Text Classification 2

Motalib Rahim (mxr170012)

Dr. Karen Mazidi

```
# import libraries
import pandas as pd
from tensorflow import keras
from sklearn.model_selection import train_test_split
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
import numpy as np
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.linear_model import LogisticRegression
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Embedding, LSTM
from sklearn.metrics import accuracy_score
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from gensim.models import Word2Vec


df = pd.read_csv("spam_ham_dataset.csv")  # load dataset


df # dataframe
```

	Unnamed: 0	label	text	label_num
0	605	ham	Subject: enron methanol ; meter # : 988291\r\n...	0
1	2349	ham	Subject: hpl nom for january 9 , 2001\r\n(see...	0
2	3624	ham	Subject: neon retreat\r\nho ho ho , we ' re ar...	0
3	4685	spam	Subject: photoshop , windows , office . cheap ...	1
4	2030	ham	Subject: re : indian springs\r\nthis deal is t...	0
...	...	...	...	...
5166	1518	ham	Subject: put the 10 on the ft\r\nthe transport...	0
5167	404	ham	Subject: 3 / 4 / 2000 and following noms\r\nhp...	0
5168	2933	ham	Subject: calpine daily gas nomination\r\n>\r\n...	0

2m 34s completed at 7:12 PM

5170	4807	spam	Subject: important online banking alert\r\ndea...	1

5171 rows × 4 columns

```
df.head(10) # show top 10
```

	Unnamed: 0	label	text	label_num
0	605	ham	Subject: enron methanol ; meter # : 988291\r\n...	0
1	2349	ham	Subject: hpl nom for january 9 , 2001\r\n(see...	0
2	3624	ham	Subject: neon retreat\r\nho ho ho , we ' re ar...	0
3	4685	spam	Subject: photoshop , windows , office . cheap ...	1
4	2030	ham	Subject: re : indian springs\r\nthis deal is t...	0
5	2949	ham	Subject: ehronline web address change\r\nthis ...	0
6	2793	ham	Subject: spring savings certificate - take 30 ...	0
7	4185	spam	Subject: looking for medication ? we ` re the ...	1
8	2641	ham	Subject: noms / actual flow for 2 / 26\r\nwe a...	0
9	1870	ham	Subject: nominations for oct . 21 - 23 , 2000\...	0

```
df.info # information df
```

```
<bound method DataFrame.info of       Unnamed: 0 label
text  \
0            605   ham  Subject: enron methanol ; meter # : 988291\r\n...
1           2349   ham  Subject: hpl nom for january 9 , 2001\r\n( see...
2           3624   ham  Subject: neon retreat\r\nho ho ho , we ' re ar...
3           4685  spam  Subject: photoshop , windows , office . cheap ...
4           2030   ham  Subject: re : indian springs\r\nthis deal is t...
...          ...   ...                                                ...
5166        1518   ham  Subject: put the 10 on the ft\r\nthe transport...
5167         404   ham  Subject: 3 / 4 / 2000 and following noms\r\nhp...
5168        2933   ham  Subject: calpine daily gas nomination\r\n>\r\n...
5169        1409   ham  Subject: industrial worksheets for august 2000...
5170        4807  spam  Subject: important online banking alert\r\ndea...

      label_num
0             0
1             0
2             0
3             1
4             0
...         ...
5166          0
```

```
5167          0
5168          0
5169          0
5170          1

[5171 rows x 4 columns]>
```

```
df['label'].value_counts() # counts
```

```
ham     3672
spam    1499
Name: label, dtype: int64
```

```
df['label_num'].value_counts()
```

```
0    3672
1    1499
Name: label_num, dtype: int64
```

```
X = df['text']
y = df['label_num']
```

```
# preprocess
vectorizer = CountVectorizer(min_df=0, lowercase=False)
vectorizer.fit(X)
vectorizer.vocabulary_
```

```
{'Subject': 4705,
 'enron': 18720,
 'methanol': 31037,
 'meter': 31030,
 '988291': 4629,
 'this': 44990,
 'is': 26372,
 'follow': 20851,
 'up': 46832,
 'to': 45371,
 'the': 44868,
 'note': 33216,
 'gave': 21766,
 'you': 49946,
 'on': 33929,
 'monday': 31724,
 '00': 0,
 'preliminary': 36689,
 'flow': 20740,
 'data': 15289,
 'provided': 37232,
 'by': 10767,
 'daren': 15232,
 'please': 36060,
```

```
 'override': 34493,
 'pop': 36320,
 'daily': 15134,
 'volume': 47879,
 'presently': 36754,
 'zero': 50177,
 'reflect': 38657,
 'activity': 5145,
 'can': 11043,
 'obtain': 33557,
 'from': 21288,
 'gas': 21711,
 'control': 13943,
 'change': 11912,
 'needed': 32622,
 'asap': 7248,
 'for': 20912,
 'economics': 17899,
 'purposes': 37476,
 'hpl': 24487,
 'nom': 33091,
 'january': 26666,
 '2001': 1116,
 'see': 40994,
 'attached': 7533,
 'file': 20380,
 'hplnol': 24494,
 '09': 214,
 'xls': 49537,
 'neon': 32689,
 'retreat': 39310,
 'ho': 24144,
 'we': 48346,
 're': 38306,
```

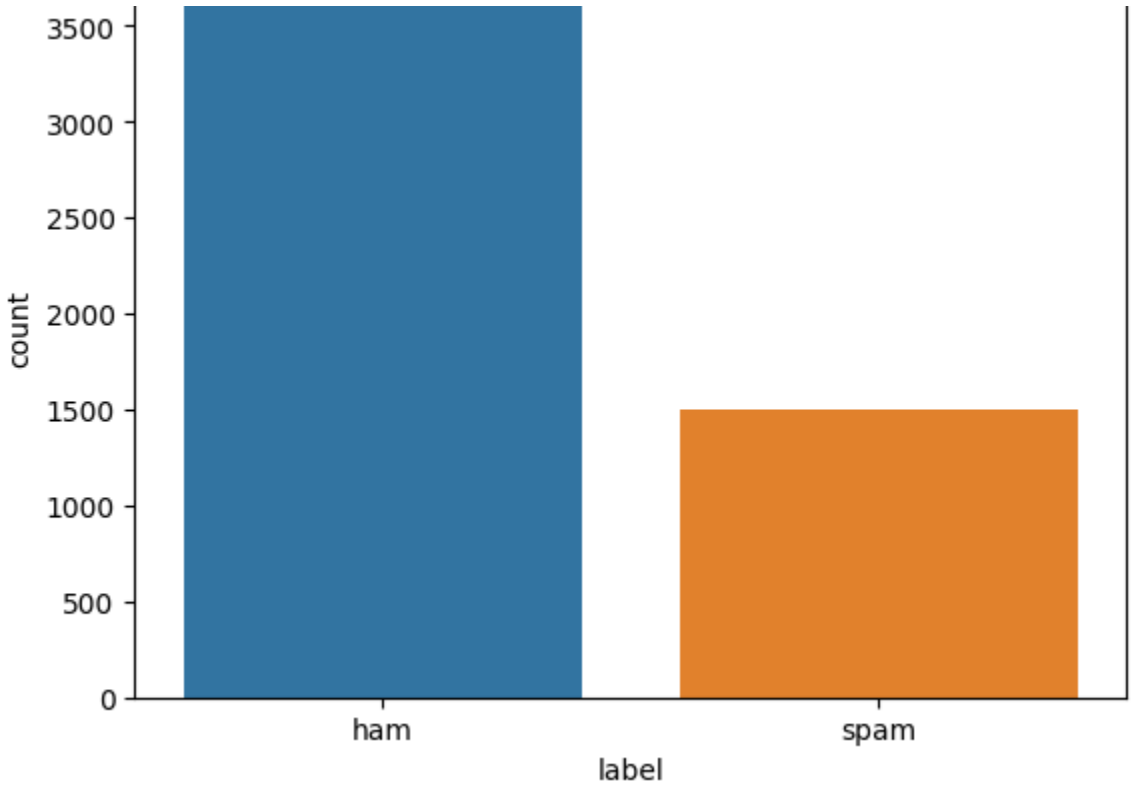
```
vectorizer.transform(X).toarray()
```

```
array([[1, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       ...,
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0]])
```

```
# train, test, split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=42)
```

```
sns.countplot(x="label", data = df) # graph
```

```
<Axes: xlabel='label', ylabel='count'>
```



Description

The target classes are that of what data consists of in the label data. Here label data have emails that are classified as spam or ham. Spam emails are unecessary emails and ham are the necessary ones. The model should be able to predict the target classes for every data. In other words it should bee able to predict whether an email/ message is classified as ham or spam.

```
# sequential model
model = Sequential()

# dense layer
model.add(Dense(64, input_shape=(len(vectorizer.vocabulary_),), activation='relu'))
model.add(Dense(1, activation='sigmoid'))

# compile
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
# fit
model.fit(vectorizer.transform(X_train).toarray(), y_train, epochs=10, batch_size=32, verbo
# evaluate
loss, accuracy = model.evaluate(vectorizer.transform(X_test).toarray(), y_test, batch_size=
print('loss:', loss)
print('accuracy:', accuracy)
```

```
Epoch 1/10
109/109 [==============================] - 12s 101ms/step - loss: 0.1888 - accuracy:
```

```
Epoch 2/10
109/109 [==============================] - 8s 71ms/step - loss: 0.0373 - accuracy: 0.
Epoch 3/10
109/109 [==============================] - 9s 83ms/step - loss: 0.0181 - accuracy: 0.
Epoch 4/10
109/109 [==============================] - 8s 75ms/step - loss: 0.0111 - accuracy: 0.
Epoch 5/10
109/109 [==============================] - 8s 76ms/step - loss: 0.0075 - accuracy: 0.
Epoch 6/10
109/109 [==============================] - 9s 80ms/step - loss: 0.0055 - accuracy: 0.
Epoch 7/10
109/109 [==============================] - 10s 94ms/step - loss: 0.0041 - accuracy: 0
Epoch 8/10
109/109 [==============================] - 9s 84ms/step - loss: 0.0032 - accuracy: 0.
Epoch 9/10
109/109 [==============================] - 10s 96ms/step - loss: 0.0026 - accuracy: 1
Epoch 10/10
109/109 [==============================] - 9s 84ms/step - loss: 0.0021 - accuracy: 1.
54/54 [==============================] - 1s 19ms/step - loss: 0.0908 - accuracy: 0.98
loss: 0.09078647196292877
accuracy: 0.9835969805717468
```

```
# tokenize
tokenizer = Tokenizer()
tokenizer.fit_on_texts(X)
# Convert text data to sequences
seq = tokenizer.texts_to_sequences(X)
# pad sequences
max_length = max([len(x) for x in seq])
X_padded = pad_sequences(seq, maxlen=max_length, padding='post')
X_train, X_test, y_train, y_test = train_test_split(X_padded, y, test_size=0.33, random_sta
#sequential model
model = Sequential()
# embedding layer
model.add(Embedding(input_dim=len(tokenizer.word_index) + 1, output_dim=64, input_length=ma
# LSTM layer
model.add(LSTM(units=128))
model.add(Dense(units=1, activation='sigmoid'))
# compile
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
# fit
model.fit(X_train, y_train, epochs=10, batch_size=32, verbose=1)
# evaluate
loss, accuracy = model.evaluate(X_test, y_test, batch_size=32, verbose=1)
print('loss:', loss)
print('accuracy:', accuracy)
```

```
Epoch 1/10
  4/109 [>.............................] - ETA: 38:00 - loss: 0.6744 - accuracy: 0.67
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-51-63580dd1f67d> in <cell line: 20>()
```

```
     18 model.compile(loss='binary_crossentropy', optimizer='adam', metrics=
['accuracy'])
     19 # fit
---> 20 model.fit(X_train, y_train, epochs=10, batch_size=32, verbose=1)
     21 # evaluate
     22 loss, accuracy = model.evaluate(X_test, y_test, batch_size=32, verbose=1)
```

8 frames

```
/usr/local/lib/python3.9/dist-packages/tensorflow/python/eager/execute.py in
quick_execute(op_name, num_outputs, inputs, attrs, ctx, name)
     50   try:
     51     ctx.ensure_initialized()
---> 52     tensors = pywrap_tfe.TFE_Py_Execute(ctx._handle, device_name, op_name,
     53                                         inputs, attrs, num_outputs)
     54   except core._NotOkStatusException as e:

KeyboardInterrupt:
```

^ takes a while

```
# train
sentences = [text.split() for text in X]
model = Word2Vec(sentences, vector_size=100, window=5, min_count=1, workers=4)
# embedding matrix
matrix = np.zeros((len(tokenizer.word_index) + 1, 100))
for word, i in tokenizer.word_index.items():
    if word in model.wv:
        matrix[i] = model.wv[word]
# sequential model
model = Sequential()
# embedding layer
model.add(Embedding(input_dim=len(tokenizer.word_index) + 1, output_dim=100, weights=[embed
# LSTM layer
model.add(LSTM(units=128))
# output layer - binary classification
model.add(Dense(units=1, activation='sigmoid'))
# compile
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
# fit
model.fit(X_train, y_train, epochs=10, batch_size=32, verbose=1)
# evaluate
loss, accuracy = model.evaluate(X_test, y_test, batch_size=32, verbose=1)
print('loss:', loss)
print('accuracy:', accuracy)
```

```
Epoch 1/10
WARNING:tensorflow:5 out of the last 1100 calls to <function Model.make_train_functio
 12/109 [==>.....................] - ETA: 17:23 - loss: 0.6940 - accuracy: 0.67
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
```

```
<ipython-input-52-07b8b876a467> in <cell line: 20>()
     18 model.compile(loss='binary_crossentropy', optimizer='adam', metrics=
['accuracy'])
     19 # fit
---> 20 model.fit(X_train, y_train, epochs=10, batch_size=32, verbose=1)
     21 # evaluate
     22 loss, accuracy = model.evaluate(X_test, y_test, batch_size=32, verbose=1)

                                    8 frames
/usr/local/lib/python3.9/dist-packages/tensorflow/python/eager/execute.py in
quick_execute(op_name, num_outputs, inputs, attrs, ctx, name)
     50   try:
     51     ctx.ensure_initialized()
---> 52     tensors = pywrap_tfe.TFE_Py_Execute(ctx._handle, device_name, op_name,
     53                                         inputs, attrs, num_outputs)
     54   except core._NotOkStatusException as e:

KeyboardInterrupt:
```

SEARCH STACK OVERFLOW

^ takes a while

Analysis

Given the data of the text and the label telling whether the text falls under ham or spam. This is the main purpose of the model. Various different techniques have been used like the count vectorizer, lstm and word2vec. With it an evaluation is to be done. The accuracy shows the performance metric, this measures the predictions made. The count vectorizer is used to convert text to numerical representations. To a matrix then it logistic regression is done. The accuracy calculated through cross validation. The resultt is 0.98 which is very good. LSTM for RNN learns the word representations. The accuracy here is 0.97 which is also really good. For the embedding part Word2Vec has been used. It make a more dense vectors. Word2Vec take in account the sematancis and the contexts of the data. Here the accuracy value is pretty low. The performance on these approaches matters on the training data, parameters, tuning and architecture. Overall the model performs well according to the accuracies.

Colab paid products - Cancel contracts here