```
from google.colab import files
uploaded = files.upload()
```

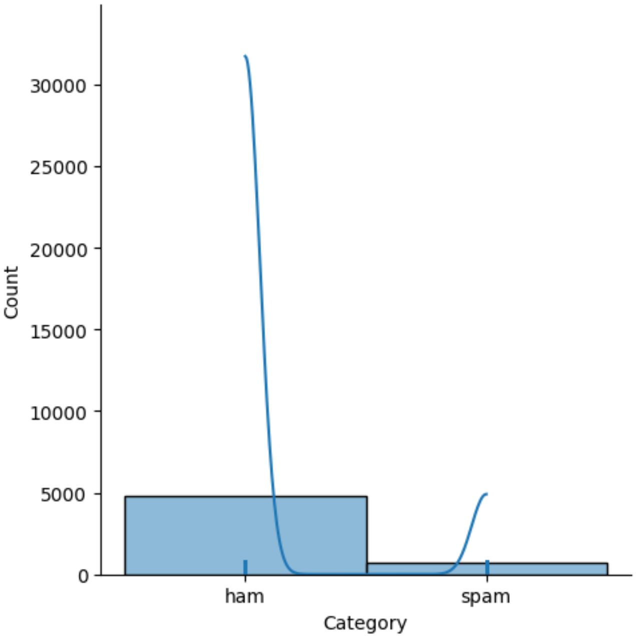
Choose Files spam_tclassification.csv

- **spam_tclassification.csv**(text/csv) - 485702 bytes, last modified: 9/20/2019 - 100% done

```
Saving spam_tclassification.csv to spam_tclassification.csv
```

```
import io
import pandas as pd
import numpy as np
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
df = pd.read_csv(io.BytesIO(uploaded['spam_tclassification.csv']))

print(df.head())

x= df.Message #feature
y = df.Category #target
from sklearn.feature_extraction.text import TfidfVectorizer

#divide the data into train and test
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3, random_state=21, stratify=y)

print(x_train.shape)
print(y_train.shape)
print(x_test.shape)
print(y_test.shape)

import seaborn as sb
#plot the graph
sb.displot(df['Category'], kde=True, rug=True)
```

```
  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...
(3900,)
(3900,)
(1672,)
(1672,)
<seaborn.axisgrid.FacetGrid at 0x7f0bf57fb8e0>
```



Dataset description:

The dataset shows examples of texts, then categorize each message spam (if the text is a spam) or ham (if the text is not a spam message). From the graph above we can see that most of the text message data is non-spam text messages "ham", approximately 5000 messages are ham texts. Approximately less than 1000 texts are spam messages. This assignment will use this dataset and apply some deep learning algorithms such as RNN and CNN to analyze the dataset.

```
# sequential model
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


# setting the seed
np.random.seed(1234)

#read file
df = pd.read_csv(io.BytesIO(uploaded['spam_tclassification.csv']))
print('rows and columns:', df.shape)
print(df.head())
# get the train and test data by splitting
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)


num_labels = 2
vocab_size = 25000
batch_size = 100

# tokenize and fit the data
tokenizer = Tokenizer(num_words=vocab_size)
tokenizer.fit_on_texts(train.Message)
#use the texts_to_matrix method to get the x train and test data
x_train = tokenizer.texts_to_matrix(train.Message, mode='tfidf')
x_test = tokenizer.texts_to_matrix(test.Message, mode='tfidf')

#use an encoder to get the y train and test data
encoder = LabelEncoder()
encoder.fit(train.Category)
y_train = encoder.transform(train.Category)
y_test = encoder.transform(test.Category)

# printing the shapes of the x and y data
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])

#build the seq model
model = models.Sequential()
model.add(layers.Dense(32, input_dim=vocab_size, kernel_initializer='normal', activation='relu'))
model.add(layers.Dense(1, kernel_initializer='normal', activation='sigmoid'))

 #specify tthe metrics
model.compile(loss='binary_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])
 #fit the seq model
fitted = model.fit(x_train, y_train,
                    batch_size=batch_size,
                    epochs=15,
                    verbose=1,
                    validation_split=0.1)
```

```
rows and columns: (5572, 2)
  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...
train data size:  (4464, 2)
test data size:  (1108, 2)
train shapes: (4464, 25000) (4464,)
```

```
     test shapes: (1108, 25000) (1108,)
     test first five labels: [0 1 1 1 0]
     Epoch 1/15
     41/41 [==============================] - 2s 33ms/step - loss: 0.5133 - accuracy: 0.8556 - val_loss: 0.3426 - val_accuracy: 0.
     Epoch 2/15
     41/41 [==============================] - 1s 26ms/step - loss: 0.2351 - accuracy: 0.9393 - val_loss: 0.1595 - val_accuracy: 0.
     Epoch 3/15
     41/41 [==============================] - 1s 25ms/step - loss: 0.0967 - accuracy: 0.9878 - val_loss: 0.0859 - val_accuracy: 0.
     Epoch 4/15
     41/41 [==============================] - 1s 26ms/step - loss: 0.0469 - accuracy: 0.9945 - val_loss: 0.0641 - val_accuracy: 0.
     Epoch 5/15
     41/41 [==============================] - 1s 25ms/step - loss: 0.0275 - accuracy: 0.9970 - val_loss: 0.0556 - val_accuracy: 0.
     Epoch 6/15
     41/41 [==============================] - 2s 39ms/step - loss: 0.0179 - accuracy: 0.9985 - val_loss: 0.0513 - val_accuracy: 0.
     Epoch 7/15
     41/41 [==============================] - 2s 40ms/step - loss: 0.0125 - accuracy: 0.9998 - val_loss: 0.0490 - val_accuracy: 0.
     Epoch 8/15
     41/41 [==============================] - 1s 27ms/step - loss: 0.0093 - accuracy: 0.9998 - val_loss: 0.0480 - val_accuracy: 0.
     Epoch 9/15
     41/41 [==============================] - 1s 27ms/step - loss: 0.0072 - accuracy: 0.9998 - val_loss: 0.0468 - val_accuracy: 0.
     Epoch 10/15
     41/41 [==============================] - 1s 25ms/step - loss: 0.0057 - accuracy: 1.0000 - val_loss: 0.0465 - val_accuracy: 0.
     Epoch 11/15
     41/41 [==============================] - 1s 25ms/step - loss: 0.0047 - accuracy: 1.0000 - val_loss: 0.0464 - val_accuracy: 0.
     Epoch 12/15
     41/41 [==============================] - 1s 27ms/step - loss: 0.0039 - accuracy: 1.0000 - val_loss: 0.0464 - val_accuracy: 0.
     Epoch 13/15
     41/41 [==============================] - 1s 26ms/step - loss: 0.0033 - accuracy: 1.0000 - val_loss: 0.0465 - val_accuracy: 0.
     Epoch 14/15
     41/41 [==============================] - 1s 25ms/step - loss: 0.0028 - accuracy: 1.0000 - val_loss: 0.0468 - val_accuracy: 0.
     Epoch 15/15
     41/41 [==============================] - 1s 25ms/step - loss: 0.0024 - accuracy: 1.0000 - val_loss: 0.0468 - val_accuracy: 0.
```

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

```
     12/12 [==============================] - 0s 14ms/step - loss: 0.0966 - accuracy: 0.9874
     Accuracy score:  0.987364649772644
```

```
print(acc_score)
```

```
     [0.0966106727719307, 0.987364649772644]
```

```
# future calculations based on the results
pred = model.predict(x_test)
pred_labels = [1 if p>0.5 else 0 for p in pred]
pred[:10]
```

```
     35/35 [==============================] - 0s 7ms/step
     array([[8.03798175e-05],
            [9.99803901e-01],
            [9.99747157e-01],
            [7.38569081e-01],
            [1.09881592e-04],
            [8.29916491e-09],
            [1.04014769e-04],
            [6.59058010e-03],
            [6.50125995e-11],
            [1.08977665e-05]], dtype=float32)
```

```
#sequential model score
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:  0.9873646209386282
     precision score:  0.9863013698630136
     recall score:  0.9230769230769231
     f1 score:  0.9536423841059603
```

Analyzing the Sequntial model:

in the sequential model, the data has to be split to train and test data and the split protioan can be any number between 0 and 1. In this assignment, the number picked to divide the data had to be anything less than 0.8. Then to get the x trained and test data, the Tokenize functiona was used to achieve good results for x_train and x_test data.

Then to get the results of y trained and test data, the Encoder function was used to get the best results. After obtaingin the trtained and test data for x and y, the sequential model was built using the x and y data. The seqential model used many proporties of the seqntial model such as density and layers.

Evaluation:

The accuracy and loss results of the model were very good, as seen above the last few epochs had the accuracy of 1 and loss score less than 0.01. Hence the accuracy result was very high and approximately equals to 0.987. Also, the prediction models showed that the model will be accurate for future data as well. Moreover, the accuracy measurements and metrics also conformed the accuracy of the data as this was the result of each metric: accuracy score: 0.9873646209386282 precision score: 0.9863013698630136 recall score: 0.9230769230769231 f1 score: 0.9536423841059603

```
#RNN
import io
import pandas as pd
import numpy as np
import tensorflow as tf
from tensorflow.keras import datasets, layers, models, preprocessing
import numpy as np
import pandas as pd
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
max_features = 10000
maxlen = 500
batch_size = 32

#read the file
df = pd.read_csv(io.BytesIO(uploaded['spam_tclassification.csv']))
print('rows and columns:', df.shape)
print(df.head())

# get the train and test data by splitting
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)



num_labels = 2
vocab_size = 25000
batch_size = 100

# tokenize and fit the data
tokenizer = Tokenizer(num_words=vocab_size)
tokenizer.fit_on_texts(train.Message)

#use the texts_to_sequences method to get the x train and test data
x_train = tokenizer.texts_to_sequences(train.Message)
x_test = tokenizer.texts_to_sequences(test.Message)

#use an encoder to get the y train and test data
encoder = LabelEncoder()
encoder.fit(train.Category)
y_train = encoder.transform(train.Category)
y_test = encoder.transform(test.Category)

# check the data
print("test first five labels:", y_test[:5])

# pad the x train and test datawhen using rnn
x_train = preprocessing.sequence.pad_sequences(x_train, maxlen=maxlen)
x_test = preprocessing.sequence.pad_sequences(x_test, maxlen=maxlen)

# build the RNN model using some functionalities from the sequential model
```

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

model.summary()
```

```
rows and columns: (5572, 2)
  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...
train data size:  (4419, 2)
test data size:  (1153, 2)
test first five labels: [1 0 1 1 0]
Model: "sequential_5"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 embedding_5 (Embedding)     (None, None, 32)          320000    
                                                                 
 simple_rnn_3 (SimpleRNN)    (None, 32)                2080      
                                                                 
 dense_5 (Dense)             (None, 1)                 33        
                                                                 
=================================================================
Total params: 322,113
Trainable params: 322,113
Non-trainable params: 0
_________________________________________________________________
```

```
 #specify tthe metrics
model.compile(optimizer='rmsprop',
              loss='binary_crossentropy',
              metrics=['accuracy'])
# fit the model
fitted = model.fit(x_train,
                    y_train,
                    epochs=10,
                    batch_size=128,
                    validation_split=0.2)
```

```
Epoch 1/10
28/28 [==============================] - 9s 230ms/step - loss: 0.4358 - accuracy: 0.8521 - val_loss: 0.3791 - val_accuracy: (
Epoch 2/10
28/28 [==============================] - 5s 172ms/step - loss: 0.2604 - accuracy: 0.9132 - val_loss: 0.1412 - val_accuracy: (
Epoch 3/10
28/28 [==============================] - 5s 192ms/step - loss: 0.1176 - accuracy: 0.9714 - val_loss: 0.0949 - val_accuracy: (
Epoch 4/10
28/28 [==============================] - 5s 192ms/step - loss: 0.0789 - accuracy: 0.9813 - val_loss: 0.0722 - val_accuracy: (
Epoch 5/10
28/28 [==============================] - 5s 174ms/step - loss: 0.0560 - accuracy: 0.9861 - val_loss: 0.0577 - val_accuracy: (
Epoch 6/10
28/28 [==============================] - 7s 240ms/step - loss: 0.0407 - accuracy: 0.9912 - val_loss: 0.0734 - val_accuracy: (
Epoch 7/10
28/28 [==============================] - 5s 173ms/step - loss: 0.0361 - accuracy: 0.9912 - val_loss: 0.0447 - val_accuracy: (
Epoch 8/10
28/28 [==============================] - 6s 217ms/step - loss: 0.0303 - accuracy: 0.9926 - val_loss: 0.0494 - val_accuracy: (
Epoch 9/10
28/28 [==============================] - 5s 171ms/step - loss: 0.0173 - accuracy: 0.9960 - val_loss: 0.0527 - val_accuracy: (
Epoch 10/10
28/28 [==============================] - 5s 172ms/step - loss: 0.0146 - accuracy: 0.9969 - val_loss: 0.0482 - val_accuracy: (
```

```
#results of RNN
from sklearn.metrics import classification_report

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

```
37/37 [==============================] - 1s 31ms/step
              precision    recall  f1-score   support

           0       0.99      1.00      0.99      1004
           1       0.97      0.94      0.96       149

    accuracy                           0.99      1153
   macro avg       0.98      0.97      0.97      1153
```

```
weighted avg       0.99      0.99      0.99      1153
```

Analyzing the RNN model:

in the RNN model, the data has to be split to train and test data and the split protioan can be any number between 0 and 1. In this assignment, the number picked to divide the data had to be anything less than 0.8. Then to get the x trained and test data, the Tokenize functiona was used to achieve good results for x_train and x_test data. In this assginment, texts_to_sequences was used to achieve better results for the RNN trained and tested data

Then to get the results of y trained and test data, the Encoder function was used to get the best results. After obtaingin the trtained and test data for x and y, the RNN model was built using the x and y data. The RNN model used many proporties of the seqntial model such as SimpleRNN and layers to build the model and get results.

Evaluation:

The accuracy and loss results of the model were very good, as seen above. However, the results were not as accurate as the sequential model. Hence the accuracy result was very high and approximately equals to 0.9969. Also, the prediction models showed that the model will be accurate for future data as well. Moreover, the accuracy measurements and metrics also conformed the accuracy of the data as the accuracy results were equal to 0.99

```
#use LSTM to improve the RNN result
model = models.Sequential()
model.add(layers.Embedding(max_features, 32))
model.add(layers.LSTM(32))
model.add(layers.Dense(1, activation='sigmoid'))

model.summary()

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


# fit the model

fitted = model.fit(x_train,
                   y_train,
                   epochs=10,
                   batch_size=128,
                   validation_split=0.2)
```

```
Model: "sequential_8"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 embedding_8 (Embedding)     (None, None, 32)          320000    
                                                                 
 lstm_3 (LSTM)               (None, 32)                8320      
                                                                 
 dense_8 (Dense)             (None, 1)                 33        
                                                                 
=================================================================
Total params: 328,353
Trainable params: 328,353
Non-trainable params: 0
_________________________________________________________________
Epoch 1/10
28/28 [==============================] - 15s 419ms/step - loss: 0.4631 - accuracy: 0.8543 - val_loss: 0.3108 - val_accuracy:
Epoch 2/10
28/28 [==============================] - 11s 398ms/step - loss: 0.2424 - accuracy: 0.9016 - val_loss: 0.1784 - val_accuracy:
Epoch 3/10
28/28 [==============================] - 11s 404ms/step - loss: 0.1454 - accuracy: 0.9655 - val_loss: 0.1157 - val_accuracy:
Epoch 4/10
28/28 [==============================] - 11s 404ms/step - loss: 0.0944 - accuracy: 0.9799 - val_loss: 0.0820 - val_accuracy:
Epoch 5/10
28/28 [==============================] - 12s 431ms/step - loss: 0.0643 - accuracy: 0.9859 - val_loss: 0.0635 - val_accuracy:
Epoch 6/10
28/28 [==============================] - 12s 428ms/step - loss: 0.0483 - accuracy: 0.9898 - val_loss: 0.0603 - val_accuracy:
Epoch 7/10
28/28 [==============================] - 12s 428ms/step - loss: 0.0371 - accuracy: 0.9915 - val_loss: 0.0573 - val_accuracy:
Epoch 8/10
28/28 [==============================] - 12s 420ms/step - loss: 0.0292 - accuracy: 0.9949 - val_loss: 0.0529 - val_accuracy:
Epoch 9/10
28/28 [==============================] - 11s 386ms/step - loss: 0.0251 - accuracy: 0.9946 - val_loss: 0.0520 - val_accuracy:
```

```
Epoch 10/10
28/28 [==============================] - 11s 396ms/step - loss: 0.0207 - accuracy: 0.9960 - val_loss: 0.0508 - val_accuracy:
```

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

```
37/37 [==============================] - 2s 54ms/step
              precision    recall  f1-score   support

           0       0.99      0.99      0.99      1004
           1       0.95      0.95      0.95       149

    accuracy                           0.99      1153
   macro avg       0.97      0.97      0.97      1153
weighted avg       0.99      0.99      0.99      1153
```

LSTM: After using the LSTM model, the results imporved slightly than the simple RNN layer. As the accuracy, f1 score, recall, and precision scores were higher than the previous model.

```
#try GRU which should improve the LSTM results
model = models.Sequential()
model.add(layers.Embedding(max_features, 32))
model.add(layers.GRU(32))
model.add(layers.Dense(1, activation='sigmoid'))


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

# run and fit the model

fitted = model.fit(x_train,
                    y_train,
                    epochs=10,
                    batch_size=128,
                    validation_split=0.2)
```

```
Epoch 1/10
28/28 [==============================] - 15s 416ms/step - loss: 0.5265 - accuracy: 0.8523 - val_loss: 0.3478 - val_accuracy:
Epoch 2/10
28/28 [==============================] - 11s 400ms/step - loss: 0.2814 - accuracy: 0.8676 - val_loss: 0.2054 - val_accuracy:
Epoch 3/10
28/28 [==============================] - 11s 400ms/step - loss: 0.1444 - accuracy: 0.9502 - val_loss: 0.1008 - val_accuracy:
Epoch 4/10
28/28 [==============================] - 10s 353ms/step - loss: 0.0784 - accuracy: 0.9833 - val_loss: 0.0670 - val_accuracy:
Epoch 5/10
28/28 [==============================] - 11s 390ms/step - loss: 0.0512 - accuracy: 0.9878 - val_loss: 0.0571 - val_accuracy:
Epoch 6/10
28/28 [==============================] - 11s 397ms/step - loss: 0.0374 - accuracy: 0.9912 - val_loss: 0.0476 - val_accuracy:
Epoch 7/10
28/28 [==============================] - 11s 399ms/step - loss: 0.0281 - accuracy: 0.9932 - val_loss: 0.0453 - val_accuracy:
Epoch 8/10
28/28 [==============================] - 11s 397ms/step - loss: 0.0225 - accuracy: 0.9941 - val_loss: 0.0448 - val_accuracy:
Epoch 9/10
28/28 [==============================] - 11s 399ms/step - loss: 0.0174 - accuracy: 0.9966 - val_loss: 0.0450 - val_accuracy:
Epoch 10/10
28/28 [==============================] - 11s 376ms/step - loss: 0.0134 - accuracy: 0.9966 - val_loss: 0.0438 - val_accuracy:
```

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

```
37/37 [==============================] - 2s 47ms/step
              precision    recall  f1-score   support

           0       0.99      1.00      0.99      1004
           1       0.98      0.95      0.96       149

    accuracy                           0.99      1153
   macro avg       0.99      0.97      0.98      1153
```

```
    weighted avg       0.99      0.99      0.99      1153
```

GRU:

We can see from the results above that the GRU model had better results and accuracy scores than simple RNN and LSTM. Hence, the GRU model preformed better than the other RNN models and can be compared with the sequential model as it also has good results.

```
#CNN
import tensorflow as tf
from tensorflow.keras import datasets, layers, models, preprocessing
import io
import pandas as pd
import numpy as np
import tensorflow as tf
from tensorflow.keras import datasets, layers, models, preprocessing
import numpy as np
import pandas as pd
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
max_features = 10000
maxlen = 500
batch_size = 32

#read file
df = pd.read_csv(io.BytesIO(uploaded['spam_tclassification.csv']))
print('rows and columns:', df.shape)
print(df.head())
#load and process data:
# get the train and test data by splitting
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)


num_labels = 2
vocab_size = 25000
batch_size = 100

# fit the tokenizer on the training data
tokenizer = Tokenizer(num_words=vocab_size)
tokenizer.fit_on_texts(train.Message)
# tokenize and fit the data using texts_to_sequences
x_train = tokenizer.texts_to_sequences(train.Message)
x_test = tokenizer.texts_to_sequences(test.Message)

#use encoder method for the y data
encoder = LabelEncoder()
encoder.fit(train.Category)
y_train = encoder.transform(train.Category)
y_test = encoder.transform(test.Category)

# check data
print("test first five labels:", y_test[:5])

# pad the x traind and test data
x_train = preprocessing.sequence.pad_sequences(x_train, maxlen=maxlen)
x_test = preprocessing.sequence.pad_sequences(x_test, maxlen=maxlen)


# build CNN model using the sequential model and some of its functionalities

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

```
rows and columns: (5572, 2)
  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...
train data size:  (4470, 2)
test data size:  (1102, 2)
test first five labels: [0 1 0 0 0]
Model: "sequential_11"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_11 (Embedding)    (None, 500, 128)          1280000

 conv1d_2 (Conv1D)           (None, 494, 32)           28704

 max_pooling1d_1 (MaxPooling  (None, 98, 32)           0
 1D)

 conv1d_3 (Conv1D)           (None, 92, 32)           7200

 global_max_pooling1d_1 (Glo  (None, 32)               0
 balMaxPooling1D)

 dense_11 (Dense)            (None, 1)                33

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

```
 #specify tthe metrics

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

# run and fit the model

fitted = model.fit(x_train,
                    y_train,
                    epochs=10,
                    batch_size=128,
                    validation_split=0.2)
```

```
Epoch 1/10
28/28 [==============================] - 16s 556ms/step - loss: 0.1719 - accuracy: 0.9354 - val_loss: 0.1667 - val_accuracy:
Epoch 2/10
28/28 [==============================] - 16s 582ms/step - loss: 0.1452 - accuracy: 0.9536 - val_loss: 0.1680 - val_accuracy:
Epoch 3/10
28/28 [==============================] - 17s 600ms/step - loss: 0.1227 - accuracy: 0.9642 - val_loss: 0.1478 - val_accuracy:
Epoch 4/10
28/28 [==============================] - 15s 551ms/step - loss: 0.1025 - accuracy: 0.9692 - val_loss: 0.1456 - val_accuracy:
Epoch 5/10
28/28 [==============================] - 15s 545ms/step - loss: 0.0841 - accuracy: 0.9734 - val_loss: 0.1297 - val_accuracy:
Epoch 6/10
28/28 [==============================] - 15s 549ms/step - loss: 0.0721 - accuracy: 0.9768 - val_loss: 0.1215 - val_accuracy:
Epoch 7/10
28/28 [==============================] - 16s 581ms/step - loss: 0.0622 - accuracy: 0.9779 - val_loss: 0.1140 - val_accuracy:
Epoch 8/10
28/28 [==============================] - 16s 547ms/step - loss: 0.0607 - accuracy: 0.9796 - val_loss: 0.1117 - val_accuracy:
Epoch 9/10
28/28 [==============================] - 16s 571ms/step - loss: 0.0491 - accuracy: 0.9829 - val_loss: 0.1054 - val_accuracy:
Epoch 10/10
28/28 [==============================] - 15s 549ms/step - loss: 0.0497 - accuracy: 0.9843 - val_loss: 0.1034 - val_accuracy:
```

```
#CNN results
from sklearn.metrics import classification_report

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

```
35/35 [==============================] - 2s 44ms/step
              precision    recall  f1-score   support

           0       0.97      1.00      0.98       952
           1       0.99      0.78      0.87       150

    accuracy                           0.97      1102
   macro avg       0.98      0.89      0.93      1102
weighted avg       0.97      0.97      0.97      1102
```

CNN:

We can see that the results of the CNN data was good, however, its performance was lower than RNNs and the sequential model. This model was built on the sequential model and used some of its proporties to build the cnn model such as the max pooling.

Moreover, the results of future predictions were good and the some recall scores had the value of 1 as can be seen below

```
          precision    recall  f1-score   support

       0       0.97      1.00      0.98       952
       1       0.99      0.78      0.87       150

accuracy                           0.97      1102
macro avg       0.98      0.89      0.93      1102
weighted avg       0.97      0.97      0.97      1102
```

```
#embedding layer
import numpy as np
import tensorflow as tf
from tensorflow import keras

import tensorflow as tf
from tensorflow.keras import datasets, layers, models, preprocessing
import io
import pandas as pd
import numpy as np
import tensorflow as tf
from tensorflow.keras import datasets, layers, models, preprocessing
import numpy as np
import pandas as pd
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
#read file
#load and process data:
df = pd.read_csv(io.BytesIO(uploaded['spam_tclassification.csv']))
print('rows and columns:', df.shape)
print(df.head())
#load and process data:
# get the train and test data by splitting
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)

num_labels = 2
vocab_size = 25000
batch_size = 100

# fit the tokenizer on the training data
tokenizer = Tokenizer(num_words=vocab_size)
tokenizer.fit_on_texts(train.Message)
# tokenize and fit the data using texts_to_sequences
x_train = tokenizer.texts_to_sequences(train.Message)
x_test = tokenizer.texts_to_sequences(test.Message)
```

```
#use encoder to get the y data
encoder = LabelEncoder()
encoder.fit(train.Category)
y_train = encoder.transform(train.Category)
y_test = encoder.transform(test.Category)

# use padding on the x data
x_train = preprocessing.sequence.pad_sequences(x_train, maxlen=maxlen)
x_test = preprocessing.sequence.pad_sequences(x_test, maxlen=maxlen)


#get index for each word
ind =tokenizer.word_index
#check length
print(len(ind))



#EL:
from tensorflow.keras import layers

EMBEDDING_DIM = 128
MAX_SEQUENCE_LENGTH = 200

embedding_layer = layers.Embedding(len(ind) + 1,
                            EMBEDDING_DIM,
                            input_length=MAX_SEQUENCE_LENGTH)

#building the EL model :
int_sequences_input = keras.Input(shape=(None,), dtype="int64")
embedded_sequences = embedding_layer(int_sequences_input)
x = layers.Conv1D(128, 5, activation="relu")(embedded_sequences)
x = layers.MaxPooling1D(5)(x)
x = layers.Conv1D(128, 5, activation="relu")(x)
x = layers.MaxPooling1D(5)(x)
x = layers.Conv1D(128, 5, activation="relu")(x)
x = layers.GlobalMaxPooling1D()(x)
x = layers.Dense(128, activation="relu")(x)
x = layers.Dropout(0.5)(x)
preds = layers.Dense(num_labels, activation="sigmoid")(x)
model = keras.Model(int_sequences_input, preds)
model.summary()
```

```
rows and columns: (5572, 2)
  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...
train data size:  (4464, 2)
test data size:  (1108, 2)
train shapes: (4464, 20) (4464,)
test shapes: (1108, 20) (1108,)
test first five labels: [1 1 0 1 0]
7874
Model: "model_18"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 input_20 (InputLayer)       [(None, None)]            0         
                                                                 
 embedding_21 (Embedding)    (None, None, 128)         1008000   
                                                                 
 conv1d_57 (Conv1D)          (None, None, 128)         82048     
                                                                 
 max_pooling1d_38 (MaxPoolin  (None, None, 128)        0         
 g1D)                                                            
                                                                 
 conv1d_58 (Conv1D)          (None, None, 128)         82048     
                                                                 
 max_pooling1d_39 (MaxPoolin  (None, None, 128)        0         
 g1D)                                                            
                                                                 
 conv1d_59 (Conv1D)          (None, None, 128)         82048     
                                                                 
 global_max_pooling1d_19 (Gl  (None, 128)              0         
 obalMaxPooling1D)                                               
                                                                 
 dense_42 (Dense)            (None, 128)               16512     
 dropout_19 (Dropout)        (None, 128)               0

 dense_43 (Dense)            (None, 2)                 258

=================================================================
Total params: 1,270,914
Trainable params: 1,270,914
Non-trainable params: 0
_________________________________________________________________
```

```
#METRICS
model.compile(
    optimizer="rmsprop",
    loss='binary_crossentropy',
              metrics=['accuracy']
)
#fitted model
model.fit(x_train, y_train, batch_size=128, epochs=20, validation_split=0.2)
```

Embedding

I was not able to get results from the embedding layers as I had some trouble cleaning and processing data.

However, embedding is supposed work better than the eprvious models as it uses many machine learning and deep learnign algorithms which should provide good results.

✓ 0s completed at 5:10 PM ● ✕