```
!pip install transformers
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: transformers in /usr/local/lib/python3.8/dist-packages (4.25.1)
Requirement already satisfied: tqdm>=4.27 in /usr/local/lib/python3.8/dist-packages (from transformers) (4.64.1)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.8/dist-packages (from transformers) (21.3)
Requirement already satisfied: filelock in /usr/local/lib/python3.8/dist-packages (from transformers) (3.8.0)
Requirement already satisfied: huggingface-hub<1.0,>=0.10.0 in /usr/local/lib/python3.8/dist-packages (from transformers) (0.
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.8/dist-packages (from transformers) (6.0)
Requirement already satisfied: requests in /usr/local/lib/python3.8/dist-packages (from transformers) (2.23.0)
Requirement already satisfied: regex!=2019.12.17 in /usr/local/lib/python3.8/dist-packages (from transformers) (2022.6.2)
Requirement already satisfied: tokenizers!=0.11.3,<0.14,>=0.11.1 in /usr/local/lib/python3.8/dist-packages (from transformers
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.8/dist-packages (from transformers) (1.21.6)
Requirement already satisfied: typing-extensions>=3.7.4.3 in /usr/local/lib/python3.8/dist-packages (from huggingface-hub<1.(
Requirement already satisfied: pyparsing!=3.0.5,>=2.0.2 in /usr/local/lib/python3.8/dist-packages (from packaging>=20.0->tran
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.8/dist-packages (from requests->transformers) (2(
Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python3.8/dist-packages (from requests->transformers) (3.(
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.8/dist-packages (from requests->transformers) (2.10)
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in /usr/local/lib/python3.8/dist-packages (from reques
```

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

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
import seaborn as sns
```

## 1. Go to Kaggle.com. Find a text classification data set that interests you. Divide into train/test. Create a graph showing the distribution of the target classes. Describe the data set and what the model should be able to predict.

The data set contains text messages exchanged, including spam messages. THe model should be able to predict if a given message is spam or not spam (ham).

```
df = pd.read_csv('Data.csv')
```

```
df.head()
```

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

```
df.shape
```

```
(5572, 2)
```

```
df.isna().sum()
```

```
Category    0
Message     0
dtype: int64
```

Map the labels to 1(spam) and 0(ham)

```
map = {
    'spam': 1,
    'ham': 0
```

```
}

df['Category'] = df['Category'].map(map)
df['Category']
```

```
0       0
1       0
2       1
3       0
4       0
       ..
5567    1
5568    0
5569    0
5570    0
5571    0
Name: Category, Length: 5572, dtype: int64
```

```
X = df['Message']
y = df['Category']
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 42)
```

```
X_train.shape, X_test.shape
```

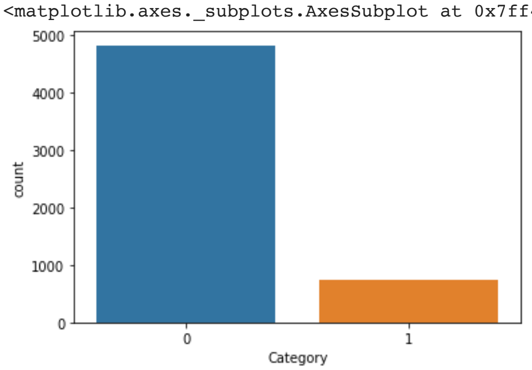
```
((4457,), (1115,))
```

```
y_train.shape, y_test.shape
```

```
((4457,), (1115,))
```

```
sns.countplot(y)
```

```
/usr/local/lib/python3.8/dist-packages/seaborn/_decorators.py:36: FutureWar
  warnings.warn(
<matplotlib.axes._subplots.AxesSubplot at 0x7ff48322f280>
```




```
len(set(y_train))
```

```
2
```

```
X_train
```

```
1978    Reply to win £100 weekly! Where will the 2006 ...
3989    Hello. Sort of out in town already. That . So ...
3935     How come guoyang go n tell her? Then u told her?
4078    Hey sathya till now we dint meet not even a si...
4086    Orange brings you ringtones from all time Char...
                              ...
3772    Hi, wlcome back, did wonder if you got eaten b...
5191                               Sorry, I'll call later
5226        Prabha..i'm soryda..realy..frm heart i'm sory
5390                           Nt joking seriously i told
860                Did he just say somebody is named tampa
Name: Message, Length: 4457, dtype: object
```

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

## Tokenize the data

```
tokenizer = AutoTokenizer.from_pretrained('bert-base-cased')
```

```
X_train_tokenized = tokenizer(list(X_train), return_tensors = 'np', padding = True)['input_ids']
X_test_tokenized = tokenizer(list(X_test), return_tensors = 'np', padding = True)['input_ids']
```

```
X_train_tokenized
```

```
array([[  101, 20777,  1193, ...,     0,     0,     0],
       [  101,  8667,   119, ...,     0,     0,     0],
       [  101,  1731,  1435, ...,     0,     0,     0],
       ...,
       [  101,   153, 17952, ...,     0,     0,     0],
       [  101,   151,  1204, ...,     0,     0,     0],
       [  101,  2966,  1119, ...,     0,     0,     0]])
```

```
X_test_tokenized
```

```
array([[ 101,  156, 3530, ...,    0,    0,    0],
       [ 101, 1262, 1145, ...,    0,    0,    0],
       [ 101,  150, 6262, ...,    0,    0,    0],
       ...,
       [ 101, 2160,  178, ...,    0,    0,    0],
       [ 101, 1731, 1132, ...,    0,    0,    0],
       [ 101,  144,  119, ...,    0,    0,    0]])
```

## Vectorize the Data

Code from https://github.com/kjmazidi/NLP/blob/master/Part_6-Deep%20Learning/Chapter_23_Keras/Keras_imbd_1_Dense_Sequential.ipynb

```
max_X_train = X_train_tokenized.max()
max_X_test = X_test_tokenized.max()

dim = max(max_X_train, max_X_test)
dim += 1
dim
```

```
28160
```

```
def vectorize_sequences(sequences, dimension = dim):
    # Create an all-zero matrix of shape (len(sequences), dimension)
    results = np.zeros((len(sequences), dimension))
    for i, sequence in enumerate(sequences):
        results[i, sequence] = 1.  # set specific indices of results[i] to 1s
    return results
```

```
X_train_vec = vectorize_sequences(X_train_tokenized)
X_test_vec = vectorize_sequences(X_test_tokenized)
```

```
X_train_vec.shape, X_test_vec.shape
```

```
((4457, 28160), (1115, 28160))
```

## Model Building

```
model = models.Sequential()
model.add(layers.Dense(16, activation='relu', input_shape=(dim,)))
model.add(layers.Dense(16, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))
```

```
model.compile(optimizer = 'rmsprop',
              loss = tf.keras.losses.BinaryCrossentropy(from_logits=True),
              metrics = ['accuracy'])
```

```
X_val_size = int(len(X_train_vec) * 0.2)
X_val = X_train_vec[:X_val_size]
```

```
partial_X_train = X_train_vec[X_val_size:]

y_val_size = int(len(y_train) * 0.2)
y_val = y_train[:y_val_size]
partial_y_train = y_train[y_val_size:]


history = model.fit(partial_X_train,
                    partial_y_train,
                    epochs = 20,
                    batch_size = 512,
                    validation_data = (X_val, y_val))
```

```
Epoch 1/20
/usr/local/lib/python3.8/dist-packages/tensorflow/python/util/dispatch.py:1082: UserWarning: "`binary_crossentropy` received
  return dispatch_target(*args, **kwargs)
7/7 [==============================] - 3s 144ms/step - loss: 0.5777 - accuracy: 0.8985 - val_loss: 0.4541 - val_accuracy: 0.9
Epoch 2/20
7/7 [==============================] - 0s 73ms/step - loss: 0.3910 - accuracy: 0.9801 - val_loss: 0.3331 - val_accuracy: 0.98
Epoch 3/20
7/7 [==============================] - 0s 63ms/step - loss: 0.2863 - accuracy: 0.9877 - val_loss: 0.2544 - val_accuracy: 0.98
Epoch 4/20
7/7 [==============================] - 0s 59ms/step - loss: 0.2153 - accuracy: 0.9907 - val_loss: 0.1983 - val_accuracy: 0.98
Epoch 5/20
7/7 [==============================] - 0s 70ms/step - loss: 0.1641 - accuracy: 0.9916 - val_loss: 0.1566 - val_accuracy: 0.98
Epoch 6/20
7/7 [==============================] - 0s 60ms/step - loss: 0.1257 - accuracy: 0.9927 - val_loss: 0.1251 - val_accuracy: 0.98
Epoch 7/20
7/7 [==============================] - 0s 61ms/step - loss: 0.0967 - accuracy: 0.9933 - val_loss: 0.1014 - val_accuracy: 0.99
Epoch 8/20
7/7 [==============================] - 1s 89ms/step - loss: 0.0748 - accuracy: 0.9955 - val_loss: 0.0835 - val_accuracy: 0.99
Epoch 9/20
7/7 [==============================] - 1s 84ms/step - loss: 0.0579 - accuracy: 0.9964 - val_loss: 0.0699 - val_accuracy: 0.99
Epoch 10/20
7/7 [==============================] - 0s 74ms/step - loss: 0.0449 - accuracy: 0.9972 - val_loss: 0.0597 - val_accuracy: 0.99
Epoch 11/20
7/7 [==============================] - 0s 58ms/step - loss: 0.0348 - accuracy: 0.9975 - val_loss: 0.0526 - val_accuracy: 0.99
Epoch 12/20
7/7 [==============================] - 0s 56ms/step - loss: 0.0272 - accuracy: 0.9978 - val_loss: 0.0463 - val_accuracy: 0.99
Epoch 13/20
7/7 [==============================] - 0s 49ms/step - loss: 0.0212 - accuracy: 0.9983 - val_loss: 0.0427 - val_accuracy: 0.99
Epoch 14/20
7/7 [==============================] - 0s 53ms/step - loss: 0.0166 - accuracy: 0.9989 - val_loss: 0.0405 - val_accuracy: 0.99
Epoch 15/20
7/7 [==============================] - 0s 53ms/step - loss: 0.0132 - accuracy: 0.9989 - val_loss: 0.0384 - val_accuracy: 0.99
Epoch 16/20
7/7 [==============================] - 0s 51ms/step - loss: 0.0105 - accuracy: 0.9994 - val_loss: 0.0364 - val_accuracy: 0.99
Epoch 17/20
7/7 [==============================] - 0s 56ms/step - loss: 0.0084 - accuracy: 0.9994 - val_loss: 0.0367 - val_accuracy: 0.99
Epoch 18/20
7/7 [==============================] - 0s 52ms/step - loss: 0.0068 - accuracy: 0.9994 - val_loss: 0.0369 - val_accuracy: 0.99
Epoch 19/20
7/7 [==============================] - 0s 58ms/step - loss: 0.0055 - accuracy: 0.9994 - val_loss: 0.0378 - val_accuracy: 0.99
Epoch 20/20
7/7 [==============================] - 0s 64ms/step - loss: 0.0046 - accuracy: 0.9994 - val_loss: 0.0377 - val_accuracy: 0.99
```

## Evaluation

```
from sklearn.metrics import classification_report


y_pred = model.predict(X_test_vec)
y_pred = [1.0 if p >= 0.5 else 0.0 for p in y_pred]
print(classification_report(y_test, y_pred))
```

```
35/35 [==============================] - 0s 3ms/step
              precision    recall  f1-score   support

           0       0.99      1.00      0.99       966
           1       1.00      0.92      0.96       149

    accuracy                           0.99      1115
   macro avg       0.99      0.96      0.98      1115
weighted avg       0.99      0.99      0.99      1115
```

```
losses_and_metrics = model.evaluate(X_test_vec, y_test, batch_size = 128)
losses_and_metrics
```

```
9/9 [==============================] - 0s 11ms/step - loss: 0.0606 - accuracy: 0.9892
[0.06058812141418457, 0.9892376661300659]
```

## 3. Try a different architecture like RNN, CNN, etc and evaluate on the test data

### RNN

```
max_features = 10000
maxlen = 500
batch_size = 32
```

```
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='rmsprop',
              loss='binary_crossentropy',
              metrics=['accuracy'])
```

```
history = model.fit(partial_X_train,
                    partial_y_train,
                    epochs = 10,
                    batch_size = 128,
                    validation_data = (X_val, y_val))
```

```
Epoch 1/10
```

```
pred = model.predict(X_test_vec)
pred = [1.0 if p>= 0.5 else 0.0 for p in pred]
print(classification_report(y_test, pred))
```

```
losses_and_metrics = model.evaluate(X_test_vec, y_test, batch_size = 128)
losses_and_metrics
```

### GRU

```
max_features = 10000
maxlen = 500
batch_size = 32
```

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

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

```
history = model.fit(partial_X_train,
                    partial_y_train,
                    epochs = 10,
                    batch_size = 128,
                    validation_data = (X_val, y_val))
```

```
Epoch 1/10
28/28 [==============================] - 28s 841ms/step - loss: 0.4787 - accuracy: 0.8390 - val_loss: 0.3816 - val_accuracy:
Epoch 2/10
28/28 [==============================] - 23s 832ms/step - loss: 0.3998 - accuracy: 0.8637 - val_loss: 0.3790 - val_accuracy:
Epoch 3/10
28/28 [==============================] - 31s 1s/step - loss: 0.3989 - accuracy: 0.8637 - val_loss: 0.3833 - val_accuracy: 0.8
Epoch 4/10
28/28 [==============================] - 24s 853ms/step - loss: 0.3988 - accuracy: 0.8637 - val_loss: 0.3801 - val_accuracy:
```

```
Epoch 5/10
28/28 [==============================] - 23s 830ms/step - loss: 0.3981 - accuracy: 0.8637 - val_loss: 0.3747 - val_accuracy:
Epoch 6/10
28/28 [==============================] - 25s 908ms/step - loss: 0.3963 - accuracy: 0.8637 - val_loss: 0.3753 - val_accuracy:
Epoch 7/10
28/28 [==============================] - 23s 830ms/step - loss: 0.3949 - accuracy: 0.8637 - val_loss: 0.3729 - val_accuracy:
Epoch 8/10
28/28 [==============================] - 23s 823ms/step - loss: 0.3941 - accuracy: 0.8654 - val_loss: 0.3721 - val_accuracy:
Epoch 9/10
28/28 [==============================] - 24s 843ms/step - loss: 0.3932 - accuracy: 0.8676 - val_loss: 0.3780 - val_accuracy:
Epoch 10/10
28/28 [==============================] - 23s 819ms/step - loss: 0.3929 - accuracy: 0.8676 - val_loss: 0.3721 - val_accuracy:
```

```
pred = model.predict(X_test_vec)
pred = [1.0 if p>= 0.5 else 0.0 for p in pred]
print(classification_report(y_test, pred))
```

```
35/35 [==============================] - 10s 288ms/step
              precision    recall  f1-score   support

           0       0.87      1.00      0.93       966
           1       1.00      0.03      0.05       149

    accuracy                           0.87      1115
   macro avg       0.93      0.51      0.49      1115
weighted avg       0.89      0.87      0.81      1115
```

```
losses_and_metrics = model.evaluate(X_test_vec, y_test, batch_size = 128)
losses_and_metrics
```

```
9/9 [==============================] - 3s 323ms/step - loss: 0.3909 - accuracy: 0.8700
[0.39092352986335754, 0.8699551820755005]
```

## CNN

```
model = models.Sequential()
model.add(layers.Embedding(max_features, 128, input_length = dim))
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.summary()
```

```
Model: "sequential_6"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_4 (Embedding)     (None, 28160, 128)        1280000

 conv1d_6 (Conv1D)           (None, 28154, 32)         28704

 max_pooling1d_3 (MaxPooling  (None, 5630, 32)         0
 1D)

 conv1d_7 (Conv1D)           (None, 5624, 32)          7200

 global_max_pooling1d_3 (Glo  (None, 32)               0
 balMaxPooling1D)

 dense_10 (Dense)            (None, 1)                 33

=================================================================
Total params: 1,315,937
Trainable params: 1,315,937
Non-trainable params: 0
_________________________________________________________________
```

```
model.compile(optimizer=tf.keras.optimizers.RMSprop(lr=1e-4),  # set learning rate
              loss='binary_crossentropy',
              metrics=['accuracy'])
```

```
/usr/local/lib/python3.8/dist-packages/keras/optimizers/optimizer_v2/rmsprop.py:135: UserWarning: The `lr` argument is deprec
  super(RMSprop, self).__init__(name, **kwargs)
```

```
history = model.fit(partial_X_train,
                    partial_y_train,
                    epochs = 10,
                    batch_size = 128,
                    validation_data = (X_val, y_val))
```

```
Epoch 1/10
28/28 [==============================] - 17s 573ms/step - loss: 0.3818 - accuracy: 0.8637 - val_loss: 0.3494 - val_accuracy:
Epoch 2/10
28/28 [==============================] - 14s 485ms/step - loss: 0.3626 - accuracy: 0.8637 - val_loss: 0.3298 - val_accuracy:
Epoch 3/10
28/28 [==============================] - 14s 492ms/step - loss: 0.3448 - accuracy: 0.8637 - val_loss: 0.3126 - val_accuracy:
Epoch 4/10
28/28 [==============================] - 14s 498ms/step - loss: 0.3324 - accuracy: 0.8637 - val_loss: 0.3041 - val_accuracy:
Epoch 5/10
28/28 [==============================] - 14s 499ms/step - loss: 0.3234 - accuracy: 0.8637 - val_loss: 0.2975 - val_accuracy:
Epoch 6/10
28/28 [==============================] - 14s 499ms/step - loss: 0.3161 - accuracy: 0.8637 - val_loss: 0.2896 - val_accuracy:
Epoch 7/10
28/28 [==============================] - 14s 504ms/step - loss: 0.3145 - accuracy: 0.8637 - val_loss: 0.2853 - val_accuracy:
Epoch 8/10
28/28 [==============================] - 14s 502ms/step - loss: 0.3107 - accuracy: 0.8637 - val_loss: 0.3086 - val_accuracy:
Epoch 9/10
28/28 [==============================] - 14s 499ms/step - loss: 0.3184 - accuracy: 0.8643 - val_loss: 0.2800 - val_accuracy:
Epoch 10/10
28/28 [==============================] - 14s 497ms/step - loss: 0.3092 - accuracy: 0.8651 - val_loss: 0.2799 - val_accuracy:
```

```
from sklearn.metrics import classification_report

pred = model.predict(X_test_vec)
pred = [1.0 if p>= 0.5 else 0.0 for p in pred]
print(classification_report(y_test, pred))
```

```
35/35 [==============================] - 2s 49ms/step
              precision    recall  f1-score   support

           0       0.87      1.00      0.93       966
           1       0.67      0.01      0.03       149

    accuracy                           0.87      1115
   macro avg       0.77      0.51      0.48      1115
weighted avg       0.84      0.87      0.81      1115
```

```
losses_and_metrics = model.evaluate(X_test_vec, y_test, batch_size = 128)
losses_and_metrics
```

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

Looking at the performance of the Sequential model, we see that it acheives very high accuracies in testing, training and validation. The model wsd able to achieve 0.9994 for training, 0.9933 in validation, and 0.9883 in testing. Also, the model had very low loss in these three areas as well: 0.0047 in training, 0.0355 in validation, and 0.0574 in testing. Looking at the classifiation report, for bith classes (0 and 1) the model acheived very high perecision and recall, both near 1. Finally, the model had a very quick compute time of around 47 ms per step in each epoch.

Looking at the performnce of the CNN model, we see that it acheives high accuracy and low loss in testing, training and validation, but not as high as the Sequential model. The model was able to achive 0.8651 accuracy and 0.3092 loss in training, 0.8765 accuracy and 0.2799 loss in validation and 0.8655 accuracy and 0.3022 loss in testing. While these results are good, they are not as good as the Sequential model. Looking at the classification report, we see that the precision and recall are high for only one class, 0. FOr 1, both metrics are 0, indicating an imbalanced model. Thus, the Sequential model performed better.

The RNN model took an insanely long time to run, so much so that I ran out of compute units during training. For this reason, I decided to also build a GRU model because it is a variation of the RNN. Specifically, the GRU is a simpler version of LSTM which is an RNN variation that solves the vanishing gradient problem.

Looking at the performance of the GRU model, we see that it acheives better accuracy and loss in training, testing and validation than the CNN, but worse than the Sequential model. The model achieved 0.8676 accuracy and 0.3929 loss in training, 0.8788 accuracy and 0.3721 loss in validation, and 0.8700 accuracy and 0.3909 loss in testing. Looking at the classfication report, we see high preicsion and recall for class 0 and high preicsion but low recall for class 1. This still indicates a slighly imabalnced model, not as much as the CNN. We can expect the RNN model to perform similarly to the GRU and likely better since RNNs are typially used for text classification.

Overall, the Sequential model performed the best based on the experiments ran. However, the RNN model would likely perform better if given the resources to run.

✓ 3s completed at 2:16 PM