# Text Classification 2

By Ahmed Iqbal (axi180011)

Description: The dataset I selected is on SMS text messages and whether they are spam or not. The model should be able to predict whether a message is spam or not.

# Sequential

```
import tensorflow as tf
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
import nltk
nltk.download('stopwords')
from tensorflow.keras import datasets, layers, models
from nltk.corpus import stopwords
import numpy as np
import scipy
from keras.preprocessing.text import Tokenizer
```

```
stopwords = set(stopwords.words('english'))
vectorizer = TfidfVectorizer(stop_words=list(stopwords))
```

```
df = pd.read_csv("https://raw.githubusercontent.com/ahmxdiqbal/CS4395/main/SMS_train.csv", he

# set up X and y
X = df.iloc[:,0]
y = df.iloc[:,1]

for i in range(len(y)):
  if y[i] == "Non-Spam":
    y[i] = 0
  else:
    y[i] = 1
```

```
def vectorize_sequences(sequences, dimension=10000):
    tokenizer = Tokenizer(num_words=dimension)
    tokenizer.fit_on_texts(sequences)
    sequences = tokenizer.texts_to_sequences(sequences)
    results = np.zeros((len(sequences), dimension))
    for i, sequence in enumerate(sequences):
```

```
        results[i, sequence] = 1.
    return results

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, train_size=0.8, rand



# Our vectorized labels
y_train = np.asarray(y_train).astype('float32')
y_test = np.asarray(y_test).astype('float32')

X_train.shape

# apply tfidf vectorizer
x_train = vectorize_sequences(X_train)  # fit and transform the train data
x_test = vectorize_sequences(X_test)        # transform only the test data

# build the model

model = models.Sequential()
model.add(layers.Dense(16, activation='relu', input_shape=(10000,)))
model.add(layers.Dense(16, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))

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

# create a validation set
x_val = x_train[:100]
partial_x_train = x_train[100:]
#
y_val = y_train[:100]
partial_y_train = y_train[100:]
# train

history = model.fit(x_train, y_train, epochs=20, batch_size=512, validation_data=(x_val, y_va

# use sklearn evaluation

from sklearn.metrics import classification_report

print("\n\n")

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))
```

```
     [nltk_data] Downloading package stopwords to /root/nltk_data...
     [nltk_data]   Package stopwords is already up-to-date!
     Epoch 1/20
     2/2 [==============================] - 1s 242ms/step - loss: 0.6985 - accuracy: 0.3508 -
     Epoch 2/20
```

```
2/2 [==============================] - 0s 65ms/step - loss: 0.6784 - accuracy: 0.9359 -
Epoch 3/20
2/2 [==============================] - 0s 65ms/step - loss: 0.6630 - accuracy: 0.9686 -
Epoch 4/20
2/2 [==============================] - 0s 49ms/step - loss: 0.6450 - accuracy: 0.9777 -
Epoch 5/20
2/2 [==============================] - 0s 48ms/step - loss: 0.6247 - accuracy: 0.9843 -
Epoch 6/20
2/2 [==============================] - 0s 65ms/step - loss: 0.6026 - accuracy: 0.9856 -
Epoch 7/20
2/2 [==============================] - 0s 65ms/step - loss: 0.5800 - accuracy: 0.9869 -
Epoch 8/20
2/2 [==============================] - 0s 52ms/step - loss: 0.5578 - accuracy: 0.9908 -
Epoch 9/20
2/2 [==============================] - 0s 48ms/step - loss: 0.5360 - accuracy: 0.9908 -
Epoch 10/20
2/2 [==============================] - 0s 67ms/step - loss: 0.5148 - accuracy: 0.9935 -
Epoch 11/20
2/2 [==============================] - 0s 53ms/step - loss: 0.4944 - accuracy: 0.9935 -
Epoch 12/20
2/2 [==============================] - 0s 64ms/step - loss: 0.4746 - accuracy: 0.9948 -
Epoch 13/20
2/2 [==============================] - 0s 65ms/step - loss: 0.4555 - accuracy: 0.9948 -
Epoch 14/20
2/2 [==============================] - 0s 67ms/step - loss: 0.4371 - accuracy: 0.9948 -
Epoch 15/20
2/2 [==============================] - 0s 65ms/step - loss: 0.4192 - accuracy: 0.9948 -
Epoch 16/20
2/2 [==============================] - 0s 62ms/step - loss: 0.4019 - accuracy: 0.9961 -
Epoch 17/20
2/2 [==============================] - 0s 61ms/step - loss: 0.3852 - accuracy: 0.9961 -
Epoch 18/20
2/2 [==============================] - 0s 53ms/step - loss: 0.3691 - accuracy: 0.9961 -
Epoch 19/20
2/2 [==============================] - 0s 55ms/step - loss: 0.3536 - accuracy: 0.9961 -
Epoch 20/20
2/2 [==============================] - 0s 74ms/step - loss: 0.3386 - accuracy: 0.9974 -



6/6 [==============================] - 0s 3ms/step
              precision    recall  f1-score   support

         0.0       0.87      0.99      0.93       168
         1.0       0.00      0.00      0.00        24

    accuracy                           0.86       192
   macro avg       0.44      0.49      0.46       192
weighted avg       0.76      0.86      0.81       192
```

## RNN

```
import tensorflow as tf
from tensorflow.keras import datasets, layers, models, preprocessing
max_features = 10000
maxlen = 500
batch_size = 32
```

```
#train_data = preprocessing.sequence.pad_sequences(X_train, maxlen=maxlen)
#test_data = preprocessing.sequence.pad_sequences(X_test, maxlen=maxlen)

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

model.summary()

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

#y = np.asarray(y_train).astype('float32')
```

```
history = model.fit(x_train,
                    y_train,
                    epochs=1,
                    batch_size=128,
                    validation_split=0.2)

print("\n\n")

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))
```

```
el: "sequential_56"
_________________________________________________________________
yer (type)                 Output Shape              Param #
=================================================================
bedding_45 (Embedding)     (None, None, 32)          320000
```

```
mple_rnn_20 (SimpleRNN)   (None, 32)                2080

nse_152 (Dense)           (None, 1)                 33

=================================================================
al params: 322,113
inable params: 322,113
-trainable params: 0
_________________________________________________________________
 [==============================] - 3s 155ms/step - loss: 0.6717 - accuracy: 0.6738 - va
 [==============================] - 3s 477ms/step
              precision    recall  f1-score   support

         0.0       0.88      1.00      0.93       168
         1.0       0.00      0.00      0.00        24

    accuracy                           0.88       192
   macro avg       0.44      0.50      0.47       192
ghted avg       0.77      0.88      0.82       192

r/local/lib/python3.9/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMe
warn_prf(average, modifier, msg_start, len(result))
r/local/lib/python3.9/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMe
warn_prf(average, modifier, msg_start, len(result))
r/local/lib/python3.9/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMe
warn_prf(average, modifier, msg_start, len(result))
```

## Embedding

```
import numpy as np
import tensorflow as tf
from tensorflow import keras

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(X_train).batch(128)
vectorizer.adapt(text_ds)

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

from tensorflow.keras import layers

EMBEDDING_DIM = 128
MAX_SEQUENCE_LENGTH = 200

embedding_layer = layers.Embedding(len(word_index) + 1,
                            EMBEDDING_DIM,
```

```
                            input_length=MAX_SEQUENCE_LENGTH)

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(len(class_names), activation="softmax")(x)
model = keras.Model(int_sequences_input, x)
model.build((None, None))
model.summary()

x_train = vectorizer(np.array([[s] for s in X_train])).numpy()
x_val = vectorizer(np.array([[s] for s in X_train[:100]])).numpy()
x_train = np.array(x_train[:763])
y_train = np.array(y_train)
y_val = np.array(y_train[:100])

print(x_train.shape)
print(y_train.shape)
print(x_val.shape)
print(y_val.shape)



model.compile(
    loss="sparse_categorical_crossentropy", optimizer="rmsprop", metrics=["acc"]
)
model.fit(x_train, y_train, batch_size=128, epochs=1, validation_data=(x_val, y_val))

test_x = vectorizer(np.array([[s] for s in X_test])).numpy()

preds = model.predict(test_x)
pred_labels = [np.argmax(p) for p in preds]

from sklearn.metrics import classification_report

print("\n\n")

print(classification_report(y_test, pred_labels))
```

```
     Model: "model_21"
     _________________________________________________________________
      Layer (type)                Output Shape              Param #
     =================================================================
      input_24 (InputLayer)       [(None, None)]            0
```

```
 embedding_44 (Embedding)    (None, None, 128)         368256

 conv1d_69 (Conv1D)          (None, None, 128)         82048

 max_pooling1d_46 (MaxPoolin  (None, None, 128)        0
 g1D)

 conv1d_70 (Conv1D)          (None, None, 128)         82048

 max_pooling1d_47 (MaxPoolin  (None, None, 128)        0
 g1D)

 conv1d_71 (Conv1D)          (None, None, 128)         82048

 global_max_pooling1d_23 (Gl  (None, 128)              0
 obalMaxPooling1D)

 dense_151 (Dense)           (None, 128)               16512

 dropout_23 (Dropout)        (None, 128)               0

=================================================================
Total params: 630,912
Trainable params: 630,912
Non-trainable params: 0
_________________________________________________________________
(763, 200)
(763,)
(100, 200)
(100,)
6/6 [==============================] - 5s 450ms/step - loss: 8.1955 - acc: 0.3906 - val_
6/6 [==============================] - 0s 28ms/step
              precision    recall  f1-score   support

         0.0       0.88      1.00      0.93       168
         1.0       0.00      0.00      0.00        24

    accuracy                           0.88       192
   macro avg       0.44      0.50      0.47       192
weighted avg       0.77      0.88      0.82       192

/usr/local/lib/python3.9/dist-packages/sklearn/metrics/_classification.py:1344: Undefine
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.9/dist-packages/sklearn/metrics/_classification.py:1344: Undefine
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.9/dist-packages/sklearn/metrics/_classification.py:1344: Undefine
  _warn_prf(average, modifier, msg_start, len(result))
```

## Analysis

The three approaches are all really similar when it come to their performance. From the f1 score, we can see that the embedding approach and the RNN approach both had an f1 score of 0.82, and the

sequential approach was not far behind with ascore of 0.81. All in all, these approaches did not perform terribly poor or terribly well. There is definitely a lot of room for improvement, but I will say that this is a solid first attempt.

We see a similar pattern playing out with the precision score, which is 0.77 for the RNN and embedding approaches and 0.76 for the sequential approach. This mirroring of the relative performances of each approach could speak to methodelogical errors in carrying out each approach.

Again, the same pattern repats with the recall score. The RNN and embedding approaches are tied for first place with 0.88, and sequential is just barely behind them with a score of 0.86.

All in all, perhaps the dataset was not the best. That could be one issue. Another issue could be my overall inexperience with python and all of these datat libraries means that I could have made some mistakes in implementing these approaches.