# Assignment 8: Text Classification 2

Name: Aarya Patil

Date: 04/20/23

---

> Data set used: [SMS Spam Collection (Text Classification)

](https://www.kaggle.com/datasets/thedevastator/sms-spam-collection-a-more-diverse-dataset)

In [ ]:
```
# Import libraries
import pandas as pd
import seaborn as sb
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras import layers, models
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import accuracy_score, precision_score, recall_score, f
```

In [ ]:
```
# Save the data file name into a variable
input_file = "text_classification_data.csv"
data = pd.read_csv(input_file, header = 0) #load in data
print(data.head())
```

```
                                                 sms  label
0  Go until jurong point, crazy.. Available only ...      0
1                      Ok lar... Joking wif u oni...\n      0
2  Free entry in 2 a wkly comp to win FA Cup fina...      1
3  U dun say so early hor... U c already then say...      0
4  Nah I don't think he goes to usf, he lives aro...      0
```

In [ ]:
```
# Split into train and test data
i = np.random.rand(len(data)) < 0.8
train = data[i]
test = data[~i]
print("Train data size: ", train.shape)
print("Test data size: ", test.shape)
```

```
Train data size:  (4455, 2)
Test data size:  (1119, 2)
```

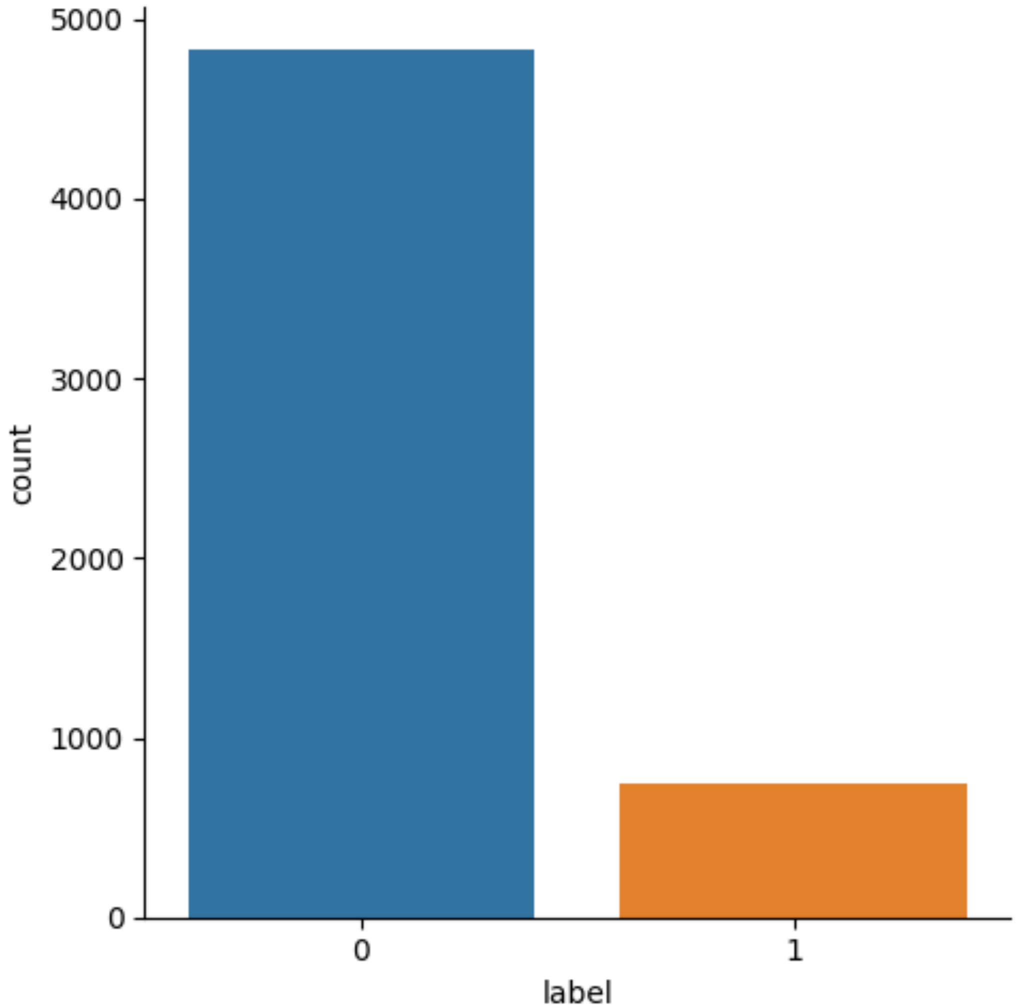
In [ ]:
```
# Set up X and Y
vocab_size = 25000
batch_size = 100
num_labels = 2

# Fit the tokenizer
tokenizer = Tokenizer(num_words = vocab_size)
tokenizer.fit_on_texts(train.sms)

encoder = LabelEncoder()
encoder.fit(train.label)

X_train = tokenizer.texts_to_matrix(train.sms, mode = 'tfidf')
X_test = tokenizer.texts_to_matrix(test.sms, mode = 'tfidf')

y_train = encoder.transform(train.label)
y_test = encoder.transform(test.label)

# Print out shape
print("X-train shape:", X_train.shape)
print("y-train shape:", y_train.shape)
print("X_test shape:", X_test.shape)
print("y_test shape:", y_test.shape)
print("First five test labels:", y_test[:5])
```

```
X-train shape: (4455, 25000)
y-train shape: (4455,)
X_test shape: (1119, 25000)
y_test shape: (1119,)
First five test labels: [0 1 0 1 0]
```

## Create a graph showing the distribution of the target classes

In [ ]:
```
# Create a graph showing the distribution of the target classes
sb.catplot(x = "label", kind = 'count', data = data)
```

Out[ ]:
```
<seaborn.axisgrid.FacetGrid at 0x7f3cdb74ecd0>
```




## Describe the data set and what the model should be able to predict

This data set holds SMS labeled messages that have been collected for mobile phone spam research. Messages that are not spam are labeled as '0' and messages that are spam are labeled as '1'.

The model should be able to predict whether a message is spam or not.

## Create a sequential model and evaluate on the test data

In [ ]:
```
# Fit model
model = models.Sequential()

model.add(layers.Dense(32, input_dim = vocab_size, kernel_initializer = 'nor
model.add(layers.Dense(1, kernel_initializer = 'normal', activation = 'sigmo

model.compile(loss = 'binary_crossentropy', optimizer = 'adam', metrics = ['

model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_4 (Dense)             (None, 32)                800032

 dense_5 (Dense)             (None, 1)                 33

=================================================================
Total params: 800,065
Trainable params: 800,065
Non-trainable params: 0
_________________________________________________________________
```

In [ ]:
```
history = model.fit(X_train, y_train, batch_size = batch_size, epochs = 30,
```

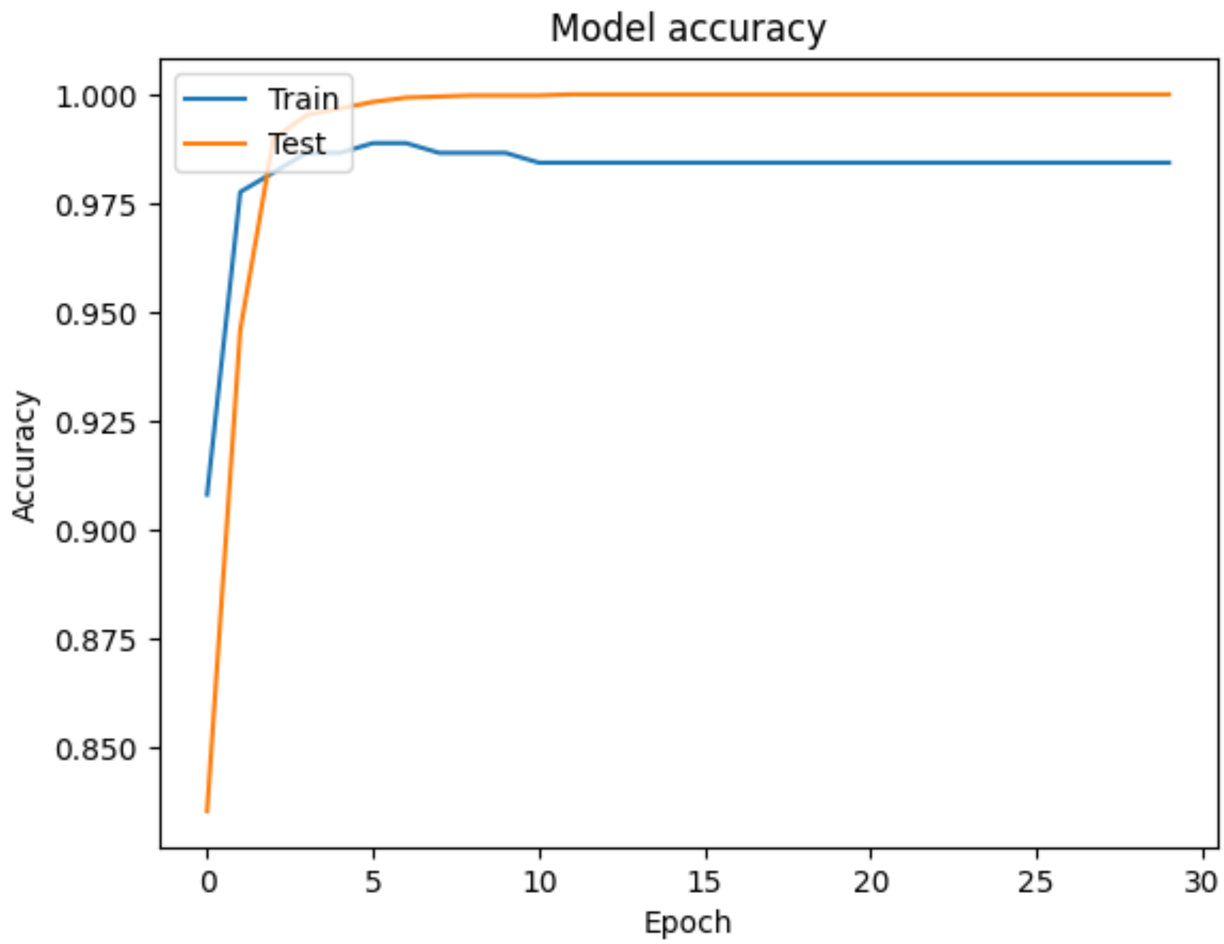
```
Epoch 1/30
41/41 [==============================] - 4s 50ms/step - loss: 0.5295 - accu
racy: 0.8354 - val_loss: 0.3653 - val_accuracy: 0.9081
Epoch 2/30
41/41 [==============================] - 1s 26ms/step - loss: 0.2467 - accu
racy: 0.9459 - val_loss: 0.1758 - val_accuracy: 0.9776
Epoch 3/30
41/41 [==============================] - 1s 30ms/step - loss: 0.1048 - accu
racy: 0.9898 - val_loss: 0.1046 - val_accuracy: 0.9821
Epoch 4/30
41/41 [==============================] - 1s 26ms/step - loss: 0.0543 - accu
racy: 0.9953 - val_loss: 0.0822 - val_accuracy: 0.9865
Epoch 5/30
41/41 [==============================] - 1s 24ms/step - loss: 0.0330 - accu
racy: 0.9968 - val_loss: 0.0733 - val_accuracy: 0.9865
Epoch 6/30
41/41 [==============================] - 1s 24ms/step - loss: 0.0223 - accu
racy: 0.9983 - val_loss: 0.0685 - val_accuracy: 0.9888
Epoch 7/30
41/41 [==============================] - 1s 25ms/step - loss: 0.0161 - accu
racy: 0.9993 - val_loss: 0.0670 - val_accuracy: 0.9888
Epoch 8/30
41/41 [==============================] - 1s 24ms/step - loss: 0.0122 - accu
racy: 0.9995 - val_loss: 0.0657 - val_accuracy: 0.9865
```

```
Epoch 9/30
41/41 [==============================] - 1s 25ms/step - loss: 0.0095 - accu
racy: 0.9998 - val_loss: 0.0649 - val_accuracy: 0.9865
Epoch 10/30
41/41 [==============================] - 1s 30ms/step - loss: 0.0075 - accu
racy: 0.9998 - val_loss: 0.0652 - val_accuracy: 0.9865
Epoch 11/30
41/41 [==============================] - 1s 34ms/step - loss: 0.0060 - accu
racy: 0.9998 - val_loss: 0.0655 - val_accuracy: 0.9843
Epoch 12/30
41/41 [==============================] - 2s 38ms/step - loss: 0.0049 - accu
racy: 1.0000 - val_loss: 0.0661 - val_accuracy: 0.9843
Epoch 13/30
41/41 [==============================] - 1s 31ms/step - loss: 0.0041 - accu
racy: 1.0000 - val_loss: 0.0670 - val_accuracy: 0.9843
Epoch 14/30
41/41 [==============================] - 1s 28ms/step - loss: 0.0035 - accu
racy: 1.0000 - val_loss: 0.0678 - val_accuracy: 0.9843
Epoch 15/30
41/41 [==============================] - 1s 32ms/step - loss: 0.0030 - accu
racy: 1.0000 - val_loss: 0.0687 - val_accuracy: 0.9843
Epoch 16/30
41/41 [==============================] - 1s 30ms/step - loss: 0.0027 - accu
racy: 1.0000 - val_loss: 0.0695 - val_accuracy: 0.9843
Epoch 17/30
41/41 [==============================] - 1s 31ms/step - loss: 0.0023 - accu
racy: 1.0000 - val_loss: 0.0704 - val_accuracy: 0.9843
Epoch 18/30
41/41 [==============================] - 1s 26ms/step - loss: 0.0021 - accu
racy: 1.0000 - val_loss: 0.0713 - val_accuracy: 0.9843
Epoch 19/30
41/41 [==============================] - 1s 26ms/step - loss: 0.0019 - accu
racy: 1.0000 - val_loss: 0.0722 - val_accuracy: 0.9843
Epoch 20/30
41/41 [==============================] - 1s 25ms/step - loss: 0.0017 - accu
racy: 1.0000 - val_loss: 0.0729 - val_accuracy: 0.9843
Epoch 21/30
41/41 [==============================] - 1s 25ms/step - loss: 0.0015 - accu
racy: 1.0000 - val_loss: 0.0737 - val_accuracy: 0.9843
Epoch 22/30
41/41 [==============================] - 1s 33ms/step - loss: 0.0014 - accu
racy: 1.0000 - val_loss: 0.0743 - val_accuracy: 0.9843
Epoch 23/30
41/41 [==============================] - 1s 36ms/step - loss: 0.0013 - accu
racy: 1.0000 - val_loss: 0.0750 - val_accuracy: 0.9843
Epoch 24/30
41/41 [==============================] - 1s 31ms/step - loss: 0.0011 - accu
racy: 1.0000 - val_loss: 0.0757 - val_accuracy: 0.9843
Epoch 25/30
41/41 [==============================] - 1s 25ms/step - loss: 0.0011 - accu
```

```
racy: 1.0000 - val_loss: 0.0763 - val_accuracy: 0.9843
Epoch 26/30
41/41 [==============================] - 1s 26ms/step - loss: 9.7778e-04 -
accuracy: 1.0000 - val_loss: 0.0771 - val_accuracy: 0.9843
Epoch 27/30
41/41 [==============================] - 1s 27ms/step - loss: 9.0370e-04 -
accuracy: 1.0000 - val_loss: 0.0778 - val_accuracy: 0.9843
Epoch 28/30
41/41 [==============================] - 1s 30ms/step - loss: 8.4027e-04 -
accuracy: 1.0000 - val_loss: 0.0784 - val_accuracy: 0.9843
Epoch 29/30
41/41 [==============================] - 1s 32ms/step - loss: 7.8206e-04 -
accuracy: 1.0000 - val_loss: 0.0791 - val_accuracy: 0.9843
Epoch 30/30
41/41 [==============================] - 1s 26ms/step - loss: 7.2971e-04 -
accuracy: 1.0000 - val_loss: 0.0796 - val_accuracy: 0.9843
```

In [ ]:

```
# Plot training and validation accuracy values
plt.plot(history.history['val_accuracy'])
plt.plot(history.history['accuracy'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc = 'upper left')
plt.show()
```




In [ ]:
```
# Evaluate
score = model.evaluate(X_test, y_test, batch_size = batch_size, verbose = 1)
print('Accuracy: ', score[1])
print('Loss: ', score[0])
```

```
12/12 [==============================] - 0s 12ms/step - loss: 0.0970 - accu
racy: 0.9830
Accuracy:  0.983020544052124
Loss:  0.09702974557876587
```

In [ ]:
```
# Get predictions
pred = model.predict(X_test)
pred_labels = [1 if p > 0.5 else 0 for p in pred]

print(pred[:10])
print(pred_labels[:10])

print('Accuracy score: ', accuracy_score(y_test, pred_labels))
print('Precision score: ', precision_score(y_test, pred_labels))
print('Recall score: ', recall_score(y_test, pred_labels))
print('f1 score: ', f1_score(y_test, pred_labels))
```

```
35/35 [==============================] - 0s 4ms/step
[[1.2931503e-04]
 [9.9997008e-01]
 [3.7817091e-10]
 [9.9999976e-01]
 [2.1797841e-02]
 [9.9916261e-01]
 [6.5194463e-09]
 [7.4062531e-04]
 [5.6787594e-03]
 [7.5629170e-10]]
[0, 1, 0, 1, 0, 1, 0, 0, 0, 0]
Accuracy score:  0.9830205540661304
Precision score:  0.984375
Recall score:  0.8811188811188811
f1 score:  0.9298892988929889
```

## Try a different architecture (CNN) and evaluate on the test data

In [ ]:

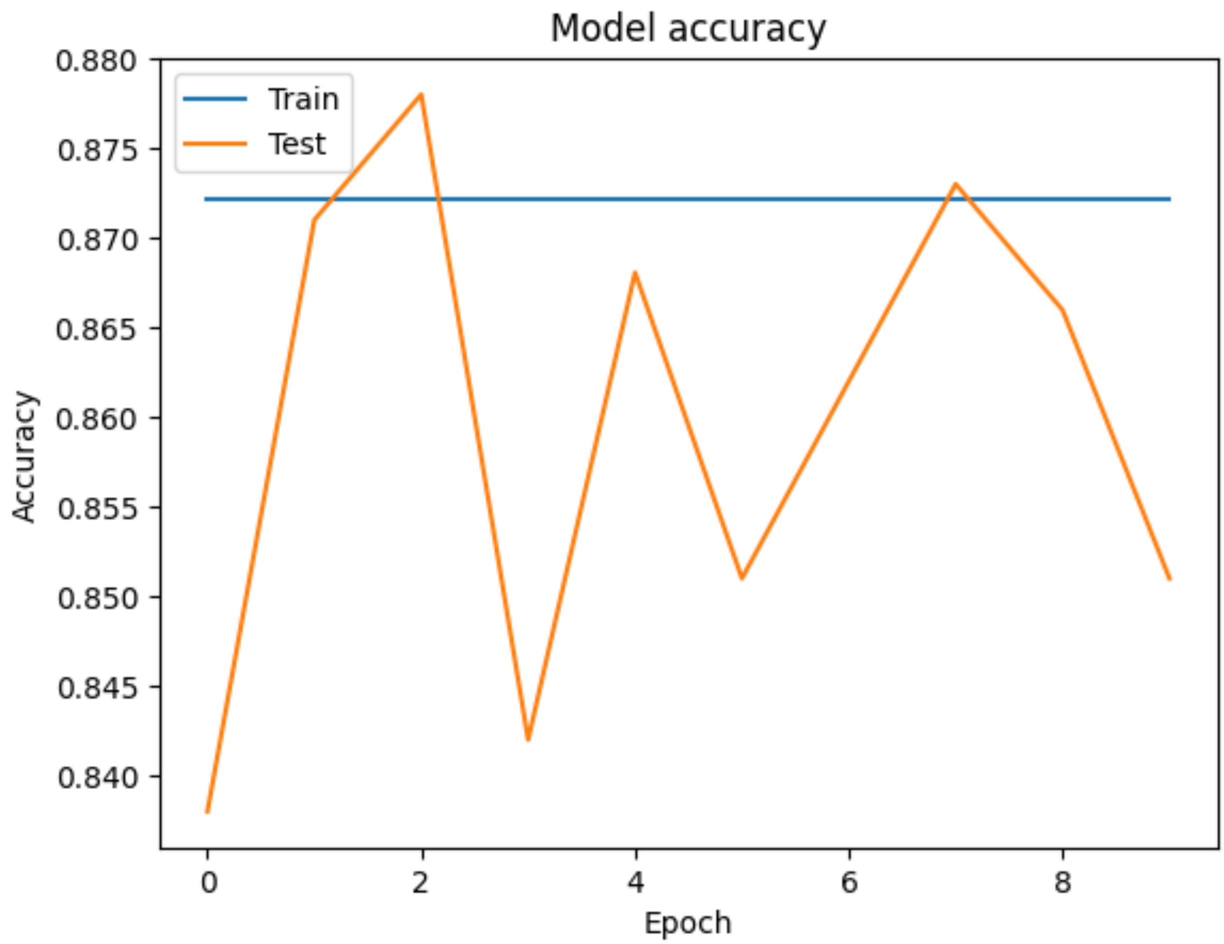
```
embedding_dim = 100

model = models.Sequential()

model.add(layers.Embedding(vocab_size, embedding_dim))
model.add(layers.Conv1D(128, 5, activation = 'relu'))
model.add(layers.GlobalMaxPooling1D())
model.add(layers.Dense(10, activation = 'relu'))
model.add(layers.Dense(1, activation = 'sigmoid'))

model.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics = ['

model.summary()
```

```
Model: "sequential_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 embedding_1 (Embedding)     (None, None, 100)         2500000   
                                                                 
 conv1d_1 (Conv1D)           (None, None, 128)         64128     
                                                                 
 global_max_pooling1d_1 (Glo  (None, 128)              0         
 balMaxPooling1D)                                                
                                                                 
 dense_8 (Dense)             (None, 10)                1290      
                                                                 
 dense_9 (Dense)             (None, 1)                 11        
                                                                 
=================================================================
Total params: 2,565,429
Trainable params: 2,565,429
Non-trainable params: 0
_________________________________________________________________
```

In [ ]:
```
history = model.fit(X_train, y_train, batch_size=batch_size, epochs = 10, st
```

```
Epoch 1/10
10/10 [==============================] - 489s 51s/step - loss: 0.5435 - acc
uracy: 0.8380 - val_loss: 0.4233 - val_accuracy: 0.8722
Epoch 2/10
10/10 [==============================] - 446s 45s/step - loss: 0.3978 - acc
uracy: 0.8710 - val_loss: 0.3972 - val_accuracy: 0.8722
Epoch 3/10
10/10 [==============================] - 426s 44s/step - loss: 0.3807 - acc
uracy: 0.8780 - val_loss: 0.3888 - val_accuracy: 0.8722
Epoch 4/10
10/10 [==============================] - 428s 44s/step - loss: 0.4395 - acc
uracy: 0.8420 - val_loss: 0.3868 - val_accuracy: 0.8722
Epoch 5/10
10/10 [==============================] - 403s 42s/step - loss: 0.3858 - acc
uracy: 0.8681 - val_loss: 0.3750 - val_accuracy: 0.8722
Epoch 6/10
10/10 [==============================] - 425s 44s/step - loss: 0.3980 - acc
uracy: 0.8510 - val_loss: 0.3606 - val_accuracy: 0.8722
Epoch 7/10
10/10 [==============================] - 421s 44s/step - loss: 0.3621 - acc
uracy: 0.8620 - val_loss: 0.3442 - val_accuracy: 0.8722
Epoch 8/10
10/10 [==============================] - 418s 43s/step - loss: 0.3287 - acc
uracy: 0.8730 - val_loss: 0.3479 - val_accuracy: 0.8722
Epoch 9/10
10/10 [==============================] - 404s 42s/step - loss: 0.3465 - acc
uracy: 0.8660 - val_loss: 0.3378 - val_accuracy: 0.8722
Epoch 10/10
10/10 [==============================] - 419s 44s/step - loss: 0.3580 - acc
uracy: 0.8510 - val_loss: 0.3253 - val_accuracy: 0.8722
```

In [ ]:
```
# Plot training and validation accuracy values
plt.plot(history.history['val_accuracy'])
plt.plot(history.history['accuracy'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc = 'upper left')
plt.show()
```




In [ ]:

```
# Evaluate
score = model.evaluate(X_test, y_test, batch_size = batch_size, verbose = 1)
print('Accuracy: ', score[1])
print('Loss: ', score[0])
```

```
12/12 [==============================] - 102s 9s/step - loss: 0.3253 - accu
racy: 0.8722
Accuracy:  0.8722073435783386
Loss:  0.32531261444091797
```

## Try different embedding approaches and evaluate on the test data

In [ ]:
```
# Fit model
embedding_dim = 50

model = models.Sequential()

model.add(layers.Dense(64, input_dim = vocab_size, kernel_initializer = 'nor
model.add(layers.Dense(1, kernel_initializer = 'normal', activation = 'sigmo

model.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics = ['

model.summary()
```

```
Model: "sequential_5"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_10 (Dense)            (None, 64)                1600064

 dense_11 (Dense)            (None, 1)                 65

=================================================================
Total params: 1,600,129
Trainable params: 1,600,129
Non-trainable params: 0
_________________________________________________________________
```

In [ ]:
```
history = model.fit(X_train, y_train, epochs = 50, verbose = 1, validation_d
```

```
Epoch 1/50
45/45 [==============================] - 4s 57ms/step - loss: 0.4188 - accu
racy: 0.8788 - val_loss: 0.2355 - val_accuracy: 0.9446
Epoch 2/50
45/45 [==============================] - 2s 46ms/step - loss: 0.1316 - accu
racy: 0.9776 - val_loss: 0.1038 - val_accuracy: 0.9866
Epoch 3/50
45/45 [==============================] - 2s 46ms/step - loss: 0.0501 - accu
racy: 0.9946 - val_loss: 0.0741 - val_accuracy: 0.9920
Epoch 4/50
45/45 [==============================] - 3s 75ms/step - loss: 0.0248 - accu
racy: 0.9980 - val_loss: 0.0665 - val_accuracy: 0.9893
Epoch 5/50
45/45 [==============================] - 2s 49ms/step - loss: 0.0142 - accu
racy: 0.9991 - val_loss: 0.0658 - val_accuracy: 0.9875
Epoch 6/50
45/45 [==============================] - 2s 44ms/step - loss: 0.0091 - accu
racy: 0.9993 - val_loss: 0.0657 - val_accuracy: 0.9875
Epoch 7/50
45/45 [==============================] - 2s 55ms/step - loss: 0.0063 - accu
racy: 0.9998 - val_loss: 0.0673 - val_accuracy: 0.9866
Epoch 8/50
```

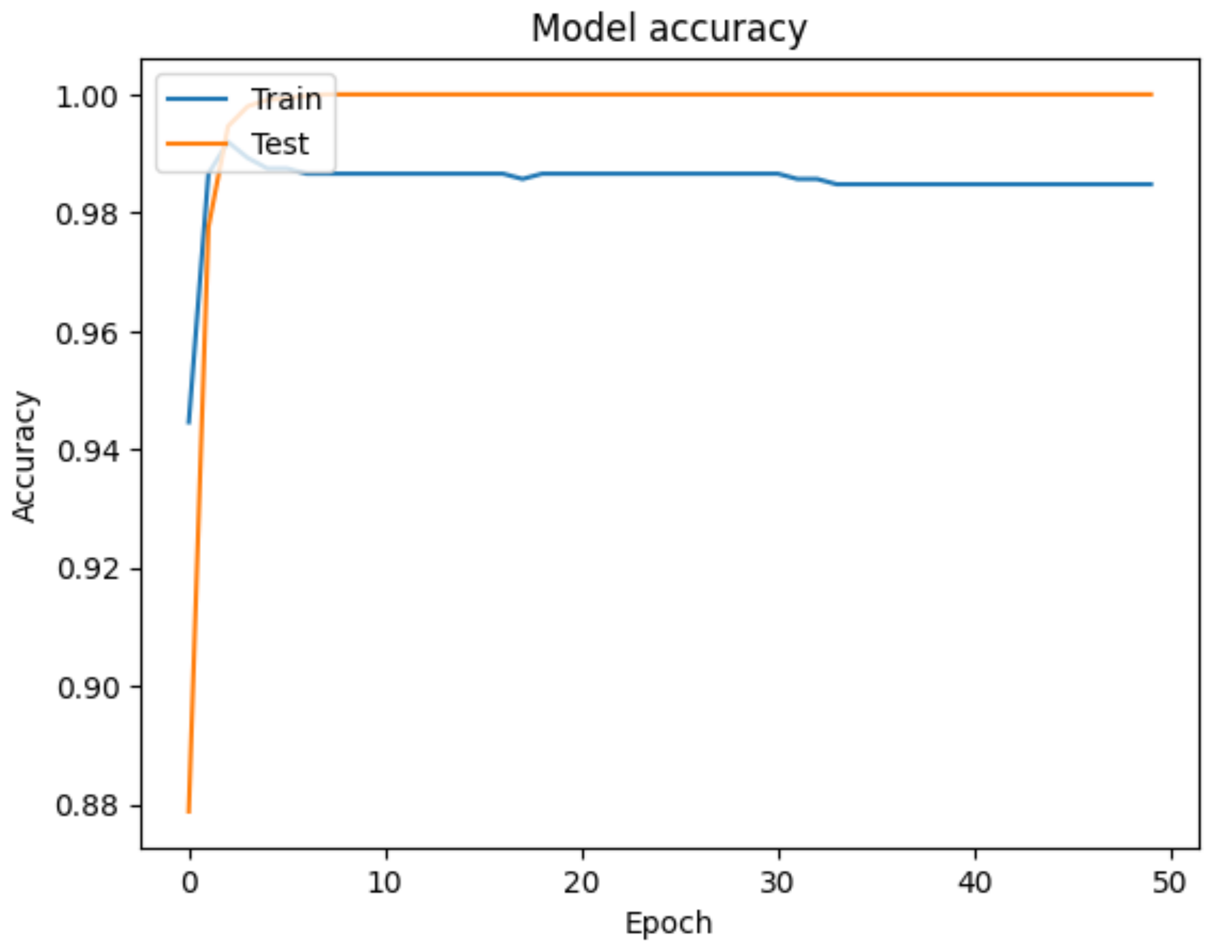
```
45/45 [==============================] - 2s 50ms/step - loss: 0.0046 - accu
racy: 1.0000 - val_loss: 0.0696 - val_accuracy: 0.9866
Epoch 9/50
45/45 [==============================] - 3s 63ms/step - loss: 0.0035 - accu
racy: 1.0000 - val_loss: 0.0717 - val_accuracy: 0.9866
Epoch 10/50
45/45 [==============================] - 3s 57ms/step - loss: 0.0027 - accu
racy: 1.0000 - val_loss: 0.0739 - val_accuracy: 0.9866
Epoch 11/50
45/45 [==============================] - 2s 46ms/step - loss: 0.0022 - accu
racy: 1.0000 - val_loss: 0.0758 - val_accuracy: 0.9866
Epoch 12/50
45/45 [==============================] - 2s 44ms/step - loss: 0.0018 - accu
racy: 1.0000 - val_loss: 0.0777 - val_accuracy: 0.9866
Epoch 13/50
45/45 [==============================] - 2s 47ms/step - loss: 0.0015 - accu
racy: 1.0000 - val_loss: 0.0796 - val_accuracy: 0.9866
Epoch 14/50
45/45 [==============================] - 2s 46ms/step - loss: 0.0013 - accu
racy: 1.0000 - val_loss: 0.0815 - val_accuracy: 0.9866
Epoch 15/50
45/45 [==============================] - 3s 70ms/step - loss: 0.0011 - accu
racy: 1.0000 - val_loss: 0.0833 - val_accuracy: 0.9866
Epoch 16/50
45/45 [==============================] - 2s 52ms/step - loss: 9.7603e-04 -
accuracy: 1.0000 - val_loss: 0.0850 - val_accuracy: 0.9866
Epoch 17/50
45/45 [==============================] - 2s 48ms/step - loss: 8.5732e-04 -
accuracy: 1.0000 - val_loss: 0.0863 - val_accuracy: 0.9866
Epoch 18/50
45/45 [==============================] - 2s 50ms/step - loss: 7.6022e-04 -
accuracy: 1.0000 - val_loss: 0.0880 - val_accuracy: 0.9857
Epoch 19/50
45/45 [==============================] - 2s 49ms/step - loss: 6.7835e-04 -
accuracy: 1.0000 - val_loss: 0.0897 - val_accuracy: 0.9866
Epoch 20/50
45/45 [==============================] - 2s 52ms/step - loss: 6.0622e-04 -
accuracy: 1.0000 - val_loss: 0.0910 - val_accuracy: 0.9866
Epoch 21/50
45/45 [==============================] - 3s 67ms/step - loss: 5.4742e-04 -
accuracy: 1.0000 - val_loss: 0.0923 - val_accuracy: 0.9866
Epoch 22/50
45/45 [==============================] - 2s 48ms/step - loss: 4.9713e-04 -
accuracy: 1.0000 - val_loss: 0.0935 - val_accuracy: 0.9866
Epoch 23/50
45/45 [==============================] - 2s 48ms/step - loss: 4.5323e-04 -
accuracy: 1.0000 - val_loss: 0.0948 - val_accuracy: 0.9866
Epoch 24/50
45/45 [==============================] - 2s 49ms/step - loss: 4.1460e-04 -
accuracy: 1.0000 - val_loss: 0.0959 - val_accuracy: 0.9866
```

```
Epoch 25/50
45/45 [==============================] - 2s 46ms/step - loss: 3.8099e-04 -
accuracy: 1.0000 - val_loss: 0.0969 - val_accuracy: 0.9866
Epoch 26/50
45/45 [==============================] - 3s 64ms/step - loss: 3.5201e-04 -
accuracy: 1.0000 - val_loss: 0.0981 - val_accuracy: 0.9866
Epoch 27/50
45/45 [==============================] - 3s 60ms/step - loss: 3.2486e-04 -
accuracy: 1.0000 - val_loss: 0.0992 - val_accuracy: 0.9866
Epoch 28/50
45/45 [==============================] - 2s 48ms/step - loss: 3.0162e-04 -
accuracy: 1.0000 - val_loss: 0.1002 - val_accuracy: 0.9866
Epoch 29/50
45/45 [==============================] - 2s 48ms/step - loss: 2.8064e-04 -
accuracy: 1.0000 - val_loss: 0.1011 - val_accuracy: 0.9866
Epoch 30/50
45/45 [==============================] - 9s 203ms/step - loss: 2.6160e-04 -
accuracy: 1.0000 - val_loss: 0.1023 - val_accuracy: 0.9866
Epoch 31/50
45/45 [==============================] - 3s 77ms/step - loss: 2.4457e-04 -
accuracy: 1.0000 - val_loss: 0.1030 - val_accuracy: 0.9866
Epoch 32/50
45/45 [==============================] - 2s 50ms/step - loss: 2.2897e-04 -
accuracy: 1.0000 - val_loss: 0.1041 - val_accuracy: 0.9857
Epoch 33/50
45/45 [==============================] - 2s 46ms/step - loss: 2.1509e-04 -
accuracy: 1.0000 - val_loss: 0.1050 - val_accuracy: 0.9857
Epoch 34/50
45/45 [==============================] - 2s 48ms/step - loss: 2.0218e-04 -
accuracy: 1.0000 - val_loss: 0.1057 - val_accuracy: 0.9848
Epoch 35/50
45/45 [==============================] - 2s 47ms/step - loss: 1.9032e-04 -
accuracy: 1.0000 - val_loss: 0.1066 - val_accuracy: 0.9848
Epoch 36/50
45/45 [==============================] - 4s 84ms/step - loss: 1.7988e-04 -
accuracy: 1.0000 - val_loss: 0.1074 - val_accuracy: 0.9848
Epoch 37/50
45/45 [==============================] - 2s 48ms/step - loss: 1.6979e-04 -
accuracy: 1.0000 - val_loss: 0.1083 - val_accuracy: 0.9848
Epoch 38/50
45/45 [==============================] - 2s 47ms/step - loss: 1.6092e-04 -
accuracy: 1.0000 - val_loss: 0.1093 - val_accuracy: 0.9848
Epoch 39/50
45/45 [==============================] - 2s 46ms/step - loss: 1.5233e-04 -
accuracy: 1.0000 - val_loss: 0.1100 - val_accuracy: 0.9848
Epoch 40/50
45/45 [==============================] - 2s 45ms/step - loss: 1.4470e-04 -
accuracy: 1.0000 - val_loss: 0.1107 - val_accuracy: 0.9848
Epoch 41/50
45/45 [==============================] - 3s 58ms/step - loss: 1.3762e-04 -
```

```
accuracy: 1.0000 - val_loss: 0.1117 - val_accuracy: 0.9848
Epoch 42/50
45/45 [==============================] - 3s 64ms/step - loss: 1.3089e-04 -
accuracy: 1.0000 - val_loss: 0.1125 - val_accuracy: 0.9848
Epoch 43/50
45/45 [==============================] - 2s 45ms/step - loss: 1.2486e-04 -
accuracy: 1.0000 - val_loss: 0.1133 - val_accuracy: 0.9848
Epoch 44/50
45/45 [==============================] - 2s 46ms/step - loss: 1.1883e-04 -
accuracy: 1.0000 - val_loss: 0.1138 - val_accuracy: 0.9848
Epoch 45/50
45/45 [==============================] - 2s 51ms/step - loss: 1.1354e-04 -
accuracy: 1.0000 - val_loss: 0.1147 - val_accuracy: 0.9848
Epoch 46/50
45/45 [==============================] - 2s 50ms/step - loss: 1.0845e-04 -
accuracy: 1.0000 - val_loss: 0.1153 - val_accuracy: 0.9848
Epoch 47/50
45/45 [==============================] - 3s 68ms/step - loss: 1.0375e-04 -
accuracy: 1.0000 - val_loss: 0.1162 - val_accuracy: 0.9848
Epoch 48/50
45/45 [==============================] - 2s 54ms/step - loss: 9.9288e-05 -
accuracy: 1.0000 - val_loss: 0.1168 - val_accuracy: 0.9848
Epoch 49/50
45/45 [==============================] - 3s 65ms/step - loss: 9.5161e-05 -
accuracy: 1.0000 - val_loss: 0.1175 - val_accuracy: 0.9848
Epoch 50/50
45/45 [==============================] - 2s 48ms/step - loss: 9.1293e-05 -
accuracy: 1.0000 - val_loss: 0.1182 - val_accuracy: 0.9848
```

In [ ]:

```
# Plot training and validation accuracy values
plt.plot(history.history['val_accuracy'])
plt.plot(history.history['accuracy'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc = 'upper left')
plt.show()
```




In [ ]:
```
# Evaluate
score = model.evaluate(X_test, y_test, batch_size = batch_size, verbose = 1)
print('Accuracy: ', score[1])
print('Loss: ', score[0])
```

```
12/12 [==============================] - 0s 16ms/step - loss: 0.1182 - accu
racy: 0.9848
Accuracy:  0.9848078489303589
Loss:  0.1181943342089653
```

In [ ]:
```
# Get predictions
pred = model.predict(X_test)
pred_labels = [1 if p > 0.5 else 0 for p in pred]

print(pred[:10])
print(pred_labels[:10])

print('Accuracy score: ', accuracy_score(y_test, pred_labels))
print('Precision score: ', precision_score(y_test, pred_labels))
print('Recall score: ', recall_score(y_test, pred_labels))
print('f1 score: ', f1_score(y_test, pred_labels))
```

```
35/35 [==============================] - 0s 7ms/step
[[8.6365014e-07]
 [9.9998045e-01]
 [1.7437482e-13]
 [1.0000000e+00]
 [3.6307354e-03]
 [9.9935097e-01]
 [5.8593091e-12]
 [1.1403668e-05]
 [1.3000194e-04]
 [2.0403445e-13]]
[0, 1, 0, 1, 0, 1, 0, 0, 0, 0]
Accuracy score:  0.9848078641644326
Precision score:  0.9921875
Recall score:  0.8881118881118881
f1 score:  0.9372693726937269
```

## Write up your analysis of the performance of various approaches

- In regards to both speed and accuracy, the regular Sequential model outperformed the CNN model by a very large margin. The final testing accuracy was 98.38% with a loss of 0.097. There were times during training where the model performed with 100% accuracy.
- In regards to both accuracy and time to train, the CNN model performed the worst. The final accuracy was 87% and the loss was 0.3253 after about 1 hour of training.
- The Sequential model had an accuracy of 98.48% and a loss of 0.1181, meaning it outperformed the regular Sequential model by only a little. However, the downside to this model was that it took 3 times as long to train.
- Overall, the third model performed better than the other models in both accuracy and loss metrics, with the only drawback being training time.