Hm4

March 30, 2024

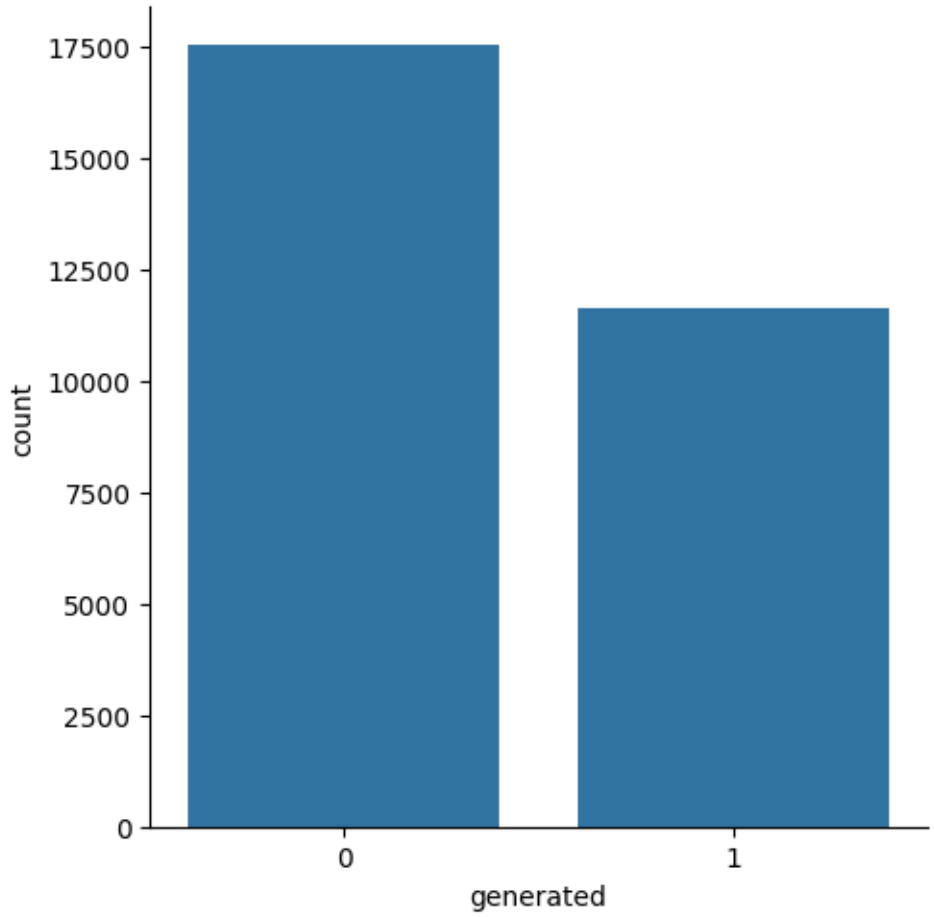
Dataset - https://www.kaggle.com/datasets/sunilthite/llm-detect-ai-generated-text-dataset

[ ]:
```
import pandas as pd
import seaborn as sb
import numpy as np
import tensorflow as tf
import keras
```

[ ]:
```
df = pd.read_csv('Training_Essay_Data.csv')
```

[ ]:
```
sb.catplot(x = 'generated', kind = 'count', data = df)
```

[ ]:
```
<seaborn.axisgrid.FacetGrid at 0x23a04075f10>
```



The data set is a set of essays which have either been AI generated or written by humans. The data set seems to be around roughly 2/3's human vs 1/3 AI generated. This model should be able to predict whether or not an essay was AI generated or not.

[ ]:
```
i = np.random.rand(len(df)) < 0.8
train = df[i]
test = df[~i]
```

[ ]:
```
from tensorflow.keras.callbacks import EarlyStopping

early_stopping = EarlyStopping(
    min_delta=0.001,
    patience=5,
    restore_best_weights=True,
)
```

[ ]:
```
from keras.preprocessing.text import Tokenizer
from keras import layers, models, preprocessing
```

```
from sklearn.preprocessing import LabelEncoder

maxlen = 1000

num_labels = 2

# fit the tokenizer on the training data
tokenizer = Tokenizer(num_words=maxlen)
tokenizer.fit_on_texts(train.text)

x_train = tokenizer.texts_to_matrix(train.text, mode='tfidf')
x_test = tokenizer.texts_to_matrix(test.text, mode='tfidf')
x_train = preprocessing.sequence.pad_sequences(x_train, maxlen)
x_test = preprocessing.sequence.pad_sequences(x_test, maxlen)


y_train = train.generated
y_test = test.generated

print("train shapes:", x_train.shape, y_train.shape)
print("test shapes:", x_test.shape, y_test.shape)
print("test first five labels:", y_test[:5])
```

```
train shapes: (23407, 1000) (23407,)
test shapes: (5738, 1000) (5738,)
test first five labels: 4     1
11    1
13    1
14    1
21    1
Name: generated, dtype: int64
```

[ ]:
```
batch_size = 512
vocab_size = len(tokenizer.word_index)
```

[ ]:
```
model = models.Sequential()
model.add(layers.Dense(5, kernel_initializer='normal', activation='relu'))
model.add(layers.Dense(1, kernel_initializer='normal', activation='sigmoid'))

model.compile(loss='binary_crossentropy',
              optimizer='rmsprop',
              metrics=['accuracy'])

history = model.fit(x_train, y_train,
                    batch_size = batch_size,
                    epochs=30,
                    verbose = 1,
                    validation_split=0.1)
```

```
Epoch 1/30
42/42             1s 4ms/step -
accuracy: 0.7911 - loss: 0.5989 - val_accuracy: 0.9692 - val_loss: 0.2538
Epoch 2/30
42/42             0s 2ms/step -
accuracy: 0.9593 - loss: 0.2438 - val_accuracy: 0.9863 - val_loss: 0.0764
Epoch 3/30
42/42             0s 2ms/step -
accuracy: 0.9745 - loss: 0.1222 - val_accuracy: 0.9949 - val_loss: 0.0314
Epoch 4/30
42/42             0s 2ms/step -
accuracy: 0.9811 - loss: 0.0793 - val_accuracy: 0.9966 - val_loss: 0.0190
Epoch 5/30
42/42             0s 2ms/step -
accuracy: 0.9838 - loss: 0.0620 - val_accuracy: 0.9970 - val_loss: 0.0133
Epoch 6/30
42/42             0s 2ms/step -
accuracy: 0.9887 - loss: 0.0435 - val_accuracy: 0.9970 - val_loss: 0.0109
Epoch 7/30
42/42             0s 2ms/step -
accuracy: 0.9894 - loss: 0.0384 - val_accuracy: 0.9970 - val_loss: 0.0092
Epoch 8/30
42/42             0s 2ms/step -
accuracy: 0.9916 - loss: 0.0295 - val_accuracy: 0.9966 - val_loss: 0.0114
Epoch 9/30
42/42             0s 2ms/step -
accuracy: 0.9918 - loss: 0.0255 - val_accuracy: 0.9974 - val_loss: 0.0079
Epoch 10/30
42/42             0s 2ms/step -
accuracy: 0.9941 - loss: 0.0212 - val_accuracy: 0.9970 - val_loss: 0.0078
Epoch 11/30
42/42             0s 2ms/step -
accuracy: 0.9954 - loss: 0.0180 - val_accuracy: 0.9970 - val_loss: 0.0077
Epoch 12/30
42/42             0s 2ms/step -
accuracy: 0.9958 - loss: 0.0174 - val_accuracy: 0.9970 - val_loss: 0.0078
Epoch 13/30
42/42             0s 2ms/step -
accuracy: 0.9965 - loss: 0.0139 - val_accuracy: 0.9970 - val_loss: 0.0073
Epoch 14/30
42/42             0s 2ms/step -
accuracy: 0.9973 - loss: 0.0124 - val_accuracy: 0.9970 - val_loss: 0.0070
Epoch 15/30
42/42             0s 2ms/step -
accuracy: 0.9972 - loss: 0.0121 - val_accuracy: 0.9970 - val_loss: 0.0065
Epoch 16/30
42/42             0s 2ms/step -
accuracy: 0.9981 - loss: 0.0101 - val_accuracy: 0.9970 - val_loss: 0.0061
```

```
Epoch 17/30
42/42             0s 2ms/step -
accuracy: 0.9977 - loss: 0.0092 - val_accuracy: 0.9966 - val_loss: 0.0080
Epoch 18/30
42/42             0s 2ms/step -
accuracy: 0.9984 - loss: 0.0083 - val_accuracy: 0.9966 - val_loss: 0.0073
Epoch 19/30
42/42             0s 2ms/step -
accuracy: 0.9987 - loss: 0.0076 - val_accuracy: 0.9970 - val_loss: 0.0078
Epoch 20/30
42/42             0s 2ms/step -
accuracy: 0.9988 - loss: 0.0064 - val_accuracy: 0.9970 - val_loss: 0.0067
Epoch 21/30
42/42             0s 2ms/step -
accuracy: 0.9989 - loss: 0.0058 - val_accuracy: 0.9970 - val_loss: 0.0076
Epoch 22/30
42/42             0s 2ms/step -
accuracy: 0.9989 - loss: 0.0051 - val_accuracy: 0.9979 - val_loss: 0.0052
Epoch 23/30
42/42             0s 2ms/step -
accuracy: 0.9994 - loss: 0.0042 - val_accuracy: 0.9979 - val_loss: 0.0053
Epoch 24/30
42/42             0s 2ms/step -
accuracy: 0.9994 - loss: 0.0040 - val_accuracy: 0.9979 - val_loss: 0.0065
Epoch 25/30
42/42             0s 2ms/step -
accuracy: 0.9994 - loss: 0.0039 - val_accuracy: 0.9979 - val_loss: 0.0067
Epoch 26/30
42/42             0s 2ms/step -
accuracy: 0.9997 - loss: 0.0032 - val_accuracy: 0.9979 - val_loss: 0.0070
Epoch 27/30
42/42             0s 2ms/step -
accuracy: 0.9998 - loss: 0.0028 - val_accuracy: 0.9979 - val_loss: 0.0061
Epoch 28/30
42/42             0s 2ms/step -
accuracy: 0.9999 - loss: 0.0025 - val_accuracy: 0.9979 - val_loss: 0.0058
Epoch 29/30
42/42             0s 2ms/step -
accuracy: 0.9999 - loss: 0.0022 - val_accuracy: 0.9979 - val_loss: 0.0066
Epoch 30/30
42/42             0s 2ms/step -
accuracy: 0.9999 - loss: 0.0019 - val_accuracy: 0.9979 - val_loss: 0.0069
```

[ ]:
```python
from sklearn import metrics
pred = model.predict(x_test)
pred = [1.0 if p>= 0.5 else 0.0 for p in pred]
```

```
print('accuracy score: ', metrics.accuracy_score(y_test, pred))
print(metrics.classification_report(y_test, pred))
print("Confusion matrix:\n", metrics.confusion_matrix(y_test, pred))
```

```
180/180             0s 584us/step
accuracy score:  0.9898919484140816
              precision    recall  f1-score   support

           0       0.99      0.99      0.99      3439
           1       0.99      0.99      0.99      2299

    accuracy                           0.99      5738
   macro avg       0.99      0.99      0.99      5738
weighted avg       0.99      0.99      0.99      5738

Confusion matrix:
 [[3405   34]
 [  24 2275]]
```

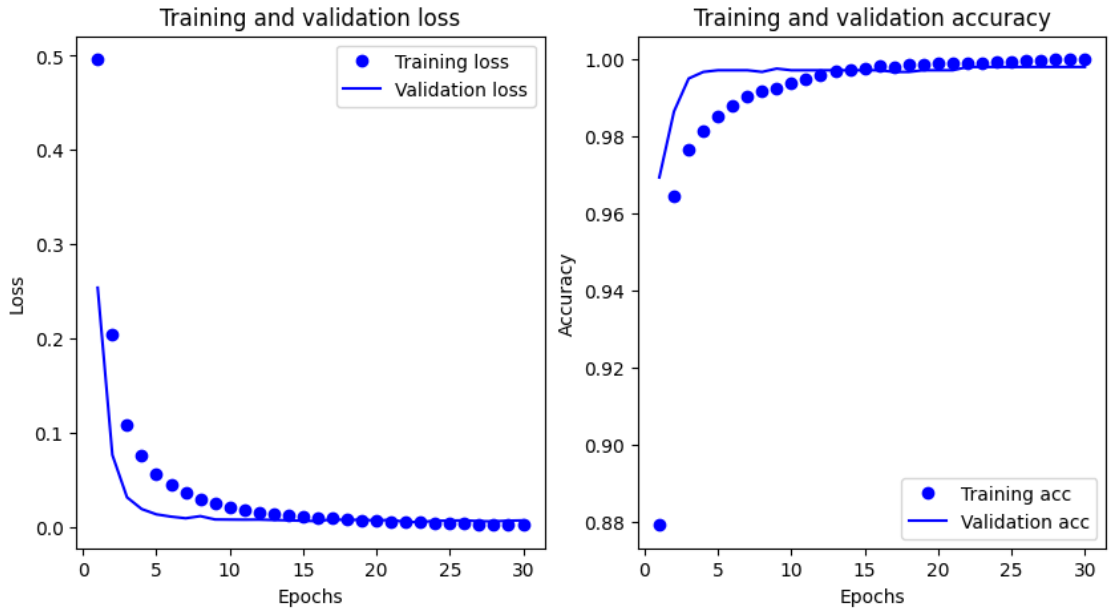
[ ]:
```
import matplotlib.pyplot as plt

loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(1, len(loss)+1)

fig, axs = plt.subplots(1, 2, figsize=(10, 5))
axs[0].plot(epochs, loss, 'bo', label='Training loss')
axs[0].plot(epochs, val_loss, 'b', label='Validation loss')
axs[0].set_title('Training and validation loss')
axs[0].set_xlabel('Epochs')
axs[0].set_ylabel('Loss')
axs[0].legend()
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

axs[1].plot(epochs, acc, 'bo', label='Training acc')
axs[1].plot(epochs, val_acc, 'b', label='Validation acc')
axs[1].set_title('Training and validation accuracy')
axs[1].set_xlabel('Epochs')
axs[1].set_ylabel('Accuracy')
axs[1].legend()

plt.show()
```



[ ]:
```
model = models.Sequential()
model.add(layers.Embedding(vocab_size, 32))
model.add(layers.SimpleRNN(32, activation='relu', kernel_initializer='normal'))
model.add(layers.Dense(1, activation='sigmoid', kernel_initializer='normal'))
model.compile(optimizer='rmsprop',
              loss='binary_crossentropy',
              metrics=['accuracy'])
```

[ ]:
```
history = model.fit(x_train,
                    y_train,
                    epochs=15,
                    batch_size = batch_size,
                    validation_split=0.1,
                    verbose = 1)
```

```
Epoch 1/15
42/42             13s 271ms/step -
accuracy: 0.6205 - loss: 0.6823 - val_accuracy: 0.4699 - val_loss: 0.7130
Epoch 2/15
42/42             11s 271ms/step -
accuracy: 0.6141 - loss: 0.6657 - val_accuracy: 0.4699 - val_loss: 0.7145
Epoch 3/15
42/42             11s 266ms/step -
accuracy: 0.6122 - loss: 0.6639 - val_accuracy: 0.4703 - val_loss: 0.7084
Epoch 4/15
42/42             12s 280ms/step -
```

```
accuracy: 0.6268 - loss: 0.6540 - val_accuracy: 0.4793 - val_loss: 0.6978
Epoch 5/15
42/42             11s 263ms/step -
accuracy: 0.6260 - loss: 0.6578 - val_accuracy: 0.4857 - val_loss: 0.7141
Epoch 6/15
42/42             11s 270ms/step -
accuracy: 0.6272 - loss: 0.6556 - val_accuracy: 0.4746 - val_loss: 0.7355
Epoch 7/15
42/42             11s 261ms/step -
accuracy: 0.6330 - loss: 0.6512 - val_accuracy: 0.4883 - val_loss: 0.7113
Epoch 8/15
42/42             11s 267ms/step -
accuracy: 0.6267 - loss: 0.6472 - val_accuracy: 0.4883 - val_loss: 0.7060
Epoch 9/15
42/42             11s 265ms/step -
accuracy: 0.6407 - loss: 0.6460 - val_accuracy: 0.5741 - val_loss: 0.6846
Epoch 10/15
42/42             11s 273ms/step -
accuracy: 0.6222 - loss: 0.6506 - val_accuracy: 0.4806 - val_loss: 0.7247
Epoch 11/15
42/42             11s 261ms/step -
accuracy: 0.6216 - loss: 0.6567 - val_accuracy: 0.4870 - val_loss: 0.7169
Epoch 12/15
42/42             11s 265ms/step -
accuracy: 0.6346 - loss: 0.6482 - val_accuracy: 0.5011 - val_loss: 0.7106
Epoch 13/15
42/42             11s 264ms/step -
accuracy: 0.6312 - loss: 0.6439 - val_accuracy: 0.4780 - val_loss: 0.7453
Epoch 14/15
42/42             11s 271ms/step -
accuracy: 0.6416 - loss: 0.6427 - val_accuracy: 0.4844 - val_loss: 0.7144
Epoch 15/15
42/42             11s 261ms/step -
accuracy: 0.6339 - loss: 0.6399 - val_accuracy: 0.4895 - val_loss: 0.7086
```

[ ]:
```python
pred = model.predict(x_test)
pred = [1.0 if p>= 0.5 else 0.0 for p in pred]

print('accuracy score: ', metrics.accuracy_score(y_test, pred))
print(metrics.classification_report(y_test, pred))
print("Confusion matrix:\n", metrics.confusion_matrix(y_test, pred))
```

```
180/180             4s 22ms/step
accuracy score:  0.6270477518299059
              precision    recall  f1-score   support

           0       0.62      0.96      0.76      3439
           1       0.68      0.13      0.22      2299
```

```
    accuracy                           0.63      5738
   macro avg       0.65      0.54      0.49      5738
weighted avg       0.65      0.63      0.54      5738

Confusion matrix:
 [[3303  136]
 [2004  295]]
```

[ ]:
```
model = models.Sequential()
model.add(layers.Embedding(vocab_size, 32))
model.add(layers.LSTM(32, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))
model.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['accuracy'])
```

[ ]:
```
history = model.fit(x_train,
                    y_train,
                    epochs=15,
                    batch_size = batch_size,
                    validation_split=0.1,
                    verbose = 1)
```

```
Epoch 1/15
42/42             32s 725ms/step -
accuracy: 0.5896 - loss: 0.6760 - val_accuracy: 0.4699 - val_loss: 0.7110
Epoch 2/15
42/42             31s 729ms/step -
accuracy: 0.6145 - loss: 0.6652 - val_accuracy: 0.4707 - val_loss: 0.7118
Epoch 3/15
42/42             30s 718ms/step -
accuracy: 0.6185 - loss: nan - val_accuracy: 0.4699 - val_loss: nan
Epoch 4/15
42/42             30s 719ms/step -
accuracy: 0.6134 - loss: nan - val_accuracy: 0.4699 - val_loss: nan
Epoch 5/15
42/42             30s 712ms/step -
accuracy: 0.6174 - loss: nan - val_accuracy: 0.4699 - val_loss: nan
Epoch 6/15
42/42             30s 710ms/step -
accuracy: 0.6222 - loss: nan - val_accuracy: 0.4699 - val_loss: nan
Epoch 7/15
42/42             30s 711ms/step -
accuracy: 0.6153 - loss: nan - val_accuracy: 0.4699 - val_loss: nan
Epoch 8/15
42/42             30s 719ms/step -
accuracy: 0.6123 - loss: nan - val_accuracy: 0.4699 - val_loss: nan
```

```
Epoch 9/15
42/42             30s 710ms/step -
accuracy: 0.6168 - loss: nan - val_accuracy: 0.4699 - val_loss: nan
Epoch 10/15
42/42             30s 713ms/step -
accuracy: 0.6173 - loss: nan - val_accuracy: 0.4699 - val_loss: nan
Epoch 11/15
42/42             30s 713ms/step -
accuracy: 0.6128 - loss: nan - val_accuracy: 0.4699 - val_loss: nan
Epoch 12/15
42/42             31s 748ms/step -
accuracy: 0.6137 - loss: nan - val_accuracy: 0.4699 - val_loss: nan
Epoch 13/15
42/42             34s 803ms/step -
accuracy: 0.6180 - loss: nan - val_accuracy: 0.4699 - val_loss: nan
Epoch 14/15
42/42             32s 766ms/step -
accuracy: 0.6140 - loss: nan - val_accuracy: 0.4699 - val_loss: nan
Epoch 15/15
42/42             30s 716ms/step -
accuracy: 0.6143 - loss: nan - val_accuracy: 0.4699 - val_loss: nan
```

[ ]:
```
pred = model.predict(x_test)
pred = [1.0 if p>= 0.5 else 0.0 for p in pred]

print('accuracy score: ', metrics.accuracy_score(y_test, pred))
print(metrics.classification_report(y_test, pred))
print("Confusion matrix:\n", metrics.confusion_matrix(y_test, pred))
```

```
180/180             7s 38ms/step
accuracy score:  0.5993377483443708
              precision    recall  f1-score   support

           0       0.60      1.00      0.75      3439
           1       0.00      0.00      0.00      2299

    accuracy                           0.60      5738
   macro avg       0.30      0.50      0.37      5738
weighted avg       0.36      0.60      0.45      5738

Confusion matrix:
 [[3439    0]
 [2299    0]]

C:\Users\ashur\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.11_qbz5n
2kfra8p0\LocalCache\local-packages\Python311\site-
packages\sklearn\metrics\_classification.py:1509: UndefinedMetricWarning:
Precision is ill-defined and being set to 0.0 in labels with no predicted
samples. Use `zero_division` parameter to control this behavior.
```

```
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
C:\Users\ashur\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.11_qbz5n
2kfra8p0\LocalCache\local-packages\Python311\site-
packages\sklearn\metrics\_classification.py:1509: UndefinedMetricWarning:
Precision is ill-defined and being set to 0.0 in labels with no predicted
samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
C:\Users\ashur\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.11_qbz5n
2kfra8p0\LocalCache\local-packages\Python311\site-
packages\sklearn\metrics\_classification.py:1509: UndefinedMetricWarning:
Precision is ill-defined and being set to 0.0 in labels with no predicted
samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
```

[ ]:
```
model = models.Sequential()
model.add(layers.Embedding(vocab_size, 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))

model.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['accuracy'])
```

[ ]:
```
history = model.fit(x_train,
                    y_train,
                    epochs=15,
                    batch_size = batch_size,
                    validation_split=0.1,
                    verbose = 1)
```

```
Epoch 1/15
42/42             6s 122ms/step -
accuracy: 0.6138 - loss: 5.6039 - val_accuracy: 0.4699 - val_loss: 0.7129
Epoch 2/15
42/42             5s 117ms/step -
accuracy: 0.6158 - loss: 0.6738 - val_accuracy: 0.4699 - val_loss: 0.7197
Epoch 3/15
42/42             5s 117ms/step -
accuracy: 0.6138 - loss: 0.6670 - val_accuracy: 0.4699 - val_loss: 0.7061
Epoch 4/15
42/42             5s 118ms/step -
accuracy: 0.6127 - loss: 0.6645 - val_accuracy: 0.4699 - val_loss: 0.7020
Epoch 5/15
42/42             5s 119ms/step -
accuracy: 0.6179 - loss: 0.6587 - val_accuracy: 0.4699 - val_loss: 0.6928
```

```
Epoch 6/15
42/42             5s 117ms/step -
accuracy: 0.6177 - loss: 0.6545 - val_accuracy: 0.4699 - val_loss: 0.6907
Epoch 7/15
42/42             5s 117ms/step -
accuracy: 0.6216 - loss: 0.6449 - val_accuracy: 0.4699 - val_loss: 0.6541
Epoch 8/15
42/42             5s 116ms/step -
accuracy: 0.6225 - loss: 0.6212 - val_accuracy: 0.7172 - val_loss: 0.5267
Epoch 9/15
42/42             5s 118ms/step -
accuracy: 0.7301 - loss: 0.5438 - val_accuracy: 0.9483 - val_loss: 0.3015
Epoch 10/15
42/42             5s 123ms/step -
accuracy: 0.7879 - loss: 0.4744 - val_accuracy: 0.7945 - val_loss: 0.3719
Epoch 11/15
42/42             5s 116ms/step -
accuracy: 0.8202 - loss: 0.4391 - val_accuracy: 0.8014 - val_loss: 0.3605
Epoch 12/15
42/42             5s 123ms/step -
accuracy: 0.7882 - loss: 0.4796 - val_accuracy: 0.8513 - val_loss: 0.3362
Epoch 13/15
42/42             5s 124ms/step -
accuracy: 0.8502 - loss: 0.3767 - val_accuracy: 0.8159 - val_loss: 0.3377
Epoch 14/15
42/42             5s 118ms/step -
accuracy: 0.8608 - loss: 0.3613 - val_accuracy: 0.8244 - val_loss: 0.3398
Epoch 15/15
42/42             5s 119ms/step -
accuracy: 0.8575 - loss: 0.3964 - val_accuracy: 0.7437 - val_loss: 0.4189
```

[ ]:
```
pred = model.predict(x_test)
pred = [1.0 if p>= 0.5 else 0.0 for p in pred]

print('accuracy score: ', metrics.accuracy_score(y_test, pred))
print(metrics.classification_report(y_test, pred))
print("Confusion matrix:\n", metrics.confusion_matrix(y_test, pred))
```

```
180/180             1s 3ms/step
accuracy score:  0.8387940048797491
              precision    recall  f1-score   support

           0       0.81      0.96      0.88      3439
           1       0.91      0.66      0.77      2299

    accuracy                           0.84      5738
   macro avg       0.86      0.81      0.82      5738
weighted avg       0.85      0.84      0.83      5738
```

```
Confusion matrix:
 [[3298  141]
 [ 784 1515]]
```

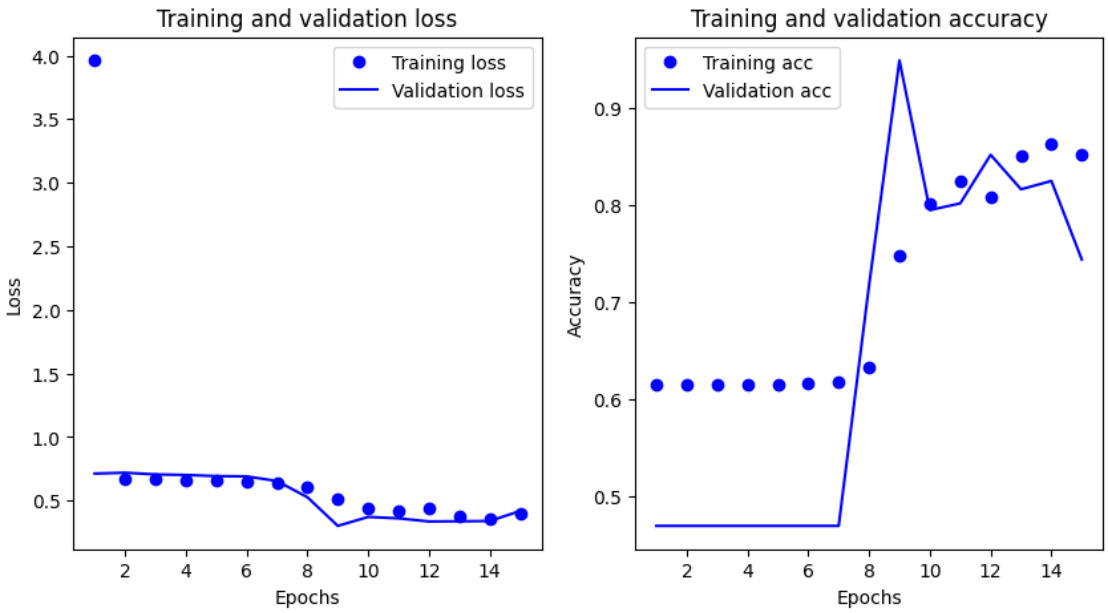
[ ]:
```
import matplotlib.pyplot as plt

loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(1, len(loss)+1)

fig, axs = plt.subplots(1, 2, figsize=(10, 5))
axs[0].plot(epochs, loss, 'bo', label='Training loss')
axs[0].plot(epochs, val_loss, 'b', label='Validation loss')
axs[0].set_title('Training and validation loss')
axs[0].set_xlabel('Epochs')
axs[0].set_ylabel('Loss')
axs[0].legend()
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

axs[1].plot(epochs, acc, 'bo', label='Training acc')
axs[1].plot(epochs, val_acc, 'b', label='Validation acc')
axs[1].set_title('Training and validation accuracy')
axs[1].set_xlabel('Epochs')
axs[1].set_ylabel('Accuracy')
axs[1].legend()

plt.show()
```



[ ]:
```
embeddings_index = {}
with open("glove.6B.100d.txt") as f:
    for line in f:
        word, coefs = line.split(maxsplit=1)
        coefs = np.fromstring(coefs, "f", sep=" ")
        embeddings_index[word] = coefs

print("Found %s word vectors." % len(embeddings_index))
```

```
Found 400000 word vectors.
```

[ ]:
```
from tensorflow.keras.layers import TextVectorization

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

[ ]:
```
voc = vectorizer.get_vocabulary()
word_index = dict(zip(voc, range(len(voc))))
```

[ ]:
```
num_tokens = len(voc) + 2
embedding_dim = 100
hits = 0
misses = 0

# Prepare embedding matrix
embedding_matrix = np.zeros((num_tokens, 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 16200 words (3800 misses)
```

[ ]:
```
from keras.layers import Embedding

embedding_layer = Embedding(
    num_tokens,
    embedding_dim,
    embeddings_initializer=keras.initializers.Constant(embedding_matrix),
```

```
    trainable=False,
)
```

[ ]:
```
int_sequences_input = keras.Input(shape=(None,), dtype="int64")
embedded_sequences = embedding_layer(int_sequences_input)
x = layers.Conv1D(32, 7, activation='relu')(embedded_sequences)
x = layers.MaxPooling1D(5)(x)
x = layers.Conv1D(32, 7, activation='relu')(x)
x = layers.GlobalMaxPooling1D()(x)
preds = layers.Dense(1, activation="sigmoid")(x)
model = keras.Model(int_sequences_input, preds)
model.summary()
```

```
Model: "functional_47"
```

Layer (type)	Output Shape	Param #
input_layer_49 (InputLayer)	(None, None)	0
embedding_38 (Embedding)	(None, None, 100)	2,000,200
conv1d_8 (Conv1D)	(None, None, 32)	22,432
max_pooling1d_3 (MaxPooling1D)	(None, None, 32)	0
conv1d_9 (Conv1D)	(None, None, 32)	7,200
global_max_pooling1d_3 (GlobalMaxPooling1D)	(None, 32)	0
dense_64 (Dense)	(None, 1)	33

```
Total params: 2,029,865 (7.74 MB)

Trainable params: 2,029,865 (7.74 MB)

Non-trainable params: 0 (0.00 B)
```

[ ]:
```
model.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['accuracy'])
```

```
[ ]: history = model.fit(x_train,
                         y_train,
                         epochs=15,
                         batch_size = batch_size,
                         validation_split=0.1,
                         verbose = 1)
```

```
Epoch 1/15
42/42             12s 267ms/step -
accuracy: 0.6539 - loss: 0.6048 - val_accuracy: 0.7933 - val_loss: 0.3832
Epoch 2/15
42/42             11s 260ms/step -
accuracy: 0.8365 - loss: 0.3676 - val_accuracy: 0.8466 - val_loss: 0.3005
Epoch 3/15
42/42             11s 269ms/step -
accuracy: 0.8869 - loss: 0.2790 - val_accuracy: 0.9466 - val_loss: 0.1664
Epoch 4/15
42/42             11s 271ms/step -
accuracy: 0.8988 - loss: 0.2474 - val_accuracy: 0.9009 - val_loss: 0.2140
Epoch 5/15
42/42             11s 264ms/step -
accuracy: 0.9150 - loss: 0.2196 - val_accuracy: 0.9389 - val_loss: 0.1521
Epoch 6/15
42/42             11s 260ms/step -
accuracy: 0.9193 - loss: 0.2039 - val_accuracy: 0.9419 - val_loss: 0.1445
Epoch 7/15
42/42             11s 257ms/step -
accuracy: 0.9238 - loss: 0.1938 - val_accuracy: 0.9650 - val_loss: 0.1182
Epoch 8/15
42/42             11s 259ms/step -
accuracy: 0.9248 - loss: 0.1948 - val_accuracy: 0.9564 - val_loss: 0.1162
Epoch 9/15
42/42             11s 264ms/step -
accuracy: 0.9280 - loss: 0.1882 - val_accuracy: 0.9663 - val_loss: 0.1070
Epoch 10/15
42/42             11s 260ms/step -
accuracy: 0.9359 - loss: 0.1741 - val_accuracy: 0.9440 - val_loss: 0.1351
Epoch 11/15
42/42             11s 259ms/step -
accuracy: 0.9366 - loss: 0.1709 - val_accuracy: 0.9684 - val_loss: 0.1007
Epoch 12/15
42/42             11s 260ms/step -
accuracy: 0.9413 - loss: 0.1595 - val_accuracy: 0.9774 - val_loss: 0.0877
Epoch 13/15
42/42             11s 259ms/step -
accuracy: 0.9390 - loss: 0.1581 - val_accuracy: 0.9210 - val_loss: 0.1718
Epoch 14/15
42/42             11s 260ms/step -
```

```
accuracy: 0.9438 - loss: 0.1514 - val_accuracy: 0.9795 - val_loss: 0.0740
Epoch 15/15
42/42             11s 262ms/step -
accuracy: 0.9425 - loss: 0.1552 - val_accuracy: 0.9778 - val_loss: 0.0777
```

[ ]:
```
pred = model.predict(x_test)
pred = [1.0 if p>= 0.5 else 0.0 for p in pred]

print('accuracy score: ', metrics.accuracy_score(y_test, pred))
print(metrics.classification_report(y_test, pred))
print("Confusion matrix:\n", metrics.confusion_matrix(y_test, pred))
```

```
180/180             1s 5ms/step
accuracy score:  0.9349947716974556
              precision    recall  f1-score   support

           0       0.95      0.94      0.95      3439
           1       0.92      0.92      0.92      2299

    accuracy                           0.93      5738
   macro avg       0.93      0.93      0.93      5738
weighted avg       0.94      0.93      0.94      5738

Confusion matrix:
 [[3247  192]
 [ 181 2118]]
```

Accuracy Scores

sequential - 98.98%

SimpleRNN - 62.70%

LSTM - 59.93%

CNN - 83.87

CNN with GloVe embedding - 93.49%

Runtime

sequential - 3s

SimpleRNN - 2m 50s

LSTM - 7m 40s

CNN - 1m 16s

CNN with GloVe embedding - 2m 46s

Analyzing the data, it seems that the best performing model was the sequential model that I started with. This one was by far the easiest to work with and was also the fastest and most accurate. Out of the other 3 models I tested the Convolutional model was the best. It was able to learn the data well although the cnn did train weirdly seeing no gains for around 7 epochs and then randomly

shooting up. The other two models struggled to learn the data in the 15 epochs I set with the lstm only guessing one way and the rnn model barely doing better. They may have eventually learned the data if I set a higher epoch count. For the data and model, I used the same data and kept the model settings the same for all of them to be able to directly compare them.

I chose to proceed with the cnn as it performed the best out of the three. I did the embedding with GloVe 6b 100d and it performed better than with the default embedding achieving 10 percent higher accuracy. It did take twice the time to train but it was not a drastically large difference.