CRUZ, JUAN CARLOS M.

FINAL CAPSTONE PROJECT

IE 198

Raw data: SPAM Dataset

• 5,572 messages
- 4,825 <u>legitimate</u> messages (86.6%)
- 747 <u>spam</u> messages (13.41%)

Task: Create a predictive model to predict Spam and Legit messages.

STEPS TO CREATING THIS PREDICTIVE MODEL

1. **Pre-processing**
   a. Data was imported into Spyder

```python
# import, export, plotting libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

sms_raw_df = pd.read_csv("/Users/jccruz/Desktop/School Documents/UP Diliman/AY 2021-2022/IE 198/Capstone Case/Spam.csv",
                     encoding= 'unicode_escape')

spam_msg = sms_raw_df[sms_raw_df.Class == 'spam']
legit_msg = sms_raw_df[sms_raw_df.Class == 'legit']
#%%
```

   b. Since the data contains an imbalanced proportion of legitimate and spam messages, <u>the first step is to **downsample the data**</u> so that the number of legit messages and spam messages to be fed into the model are equal.

```python
legit_msg = sms_raw_df[sms_raw_df.Class == 'legit']
#%%
# Downsampling legit messages
legit_msg_df = legit_msg.sample(n = len(spam_msg), random_state = 30)
spam_msg_df = spam_msg
sms_df = spam_msg.append(legit_msg_df).reset_index(drop=True)
#%%
```

   c. After this data was downsampled, the predictor variable "Class" was <u>**mapped into another column**</u> to <u>express its numeric equivalent</u> (for use when running, evaluating and assessing the model's accuracy) [0 = Legit, 1 = Spam]

```python
#%%
# Mapping into array
sms_df['msg_type'] = sms_df['Class'].map({'legit': 0, 'spam': 1})
sms_label = sms_df['msg_type'].values
sms_label = np.asarray(sms_label).astype(np.int32)
```

d. The predictor and class variables were split into test and training data. (25% test / 75% training)

```
#%%
# library for train test split
from sklearn.model_selection import train_test_split

# Split into Train and Test Data
train_sms, test_sms, train_labels, test_labels = train_test_split(sms_df['Text'], sms_label, test_size = 0.25, random_state = 100)
#%%
```

e. The "Text" predictor data was fed through a "Tokenizer" module which assigned a corresponding numerical value to every unique word found in the dataset. After both the testing and training data were tokenized, each instance of data was fed through the "texts_to_sequences" and "pad_sequences" function to represent the "Text" data as an array of numbers, with each subarray representing a sentence in the preprocessed dataset (each subarray is formed by padding each formed sequence according on the maximum length parameter [see: max_len variable which is 60 numbers (words)]).

The Tokenizer in question employed a vocabulary size of 1000, meaning that 1000 unique words were identified from the train and test data and used as the numerical reference when retrieving the numerical array equivalent of said data.

```
#%%
# deep learning libraries for text pre-processing
import tensorflow as tf
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences

# Hyperparameters for Tokenization
max_len = 60 # Maximum length of each sentence when tokenized and fed to the model
trunc_type = "post"
padding_type = "post"
oov_tok = "<X>"
vocab_size = 1000

# Turn into array
#train_labels = np.asarray(train_labels).astype(np.float32)
#test_labels = np.asarray(test_labels).astype(np.float32)

# Tokenizer Parameters
tokenizer = Tokenizer(num_words = vocab_size, char_level = False, oov_token = oov_tok)
tokenizer.fit_on_texts(train_sms)
tokenizer.fit_on_texts(test_sms)

# Sequencing representation
train_sequences = tokenizer.texts_to_sequences(train_sms)
train_padded = pad_sequences(train_sequences, maxlen = max_len, padding = padding_type,
                             truncating = trunc_type)

test_sequences = tokenizer.texts_to_sequences(test_sms)
test_padded = pad_sequences(test_sequences, maxlen = max_len, padding = padding_type,
                            truncating = trunc_type)
#%%
```

f. The data is now fully preprocessed. It is ready to be fed into the Model.

2. **Running the Model**

```
# Modeling with Neural Networks
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, Dense, Dropout, Conv1D, GlobalAveragePooling1D

vocab_size = 1000
embeding_dim = 32

Model = Sequential()
Model.add(Embedding(vocab_size, embeding_dim, input_length=60))
Model.add(GlobalAveragePooling1D())
Model.add(Dense(24, activation='relu'))
Model.add(Dense(48, activation='relu'))
Model.add(Dropout(0.20)) # Prevent overfitting
Model.add(Dense(1, activation='sigmoid')) # Predict average class accuracy of spam and legit

Model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy', 'mean_squared_error'])

early_stop = EarlyStopping(monitor='val_loss',patience=2) #Overfitting account 2
Model.fit(train_padded, train_labels, epochs=15, validation_data=(test_padded, test_labels),
          callbacks=[early_stop], verbose=2)
Model.metrics_names

print(Model.summary())

loss = pd.DataFrame(Model.history.history)
loss[['accuracy','loss']].plot()
loss[['val_accuracy','val_loss']].plot()
loss[['val_mean_squared_error', 'val_loss']].plot()
```

a. Details of the Model
   i. Artificial Neural Network with **five** (5) hidden layers and **one** (1) output layer
      1. An Embedding layer with input_dim = 1000, output_dim = 32, and input_length = 60 (input_length corresponding to the length of each subarray from the preprocessed data)
      2. A Global Average Pooling 1D layer
      3. Two density layers with 24 and 48 nodes respectively, using the ReLu activation function
      4. A Dropout layer which removes 20% of the data to prevent overfitting
      5. An output layer with a sigmoid function
   ii. Compiled with the "binary_crossentropy" loss function, uses the "Adam" optimizer and identifies accuracy and MSE metrics
   iii. EarlyStopping function which checks the val_loss at a patience value of 2
   iv. Model is tested for 15 epochs

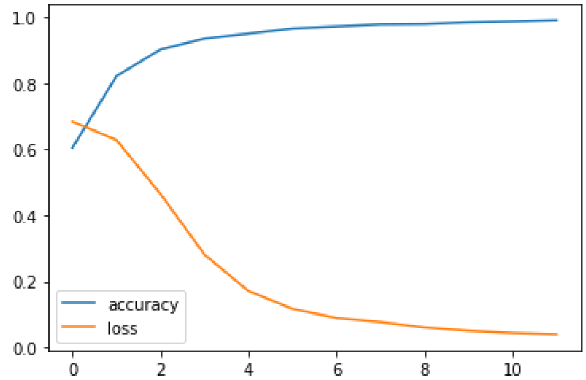
b. <u>Results and Values</u>

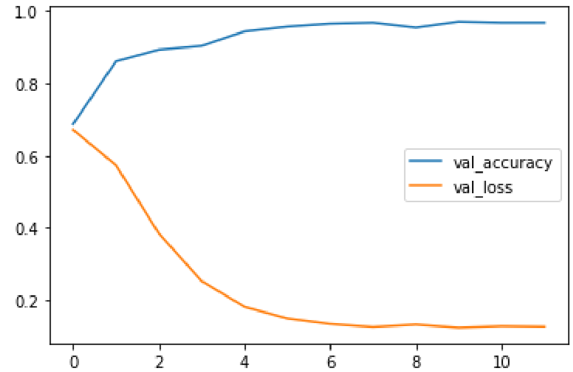
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_8 (Embedding)      (None, 60, 32)            32000
_________________________________________________________________
global_average_pooling1d_8 ( (None, 32)                0
_________________________________________________________________
dense_24 (Dense)             (None, 24)                792
_________________________________________________________________
dense_25 (Dense)             (None, 48)                1200
_________________________________________________________________
dropout_8 (Dropout)          (None, 48)                0
_________________________________________________________________
dense_26 (Dense)             (None, 1)                 49
=================================================================
Total params: 34,041
Trainable params: 34,041
Non-trainable params: 0
_________________________________________________________________
```

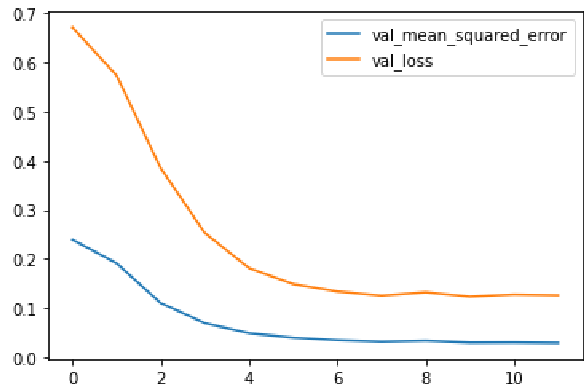
```
Epoch 1/15
35/35 - 1s - loss: 0.6844 - accuracy: 0.6054 - mean_squared_error: 0.2456 -
val_loss: 0.6715 - val_accuracy: 0.6872 - val_mean_squared_error: 0.2392
Epoch 2/15
35/35 - 0s - loss: 0.6284 - accuracy: 0.8223 - mean_squared_error: 0.2179 -
val_loss: 0.5735 - val_accuracy: 0.8610 - val_mean_squared_error: 0.1913
Epoch 3/15
35/35 - 0s - loss: 0.4650 - accuracy: 0.9027 - mean_squared_error: 0.1427 -
val_loss: 0.3847 - val_accuracy: 0.8930 - val_mean_squared_error: 0.1103
Epoch 4/15
35/35 - 0s - loss: 0.2813 - accuracy: 0.9357 - mean_squared_error: 0.0728 -
val_loss: 0.2528 - val_accuracy: 0.9037 - val_mean_squared_error: 0.0694
Epoch 5/15
35/35 - 0s - loss: 0.1709 - accuracy: 0.9509 - mean_squared_error: 0.0416 -
val_loss: 0.1815 - val_accuracy: 0.9439 - val_mean_squared_error: 0.0489
Epoch 6/15
35/35 - 0s - loss: 0.1170 - accuracy: 0.9661 - mean_squared_error: 0.0275 -
val_loss: 0.1491 - val_accuracy: 0.9572 - val_mean_squared_error: 0.0399
Epoch 7/15
35/35 - 0s - loss: 0.0891 - accuracy: 0.9723 - mean_squared_error: 0.0218 -
val_loss: 0.1341 - val_accuracy: 0.9652 - val_mean_squared_error: 0.0348
Epoch 8/15
35/35 - 0s - loss: 0.0769 - accuracy: 0.9786 - mean_squared_error: 0.0184 -
val_loss: 0.1257 - val_accuracy: 0.9679 - val_mean_squared_error: 0.0318
Epoch 9/15
35/35 - 0s - loss: 0.0606 - accuracy: 0.9795 - mean_squared_error: 0.0146 -
val_loss: 0.1324 - val_accuracy: 0.9545 - val_mean_squared_error: 0.0337
Epoch 10/15
35/35 - 0s - loss: 0.0507 - accuracy: 0.9848 - mean_squared_error: 0.0118 -
val_loss: 0.1236 - val_accuracy: 0.9706 - val_mean_squared_error: 0.0300
Epoch 11/15
35/35 - 0s - loss: 0.0437 - accuracy: 0.9875 - mean_squared_error: 0.0101 -
val_loss: 0.1279 - val_accuracy: 0.9679 - val_mean_squared_error: 0.0302
Epoch 12/15
35/35 - 0s - loss: 0.0397 - accuracy: 0.9911 - mean_squared_error: 0.0087 -
val_loss: 0.1264 - val_accuracy: 0.9679 - val_mean_squared_error: 0.0294
Model: "sequential_8"
```

MODEL STOPS AT 12/15 EPOCHS
Accuracy of Model: 99.11%
Value of Accuracy: 96.79%
Loss value: 0.0397
MSE: 0.0087
Value of MSE: 0.0294

GRAPHS







CONFUSION MATRIX, CLASSIFICATION REPORT AND AVERAGE CLASS ACCURACY OF PREDICTIONS

Code

```
#%%
from sklearn.metrics import confusion_matrix, classification_report

text_predictions = Model.predict(test_padded)
text_predictions_rounded = np.round(text_predictions)
cm = confusion_matrix(np.round(test_labels), text_predictions_rounded)

print("Confusion Matrix:", "\n", confusion_matrix(np.round(test_labels),text_predictions_rounded))
print(classification_report(np.round(test_labels), text_predictions_rounded))

print("'Spam' accuracy: {}%".format(round((cm[0][0]/(cm[0][0] + cm[0][1])*100), 2)))
print("'Legit' accuracy: {}%".format(round((((cm[1][1]/(cm[1][0]+cm[1][1])))*100, 2)))
```

RESULTS

```
None
Confusion Matrix:
 [[181   3]
 [  9 181]]
              precision    recall  f1-score   support

           0       0.95      0.98      0.97       184
           1       0.98      0.95      0.97       190

    accuracy                           0.97       374
   macro avg       0.97      0.97      0.97       374
weighted avg       0.97      0.97      0.97       374

'Spam' accuracy: 98.37%
'Legit' accuracy: 95.26%
```

INTERPRETATION

The model has 98.37% accuracy of correctly predicting a message as spam, while it has a 95.26% accuracy of correctly predicting a message as legitimate. The recall values of each legit and spam prediction are 98% and 95%, respectively, while both have an F1-score of 97%.