.7533249855041504,
 'precision': 0.7514286637306213,
 'recall': 0.7574093341827393}
```

Training 25% of the dataset

In [45]:
```
# put your code here

# Build the model
model_25 = Sequential([
    encoder100,
    Embedding(input_dim=len(encoder100.get_vocabulary()), output_dim=EMBE
    LSTM(4),
    Dense(1, activation='sigmoid')
])

# Compile the model
model_25.compile(loss='binary_crossentropy',
              optimizer='adam',
              metrics=fresh_metrics())
```

In [46]:
```
# Train the model
history_25 = model_25.fit(
    train_data25,
    validation_data=validation_data,
    epochs=15,
    verbose=1)

# Save the model and training history
model_25.save('model_25.keras')
with open('history_25.json', 'w') as f:
    json.dump(history_25.history, f)
```

```
Epoch 1/15
879/879 [==============================] - 65s 34ms/step - loss: 0.5710 -
tp: 306801.0000 - fp: 122000.0000 - tn: 327624.0000 - fn: 143671.0000 - ac
curacy: 0.7048 - precision: 0.7155 - recall: 0.6811 - val_loss: 0.5426 - v
al_tp: 28306.0000 - val_fp: 10594.0000 - val_tn: 29474.0000 - val_fn: 1162
6.0000 - val_accuracy: 0.7222 - val_precision: 0.7277 - val_recall: 0.7089
Epoch 2/15
879/879 [==============================] - 28s 32ms/step - loss: 0.5383 -
tp: 316767.0000 - fp: 113890.0000 - tn: 335734.0000 - fn: 133705.0000 - ac
curacy: 0.7249 - precision: 0.7355 - recall: 0.7032 - val_loss: 0.5319 - v
al_tp: 27591.0000 - val_fp: 9333.0000 - val_tn: 30735.0000 - val_fn: 1234
1.0000 - val_accuracy: 0.7291 - val_precision: 0.7472 - val_recall: 0.6909
Epoch 3/15
879/879 [==============================] - 28s 32ms/step - loss: 0.5276 -
tp: 318300.0000 - fp: 107938.0000 - tn: 341686.0000 - fn: 132172.0000 - ac
curacy: 0.7332 - precision: 0.7468 - recall: 0.7066 - val_loss: 0.5210 - v
al_tp: 29276.0000 - val_fp: 10374.0000 - val_tn: 29694.0000 - val_fn: 1065
6.0000 - val_accuracy: 0.7371 - val_precision: 0.7384 - val_recall: 0.7331
Epoch 4/15
879/879 [==============================] - 28s 32ms/step - loss: 0.5167 -
tp: 322800.0000 - fp: 105577.0000 - tn: 344047.0000 - fn: 127672.0000 - ac
curacy: 0.7409 - precision: 0.7535 - recall: 0.7166 - val_loss: 0.5108 - v
al_tp: 28795.0000 - val_fp: 9274.0000 - val_tn: 30794.0000 - val_fn: 1113
7.0000 - val_accuracy: 0.7449 - val_precision: 0.7564 - val_recall: 0.7211
Epoch 5/15
879/879 [==============================] - 28s 32ms/step - loss: 0.5093 -
tp: 326586.0000 - fp: 104549.0000 - tn: 345075.0000 - fn: 123886.0000 - ac
curacy: 0.7462 - precision: 0.7575 - recall: 0.7250 - val_loss: 0.5064 - v
al_tp: 28588.0000 - val_fp: 8763.0000 - val_tn: 31305.0000 - val_fn: 1134
4.0000 - val_accuracy: 0.7487 - val_precision: 0.7654 - val_recall: 0.7159
Epoch 6/15
879/879 [==============================] - 28s 32ms/step - loss: 0.5052 -
tp: 328694.0000 - fp: 104113.0000 - tn: 345511.0000 - fn: 121778.0000 - ac
curacy: 0.7490 - precision: 0.7594 - recall: 0.7297 - val_loss: 0.5026 - v
al_tp: 29194.0000 - val_fp: 9168.0000 - val_tn: 30900.0000 - val_fn: 1073
8.0000 - val_accuracy: 0.7512 - val_precision: 0.7610 - val_recall: 0.7311
Epoch 7/15
879/879 [==============================] - 28s 32ms/step - loss: 0.5021 -
tp: 330108.0000 - fp: 103689.0000 - tn: 345935.0000 - fn: 120364.0000 - ac
curacy: 0.7511 - precision: 0.7610 - recall: 0.7328 - val_loss: 0.5023 - v
al_tp: 29561.0000 - val_fp: 9544.0000 - val_tn: 30524.0000 - val_fn: 1037
1.0000 - val_accuracy: 0.7511 - val_precision: 0.7559 - val_recall: 0.7403
Epoch 8/15
879/879 [==============================] - 28s 32ms/step - loss: 0.4997 -
tp: 331433.0000 - fp: 103439.0000 - tn: 346185.0000 - fn: 119039.0000 - ac
curacy: 0.7528 - precision: 0.7621 - recall: 0.7357 - val_loss: 0.4994 - v
al_tp: 28271.0000 - val_fp: 8042.0000 - val_tn: 32026.0000 - val_fn: 1166
1.0000 - val_accuracy: 0.7537 - val_precision: 0.7785 - val_recall: 0.7080
Epoch 9/15
879/879 [==============================] - 28s 32ms/step - loss: 0.4981 -
tp: 332234.0000 - fp: 103015.0000 - tn: 346609.0000 - fn: 118238.0000 - ac
curacy: 0.7542 - precision: 0.7633 - recall: 0.7375 - val_loss: 0.4991 - v
al_tp: 31079.0000 - val_fp: 10757.0000 - val_tn: 29311.0000 - val_fn: 885
3.0000 - val_accuracy: 0.7549 - val_precision: 0.7429 - val_recall: 0.7783
Epoch 10/15
879/879 [==============================] - 28s 32ms/step - loss: 0.4962 -
tp: 333209.0000 - fp: 103042.0000 - tn: 346582.0000 - fn: 117263.0000 - ac
curacy: 0.7552 - precision: 0.7638 - recall: 0.7397 - val_loss: 0.4960 - v
al_tp: 28894.0000 - val_fp: 8522.0000 - val_tn: 31546.0000 - val_fn: 1103
8.0000 - val_accuracy: 0.7555 - val_precision: 0.7722 - val_recall: 0.7236
```

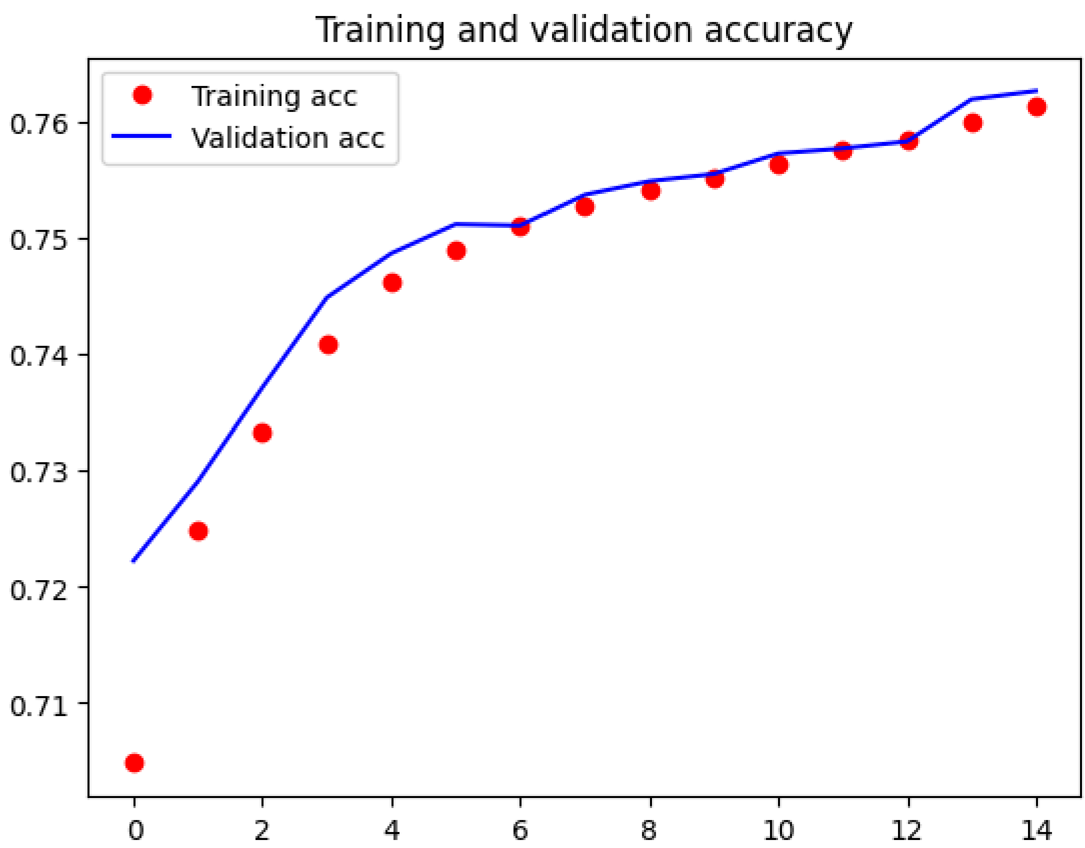
```
Epoch 11/15
879/879 [==============================] - 28s 32ms/step - loss: 0.4948 -
tp: 334048.0000 - fp: 102858.0000 - tn: 346766.0000 - fn: 116424.0000 - ac
curacy: 0.7564 - precision: 0.7646 - recall: 0.7416 - val_loss: 0.4950 - v
al_tp: 30380.0000 - val_fp: 9867.0000 - val_tn: 30201.0000 - val_fn: 9552.
0000 - val_accuracy: 0.7573 - val_precision: 0.7548 - val_recall: 0.7608
Epoch 12/15
879/879 [==============================] - 28s 32ms/step - loss: 0.4932 -
tp: 334928.0000 - fp: 102590.0000 - tn: 347034.0000 - fn: 115544.0000 - ac
curacy: 0.7577 - precision: 0.7655 - recall: 0.7435 - val_loss: 0.4933 - v
al_tp: 28813.0000 - val_fp: 8265.0000 - val_tn: 31803.0000 - val_fn: 1111
9.0000 - val_accuracy: 0.7577 - val_precision: 0.7771 - val_recall: 0.7216
Epoch 13/15
879/879 [==============================] - 28s 32ms/step - loss: 0.4919 -
tp: 335708.0000 - fp: 102630.0000 - tn: 346994.0000 - fn: 114764.0000 - ac
curacy: 0.7585 - precision: 0.7659 - recall: 0.7452 - val_loss: 0.4941 - v
al_tp: 28048.0000 - val_fp: 7452.0000 - val_tn: 32616.0000 - val_fn: 1188
4.0000 - val_accuracy: 0.7583 - val_precision: 0.7901 - val_recall: 0.7024
Epoch 14/15
879/879 [==============================] - 28s 32ms/step - loss: 0.4903 -
tp: 336466.0000 - fp: 102056.0000 - tn: 347568.0000 - fn: 114006.0000 - ac
curacy: 0.7600 - precision: 0.7673 - recall: 0.7469 - val_loss: 0.4896 - v
al_tp: 30270.0000 - val_fp: 9384.0000 - val_tn: 30684.0000 - val_fn: 9662.
0000 - val_accuracy: 0.7619 - val_precision: 0.7634 - val_recall: 0.7580
Epoch 15/15
879/879 [==============================] - 28s 32ms/step - loss: 0.4885 -
tp: 337457.0000 - fp: 101720.0000 - tn: 347904.0000 - fn: 113015.0000 - ac
curacy: 0.7614 - precision: 0.7684 - recall: 0.7491 - val_loss: 0.4873 - v
al_tp: 29601.0000 - val_fp: 8657.0000 - val_tn: 31411.0000 - val_fn: 1033
1.0000 - val_accuracy: 0.7627 - val_precision: 0.7737 - val_recall: 0.7413
```

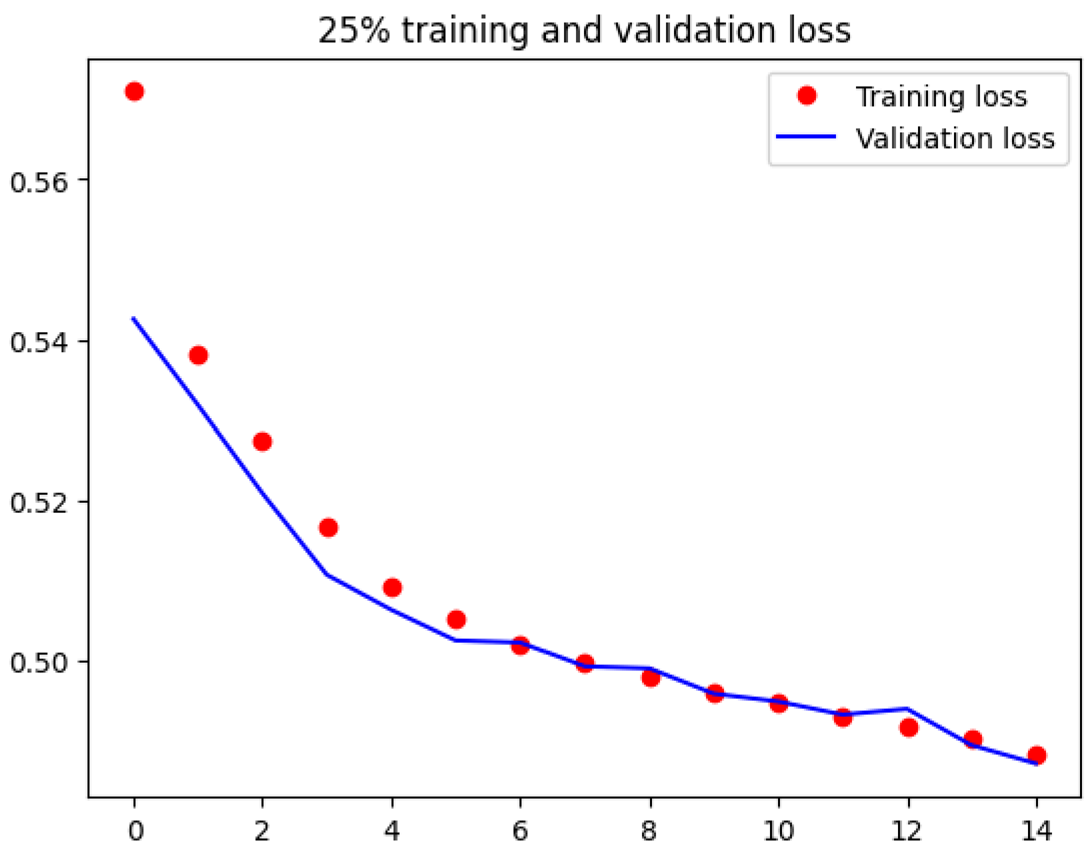
In [47]:

```python
# Reload
model_25 = tf.keras.models.load_model('model_25.keras')
with open('history_25.json') as f:
    history_25 = json.load(f)
```

In [49]:

```python
# Create the plot
acc_25 = history_25['accuracy']
val_acc_25 = history_25['val_accuracy']
loss_25 = history_25['loss']
val_loss_25 = history_25['val_loss']

epochs = range(len(acc_25))

plt.plot(epochs, acc_25, 'ro', label='Training acc')
plt.plot(epochs, val_acc_25, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.legend()

plt.figure()

plt.plot(epochs, loss_25, 'ro', label='Training loss')
plt.plot(epochs, val_loss_25, 'b', label='Validation loss')
plt.title('25% training and validation loss')
plt.legend()

plt.show()
```





Comment:

When it comes to accuracy, validation started around 0.72 with both training and validation showing parallel trends. Although there was a slight gap between training and validation up to 5 epochs, both increased, forming a curvy line, and reaching 0.76.

Regarding loss, validation started around 0.54. Subsequently, both training and validation displayed a similar trend. Like accuracy, there was a slight gap between training and validation up to 5 epochs; however, both decreased, forming a curvy line, and reaching below 0.50.

In [50]:
```
model_25.compile(metrics=fresh_metrics())
# Evaluate Model with 25% of the Training Data
model_25_results = model_25.evaluate(test_data, return_dict=True)
model_25_results
```

```
313/313 [==============================] - 8s 22ms/step - loss: 0.0000e+00
- tp: 118612.0000 - fp: 34670.0000 - tn: 125262.0000 - fn: 41456.0000 - ac
curacy: 0.7621 - precision: 0.7738 - recall: 0.7410
```

Out[50]:
```
{'loss': 0.0,
 'tp': 118612.0,
 'fp': 34670.0,
 'tn': 125262.0,
 'fn': 41456.0,
 'accuracy': 0.7621062397956848,
 'precision': 0.7738155722618103,
 'recall': 0.7410100698471069}
```

Training 50% of the dataset

In [51]:
```
# put your code here
# Build the model
model_50 = Sequential([
    encoder100,
    Embedding(input_dim=len(encoder100.get_vocabulary()), output_dim=EMBE
    LSTM(4),
    Dense(1, activation='sigmoid')
])

# Compile the model
model_50.compile(loss='binary_crossentropy',
              optimizer='adam',
              metrics=fresh_metrics())

# Train the model
history_50 = model_50.fit(
    train_data50,
    validation_data=validation_data,
    epochs=15,
    verbose=1)

# Save the model and training history
model_50.save('model_50.keras')
with open('history_50.json', 'w') as f:
    json.dump(history_50.history, f)
```

```
Epoch 1/15
1758/1758 [==============================] - 90s 32ms/step - loss: 0.5619
- tp: 606876.0000 - fp: 225038.0000 - tn: 674839.0000 - fn: 293439.0000 -
accuracy: 0.7120 - precision: 0.7295 - recall: 0.6741 - val_loss: 0.5261 -
val_tp: 29425.0000 - val_fp: 10800.0000 - val_tn: 29268.0000 - val_fn: 105
07.0000 - val_accuracy: 0.7337 - val_precision: 0.7315 - val_recall: 0.736
9
Epoch 2/15
1758/1758 [==============================] - 55s 31ms/step - loss: 0.5173
- tp: 651715.0000 - fp: 216142.0000 - tn: 683735.0000 - fn: 248600.0000 -
accuracy: 0.7418 - precision: 0.7509 - recall: 0.7239 - val_loss: 0.5085 -
val_tp: 29296.0000 - val_fp: 9596.0000 - val_tn: 30472.0000 - val_fn: 1063
6.0000 - val_accuracy: 0.7471 - val_precision: 0.7533 - val_recall: 0.7336
Epoch 3/15
1758/1758 [==============================] - 54s 31ms/step - loss: 0.5027
- tp: 664128.0000 - fp: 209807.0000 - tn: 690070.0000 - fn: 236187.0000 -
accuracy: 0.7523 - precision: 0.7599 - recall: 0.7377 - val_loss: 0.4971 -
val_tp: 30875.0000 - val_fp: 10533.0000 - val_tn: 29535.0000 - val_fn: 905
7.0000 - val_accuracy: 0.7551 - val_precision: 0.7456 - val_recall: 0.7732
Epoch 4/15
1758/1758 [==============================] - 55s 31ms/step - loss: 0.4916
- tp: 671300.0000 - fp: 203246.0000 - tn: 696631.0000 - fn: 229015.0000 -
accuracy: 0.7599 - precision: 0.7676 - recall: 0.7456 - val_loss: 0.4876 -
val_tp: 30275.0000 - val_fp: 9398.0000 - val_tn: 30670.0000 - val_fn: 965
7.0000 - val_accuracy: 0.7618 - val_precision: 0.7631 - val_recall: 0.7582
Epoch 5/15
1758/1758 [==============================] - 55s 31ms/step - loss: 0.4872
- tp: 675615.0000 - fp: 201762.0000 - tn: 698115.0000 - fn: 224700.0000 -
accuracy: 0.7631 - precision: 0.7700 - recall: 0.7504 - val_loss: 0.4880 -
val_tp: 31799.0000 - val_fp: 10940.0000 - val_tn: 29128.0000 - val_fn: 813
3.0000 - val_accuracy: 0.7616 - val_precision: 0.7440 - val_recall: 0.7963
Epoch 6/15
1758/1758 [==============================] - 56s 32ms/step - loss: 0.4843
- tp: 678854.0000 - fp: 201089.0000 - tn: 698788.0000 - fn: 221461.0000 -
accuracy: 0.7653 - precision: 0.7715 - recall: 0.7540 - val_loss: 0.4866 -
val_tp: 28178.0000 - val_fp: 7242.0000 - val_tn: 32826.0000 - val_fn: 1175
4.0000 - val_accuracy: 0.7625 - val_precision: 0.7955 - val_recall: 0.7056
Epoch 7/15
1758/1758 [==============================] - 56s 32ms/step - loss: 0.4824
- tp: 680841.0000 - fp: 200450.0000 - tn: 699427.0000 - fn: 219474.0000 -
accuracy: 0.7667 - precision: 0.7725 - recall: 0.7562 - val_loss: 0.4841 -
val_tp: 31819.0000 - val_fp: 10766.0000 - val_tn: 29302.0000 - val_fn: 811
3.0000 - val_accuracy: 0.7640 - val_precision: 0.7472 - val_recall: 0.7968
Epoch 8/15
1758/1758 [==============================] - 56s 32ms/step - loss: 0.4803
- tp: 682564.0000 - fp: 199908.0000 - tn: 699969.0000 - fn: 217751.0000 -
accuracy: 0.7680 - precision: 0.7735 - recall: 0.7581 - val_loss: 0.4798 -
val_tp: 29622.0000 - val_fp: 8278.0000 - val_tn: 31790.0000 - val_fn: 1031
0.0000 - val_accuracy: 0.7677 - val_precision: 0.7816 - val_recall: 0.7418
Epoch 9/15
1758/1758 [==============================] - 56s 32ms/step - loss: 0.4790
- tp: 683999.0000 - fp: 199659.0000 - tn: 700218.0000 - fn: 216316.0000 -
accuracy: 0.7689 - precision: 0.7741 - recall: 0.7597 - val_loss: 0.4783 -
val_tp: 31130.0000 - val_fp: 9685.0000 - val_tn: 30383.0000 - val_fn: 880
2.0000 - val_accuracy: 0.7689 - val_precision: 0.7627 - val_recall: 0.7796
Epoch 10/15
1758/1758 [==============================] - 56s 32ms/step - loss: 0.4774
- tp: 685097.0000 - fp: 198855.0000 - tn: 701022.0000 - fn: 215218.0000 -
accuracy: 0.7700 - precision: 0.7750 - recall: 0.7610 - val_loss: 0.4868 -
val_tp: 33079.0000 - val_fp: 12141.0000 - val_tn: 27927.0000 - val_fn: 685
```

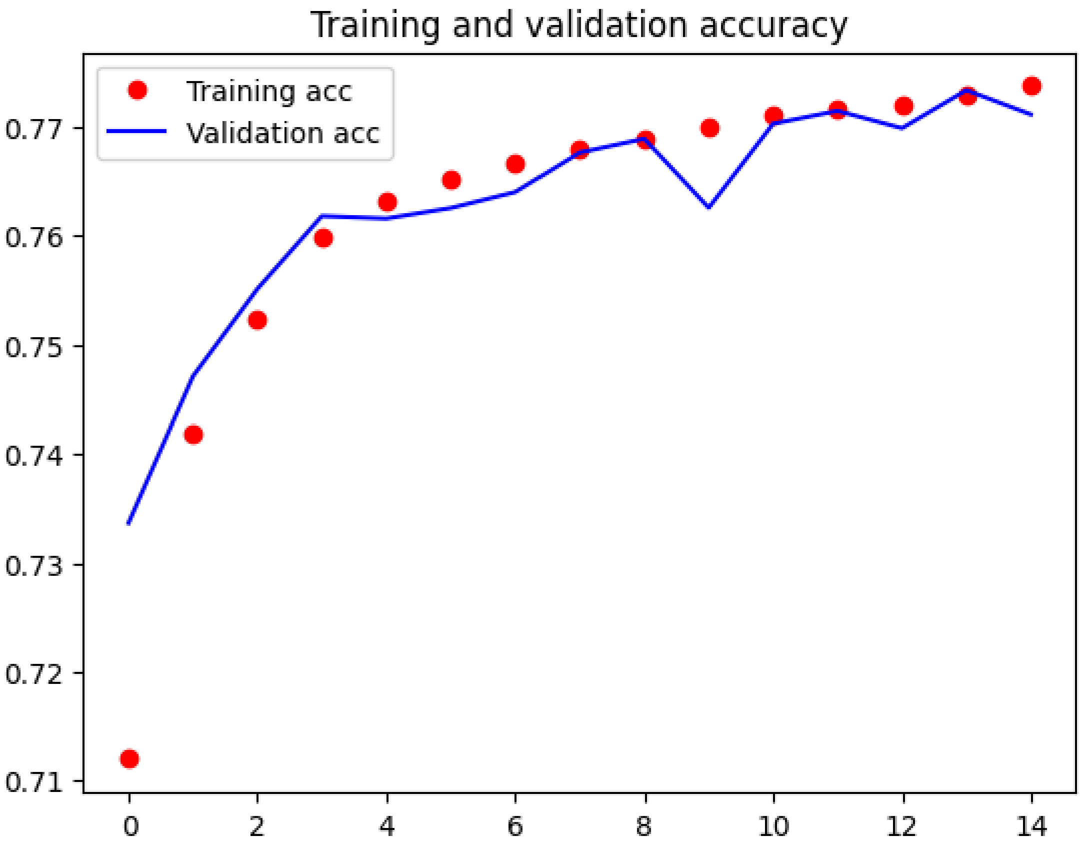
```
3.0000 - val_accuracy: 0.7626 - val_precision: 0.7315 - val_recall: 0.8284
Epoch 11/15
1758/1758 [==============================] - 56s 32ms/step - loss: 0.4757
- tp: 687043.0000 - fp: 198911.0000 - tn: 700966.0000 - fn: 213272.0000 -
accuracy: 0.7710 - precision: 0.7755 - recall: 0.7631 - val_loss: 0.4755 -
val_tp: 31054.0000 - val_fp: 9498.0000 - val_tn: 30570.0000 - val_fn: 887
8.0000 - val_accuracy: 0.7703 - val_precision: 0.7658 - val_recall: 0.7777
Epoch 12/15
1758/1758 [==============================] - 56s 32ms/step - loss: 0.4745
- tp: 687572.0000 - fp: 198271.0000 - tn: 701606.0000 - fn: 212743.0000 -
accuracy: 0.7717 - precision: 0.7762 - recall: 0.7637 - val_loss: 0.4747 -
val_tp: 31417.0000 - val_fp: 9767.0000 - val_tn: 30301.0000 - val_fn: 851
5.0000 - val_accuracy: 0.7715 - val_precision: 0.7628 - val_recall: 0.7868
Epoch 13/15
1758/1758 [==============================] - 55s 31ms/step - loss: 0.4737
- tp: 687734.0000 - fp: 197737.0000 - tn: 702140.0000 - fn: 212581.0000 -
accuracy: 0.7721 - precision: 0.7767 - recall: 0.7639 - val_loss: 0.4765 -
val_tp: 28705.0000 - val_fp: 7182.0000 - val_tn: 32886.0000 - val_fn: 1122
7.0000 - val_accuracy: 0.7699 - val_precision: 0.7999 - val_recall: 0.7188
Epoch 14/15
1758/1758 [==============================] - 56s 32ms/step - loss: 0.4726
- tp: 688971.0000 - fp: 197348.0000 - tn: 702529.0000 - fn: 211344.0000 -
accuracy: 0.7730 - precision: 0.7773 - recall: 0.7653 - val_loss: 0.4722 -
val_tp: 31434.0000 - val_fp: 9632.0000 - val_tn: 30436.0000 - val_fn: 849
8.0000 - val_accuracy: 0.7734 - val_precision: 0.7655 - val_recall: 0.7872
Epoch 15/15
1758/1758 [==============================] - 55s 31ms/step - loss: 0.4716
- tp: 689875.0000 - fp: 196830.0000 - tn: 703047.0000 - fn: 210440.0000 -
accuracy: 0.7738 - precision: 0.7780 - recall: 0.7663 - val_loss: 0.4749 -
val_tp: 32297.0000 - val_fp: 10673.0000 - val_tn: 29395.0000 - val_fn: 763
5.0000 - val_accuracy: 0.7711 - val_precision: 0.7516 - val_recall: 0.8088
```

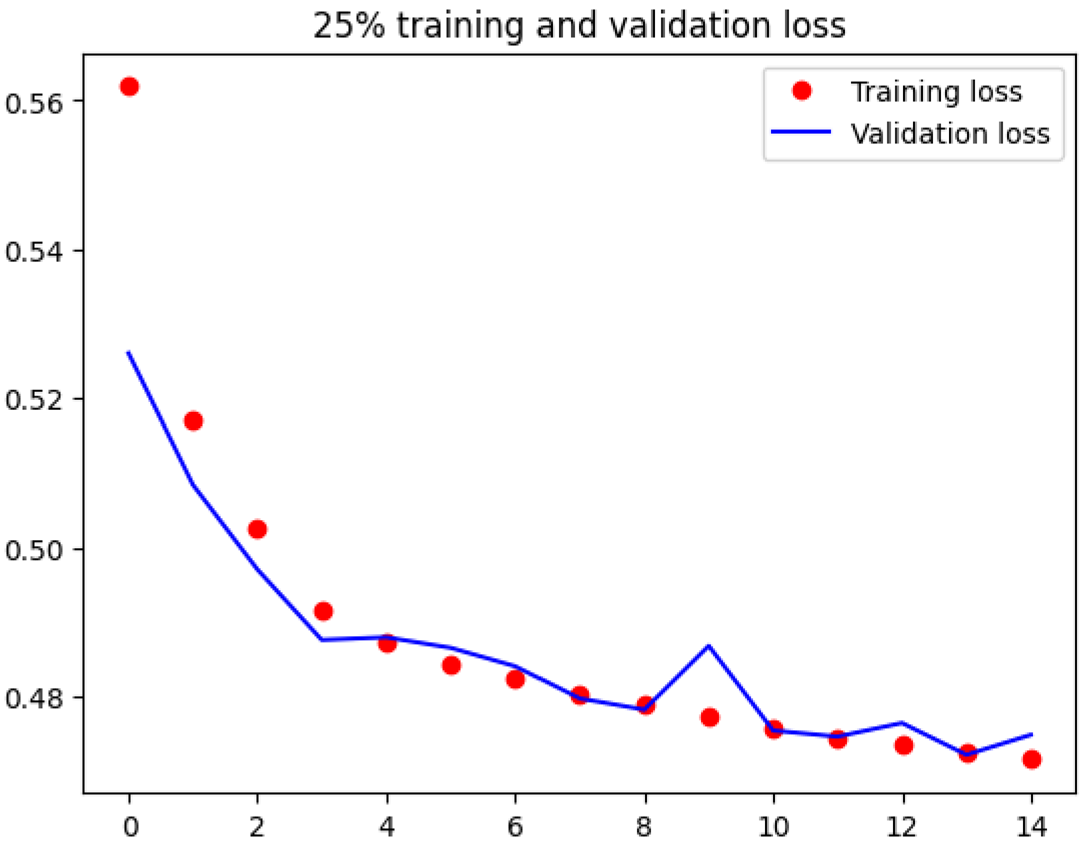
In [52]:

```
# Reload
model_50 = tf.keras.models.load_model('model_50.keras')
with open('history_50.json') as f:
    history_50 = json.load(f)
```

In [53]:

```
# create the plot
acc_50 = history_50['accuracy']
val_acc_50 = history_50['val_accuracy']
loss_50 = history_50['loss']
val_loss_50 = history_50['val_loss']

epochs = range(len(acc_50))

plt.plot(epochs, acc_50, 'ro', label='Training acc')
plt.plot(epochs, val_acc_50, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.legend()

plt.figure()

plt.plot(epochs, loss_50, 'ro', label='Training loss')
plt.plot(epochs, val_loss_50, 'b', label='Validation loss')
plt.title('25% training and validation loss')
plt.legend()

plt.show()
```





Comment:

When it comes to accuracy, validation started around 0.73. Both training and validation followed a similar trend, with a notable dip observed at the 9th epoch in the validation set. Despite this, both training and validation accuracies increased steadily, forming a curvy line with slight fluctuations and eventually reaching around 0.77.

Regarding loss, validation started around 0.52. Both training and validation exhibited a similar trend, with a significant protrusion seen at the 9th epoch in the validation set. However, both training and validation losses decreased steadily, forming a curvy line with slight fluctuations, and eventually reached below 0.48.

In [54]:
```
model_50.compile(metrics=fresh_metrics())
# Evaluate Model with 50% of the Training Data
model_50_results = model_50.evaluate(test_data, return_dict=True)
model_50_results
```

```
313/313 [==============================] - 8s 22ms/step - loss: 0.0000e+00
- tp: 129667.0000 - fp: 43030.0000 - tn: 116902.0000 - fn: 30401.0000 - ac
curacy: 0.7705 - precision: 0.7508 - recall: 0.8101
```

Out[54]:
```
{'loss': 0.0,
 'tp': 129667.0,
 'fp': 43030.0,
 'tn': 116902.0,
 'fn': 30401.0,
 'accuracy': 0.7705281376838684,
 'precision': 0.7508352994918823,
 'recall': 0.8100744485855103}
```

Compare and comment on the results

Analysing the classification metrics, as the percentage of the training dataset usage increased, the accuracy also increased. Specifically, the models trained with 10%, 25%, and 50% of the training dataset achieved accuracy percentages of 0.7533, 0.7621, and 0.7705, respectively.

However, when considering precision, the model trained with 25% of the training dataset showed the highest percentage at 0.7738. On the other hand, the recall metric reached its highest value of 0.8101 for the model trained with 50% of the training dataset. Consequently, the model trained with 50% of the dataset yielded the best performance overall.

Adjusting the number of LSTM modules

The second part your investigation is how the number of LSTM modules impacts the performance of the model.

For your models you should:

- create and use a 250-word encoder
- use 25% of the training data
- train for 15 epochs

Create two models, one with 8 LSTM units and the other with 12 LSTM units.

Training with 8 LSTM modules

In [19]:

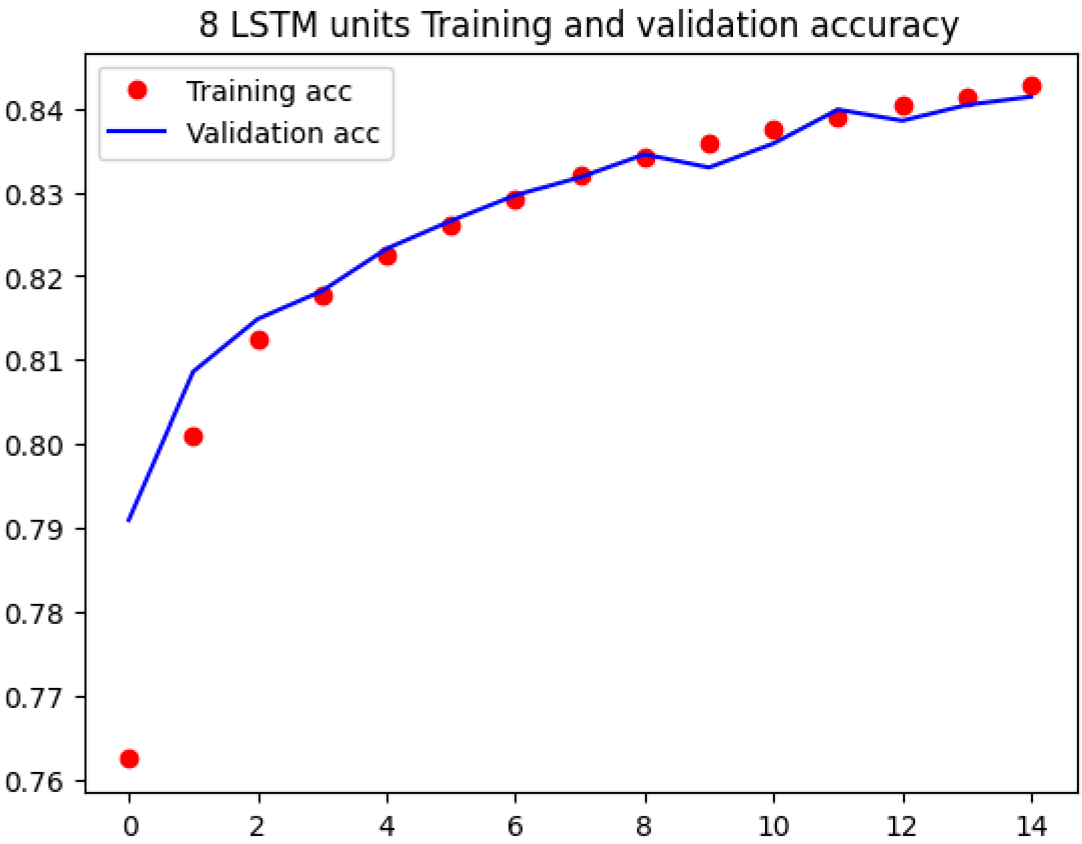
```
# put your code here
# Build the model with 8 LSTM units
model_8_units = Sequential([
    encoder250,
    Embedding(input_dim=len(encoder250.get_vocabulary()), output_dim=EMBE
    LSTM(8),
    Dense(1, activation='sigmoid')
])

# Compile the model
model_8_units.compile(loss='binary_crossentropy',
                      optimizer='adam',
                      metrics=fresh_metrics())

# Train the model
history_8_units = model_8_units.fit(
    train_data25,
    validation_data=validation_data,
    epochs=15,
    verbose=1)

# Save the model and training history
model_8_units.save('model_8_units.keras')
with open('history_8_units.json', 'w') as f:
    json.dump(history_8_units.history, f)
```

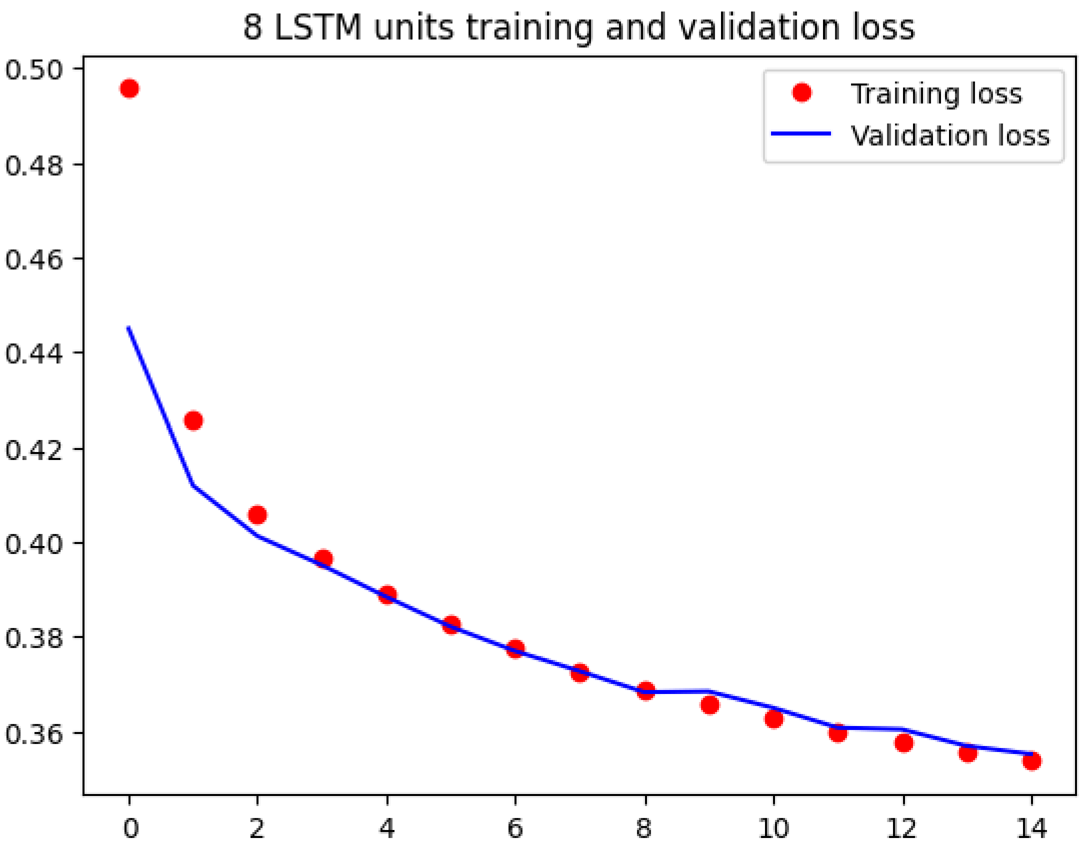
```
Epoch 1/15
879/879 [==============================] - 59s 32ms/step - loss: 0.4957 -
tp: 331814.0000 - fp: 95200.0000 - tn: 354438.0000 - fn: 118644.0000 - acc
uracy: 0.7624 - precision: 0.7771 - recall: 0.7366 - val_loss: 0.4450 - va
l_tp: 31514.0000 - val_fp: 8307.0000 - val_tn: 31761.0000 - val_fn: 8418.0
000 - val_accuracy: 0.7909 - val_precision: 0.7914 - val_recall: 0.7892
Epoch 2/15
879/879 [==============================] - 26s 30ms/step - loss: 0.4258 -
tp: 358161.0000 - fp: 86726.0000 - tn: 362912.0000 - fn: 92297.0000 - accu
racy: 0.8011 - precision: 0.8051 - recall: 0.7951 - val_loss: 0.4119 - val
_tp: 31832.0000 - val_fp: 7213.0000 - val_tn: 32855.0000 - val_fn: 8100.00
00 - val_accuracy: 0.8086 - val_precision: 0.8153 - val_recall: 0.7972
Epoch 3/15
879/879 [==============================] - 26s 30ms/step - loss: 0.4058 -
tp: 364097.0000 - fp: 82531.0000 - tn: 367107.0000 - fn: 86361.0000 - accu
racy: 0.8124 - precision: 0.8152 - recall: 0.8083 - val_loss: 0.4012 - val
_tp: 32706.0000 - val_fp: 7585.0000 - val_tn: 32483.0000 - val_fn: 7226.00
00 - val_accuracy: 0.8149 - val_precision: 0.8117 - val_recall: 0.8190
Epoch 4/15
879/879 [==============================] - 26s 30ms/step - loss: 0.3967 -
tp: 367151.0000 - fp: 80773.0000 - tn: 368865.0000 - fn: 83307.0000 - accu
racy: 0.8177 - precision: 0.8197 - recall: 0.8151 - val_loss: 0.3951 - val
_tp: 33887.0000 - val_fp: 8498.0000 - val_tn: 31570.0000 - val_fn: 6045.00
00 - val_accuracy: 0.8182 - val_precision: 0.7995 - val_recall: 0.8486
Epoch 5/15
879/879 [==============================] - 26s 30ms/step - loss: 0.3890 -
tp: 369687.0000 - fp: 79023.0000 - tn: 370615.0000 - fn: 80771.0000 - accu
racy: 0.8225 - precision: 0.8239 - recall: 0.8207 - val_loss: 0.3884 - val
_tp: 31784.0000 - val_fp: 5992.0000 - val_tn: 34076.0000 - val_fn: 8148.00
00 - val_accuracy: 0.8232 - val_precision: 0.8414 - val_recall: 0.7960
Epoch 6/15
879/879 [==============================] - 26s 30ms/step - loss: 0.3826 -
tp: 370896.0000 - fp: 76928.0000 - tn: 372710.0000 - fn: 79562.0000 - accu
racy: 0.8261 - precision: 0.8282 - recall: 0.8234 - val_loss: 0.3821 - val
_tp: 33297.0000 - val_fp: 7238.0000 - val_tn: 32830.0000 - val_fn: 6635.00
00 - val_accuracy: 0.8266 - val_precision: 0.8214 - val_recall: 0.8338
Epoch 7/15
879/879 [==============================] - 26s 30ms/step - loss: 0.3776 -
tp: 372271.0000 - fp: 75650.0000 - tn: 373988.0000 - fn: 78187.0000 - accu
racy: 0.8291 - precision: 0.8311 - recall: 0.8264 - val_loss: 0.3770 - val
_tp: 33061.0000 - val_fp: 6758.0000 - val_tn: 33310.0000 - val_fn: 6871.00
00 - val_accuracy: 0.8296 - val_precision: 0.8303 - val_recall: 0.8279
Epoch 8/15
879/879 [==============================] - 26s 30ms/step - loss: 0.3725 -
tp: 373200.0000 - fp: 73866.0000 - tn: 375772.0000 - fn: 77258.0000 - accu
racy: 0.8321 - precision: 0.8348 - recall: 0.8285 - val_loss: 0.3727 - val
_tp: 33741.0000 - val_fp: 7266.0000 - val_tn: 32802.0000 - val_fn: 6191.00
00 - val_accuracy: 0.8318 - val_precision: 0.8228 - val_recall: 0.8450
Epoch 9/15
879/879 [==============================] - 26s 30ms/step - loss: 0.3686 -
tp: 374241.0000 - fp: 73024.0000 - tn: 376614.0000 - fn: 76217.0000 - accu
racy: 0.8342 - precision: 0.8367 - recall: 0.8308 - val_loss: 0.3682 - val
_tp: 33503.0000 - val_fp: 6809.0000 - val_tn: 33259.0000 - val_fn: 6429.00
00 - val_accuracy: 0.8345 - val_precision: 0.8311 - val_recall: 0.8390
Epoch 10/15
879/879 [==============================] - 26s 30ms/step - loss: 0.3656 -
tp: 374818.0000 - fp: 72071.0000 - tn: 377567.0000 - fn: 75640.0000 - accu
racy: 0.8359 - precision: 0.8387 - recall: 0.8321 - val_loss: 0.3684 - val
_tp: 34421.0000 - val_fp: 7853.0000 - val_tn: 32215.0000 - val_fn: 5511.00
00 - val_accuracy: 0.8329 - val_precision: 0.8142 - val_recall: 0.8620
```

```
Epoch 11/15
879/879 [==============================] - 26s 30ms/step - loss: 0.3626 -
tp: 375532.0000 - fp: 71304.0000 - tn: 378334.0000 - fn: 74926.0000 - accu
racy: 0.8375 - precision: 0.8404 - recall: 0.8337 - val_loss: 0.3649 - val
_tp: 34251.0000 - val_fp: 7454.0000 - val_tn: 32614.0000 - val_fn: 5681.00
00 - val_accuracy: 0.8358 - val_precision: 0.8213 - val_recall: 0.8577
Epoch 12/15
879/879 [==============================] - 26s 30ms/step - loss: 0.3597 -
tp: 376367.0000 - fp: 70795.0000 - tn: 378843.0000 - fn: 74091.0000 - accu
racy: 0.8390 - precision: 0.8417 - recall: 0.8355 - val_loss: 0.3608 - val
_tp: 33070.0000 - val_fp: 5949.0000 - val_tn: 34119.0000 - val_fn: 6862.00
00 - val_accuracy: 0.8399 - val_precision: 0.8475 - val_recall: 0.8282
Epoch 13/15
879/879 [==============================] - 26s 30ms/step - loss: 0.3576 -
tp: 377007.0000 - fp: 70128.0000 - tn: 379510.0000 - fn: 73451.0000 - accu
racy: 0.8405 - precision: 0.8432 - recall: 0.8369 - val_loss: 0.3604 - val
_tp: 34238.0000 - val_fp: 7223.0000 - val_tn: 32845.0000 - val_fn: 5694.00
00 - val_accuracy: 0.8385 - val_precision: 0.8258 - val_recall: 0.8574
Epoch 14/15
879/879 [==============================] - 26s 30ms/step - loss: 0.3556 -
tp: 377291.0000 - fp: 69599.0000 - tn: 380039.0000 - fn: 73167.0000 - accu
racy: 0.8414 - precision: 0.8443 - recall: 0.8376 - val_loss: 0.3569 - val
_tp: 34003.0000 - val_fp: 6838.0000 - val_tn: 33230.0000 - val_fn: 5929.00
00 - val_accuracy: 0.8404 - val_precision: 0.8326 - val_recall: 0.8515
Epoch 15/15
879/879 [==============================] - 26s 30ms/step - loss: 0.3537 -
tp: 378012.0000 - fp: 69163.0000 - tn: 380475.0000 - fn: 72446.0000 - accu
racy: 0.8427 - precision: 0.8453 - recall: 0.8392 - val_loss: 0.3552 - val
_tp: 34022.0000 - val_fp: 6778.0000 - val_tn: 33290.0000 - val_fn: 5910.00
00 - val_accuracy: 0.8414 - val_precision: 0.8339 - val_recall: 0.8520
```

In [20]:
```
# Reload
model_8_units = tf.keras.models.load_model('model_8_units.keras')
with open('history_8_units.json') as f:
    history_8_units = json.load(f)
```

In [22]:
```
#create the plot
acc_8 = history_8_units['accuracy']
val_acc_8 = history_8_units['val_accuracy']
loss_8 = history_8_units['loss']
val_loss_8 = history_8_units['val_loss']

epochs = range(len(acc_8))

plt.plot(epochs, acc_8, 'ro', label='Training acc')
plt.plot(epochs, val_acc_8, 'b', label='Validation acc')
plt.title('8 LSTM units Training and validation accuracy')
plt.legend()

plt.figure()

plt.plot(epochs, loss_8, 'ro', label='Training loss')
plt.plot(epochs, val_loss_8, 'b', label='Validation loss')
plt.title('8 LSTM units training and validation loss')
plt.legend()

plt.show()
```





Comment:

When considering accuracy, validation began at approximately 0.79. Both training and validation followed a similar trend, increasing gradually and forming a slightly curved line. It is observed that there were slight fluctuations in validation after the 8th epochs, with both training and validation eventually reaching around 0.84.

Regarding loss, validation commenced at around 0.44. Both training and validation exhibited a similar trend, gradually decreasing with a slight curvature, and eventually reaching below 0.36.

In [23]:
```
model_8_units.compile(metrics=fresh_metrics())
# Evaluate Model with 8 LSTM units
model_8_units_results = model_8_units.evaluate(test_data, return_dict=Tru
model_8_units_results
```

```
313/313 [==============================] - 8s 23ms/step - loss: 0.0000e+00
- tp: 136797.0000 - fp: 27180.0000 - tn: 132752.0000 - fn: 23271.0000 - ac
curacy: 0.8423 - precision: 0.8342 - recall: 0.8546
```

Out[23]:
```
{'loss': 0.0,
 'tp': 136797.0,
 'fp': 27180.0,
 'tn': 132752.0,
 'fn': 23271.0,
 'accuracy': 0.8423406481742859,
 'precision': 0.8342450261116028,
 'recall': 0.8546180129051208}
```

Training with 12 LSTM modules

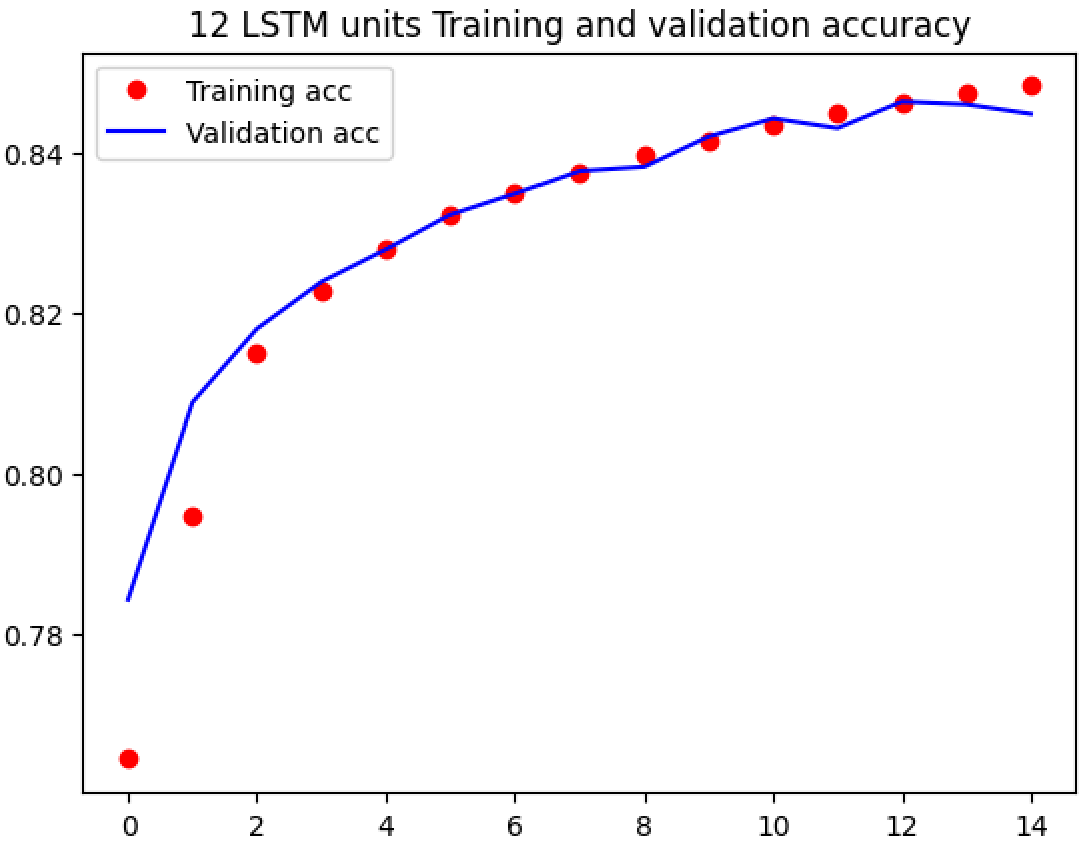
In [24]:
```
# put your code here
# Build the model with 12 LSTM units
model_12_units = Sequential([
    encoder250,
    Embedding(input_dim=len(encoder250.get_vocabulary()), output_dim=EMBE
    LSTM(12),
    Dense(1, activation='sigmoid')
])

# Compile the model
model_12_units.compile(loss='binary_crossentropy',
                       optimizer='adam',
                       metrics=fresh_metrics())

# Train the model
history_12_units = model_12_units.fit(
    train_data25,
    validation_data=validation_data,
    epochs=15,
    verbose=1)

# Save the model and training history
model_12_units.save('model_12_units.keras')
with open('history_12_units.json', 'w') as f:
    json.dump(history_12_units.history, f)
```

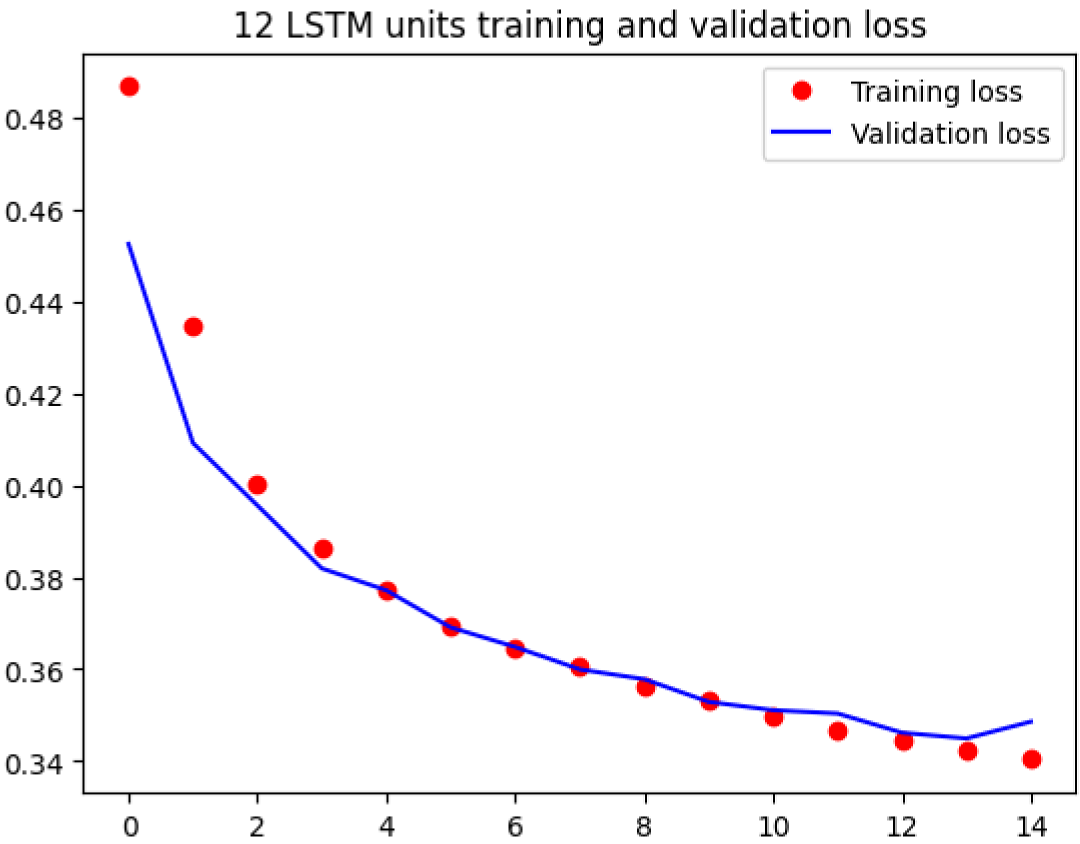
```
Epoch 1/15
879/879 [==============================] - 33s 31ms/step - loss: 0.4867 -
tp: 340193.0000 - fp: 101789.0000 - tn: 347849.0000 - fn: 110265.0000 - ac
curacy: 0.7644 - precision: 0.7697 - recall: 0.7552 - val_loss: 0.4525 - v
al_tp: 30929.0000 - val_fp: 8253.0000 - val_tn: 31815.0000 - val_fn: 9003.
0000 - val_accuracy: 0.7843 - val_precision: 0.7894 - val_recall: 0.7745
Epoch 2/15
879/879 [==============================] - 26s 30ms/step - loss: 0.4349 -
tp: 352716.0000 - fp: 86921.0000 - tn: 362717.0000 - fn: 97742.0000 - accu
racy: 0.7948 - precision: 0.8023 - recall: 0.7830 - val_loss: 0.4092 - val
_tp: 31788.0000 - val_fp: 7150.0000 - val_tn: 32918.0000 - val_fn: 8144.00
00 - val_accuracy: 0.8088 - val_precision: 0.8164 - val_recall: 0.7961
Epoch 3/15
879/879 [==============================] - 26s 30ms/step - loss: 0.4001 -
tp: 362541.0000 - fp: 78659.0000 - tn: 370979.0000 - fn: 87917.0000 - accu
racy: 0.8149 - precision: 0.8217 - recall: 0.8048 - val_loss: 0.3956 - val
_tp: 32998.0000 - val_fp: 7626.0000 - val_tn: 32442.0000 - val_fn: 6934.00
00 - val_accuracy: 0.8180 - val_precision: 0.8123 - val_recall: 0.8264
Epoch 4/15
879/879 [==============================] - 26s 30ms/step - loss: 0.3862 -
tp: 367096.0000 - fp: 76257.0000 - tn: 373381.0000 - fn: 83362.0000 - accu
racy: 0.8227 - precision: 0.8280 - recall: 0.8149 - val_loss: 0.3819 - val
_tp: 31991.0000 - val_fp: 6150.0000 - val_tn: 33918.0000 - val_fn: 7941.00
00 - val_accuracy: 0.8239 - val_precision: 0.8388 - val_recall: 0.8011
Epoch 5/15
879/879 [==============================] - 26s 30ms/step - loss: 0.3770 -
tp: 369954.0000 - fp: 74317.0000 - tn: 375321.0000 - fn: 80504.0000 - accu
racy: 0.8280 - precision: 0.8327 - recall: 0.8213 - val_loss: 0.3772 - val
_tp: 31740.0000 - val_fp: 5576.0000 - val_tn: 34492.0000 - val_fn: 8192.00
00 - val_accuracy: 0.8279 - val_precision: 0.8506 - val_recall: 0.7949
Epoch 6/15
879/879 [==============================] - 26s 30ms/step - loss: 0.3694 -
tp: 372535.0000 - fp: 73012.0000 - tn: 376626.0000 - fn: 77923.0000 - accu
racy: 0.8323 - precision: 0.8361 - recall: 0.8270 - val_loss: 0.3691 - val
_tp: 32390.0000 - val_fp: 5878.0000 - val_tn: 34190.0000 - val_fn: 7542.00
00 - val_accuracy: 0.8322 - val_precision: 0.8464 - val_recall: 0.8111
Epoch 7/15
879/879 [==============================] - 26s 30ms/step - loss: 0.3647 -
tp: 373717.0000 - fp: 71845.0000 - tn: 377793.0000 - fn: 76741.0000 - accu
racy: 0.8349 - precision: 0.8388 - recall: 0.8296 - val_loss: 0.3649 - val
_tp: 32729.0000 - val_fp: 6007.0000 - val_tn: 34061.0000 - val_fn: 7203.00
00 - val_accuracy: 0.8349 - val_precision: 0.8449 - val_recall: 0.8196
Epoch 8/15
879/879 [==============================] - 26s 30ms/step - loss: 0.3607 -
tp: 374704.0000 - fp: 70651.0000 - tn: 378987.0000 - fn: 75754.0000 - accu
racy: 0.8373 - precision: 0.8414 - recall: 0.8318 - val_loss: 0.3599 - val
_tp: 32941.0000 - val_fp: 5993.0000 - val_tn: 34075.0000 - val_fn: 6991.00
00 - val_accuracy: 0.8377 - val_precision: 0.8461 - val_recall: 0.8249
Epoch 9/15
879/879 [==============================] - 26s 30ms/step - loss: 0.3563 -
tp: 375926.0000 - fp: 69691.0000 - tn: 379947.0000 - fn: 74532.0000 - accu
racy: 0.8398 - precision: 0.8436 - recall: 0.8345 - val_loss: 0.3578 - val
_tp: 32730.0000 - val_fp: 5739.0000 - val_tn: 34329.0000 - val_fn: 7202.00
00 - val_accuracy: 0.8382 - val_precision: 0.8508 - val_recall: 0.8196
Epoch 10/15
879/879 [==============================] - 26s 30ms/step - loss: 0.3532 -
tp: 376923.0000 - fp: 69100.0000 - tn: 380538.0000 - fn: 73535.0000 - accu
racy: 0.8415 - precision: 0.8451 - recall: 0.8368 - val_loss: 0.3529 - val
_tp: 33018.0000 - val_fp: 5725.0000 - val_tn: 34343.0000 - val_fn: 6914.00
00 - val_accuracy: 0.8420 - val_precision: 0.8522 - val_recall: 0.8269
```

```
Epoch 11/15
879/879 [==============================] - 26s 30ms/step - loss: 0.3497 -
tp: 377972.0000 - fp: 68420.0000 - tn: 381218.0000 - fn: 72486.0000 - accu
racy: 0.8435 - precision: 0.8467 - recall: 0.8391 - val_loss: 0.3511 - val
_tp: 33160.0000 - val_fp: 5686.0000 - val_tn: 34382.0000 - val_fn: 6772.00
00 - val_accuracy: 0.8443 - val_precision: 0.8536 - val_recall: 0.8304
Epoch 12/15
879/879 [==============================] - 26s 30ms/step - loss: 0.3469 -
tp: 378505.0000 - fp: 67670.0000 - tn: 381968.0000 - fn: 71953.0000 - accu
racy: 0.8449 - precision: 0.8483 - recall: 0.8403 - val_loss: 0.3504 - val
_tp: 32381.0000 - val_fp: 5004.0000 - val_tn: 35064.0000 - val_fn: 7551.00
00 - val_accuracy: 0.8431 - val_precision: 0.8661 - val_recall: 0.8109
Epoch 13/15
879/879 [==============================] - 26s 30ms/step - loss: 0.3445 -
tp: 379350.0000 - fp: 67276.0000 - tn: 382362.0000 - fn: 71108.0000 - accu
racy: 0.8463 - precision: 0.8494 - recall: 0.8421 - val_loss: 0.3462 - val
_tp: 33803.0000 - val_fp: 6161.0000 - val_tn: 33907.0000 - val_fn: 6129.00
00 - val_accuracy: 0.8464 - val_precision: 0.8458 - val_recall: 0.8465
Epoch 14/15
879/879 [==============================] - 26s 30ms/step - loss: 0.3424 -
tp: 379836.0000 - fp: 66788.0000 - tn: 382850.0000 - fn: 70622.0000 - accu
racy: 0.8473 - precision: 0.8505 - recall: 0.8432 - val_loss: 0.3449 - val
_tp: 33060.0000 - val_fp: 5446.0000 - val_tn: 34622.0000 - val_fn: 6872.00
00 - val_accuracy: 0.8460 - val_precision: 0.8586 - val_recall: 0.8279
Epoch 15/15
879/879 [==============================] - 26s 30ms/step - loss: 0.3404 -
tp: 380336.0000 - fp: 66398.0000 - tn: 383240.0000 - fn: 70122.0000 - accu
racy: 0.8483 - precision: 0.8514 - recall: 0.8443 - val_loss: 0.3486 - val
_tp: 32226.0000 - val_fp: 4706.0000 - val_tn: 35362.0000 - val_fn: 7706.00
00 - val_accuracy: 0.8449 - val_precision: 0.8726 - val_recall: 0.8070
```

In [19]:
```
# Reload
model_12_units = tf.keras.models.load_model('model_12_units.keras')
with open('history_12_units.json') as f:
    history_12_units = json.load(f)
```

In [20]:
```
#creat the plot
acc_12 = history_12_units['accuracy']
val_acc_12 = history_12_units['val_accuracy']
loss_12 = history_12_units['loss']
val_loss_12 = history_12_units['val_loss']

epochs = range(len(acc_12))

plt.plot(epochs, acc_12, 'ro', label='Training acc')
plt.plot(epochs, val_acc_12, 'b', label='Validation acc')
plt.title('12 LSTM units Training and validation accuracy')
plt.legend()

plt.figure()

plt.plot(epochs, loss_12, 'ro', label='Training loss')
plt.plot(epochs, val_loss_12, 'b', label='Validation loss')
plt.title('12 LSTM units training and validation loss')
plt.legend()

plt.show()
```





Comment:

When considering accuracy, validation began at above 0.78. Both training and validation followed a similar trend, increasing gradually and forming a slightly curved line. It was observed that there were slight fluctuations in validation after the 8th epoch, with both training and validation eventually reaching around 0.84. However, while the training accuracy kept increasing, the validation loss saw a slight decrease after the 12th epoch.

Regarding loss, validation commenced at around 0.45. Both training and validation exhibited a similar trend, gradually decreasing with a slight curvature, and eventually reaching below 0.36. However, while the training loss kept decreasing, the validation loss saw a slight increase in the end.

In [21]:
```
model_12_units.compile(metrics=fresh_metrics())
# Evaluate Model with 12 LSTM units
model_12_units_results = model_12_units.evaluate(test_data, return_dict=T
model_12_units_results
```

```
313/313 [==============================] - 9s 24ms/step - loss: 0.0000e+00
- tp: 129593.0000 - fp: 19000.0000 - tn: 140932.0000 - fn: 30475.0000 - ac
curacy: 0.8454 - precision: 0.8721 - recall: 0.8096
```

Out[21]:
```
{'loss': 0.0,
 'tp': 129593.0,
 'fp': 19000.0,
 'tn': 140932.0,
 'fn': 30475.0,
 'accuracy': 0.8453906178474426,
 'precision': 0.8721339702606201,
 'recall': 0.8096121549606323}
```

Compare and comment on the results

Regarding accuracy and precision, training with 12 LSTM modules showed slightly better results compared to training with 8 LSTM modules. The accuracy and precision values were 0.8454 and 0.8721, respectively, with 12 LSTM, while they were 0.8423 and 0.8342 with 8 LSTM modules.

However, in terms of recall, 8 LSTM modules achieved a better score than 12 LSTM modules (8 LSTM: 0.8546, 12 LSTM: 0.8096). In conclusion, from the perspective of accuracy, increasing the number of LSTM modules tends to improve performance.

Adjusting the size of the vocabulary

The third investigation is how size of the encoder's vocabulary impacts the performance of the model.

For your models you should:

- use a model with 4 LSTM modules
- use 25% of the training data
- train for 15 epochs

Create two models, one using an encoder with a 250-word vocabulary and the other with a 500-word vocabulary.

Q1(c)i Training with 250 vocab length

In [22]:

```
# put your code here

# Build the model with a 250-word vocabulary encoder
model_250_vocab = Sequential([
    encoder250,
    Embedding(input_dim=len(encoder250.get_vocabulary()), output_dim=EMBE
    LSTM(4),
    Dense(1, activation='sigmoid')
])

# Compile the model
model_250_vocab.compile(loss='binary_crossentropy',
                        optimizer='adam',
                        metrics=fresh_metrics())

# Train the model
history_250_vocab = model_250_vocab.fit(
    train_data25,
    validation_data=validation_data,
    epochs=15,
    verbose=1)

# Save the model and training history
model_250_vocab.save('model_250_vocab.keras')
with open('history_250_vocab.json', 'w') as f:
    json.dump(history_250_vocab.history, f)
```

```
Epoch 1/15
879/879 [==============================] - 59s 32ms/step - loss: 0.4995 -
tp: 333922.0000 - fp: 100491.0000 - tn: 349607.0000 - fn: 116076.0000 - ac
curacy: 0.7594 - precision: 0.7687 - recall: 0.7421 - val_loss: 0.4534 - v
al_tp: 30074.0000 - val_fp: 7338.0000 - val_tn: 32730.0000 - val_fn: 9858.
0000 - val_accuracy: 0.7850 - val_precision: 0.8039 - val_recall: 0.7531
Epoch 2/15
879/879 [==============================] - 27s 31ms/step - loss: 0.4383 -
tp: 351091.0000 - fp: 86219.0000 - tn: 363879.0000 - fn: 98907.0000 - accu
racy: 0.7943 - precision: 0.8028 - recall: 0.7802 - val_loss: 0.4276 - val
_tp: 29584.0000 - val_fp: 5546.0000 - val_tn: 34522.0000 - val_fn: 10348.0
000 - val_accuracy: 0.8013 - val_precision: 0.8421 - val_recall: 0.7409
Epoch 3/15
879/879 [==============================] - 27s 31ms/step - loss: 0.4158 -
tp: 359478.0000 - fp: 82201.0000 - tn: 367897.0000 - fn: 90520.0000 - accu
racy: 0.8081 - precision: 0.8139 - recall: 0.7988 - val_loss: 0.4112 - val
_tp: 33311.0000 - val_fp: 8574.0000 - val_tn: 31494.0000 - val_fn: 6621.00
00 - val_accuracy: 0.8101 - val_precision: 0.7953 - val_recall: 0.8342
Epoch 4/15
879/879 [==============================] - 28s 31ms/step - loss: 0.4055 -
tp: 363539.0000 - fp: 80618.0000 - tn: 369480.0000 - fn: 86459.0000 - accu
racy: 0.8144 - precision: 0.8185 - recall: 0.8079 - val_loss: 0.4034 - val
_tp: 33398.0000 - val_fp: 8272.0000 - val_tn: 31796.0000 - val_fn: 6534.00
00 - val_accuracy: 0.8149 - val_precision: 0.8015 - val_recall: 0.8364
Epoch 5/15
879/879 [==============================] - 27s 31ms/step - loss: 0.4009 -
tp: 365033.0000 - fp: 79892.0000 - tn: 370206.0000 - fn: 84965.0000 - accu
racy: 0.8168 - precision: 0.8204 - recall: 0.8112 - val_loss: 0.4004 - val
_tp: 31183.0000 - val_fp: 5917.0000 - val_tn: 34151.0000 - val_fn: 8749.00
00 - val_accuracy: 0.8167 - val_precision: 0.8405 - val_recall: 0.7809
Epoch 6/15
879/879 [==============================] - 27s 31ms/step - loss: 0.3973 -
tp: 366074.0000 - fp: 79245.0000 - tn: 370853.0000 - fn: 83924.0000 - accu
racy: 0.8187 - precision: 0.8220 - recall: 0.8135 - val_loss: 0.3983 - val
_tp: 30941.0000 - val_fp: 5670.0000 - val_tn: 34398.0000 - val_fn: 8991.00
00 - val_accuracy: 0.8167 - val_precision: 0.8451 - val_recall: 0.7748
Epoch 7/15
879/879 [==============================] - 27s 31ms/step - loss: 0.3944 -
tp: 366563.0000 - fp: 78266.0000 - tn: 371832.0000 - fn: 83435.0000 - accu
racy: 0.8204 - precision: 0.8241 - recall: 0.8146 - val_loss: 0.3945 - val
_tp: 31430.0000 - val_fp: 5956.0000 - val_tn: 34112.0000 - val_fn: 8502.00
00 - val_accuracy: 0.8193 - val_precision: 0.8407 - val_recall: 0.7871
Epoch 8/15
879/879 [==============================] - 27s 31ms/step - loss: 0.3923 -
tp: 366942.0000 - fp: 77701.0000 - tn: 372397.0000 - fn: 83056.0000 - accu
racy: 0.8214 - precision: 0.8253 - recall: 0.8154 - val_loss: 0.3900 - val
_tp: 32422.0000 - val_fp: 6734.0000 - val_tn: 33334.0000 - val_fn: 7510.00
00 - val_accuracy: 0.8220 - val_precision: 0.8280 - val_recall: 0.8119
Epoch 9/15
879/879 [==============================] - 27s 30ms/step - loss: 0.3904 -
tp: 367517.0000 - fp: 77448.0000 - tn: 372650.0000 - fn: 82481.0000 - accu
racy: 0.8223 - precision: 0.8259 - recall: 0.8167 - val_loss: 0.3918 - val
_tp: 33661.0000 - val_fp: 8105.0000 - val_tn: 31963.0000 - val_fn: 6271.00
00 - val_accuracy: 0.8203 - val_precision: 0.8059 - val_recall: 0.8430
Epoch 10/15
879/879 [==============================] - 27s 31ms/step - loss: 0.3889 -
tp: 368003.0000 - fp: 77195.0000 - tn: 372903.0000 - fn: 81995.0000 - accu
racy: 0.8231 - precision: 0.8266 - recall: 0.8178 - val_loss: 0.3887 - val
_tp: 33106.0000 - val_fp: 7329.0000 - val_tn: 32739.0000 - val_fn: 6826.00
00 - val_accuracy: 0.8231 - val_precision: 0.8187 - val_recall: 0.8291
```

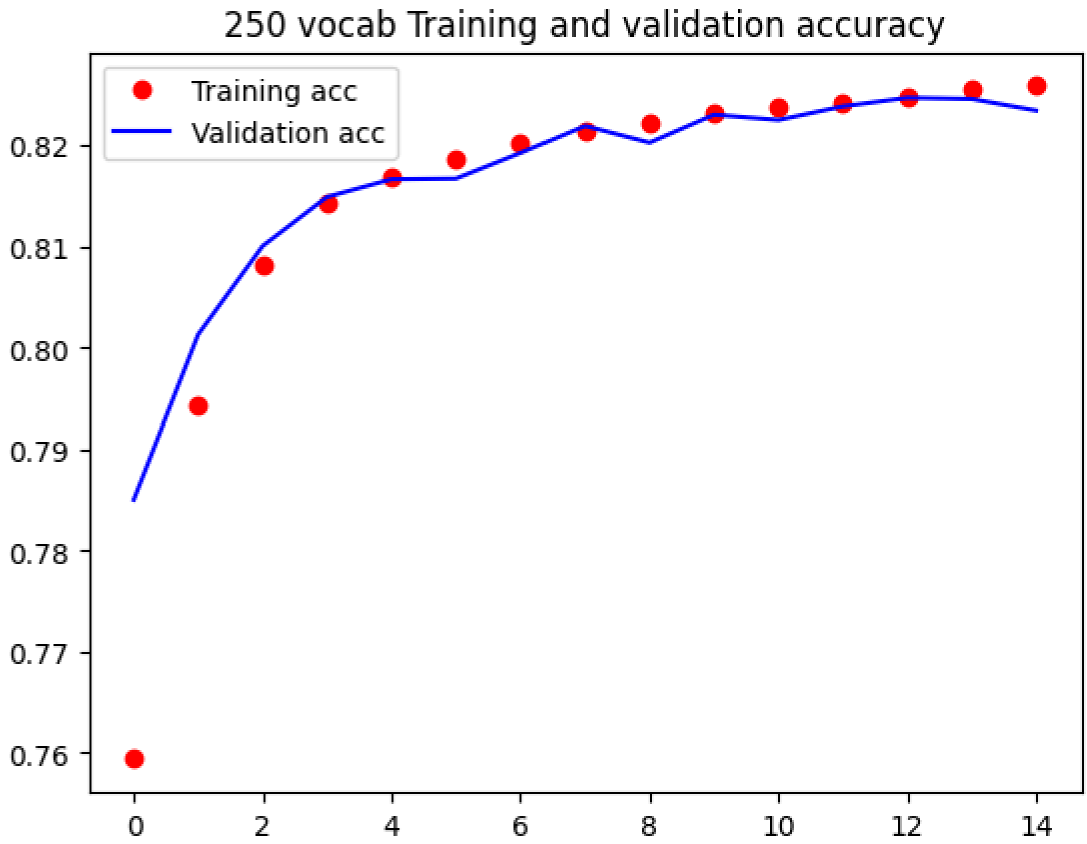
```
Epoch 11/15
879/879 [==============================] - 27s 30ms/step - loss: 0.3877 -
tp: 368425.0000 - fp: 76914.0000 - tn: 373184.0000 - fn: 81573.0000 - accu
racy: 0.8239 - precision: 0.8273 - recall: 0.8187 - val_loss: 0.3878 - val
_tp: 31814.0000 - val_fp: 6079.0000 - val_tn: 33989.0000 - val_fn: 8118.00
00 - val_accuracy: 0.8225 - val_precision: 0.8396 - val_recall: 0.7967
Epoch 12/15
879/879 [==============================] - 27s 31ms/step - loss: 0.3866 -
tp: 368540.0000 - fp: 76784.0000 - tn: 373314.0000 - fn: 81458.0000 - accu
racy: 0.8242 - precision: 0.8276 - recall: 0.8190 - val_loss: 0.3867 - val
_tp: 32266.0000 - val_fp: 6423.0000 - val_tn: 33645.0000 - val_fn: 7666.00
00 - val_accuracy: 0.8239 - val_precision: 0.8340 - val_recall: 0.8080
Epoch 13/15
879/879 [==============================] - 28s 31ms/step - loss: 0.3857 -
tp: 369089.0000 - fp: 76754.0000 - tn: 373344.0000 - fn: 80909.0000 - accu
racy: 0.8248 - precision: 0.8278 - recall: 0.8202 - val_loss: 0.3855 - val
_tp: 33232.0000 - val_fp: 7321.0000 - val_tn: 32747.0000 - val_fn: 6700.00
00 - val_accuracy: 0.8247 - val_precision: 0.8195 - val_recall: 0.8322
Epoch 14/15
879/879 [==============================] - 28s 31ms/step - loss: 0.3845 -
tp: 369454.0000 - fp: 76452.0000 - tn: 373646.0000 - fn: 80544.0000 - accu
racy: 0.8256 - precision: 0.8285 - recall: 0.8210 - val_loss: 0.3866 - val
_tp: 33693.0000 - val_fp: 7791.0000 - val_tn: 32277.0000 - val_fn: 6239.00
00 - val_accuracy: 0.8246 - val_precision: 0.8122 - val_recall: 0.8438
Epoch 15/15
879/879 [==============================] - 27s 31ms/step - loss: 0.3838 -
tp: 369563.0000 - fp: 76265.0000 - tn: 373833.0000 - fn: 80435.0000 - accu
racy: 0.8259 - precision: 0.8289 - recall: 0.8213 - val_loss: 0.3883 - val
_tp: 34048.0000 - val_fp: 8238.0000 - val_tn: 31830.0000 - val_fn: 5884.00
00 - val_accuracy: 0.8235 - val_precision: 0.8052 - val_recall: 0.8526
```

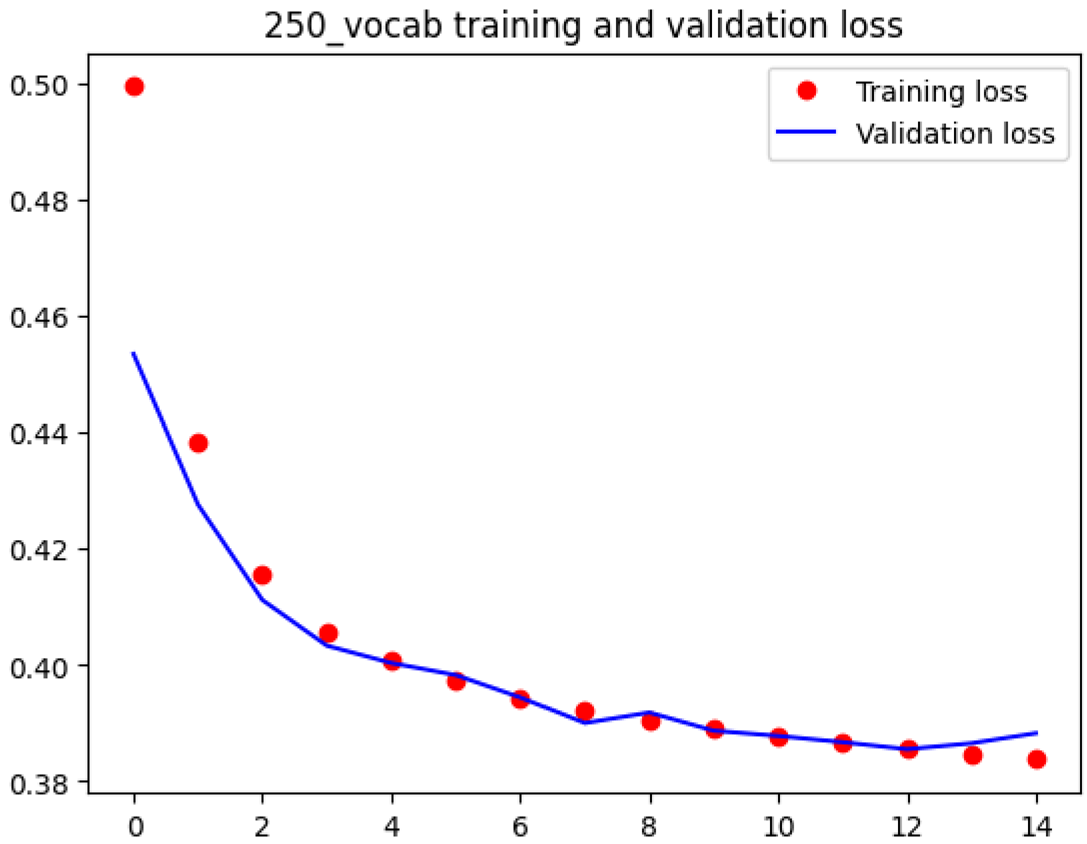
In [23]:

```python
# Reload
model_250_vocab = tf.keras.models.load_model('model_250_vocab.keras')
with open('history_250_vocab.json') as f:
    history_250_vocab = json.load(f)
```

In [24]:

```python
# Create plots
acc_250_vocab = history_250_vocab['accuracy']
val_acc_250_vocab = history_250_vocab['val_accuracy']
loss_250_vocab = history_250_vocab['loss']
val_loss_250_vocab = history_250_vocab['val_loss']

epochs = range(len(acc_250_vocab))

plt.plot(epochs, acc_250_vocab, 'ro', label='Training acc')
plt.plot(epochs, val_acc_250_vocab, 'b', label='Validation acc')
plt.title('250 vocab Training and validation accuracy')
plt.legend()

plt.figure()

plt.plot(epochs, loss_250_vocab, 'ro', label='Training loss')
plt.plot(epochs, val_loss_250_vocab, 'b', label='Validation loss')
plt.title('250_vocab training and validation loss')
plt.legend()

plt.show()
```





Comment:

When considering accuracy, validation began above 0.78. Both training and validation followed a similar trend, increasing dramatically up to the 4th epoch before gradually increasing towards the end. Both training and validation eventually reached around

0.82. However, while the training accuracy kept increasing, the validation loss saw a slight decrease after the 13th epoch.

Regarding loss, validation commenced below 0.46. Both training and validation exhibited a similar trend, dramatically decreasing up to the 3rd epoch before gradually decreasing towards the end. Eventually, they reached around 0.38. However, while the training loss kept decreasing, the validation loss saw a slight increase in the end.

In [25]:
```
model_250_vocab.compile(metrics=fresh_metrics())
# Evaluate Model with 250 vocab
model_250_vocab_results = model_250_vocab.evaluate(test_data, return_dict
model_250_vocab_results
```

```
313/313 [==============================] - 8s 21ms/step - loss: 0.0000e+00
- tp: 136861.0000 - fp: 33217.0000 - tn: 126715.0000 - fn: 23207.0000 - ac
curacy: 0.8237 - precision: 0.8047 - recall: 0.8550
```

Out[25]:
```
{'loss': 0.0,
 'tp': 136861.0,
 'fp': 33217.0,
 'tn': 126715.0,
 'fn': 23207.0,
 'accuracy': 0.8236749768257141,
 'precision': 0.8046954870223999,
 'recall': 0.8550178408622742}
```

Training with 500 vocab length

In [26]:
```
# put your code here
# Build the model with a 500-word vocabulary encoder
model_500_vocab = Sequential([
    encoder500,
    Embedding(input_dim=len(encoder500.get_vocabulary()), output_dim=EMBE
    LSTM(4),
    Dense(1, activation='sigmoid')
])

# Compile the model
model_500_vocab.compile(loss='binary_crossentropy',
                        optimizer='adam',
                        metrics=fresh_metrics())

# Train the model
history_500_vocab = model_500_vocab.fit(
    train_data25,
    validation_data=validation_data,
    epochs=15,
    verbose=1)

# Save the model and training history
model_500_vocab.save('model_500_vocab.keras')
with open('history_500_vocab.json', 'w') as f:
    json.dump(history_500_vocab.history, f)
```

```
Epoch 1/15
879/879 [==============================] - 35s 34ms/step - loss: 0.4608 -
tp: 359352.0000 - fp: 93640.0000 - tn: 356458.0000 - fn: 90646.0000 - accu
racy: 0.7953 - precision: 0.7933 - recall: 0.7986 - val_loss: 0.3968 - val
_tp: 32725.0000 - val_fp: 6920.0000 - val_tn: 33148.0000 - val_fn: 7207.00
00 - val_accuracy: 0.8234 - val_precision: 0.8255 - val_recall: 0.8195
Epoch 2/15
879/879 [==============================] - 28s 32ms/step - loss: 0.3806 -
tp: 375580.0000 - fp: 77698.0000 - tn: 372400.0000 - fn: 74418.0000 - accu
racy: 0.8310 - precision: 0.8286 - recall: 0.8346 - val_loss: 0.3668 - val
_tp: 34502.0000 - val_fp: 7519.0000 - val_tn: 32549.0000 - val_fn: 5430.00
00 - val_accuracy: 0.8381 - val_precision: 0.8211 - val_recall: 0.8640
Epoch 3/15
879/879 [==============================] - 28s 31ms/step - loss: 0.3565 -
tp: 380542.0000 - fp: 70966.0000 - tn: 379132.0000 - fn: 69456.0000 - accu
racy: 0.8440 - precision: 0.8428 - recall: 0.8457 - val_loss: 0.3512 - val
_tp: 35389.0000 - val_fp: 7844.0000 - val_tn: 32224.0000 - val_fn: 4543.00
00 - val_accuracy: 0.8452 - val_precision: 0.8186 - val_recall: 0.8862
Epoch 4/15
879/879 [==============================] - 27s 31ms/step - loss: 0.3417 -
tp: 383811.0000 - fp: 67792.0000 - tn: 382306.0000 - fn: 66187.0000 - accu
racy: 0.8512 - precision: 0.8499 - recall: 0.8529 - val_loss: 0.3384 - val
_tp: 33043.0000 - val_fp: 4901.0000 - val_tn: 35167.0000 - val_fn: 6889.00
00 - val_accuracy: 0.8526 - val_precision: 0.8708 - val_recall: 0.8275
Epoch 5/15
879/879 [==============================] - 27s 31ms/step - loss: 0.3327 -
tp: 385405.0000 - fp: 65320.0000 - tn: 384778.0000 - fn: 64593.0000 - accu
racy: 0.8557 - precision: 0.8551 - recall: 0.8565 - val_loss: 0.3318 - val
_tp: 33138.0000 - val_fp: 4724.0000 - val_tn: 35344.0000 - val_fn: 6794.00
00 - val_accuracy: 0.8560 - val_precision: 0.8752 - val_recall: 0.8299
Epoch 6/15
879/879 [==============================] - 27s 31ms/step - loss: 0.3261 -
tp: 386714.0000 - fp: 63484.0000 - tn: 386614.0000 - fn: 63284.0000 - accu
racy: 0.8592 - precision: 0.8590 - recall: 0.8594 - val_loss: 0.3270 - val
_tp: 33323.0000 - val_fp: 4630.0000 - val_tn: 35438.0000 - val_fn: 6609.00
00 - val_accuracy: 0.8595 - val_precision: 0.8780 - val_recall: 0.8345
Epoch 7/15
879/879 [==============================] - 27s 31ms/step - loss: 0.3216 -
tp: 387637.0000 - fp: 62151.0000 - tn: 387947.0000 - fn: 62361.0000 - accu
racy: 0.8617 - precision: 0.8618 - recall: 0.8614 - val_loss: 0.3236 - val
_tp: 35197.0000 - val_fp: 6399.0000 - val_tn: 33669.0000 - val_fn: 4735.00
00 - val_accuracy: 0.8608 - val_precision: 0.8462 - val_recall: 0.8814
Epoch 8/15
879/879 [==============================] - 27s 31ms/step - loss: 0.3182 -
tp: 388319.0000 - fp: 61364.0000 - tn: 388734.0000 - fn: 61679.0000 - accu
racy: 0.8633 - precision: 0.8635 - recall: 0.8629 - val_loss: 0.3208 - val
_tp: 33275.0000 - val_fp: 4407.0000 - val_tn: 35661.0000 - val_fn: 6657.00
00 - val_accuracy: 0.8617 - val_precision: 0.8830 - val_recall: 0.8333
Epoch 9/15
879/879 [==============================] - 27s 31ms/step - loss: 0.3149 -
tp: 388765.0000 - fp: 60342.0000 - tn: 389756.0000 - fn: 61233.0000 - accu
racy: 0.8649 - precision: 0.8656 - recall: 0.8639 - val_loss: 0.3153 - val
_tp: 34488.0000 - val_fp: 5351.0000 - val_tn: 34717.0000 - val_fn: 5444.00
00 - val_accuracy: 0.8651 - val_precision: 0.8657 - val_recall: 0.8637
Epoch 10/15
879/879 [==============================] - 28s 32ms/step - loss: 0.3126 -
tp: 389234.0000 - fp: 59752.0000 - tn: 390346.0000 - fn: 60764.0000 - accu
racy: 0.8661 - precision: 0.8669 - recall: 0.8650 - val_loss: 0.3135 - val
_tp: 34394.0000 - val_fp: 5210.0000 - val_tn: 34858.0000 - val_fn: 5538.00
00 - val_accuracy: 0.8656 - val_precision: 0.8684 - val_recall: 0.8613
```

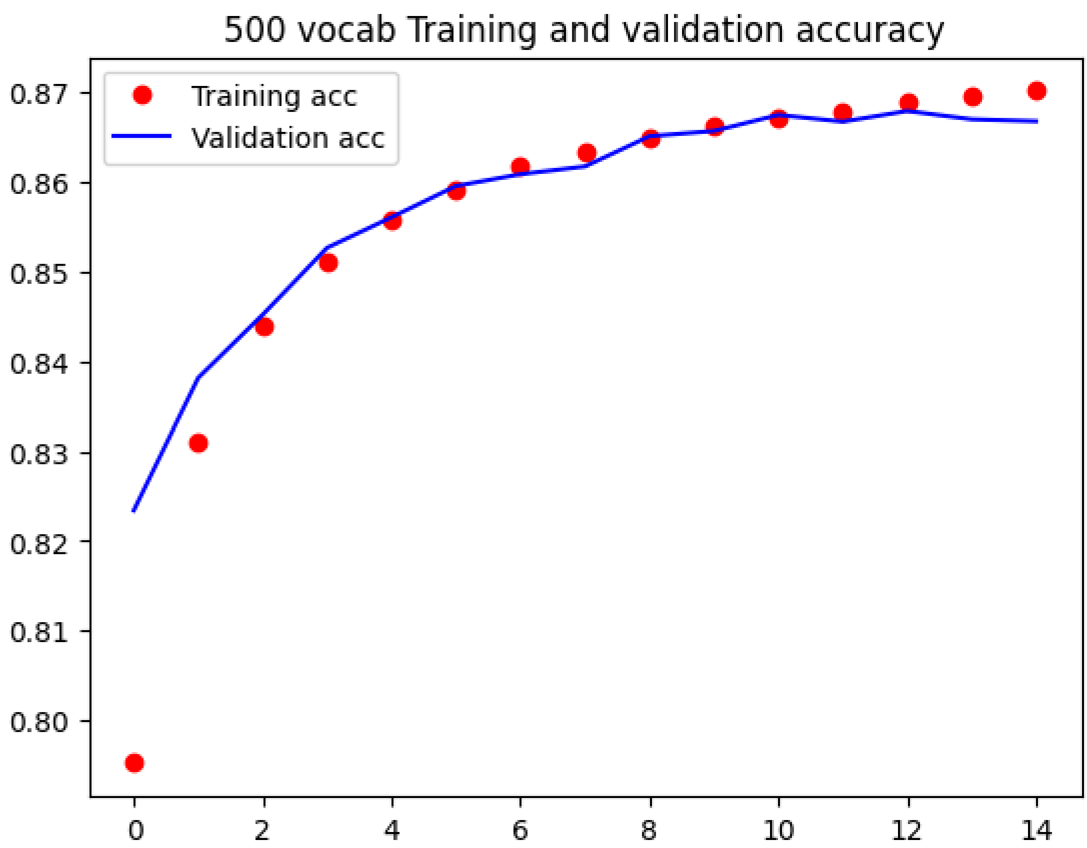
```
Epoch 11/15
879/879 [==============================] - 28s 32ms/step - loss: 0.3102 -
tp: 389578.0000 - fp: 59122.0000 - tn: 390976.0000 - fn: 60420.0000 - accu
racy: 0.8672 - precision: 0.8682 - recall: 0.8657 - val_loss: 0.3118 - val
_tp: 34757.0000 - val_fp: 5430.0000 - val_tn: 34638.0000 - val_fn: 5175.00
00 - val_accuracy: 0.8674 - val_precision: 0.8649 - val_recall: 0.8704
Epoch 12/15
879/879 [==============================] - 28s 31ms/step - loss: 0.3086 -
tp: 389707.0000 - fp: 58724.0000 - tn: 391374.0000 - fn: 60291.0000 - accu
racy: 0.8678 - precision: 0.8690 - recall: 0.8660 - val_loss: 0.3126 - val
_tp: 33646.0000 - val_fp: 4379.0000 - val_tn: 35689.0000 - val_fn: 6286.00
00 - val_accuracy: 0.8667 - val_precision: 0.8848 - val_recall: 0.8426
Epoch 13/15
879/879 [==============================] - 27s 31ms/step - loss: 0.3070 -
tp: 390405.0000 - fp: 58342.0000 - tn: 391756.0000 - fn: 59593.0000 - accu
racy: 0.8690 - precision: 0.8700 - recall: 0.8676 - val_loss: 0.3100 - val
_tp: 34027.0000 - val_fp: 4665.0000 - val_tn: 35403.0000 - val_fn: 5905.00
00 - val_accuracy: 0.8679 - val_precision: 0.8794 - val_recall: 0.8521
Epoch 14/15
879/879 [==============================] - 27s 31ms/step - loss: 0.3055 -
tp: 390751.0000 - fp: 58099.0000 - tn: 391999.0000 - fn: 59247.0000 - accu
racy: 0.8696 - precision: 0.8706 - recall: 0.8683 - val_loss: 0.3114 - val
_tp: 35604.0000 - val_fp: 6316.0000 - val_tn: 33752.0000 - val_fn: 4328.00
00 - val_accuracy: 0.8669 - val_precision: 0.8493 - val_recall: 0.8916
Epoch 15/15
879/879 [==============================] - 28s 31ms/step - loss: 0.3045 -
tp: 390867.0000 - fp: 57778.0000 - tn: 392320.0000 - fn: 59131.0000 - accu
racy: 0.8701 - precision: 0.8712 - recall: 0.8686 - val_loss: 0.3112 - val
_tp: 35825.0000 - val_fp: 6554.0000 - val_tn: 33514.0000 - val_fn: 4107.00
00 - val_accuracy: 0.8667 - val_precision: 0.8453 - val_recall: 0.8972
```

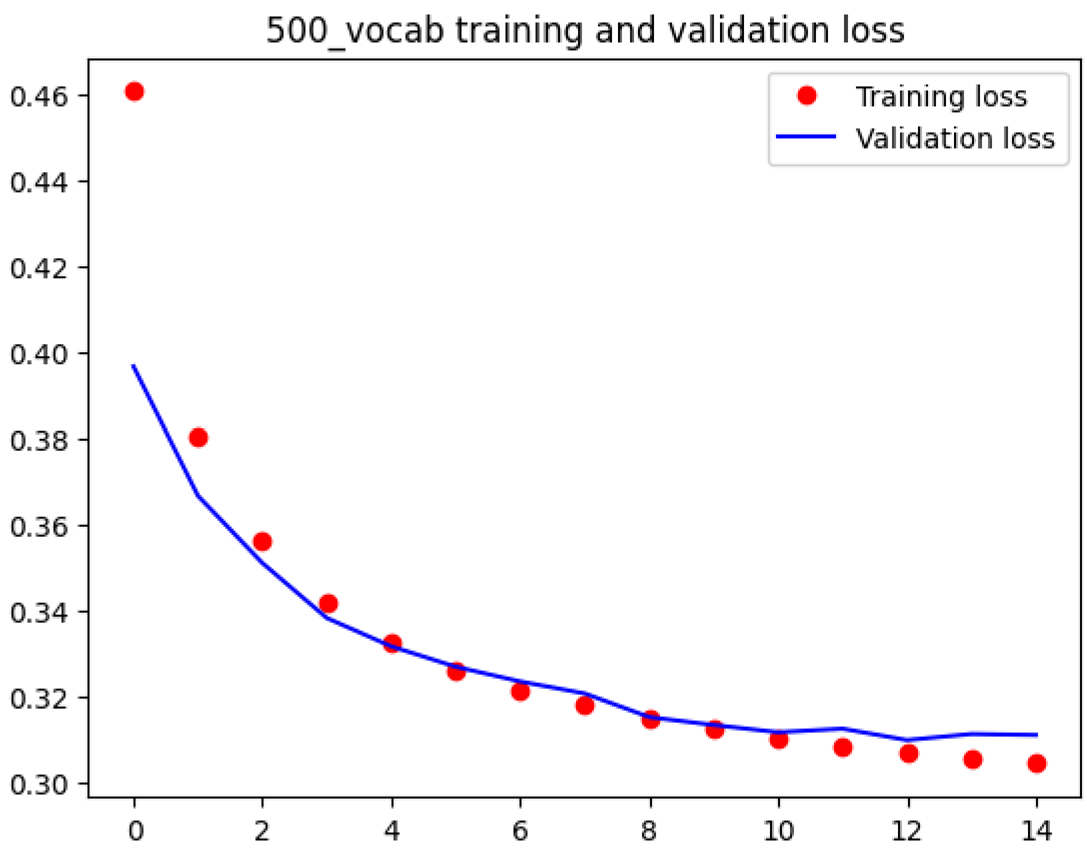
In [27]:

```
# Reload
model_500_vocab = tf.keras.models.load_model('model_500_vocab.keras')
with open('history_500_vocab.json') as f:
    history_500_vocab = json.load(f)
```

In [28]:

```
# Create the plot
acc_500_vocab = history_500_vocab['accuracy']
val_acc_500_vocab = history_500_vocab['val_accuracy']
loss_500_vocab = history_500_vocab['loss']
val_loss_500_vocab = history_500_vocab['val_loss']

epochs = range(len(acc_500_vocab))

plt.plot(epochs, acc_500_vocab, 'ro', label='Training acc')
plt.plot(epochs, val_acc_500_vocab, 'b', label='Validation acc')
plt.title('500 vocab Training and validation accuracy')
plt.legend()

plt.figure()

plt.plot(epochs, loss_500_vocab, 'ro', label='Training loss')
plt.plot(epochs, val_loss_500_vocab, 'b', label='Validation loss')
plt.title('500_vocab training and validation loss')
plt.legend()

plt.show()
```





Comment:

When considering accuracy, validation began above 0.82. Both training and validation followed a similar trend, increasing gradually and drawing a curve line. Both training and validation eventually reached around 0.87. However, while the training accuracy kept increasing, the validation loss saw a slight decrease after the 12th epoch.

Regarding loss, validation commenced below 0.40. Both training and validation exhibited a similar trend, gradually decreasing and drawing a curve. Eventually, they reached around 0.31. However, while the training loss kept decreasing, the validation loss saw a slight stayble trend in the end.

In [29]:
```
model_500_vocab.compile(metrics=fresh_metrics())
# Evaluate Model with 500 vocab
model_500_vocab_results = model_500_vocab.evaluate(test_data, return_dict
model_500_vocab_results
```

```
313/313 [==============================] - 8s 22ms/step - loss: 0.0000e+00
- tp: 143699.0000 - fp: 26633.0000 - tn: 133299.0000 - fn: 16369.0000 - ac
curacy: 0.8656 - precision: 0.8436 - recall: 0.8977
```

Out[29]:
```
{'loss': 0.0,
 'tp': 143699.0,
 'fp': 26633.0,
 'tn': 133299.0,
 'fn': 16369.0,
 'accuracy': 0.8656187653541565,
 'precision': 0.8436406254768372,
 'recall': 0.8977372050285339}
```

Compare and comment on the results

Comparing accuracy, precision, and recall, training with a vocabulary length of 500 performed better than training with a vocabulary length of 250. While the accuracy, precision, and recall with a vocabulary length of 250 were 0.8237, 0.8047, and 0.8550, respectively, those with a vocabulary length of 500 were 0.8237, 0.8047, and 0.8550, respectively. Therefore, increasing the vocabulary length resulted in better performance.

The best model consideration

What would make the best model?

The experiments so far suggest that increasing the volume of the dataset, the number of LSTM modules, and the length of the vocabulary for an encoder lead to better performance. Therefore, it is expected that the best model is the one using 12 LSTM modules and an encoder with a 500-word vocabulary trained on the full dataset.

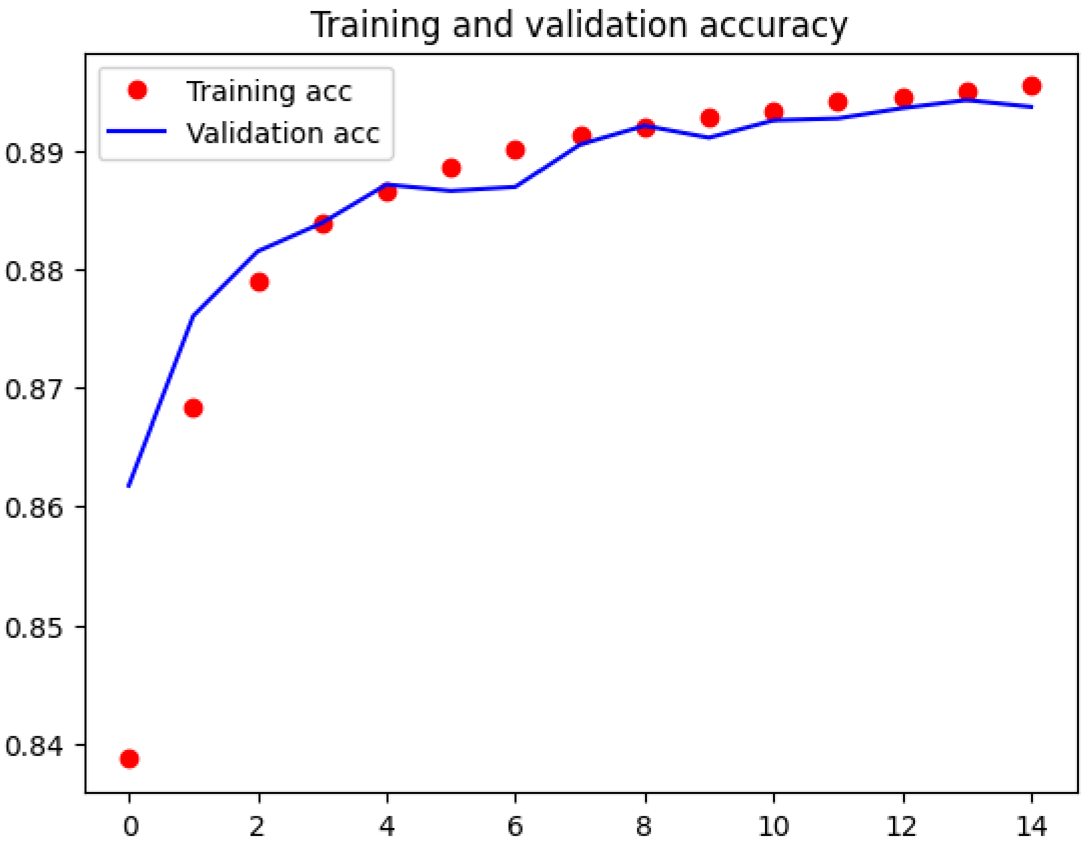
Create and train the model

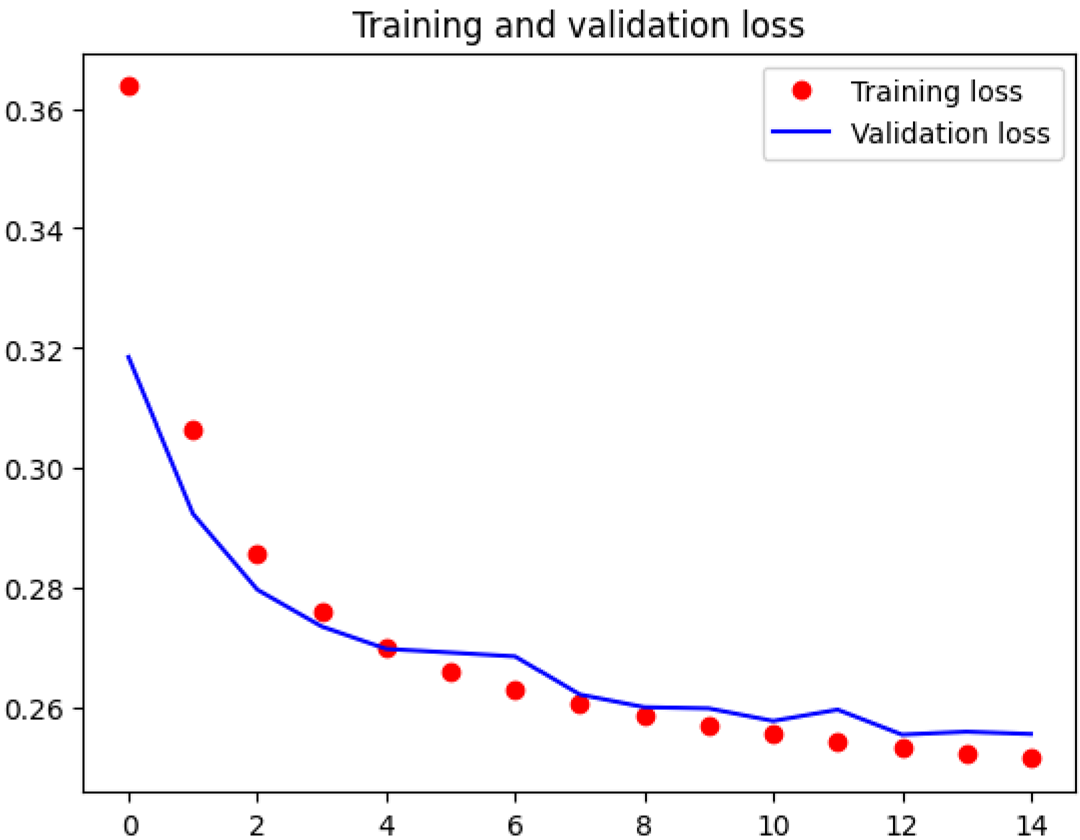
In [ ]:
```
# Put your solution here


# Build the model with a 500-word vocabulary encoder and 12 LSTM modules
model_500_vocab_12_lstm = Sequential([
    encoder500,
```

```
    Embedding(input_dim=len(encoder500.get_vocabulary()), output_dim=EMBE
    LSTM(12),
    Dense(1, activation='sigmoid')
])

# Compile the model
model_500_vocab_12_lstm.compile(loss='binary_crossentropy',
                                optimizer='adam',
                                metrics=fresh_metrics())

# Train the model
history_500_vocab_12_lstm = model_500_vocab_12_lstm.fit(
    train_data,  # Using all training data
    validation_data=validation_data,
    epochs=15,
    verbose=0)

# Save the model and training history
model_500_vocab_12_lstm.save('model_500_vocab_12_lstm.keras')
with open('history_500_vocab_12_lstm.json', 'w') as f:
    json.dump(history_500_vocab_12_lstm.history, f)
```

In [31]:
```
# Reload
model_500_vocab_12_lstm = tf.keras.models.load_model('model_500_vocab_12_
with open('history_500_vocab_12_lstm.json') as f:
    history_500_vocab_12_lstm = json.load(f)
```

In [32]:
```
# crete plots
acc_q1d = history_500_vocab_12_lstm['accuracy']
val_acc_q1d = history_500_vocab_12_lstm['val_accuracy']
loss_q1d = history_500_vocab_12_lstm['loss']
val_loss_q1d = history_500_vocab_12_lstm['val_loss']

epochs = range(len(acc_q1d))

plt.plot(epochs, acc_q1d, 'ro', label='Training acc')
plt.plot(epochs, val_acc_q1d, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.legend()

plt.figure()

plt.plot(epochs, loss_q1d, 'ro', label='Training loss')
plt.plot(epochs, val_loss_q1d, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()

plt.show()
```





Comment:

When it comes to accuracy, validation started around 0.86. Both training and validation followed a similar trend, with a slight dip observed at the 6th epoch in the validation set. Despite this, both training and validation accuracies increased steadily, forming a curvy line and eventually reaching around 0.89.

Regarding loss, validation started around 0.32. Both training and validation exhibited a similar trend, with a slight protrusion seen at the 6th epoch and 11th epoch in the validation set. However, both training and validation losses decreased steadily, forming a curvy line with slight fluctuations, and eventually reached below 0.26.

In [33]:
```
model_500_vocab_12_lstm.compile(metrics=fresh_metrics())
# Evaluate Model
model_500_vocab_12_lstm_results = model_500_vocab_12_lstm.evaluate(test_d
model_500_vocab_12_lstm_results
```

```
313/313 [==============================] - 8s 22ms/step - loss: 0.0000e+00
- tp: 140625.0000 - fp: 14647.0000 - tn: 145285.0000 - fn: 19443.0000 - ac
curacy: 0.8935 - precision: 0.9057 - recall: 0.8785
```

Out[33]:
```
{'loss': 0.0,
 'tp': 140625.0,
 'fp': 14647.0,
 'tn': 145285.0,
 'fn': 19443.0,
 'accuracy': 0.8934687376022339,
 'precision': 0.9056687355041504,
 'recall': 0.878532886505127}
```

Comparison

As anticipated, the model exhibited the best performance in terms of accuracy and precision, achieving scores of 0.8935 and 0.9057, respectively. However, the recall of 0.8785 from part (d) is lower than that of the model using an encoder with a 500-word vocabulary, 4 LSTM modules, and 25% of the training dataset, which achieved a recall of 0.8977.

Despite the slightly lower recall, as the last model attained the highest accuracy score, it is concluded that this is the best-performing model.

In [ ]:
```

```