Portfolio Assignment: Text Classification 2

Cady Baltz (cmb180010)

4/22/2023

1. Find a text classification data set

```
# necessary packages for processing text
import tensorflow as tf
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras import layers, models
from sklearn.preprocessing import LabelEncoder
import pickle
import numpy as np
import pandas as pd
import seaborn as sb

# set seed for reproducibility
np.random.seed(1234)
```

```
# using a "Real / Fake Job Posting" text dataset from Kaggle
# source: https://www.kaggle.com/datasets/shivamb/real-or-fake-fake-jobposting-predict

input_filename = 'fake_job_postings.csv'

# only imported the first 5000 entries so the program runs in a timely manner
df = pd.read_csv(input_filename)[:5000]
```

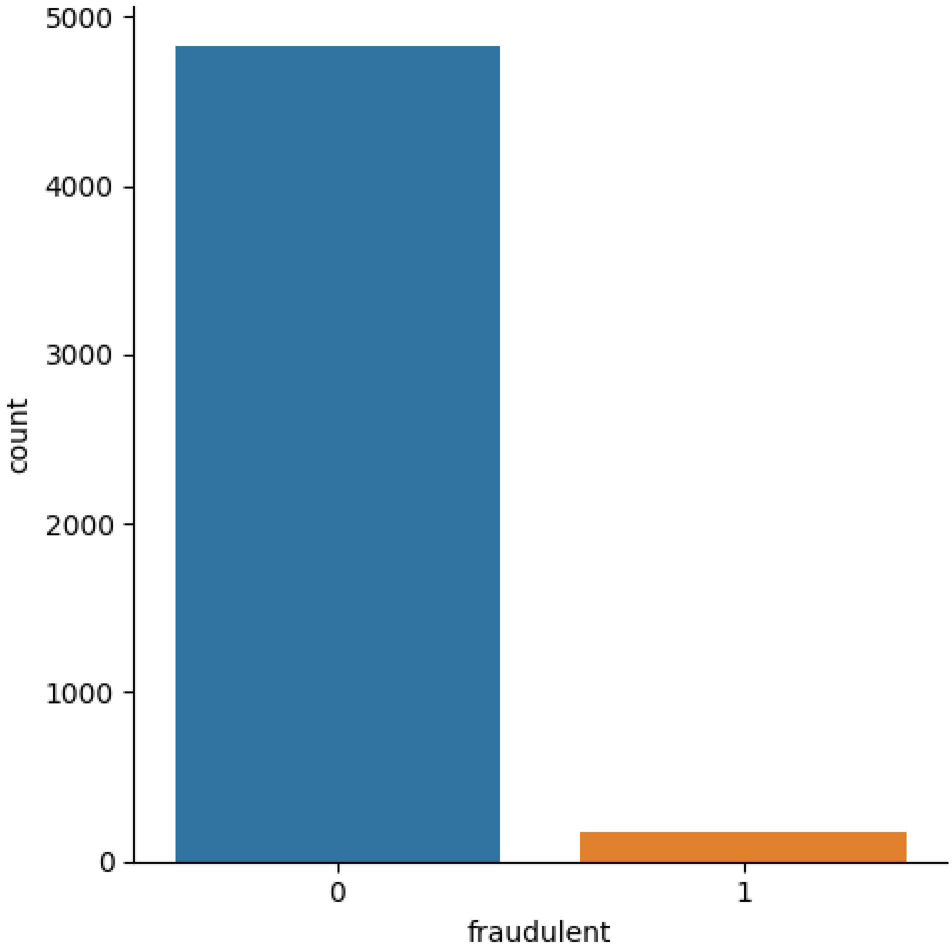
Data set description:

This dataset contains a set of job descriptions, as well as whether or not those job descriptions are fake. Given a job title from this dataset, my model should be able to predict whether the job is fraudulent or not, making this a binary text classification task.

Target class distribution

```
# create a graph showing the distribution of target classes in the dataset
sb.catplot(x="fraudulent", kind='count', data=df)
```

```
<seaborn.axisgrid.FacetGrid at 0x7f9df87ef7c0>
```



Divide into a train/test dataset

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

```
train data size:  (3990, 18)
test data size:  (1010, 18)
```

Text Pre-Processing

```
# define the same model variables to use in all types of training for fair comparison
num_labels = 2
```

```
vocab_size = 25000
batch_size = 100
epochs = 30
validation_split = 0.1
optimizer = 'rmsprop'
metrics = ['accuracy']
kernel_initializer = 'normal'

# configure the model to handle binary classification
loss = 'binary_crossentropy'
```

```
# fit the tokenizer on the training data
tokenizer = Tokenizer(num_words=vocab_size)
tokenizer.fit_on_texts(train.title)

# use tf-idf to process word frequencies in the titles
x_train = tokenizer.texts_to_matrix(train.title, mode='tfidf')
x_test = tokenizer.texts_to_matrix(test.title, mode='tfidf')

# encode whether or not the job is fraudulent as the label
encoder = LabelEncoder()
encoder.fit(train.fraudulent)
y_train = encoder.transform(train.fraudulent)
y_test = encoder.transform(test.fraudulent)
```

2. Sequential Model

```
# fit a dense sequential model on the training data
model = models.Sequential()
model.add(layers.Dense(32, input_dim=vocab_size, kernel_initializer=kernel_initialize
model.add(layers.Dense(1, kernel_initializer=kernel_initializer, activation='sigmoid')

model.compile(loss=loss,
              optimizer=optimizer,
              metrics=metrics)

history = model.fit(x_train, y_train,
                    batch_size=batch_size,
                    epochs=epochs,
                    verbose=1,
                    validation_split=validation_split)
```

```
Epoch 1/30
37/37 [==============================] - 2s 32ms/step - loss: 0.5506 - accuracy:
Epoch 2/30
37/37 [==============================] - 1s 26ms/step - loss: 0.3253 - accuracy:
Epoch 3/30
```

```
37/37 [==============================] - 1s 25ms/step - loss: 0.1850 - accuracy:
Epoch 4/30
37/37 [==============================] - 1s 24ms/step - loss: 0.1304 - accuracy:
Epoch 5/30
37/37 [==============================] - 1s 23ms/step - loss: 0.1097 - accuracy:
Epoch 6/30
37/37 [==============================] - 1s 29ms/step - loss: 0.0980 - accuracy:
Epoch 7/30
37/37 [==============================] - 1s 35ms/step - loss: 0.0897 - accuracy:
Epoch 8/30
37/37 [==============================] - 1s 36ms/step - loss: 0.0824 - accuracy:
Epoch 9/30
37/37 [==============================] - 1s 25ms/step - loss: 0.0765 - accuracy:
Epoch 10/30
37/37 [==============================] - 1s 24ms/step - loss: 0.0712 - accuracy:
Epoch 11/30
37/37 [==============================] - 1s 24ms/step - loss: 0.0673 - accuracy:
Epoch 12/30
37/37 [==============================] - 1s 24ms/step - loss: 0.0632 - accuracy:
Epoch 13/30
37/37 [==============================] - 1s 24ms/step - loss: 0.0597 - accuracy:
Epoch 14/30
37/37 [==============================] - 1s 24ms/step - loss: 0.0566 - accuracy:
Epoch 15/30
37/37 [==============================] - 1s 24ms/step - loss: 0.0538 - accuracy:
Epoch 16/30
37/37 [==============================] - 1s 24ms/step - loss: 0.0512 - accuracy:
Epoch 17/30
37/37 [==============================] - 1s 25ms/step - loss: 0.0490 - accuracy:
Epoch 18/30
37/37 [==============================] - 1s 25ms/step - loss: 0.0468 - accuracy:
Epoch 19/30
37/37 [==============================] - 1s 24ms/step - loss: 0.0450 - accuracy:
Epoch 20/30
37/37 [==============================] - 1s 34ms/step - loss: 0.0434 - accuracy:
Epoch 21/30
37/37 [==============================] - 1s 35ms/step - loss: 0.0418 - accuracy:
Epoch 22/30
37/37 [==============================] - 1s 31ms/step - loss: 0.0406 - accuracy:
Epoch 23/30
37/37 [==============================] - 1s 24ms/step - loss: 0.0392 - accuracy:
Epoch 24/30
37/37 [==============================] - 1s 23ms/step - loss: 0.0381 - accuracy:
Epoch 25/30
37/37 [==============================] - 1s 24ms/step - loss: 0.0372 - accuracy:
Epoch 26/30
37/37 [==============================] - 1s 24ms/step - loss: 0.0360 - accuracy:
Epoch 27/30
37/37 [==============================] - 1s 25ms/step - loss: 0.0353 - accuracy:
Epoch 28/30
37/37 [==============================] - 1s 25ms/step - loss: 0.0341 - accuracy:
Epoch 29/30
37/37 [==============================] - 1s 24ms/step - loss: 0.0336 - accuracy:
```

```
# evaluate
score = model.evaluate(x_test, y_test, batch_size=batch_size, verbose=1)
```

```
print('Accuracy: ', score[1])
```

```
10/10 [==============================] - 0s 14ms/step - loss: 0.1835 - accuracy:
Accuracy:  0.9497487545013428
```

```
print(score)
```

```
[0.18351320922374725, 0.9497487545013428]
```

```
# calculate more metrics using the actual predictions
pred = model.predict(x_test)
pred_labels = [1 if p>0.5 else 0 for p in pred]
```

```
32/32 [==============================] - 0s 6ms/step
```

```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
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))
```

```
accuracy score:  0.949748743718593
precision score:  0.2571428571428571
recall score:  0.2727272727272727
f1 score:  0.2647058823529411
```

3. Trying different architectures

Recurrent Neural Network (RNN)

```
import tensorflow as tf
from tensorflow.keras import datasets, layers, models, preprocessing

max_features = 10000
maxlen = 500
x_train = preprocessing.sequence.pad_sequences(x_train, maxlen=maxlen)
x_test = preprocessing.sequence.pad_sequences(x_test, maxlen=maxlen)
```

```
# fit simple RNN model
model = models.Sequential()
model.add(layers.Embedding(max_features, 32))
model.add(layers.SimpleRNN(32))
model.add(layers.Dense(1, activation='sigmoid'))
```

```
model.compile(optimizer=optimizer,
              loss=loss,
              metrics=metrics)

history = model.fit(x_train,
                    y_train,
                    epochs=epochs,
                    batch_size=batch_size,
                    validation_split=validation_split)
```

```
Epoch 1/30
37/37 [==============================] - 8s 186ms/step - loss: 0.5231 - accuracy
Epoch 2/30
37/37 [==============================] - 5s 142ms/step - loss: 0.2588 - accuracy
Epoch 3/30
37/37 [==============================] - 8s 207ms/step - loss: 0.1557 - accuracy
Epoch 4/30
37/37 [==============================] - 5s 139ms/step - loss: 0.1466 - accuracy
Epoch 5/30
37/37 [==============================] - 7s 202ms/step - loss: 0.1471 - accuracy
Epoch 6/30
37/37 [==============================] - 5s 143ms/step - loss: 0.1465 - accuracy
Epoch 7/30
37/37 [==============================] - 5s 145ms/step - loss: 0.1464 - accuracy
Epoch 8/30
37/37 [==============================] - 6s 170ms/step - loss: 0.1467 - accuracy
Epoch 9/30
37/37 [==============================] - 5s 143ms/step - loss: 0.1466 - accuracy
Epoch 10/30
37/37 [==============================] - 7s 191ms/step - loss: 0.1468 - accuracy
Epoch 11/30
37/37 [==============================] - 7s 187ms/step - loss: 0.1466 - accuracy
Epoch 12/30
37/37 [==============================] - 6s 176ms/step - loss: 0.1462 - accuracy
Epoch 13/30
37/37 [==============================] - 5s 140ms/step - loss: 0.1467 - accuracy
Epoch 14/30
37/37 [==============================] - 7s 200ms/step - loss: 0.1466 - accuracy
Epoch 15/30
37/37 [==============================] - 10s 280ms/step - loss: 0.1463 - accurac
Epoch 16/30
37/37 [==============================] - 7s 189ms/step - loss: 0.1464 - accuracy
Epoch 17/30
37/37 [==============================] - 8s 216ms/step - loss: 0.1466 - accuracy
Epoch 18/30
37/37 [==============================] - 7s 199ms/step - loss: 0.1462 - accuracy
Epoch 19/30
37/37 [==============================] - 12s 318ms/step - loss: 0.1464 - accurac
Epoch 20/30
37/37 [==============================] - 7s 193ms/step - loss: 0.1469 - accuracy
Epoch 21/30
37/37 [==============================] - 7s 190ms/step - loss: 0.1465 - accuracy
Epoch 22/30
37/37 [==============================] - 10s 269ms/step - loss: 0.1466 - accurac
Epoch 23/30
```

```
37/37 [==============================] - 12s 323ms/step - loss: 0.1467 - accurac
Epoch 24/30
37/37 [==============================] - 6s 154ms/step - loss: 0.1465 - accuracy
Epoch 25/30
37/37 [==============================] - 6s 175ms/step - loss: 0.1465 - accuracy
Epoch 26/30
37/37 [==============================] - 5s 140ms/step - loss: 0.1465 - accuracy
Epoch 27/30
37/37 [==============================] - 7s 185ms/step - loss: 0.1464 - accuracy
Epoch 28/30
37/37 [==============================] - 7s 175ms/step - loss: 0.1465 - accuracy
Epoch 29/30
37/37 [==============================] - 7s 191ms/step - loss: 0.1464 - accuracy
```

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

```
10/10 [==============================] - 1s 86ms/step - loss: 0.1459 - accuracy:
Accuracy:  0.9668341875076294
```

Convolutional Neural Network (CNN)

```
# fit CNN model (using 1D because it is text data)
model = models.Sequential()
model.add(layers.Embedding(max_features, 32))
model.add(layers.Conv1D(32, 7, activation='relu'))
model.add(layers.MaxPooling1D(5))
model.add(layers.Conv1D(32, 7, activation='relu'))
model.add(layers.GlobalMaxPooling1D())
model.add(layers.Dense(1, activation='sigmoid'))

model.compile(optimizer=optimizer,
              loss=loss,
              metrics=metrics)

history = model.fit(x_train,
                    y_train,
                    epochs=epochs,
                    batch_size=batch_size,
                    validation_split=validation_split)
```

```
Epoch 1/30
36/36 [==============================] - 6s 140ms/step - loss: 0.2878 - accuracy
Epoch 2/30
36/36 [==============================] - 7s 210ms/step - loss: 0.1433 - accuracy
Epoch 3/30
36/36 [==============================] - 8s 218ms/step - loss: 0.1443 - accuracy
Epoch 4/30
36/36 [==============================] - 7s 207ms/step - loss: 0.1437 - accuracy
```

```
Epoch 5/30
36/36 [==============================] - 6s 179ms/step - loss: 0.1430 - accuracy
Epoch 6/30
36/36 [==============================] - 8s 234ms/step - loss: 0.1447 - accuracy
Epoch 7/30
36/36 [==============================] - 6s 151ms/step - loss: 0.1435 - accuracy
Epoch 8/30
36/36 [==============================] - 6s 174ms/step - loss: 0.1442 - accuracy
Epoch 9/30
36/36 [==============================] - 5s 134ms/step - loss: 0.1439 - accuracy
Epoch 10/30
36/36 [==============================] - 6s 159ms/step - loss: 0.1434 - accuracy
Epoch 11/30
36/36 [==============================] - 5s 146ms/step - loss: 0.1439 - accuracy
Epoch 12/30
36/36 [==============================] - 5s 133ms/step - loss: 0.1440 - accuracy
Epoch 13/30
36/36 [==============================] - 6s 176ms/step - loss: 0.1440 - accuracy
Epoch 14/30
36/36 [==============================] - 5s 135ms/step - loss: 0.1439 - accuracy
Epoch 15/30
36/36 [==============================] - 6s 163ms/step - loss: 0.1440 - accuracy
Epoch 16/30
36/36 [==============================] - 5s 142ms/step - loss: 0.1442 - accuracy
Epoch 17/30
36/36 [==============================] - 5s 131ms/step - loss: 0.1442 - accuracy
Epoch 18/30
36/36 [==============================] - 6s 175ms/step - loss: 0.1431 - accuracy
Epoch 19/30
36/36 [==============================] - 5s 134ms/step - loss: 0.1437 - accuracy
Epoch 20/30
36/36 [==============================] - 6s 175ms/step - loss: 0.1439 - accuracy
Epoch 21/30
36/36 [==============================] - 5s 133ms/step - loss: 0.1443 - accuracy
Epoch 22/30
36/36 [==============================] - 5s 134ms/step - loss: 0.1443 - accuracy
Epoch 23/30
36/36 [==============================] - 6s 174ms/step - loss: 0.1436 - accuracy
Epoch 24/30
36/36 [==============================] - 5s 132ms/step - loss: 0.1442 - accuracy
Epoch 25/30
36/36 [==============================] - 6s 177ms/step - loss: 0.1437 - accuracy
Epoch 26/30
36/36 [==============================] - 5s 133ms/step - loss: 0.1438 - accuracy
Epoch 27/30
36/36 [==============================] - 5s 134ms/step - loss: 0.1437 - accuracy
Epoch 28/30
36/36 [==============================] - 6s 177ms/step - loss: 0.1435 - accuracy
Epoch 29/30
```

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

```
11/11 [==============================] - 0s 28ms/step - loss: 0.1508 - accuracy:
Accuracy:  0.9653465151786804
```

Long Short Term Memory (LSTM)

```
# fit LSTM model
model = models.Sequential()
model.add(layers.Embedding(max_features, 32))
model.add(layers.LSTM(32))
model.add(layers.Dense(1, activation='sigmoid'))

model.compile(optimizer=optimizer,
              loss=loss,
              metrics=metrics)

history = model.fit(x_train,
                    y_train,
                    epochs=epochs,
                    batch_size=batch_size,
                    validation_split=validation_split)
```

```
Epoch 2/30
37/37 [==============================] - 15s 396ms/step - loss: 0.1469 - accurac
Epoch 3/30
37/37 [==============================] - 13s 356ms/step - loss: 0.1466 - accurac
Epoch 4/30
37/37 [==============================] - 16s 429ms/step - loss: 0.1468 - accurac
Epoch 5/30
37/37 [==============================] - 15s 405ms/step - loss: 0.1464 - accurac
Epoch 6/30
37/37 [==============================] - 12s 326ms/step - loss: 0.1471 - accurac
Epoch 7/30
37/37 [==============================] - 12s 338ms/step - loss: 0.1466 - accurac
Epoch 8/30
37/37 [==============================] - 18s 491ms/step - loss: 0.1467 - accurac
Epoch 9/30
37/37 [==============================] - 13s 344ms/step - loss: 0.1466 - accurac
Epoch 10/30
37/37 [==============================] - 13s 343ms/step - loss: 0.1467 - accurac
Epoch 11/30
37/37 [==============================] - 12s 326ms/step - loss: 0.1463 - accurac
Epoch 12/30
37/37 [==============================] - 17s 467ms/step - loss: 0.1471 - accurac
Epoch 13/30
37/37 [==============================] - 15s 408ms/step - loss: 0.1463 - accurac
Epoch 14/30
37/37 [==============================] - 19s 513ms/step - loss: 0.1462 - accurac
Epoch 15/30
```

```
37/37 [==============================] - 18s 479ms/step - loss: 0.1463 - accurac
Epoch 17/30
37/37 [==============================] - 17s 464ms/step - loss: 0.1465 - accurac
Epoch 18/30
37/37 [==============================] - 13s 341ms/step - loss: 0.1465 - accurac
Epoch 19/30
37/37 [==============================] - 14s 369ms/step - loss: 0.1464 - accurac
Epoch 20/30
37/37 [==============================] - 24s 656ms/step - loss: 0.1467 - accurac
Epoch 21/30
37/37 [==============================] - 16s 419ms/step - loss: 0.1466 - accurac
Epoch 22/30
37/37 [==============================] - 16s 446ms/step - loss: 0.1465 - accurac
Epoch 23/30
37/37 [==============================] - 16s 442ms/step - loss: 0.1464 - accurac
Epoch 24/30
37/37 [==============================] - 15s 411ms/step - loss: 0.1464 - accurac
Epoch 25/30
37/37 [==============================] - 19s 507ms/step - loss: 0.1463 - accurac
Epoch 26/30
37/37 [==============================] - 15s 412ms/step - loss: 0.1461 - accurac
Epoch 27/30
37/37 [==============================] - 20s 549ms/step - loss: 0.1462 - accurac
Epoch 28/30
37/37 [==============================] - 22s 589ms/step - loss: 0.1465 - accurac
Epoch 29/30
37/37 [==============================] - 12s 316ms/step - loss: 0.1466 - accurac
Epoch 30/30
37/37 [==============================] - 12s 321ms/step - loss: 0.1465 - accurac
```

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

```
10/10 [==============================] - 1s 91ms/step - loss: 0.1459 - accuracy:
Accuracy:  0.9668341875076294
```

Different Embedding Approaches

Now that I have observed that the RNN model was the most successful, I will attempt different embedding approaches to find if I can further improve the accuracy of my predictions.

Attempt #1: Increase the dimension of the embedding length

```
# increase the embedding length from 32 -> 256
model = models.Sequential()
model.add(layers.Embedding(max_features, 256))
model.add(layers.SimpleRNN(256))
model.add(layers.Dense(1, activation='sigmoid'))
```

```
model.compile(optimizer=optimizer,
              loss=loss,
              metrics=metrics)

history = model.fit(x_train,
                    y_train,
                    epochs=epochs,
                    batch_size=batch_size,
                    validation_split=validation_split)
```

```
Epoch 1/30
37/37 [==============================] - 73s 2s/step - loss: 0.1716 - accuracy:
Epoch 2/30
37/37 [==============================] - 62s 2s/step - loss: 0.1557 - accuracy:
Epoch 3/30
37/37 [==============================] - 61s 2s/step - loss: 0.1501 - accuracy:
Epoch 4/30
37/37 [==============================] - 57s 2s/step - loss: 0.1491 - accuracy:
Epoch 5/30
37/37 [==============================] - 57s 2s/step - loss: 0.1490 - accuracy:
Epoch 6/30
37/37 [==============================] - 56s 2s/step - loss: 0.1493 - accuracy:
Epoch 7/30
37/37 [==============================] - 56s 2s/step - loss: 0.1496 - accuracy:
Epoch 8/30
37/37 [==============================] - 57s 2s/step - loss: 0.1499 - accuracy:
Epoch 9/30
37/37 [==============================] - 56s 2s/step - loss: 0.1480 - accuracy:
Epoch 10/30
37/37 [==============================] - 57s 2s/step - loss: 0.1478 - accuracy:
Epoch 11/30
37/37 [==============================] - 57s 2s/step - loss: 0.1484 - accuracy:
Epoch 12/30
37/37 [==============================] - 60s 2s/step - loss: 0.1483 - accuracy:
Epoch 13/30
37/37 [==============================] - 57s 2s/step - loss: 0.1479 - accuracy:
Epoch 14/30
23/37 [=================>............] - ETA: 21s - loss: 0.1413 - accuracy: 0.9
```

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

```
10/10 [==============================] - 0s 29ms/step - loss: 0.1460 - accuracy:
Accuracy:  0.9668341875076294
```

Attempt #2: Decrease the dimension of the embedding length

```
# decrease the embedding length from 32 -> 4
model = models.Sequential()
```

```
model.add(layers.Embedding(max_features, 4))
model.add(layers.SimpleRNN(4))
model.add(layers.Dense(1, activation='sigmoid'))

model.compile(optimizer=optimizer,
              loss=loss,
              metrics=metrics)

history = model.fit(x_train,
                    y_train,
                    epochs=epochs,
                    batch_size=batch_size,
                    validation_split=validation_split)
```

```
Epoch 2/30
36/36 [==============================] - 6s 159ms/step - loss: 0.5105 - accuracy
Epoch 3/30
36/36 [==============================] - 4s 119ms/step - loss: 0.4232 - accuracy
Epoch 4/30
36/36 [==============================] - 4s 120ms/step - loss: 0.3475 - accuracy
Epoch 5/30
36/36 [==============================] - 6s 160ms/step - loss: 0.2850 - accuracy
Epoch 6/30
36/36 [==============================] - 4s 118ms/step - loss: 0.2359 - accuracy
Epoch 7/30
36/36 [==============================] - 4s 119ms/step - loss: 0.2003 - accuracy
Epoch 8/30
36/36 [==============================] - 6s 160ms/step - loss: 0.1761 - accuracy
Epoch 9/30
36/36 [==============================] - 4s 119ms/step - loss: 0.1608 - accuracy
Epoch 10/30
36/36 [==============================] - 5s 140ms/step - loss: 0.1521 - accuracy
Epoch 11/30
36/36 [==============================] - 5s 133ms/step - loss: 0.1472 - accuracy
Epoch 12/30
36/36 [==============================] - 4s 120ms/step - loss: 0.1448 - accuracy
Epoch 13/30
36/36 [==============================] - 6s 160ms/step - loss: 0.1437 - accuracy
Epoch 14/30
36/36 [==============================] - 5s 131ms/step - loss: 0.1431 - accuracy
Epoch 15/30
36/36 [==============================] - 5s 131ms/step - loss: 0.1429 - accuracy
Epoch 16/30
36/36 [==============================] - 6s 160ms/step - loss: 0.1428 - accuracy
Epoch 17/30
36/36 [==============================] - 4s 117ms/step - loss: 0.1427 - accuracy
Epoch 18/30
36/36 [==============================] - 6s 160ms/step - loss: 0.1427 - accuracy
Epoch 19/30
36/36 [==============================] - 5s 127ms/step - loss: 0.1427 - accuracy
Epoch 20/30
36/36 [==============================] - 5s 142ms/step - loss: 0.1427 - accuracy
Epoch 21/30
```

```
36/36 [==============================] - 4s 118ms/step - loss: 0.1427 - accuracy
Epoch 23/30
36/36 [==============================] - 4s 118ms/step - loss: 0.1427 - accuracy
Epoch 24/30
36/36 [==============================] - 6s 160ms/step - loss: 0.1427 - accuracy
Epoch 25/30
36/36 [==============================] - 4s 118ms/step - loss: 0.1427 - accuracy
Epoch 26/30
36/36 [==============================] - 5s 139ms/step - loss: 0.1427 - accuracy
Epoch 27/30
36/36 [==============================] - 5s 136ms/step - loss: 0.1427 - accuracy
Epoch 28/30
36/36 [==============================] - 4s 118ms/step - loss: 0.1427 - accuracy
Epoch 29/30
36/36 [==============================] - 6s 159ms/step - loss: 0.1427 - accuracy
Epoch 30/30
36/36 [==============================] - 4s 120ms/step - loss: 0.1427 - accuracy
```

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

```
11/11 [==============================] - 0s 27ms/step - loss: 0.1507 - accuracy:
Accuracy:  0.9653465151786804
```

Attempt 3: Using a pre-trained GloVe embedding

```
# create a word and embedding dictionary to use for the pretrained embedding layer

from tensorflow.keras.layers.experimental.preprocessing import TextVectorization

vectorizer = TextVectorization(max_tokens=20000, output_sequence_length=200)
text_ds = tf.data.Dataset.from_tensor_slices(train.title).batch(128)
vectorizer.adapt(text_ds)

voc = vectorizer.get_vocabulary()
word_index = dict(zip(voc, range(len(voc))))

# downloaded the pretrained GloVe embeddings from Stanford's website
# source: http://nlp.stanford.edu/data/glove.6B.zip
path_to_glove_file = "glove.txt"

embeddings_index = {}
with open(path_to_glove_file) as f:
  for line in f:
    word, coefs = line.split(maxsplit=1)
    coefs = np.fromstring(coefs, "f", sep=" ")
    embeddings_index[word] = coefs
```

```
embedding_dim = 100
hits = 0
misses = 0

# Prepare embedding matrix
embedding_matrix = np.zeros((vocab_size, embedding_dim))
for word, i in word_index.items():
    embedding_vector = embeddings_index.get(word)
    if embedding_vector is not None:
        # Words not found in embedding index will be all-zeros.
        # This includes the representation for "padding" and "OOV"
        embedding_matrix[i] = embedding_vector
        hits += 1
    else:
        misses += 1
print("Converted %d words (%d misses)" % (hits, misses))
```

```
    Converted 39 words (2397 misses)
```

```
from tensorflow import keras

# in this model, use a pre-trained embedding rather than training during the fitting
model = models.Sequential()
model.add(layers.Embedding(max_features, 4, embeddings_initializer=keras.initializers.
model.add(layers.SimpleRNN(4))
model.add(layers.Dense(1, activation='sigmoid'))

model.compile(optimizer=optimizer,
              loss=loss,
              metrics=metrics)

history = model.fit(x_train,
                    y_train,
                    epochs=epochs,
                    batch_size=batch_size,
                    validation_split=validation_split)
```

```
    Epoch 2/30
    36/36 [==============================] - 6s 154ms/step - loss: 0.4960 - accuracy
    Epoch 3/30
    36/36 [==============================] - 4s 105ms/step - loss: 0.4501 - accuracy
    Epoch 4/30
    36/36 [==============================] - 6s 169ms/step - loss: 0.4105 - accuracy
    Epoch 5/30
    36/36 [==============================] - 5s 124ms/step - loss: 0.3759 - accuracy
    Epoch 6/30
    36/36 [==============================] - 4s 120ms/step - loss: 0.3489 - accuracy
    Epoch 7/30
    36/36 [==============================] - 7s 184ms/step - loss: 0.3242 - accuracy
    Epoch 8/30
```

```
36/36 [==============================] - 4s 113ms/step - loss: 0.2799 - accuracy
Epoch 10/30
36/36 [==============================] - 5s 133ms/step - loss: 0.2527 - accuracy
Epoch 11/30
36/36 [==============================] - 3s 96ms/step - loss: 0.2375 - accuracy:
Epoch 12/30
36/36 [==============================] - 3s 93ms/step - loss: 0.2422 - accuracy:
Epoch 13/30
36/36 [==============================] - 5s 146ms/step - loss: 0.2182 - accuracy
Epoch 14/30
36/36 [==============================] - 4s 100ms/step - loss: 0.2064 - accuracy
Epoch 15/30
36/36 [==============================] - 3s 94ms/step - loss: 0.1997 - accuracy:
Epoch 16/30
36/36 [==============================] - 3s 93ms/step - loss: 0.1917 - accuracy:
Epoch 17/30
36/36 [==============================] - 5s 153ms/step - loss: 0.1677 - accuracy
Epoch 18/30
36/36 [==============================] - 3s 92ms/step - loss: 0.1800 - accuracy:
Epoch 19/30
36/36 [==============================] - 3s 94ms/step - loss: 0.1796 - accuracy:
Epoch 20/30
36/36 [==============================] - 5s 132ms/step - loss: 0.1758 - accuracy
Epoch 21/30
36/36 [==============================] - 5s 150ms/step - loss: 0.1730 - accuracy
Epoch 22/30
36/36 [==============================] - 4s 99ms/step - loss: 0.1505 - accuracy:
Epoch 23/30
36/36 [==============================] - 4s 118ms/step - loss: 0.1726 - accuracy
Epoch 24/30
36/36 [==============================] - 6s 156ms/step - loss: 0.1473 - accuracy
Epoch 25/30
36/36 [==============================] - 4s 116ms/step - loss: 0.1650 - accuracy
Epoch 26/30
36/36 [==============================] - 5s 142ms/step - loss: 0.1669 - accuracy
Epoch 27/30
36/36 [==============================] - 6s 158ms/step - loss: 0.1446 - accuracy
Epoch 28/30
36/36 [==============================] - 5s 136ms/step - loss: 0.1666 - accuracy
Epoch 29/30
36/36 [==============================] - 3s 93ms/step - loss: 0.1437 - accuracy:
Epoch 30/30
36/36 [==============================] - 7s 194ms/step - loss: 0.1668 - accuracy
```

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

```
11/11 [==============================] - 0s 33ms/step - loss: 0.1509 - accuracy:
Accuracy:  0.9653465151786804
```

5. Analysis

Overall, the various deep learning approaches I applied using Keras produced similar outputs, with accuracies in the range 95% - 97%.

The model with the lowest accuracy was the dense sequential model at 94.975%. The next lowest accuracy was the CNN model, with an accuracy of 96.535%. The reason for this improvement is likely because the CNN was better able to learn patterns in the text data I provided. For example, in the fraudulent emails, it is possible that they tended to use certain words or phrases of words that were common across many of the instances. The CNN was better able to detect these global patterns for the given input data than the simple sequential network.

However, CNN models tend to work better with image data than text data. To further optimize my model, I also tried the RNN and LSTM approaches.

First, I tried the RNN approach, which produced another increase in accuracy up to 96.683%. The RNN approach performed better than the dense sequential model, and I believe this is because the text I was analyzing was not particularly long (I was analyzing titles as opposed to full articles). As we know, the SimpleRNN does not perform well for longer sequences.

Since we observed this increase in accuracy by applying RNN, I also tried the most powerful version of RNN - LSTM. Surprisingly, I actually observed a decrease in accuracy on the test set when I applied LSTM as compared with RNN, with the accuracy dropping to 96.535%, which was closer to the accuracy of the CNN observed previously. I believe this decrease could be because the LSTM model is overfitting the data more than the SimpleRNN model, as my text sequences are fairly short (under 20 words).

Once I determined that the RNN model produced the best accuracy when classifying the test data, I also experimented with different embedding approaches for the data.

I first tried decreasing and increasing the dimension of the embedding length when I learned my word embeddings. Decreasing the embedding length unsurprisingly reduced my accuracy to 96.535%. However, increasing my embedding length did not improve my accuracy at all. While more dimensions typically leads to a greater quality text encoding, there is a certain point where you get diminishing returns. I believe this is why my accuracy stopped improving when I increased my embedding length beyond 32.

Finally, the last approach I attempted was using a pre-trained word embedding (from GloVe), instead of training the embedding layer during the model fitting. However, this approach also slightly decreased my accuracy. I believe that in my specific dataset, because my vocabulary was in a specialized domain (words related to job titles), it was best to train the word embeddings myself.