sms-spam-detection-aditi

October 6, 2023

The SPAM dataset contains 5572 SMS messages and a label. Using this dataset, you are going to create a machine learning model that learns to detect "spam" or not "spam".

0.1 Imports Libraries

[1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

import warnings
warnings.filterwarnings("ignore")

import pickle
import tensorflow as tf
import wordcloud
```

1 SIMPLE EDA

[2]:
```
df = pd.read_csv("spam.csv",encoding='latin-1')
df.head()
```

[2]:
```
     v1                                                 v2 Unnamed: 2  \
0   ham  Go until jurong point, crazy.. Available only …         NaN
1   ham                      Ok lar… Joking wif u oni…         NaN
2  spam  Free entry in 2 a wkly comp to win FA Cup fina…         NaN
3   ham  U dun say so early hor… U c already then say…         NaN
4   ham  Nah I don't think he goes to usf, he lives aro…         NaN

  Unnamed: 3 Unnamed: 4
0        NaN        NaN
1        NaN        NaN
2        NaN        NaN
3        NaN        NaN
4        NaN        NaN
```

The Columns 2,3,4 will be dropped as they contain no relevant information

[3]:
```
data = df.copy()
data.drop(columns=["Unnamed: 2", "Unnamed: 3", "Unnamed: 4"], inplace=True)
data = data.rename(columns={"v1":"label", "v2":"text"})
```

[4]:
```
data.head()
```

[4]:
```
  label                                               text
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…
```

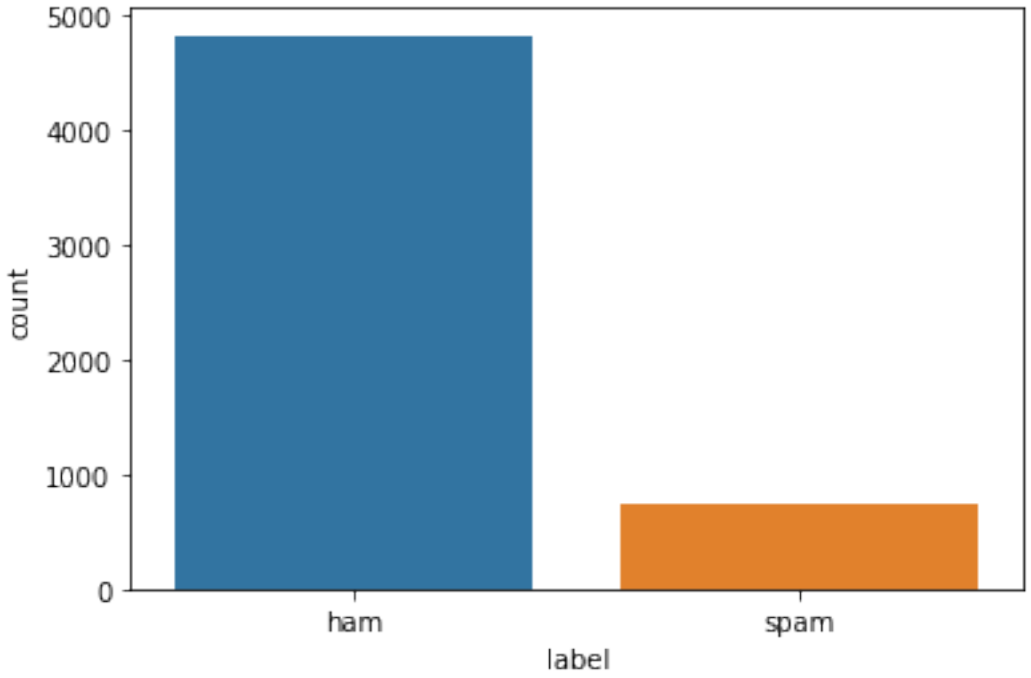
1.0.1 Distribution of the label variable

The dataset contains 4825 ham and 747 spam messages.

[5]:
```
data.label.value_counts()
```

[5]:
```
ham     4825
spam     747
Name: label, dtype: int64
```

[6]:
```
sns.countplot(data['label'])
plt.savefig("dist")
plt.show()
```



Convert labels numerical format

[7]:
```
data['label'] = data['label'].map( {'spam': 1, 'ham': 0} )
data.head()
```

[7]:
```
   label                                               text
0      0  Go until jurong point, crazy.. Available only …
1      0                      Ok lar… Joking wif u oni…
2      1  Free entry in 2 a wkly comp to win FA Cup fina…
3      0  U dun say so early hor… U c already then say…
4      0  Nah I don't think he goes to usf, he lives aro…
```

[8]:
```
data_ham  = data[data['label'] == 0].copy()
data_spam = data[data['label'] == 1].copy()
```

WordClouds

[9]:
```
def show_wordcloud(df, title):
    text = ' '.join(df['text'].astype(str).tolist())
    stopwords = set(wordcloud.STOPWORDS)

    fig_wordcloud = wordcloud.
 ↪WordCloud(stopwords=stopwords,background_color='lightgrey',
                    colormap='viridis', width=800, height=600).generate(text)

    plt.figure(figsize=(10,7), frameon=True)
    plt.imshow(fig_wordcloud)
    plt.axis('off')
    plt.title(title, fontsize=20 )
    plt.savefig("wd.png")
    plt.show()
```

1.0.2 WordCloud: Ham messages

[10]:
```
show_wordcloud(data_ham, "Ham messages")
```

1.0.3 WordCloud: Spam messages

[11]:
```
show_wordcloud(data_spam, "Spam messages")
```

Notice that majority of the word in Spam messages contains terms like FREE, CALL, URGENT. These terms will serve as important features to our model.

[12]:
```
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Flatten
from tensorflow.keras.layers import Embedding
from tensorflow.keras.callbacks import EarlyStopping
from sklearn.model_selection import train_test_split

X = data['text'].values
y = data['label'].values
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20,␣
 ↪random_state=42)
```

1.1 FEATURE ENGINEERING AND PROCESSING

[17]:
```
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Dropout
from tensorflow.keras.layers import Flatten
from tensorflow.keras.layers import Embedding
from tensorflow.keras.callbacks import EarlyStopping
```

[18]:
```
# prepare tokenizer
t = Tokenizer()
t.fit_on_texts(X_train)

# integer encode the documents
encoded_train = t.texts_to_sequences(X_train)
encoded_test = t.texts_to_sequences(X_test)

vocab_size = len(t.word_index) + 1

print(encoded_train[0:2])
```

```
[[38, 30, 8, 5, 273, 1989, 81, 116, 26, 11, 1656, 322, 10, 53, 18, 299, 30, 349,
1990], [799, 15, 2555, 1442, 1127, 192, 2556, 171, 12, 98, 1991, 44, 195, 1657,
2557, 1992, 2558, 21, 9, 4, 203, 1025, 225]]
```

[19]:
```
# pad documents to a max length of 4 words
max_length = 8
padded_train = pad_sequences(encoded_train, maxlen=max_length, padding='post')
padded_test = pad_sequences(encoded_test, maxlen=max_length, padding='post')

print(padded_train)
```

```
[[ 322   10   53 …   30  349 1990]
 [1992 2558   21 …  203 1025  225]
 [  83 1443    4 …    2 3794 3795]
 …
 [1477   30 2063 …  239   30 2064]
 [ 763 1679 1161 …    0    0    0]
 [   8  155   20 …    8  290  175]]
```

[20]:
```
# define the model
model = Sequential()
model.add(Embedding(vocab_size, 24, input_length=max_length))
model.add(Flatten())
model.add(Dense(500, activation='relu'))
model.add(Dense(200, activation='relu'))
```

```
model.add(Dropout(0.5))
model.add(Dense(100, activation='relu'))
model.add(Dense(1, activation='sigmoid'))

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

# summarize the model
print(model.summary())
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 8, 24)             190920
_________________________________________________________________
flatten_1 (Flatten)          (None, 192)               0
_________________________________________________________________
dense_2 (Dense)              (None, 500)               96500
_________________________________________________________________
dense_3 (Dense)              (None, 200)               100200
_________________________________________________________________
dropout (Dropout)            (None, 200)               0
_________________________________________________________________
dense_4 (Dense)              (None, 100)               20100
_________________________________________________________________
dense_5 (Dense)              (None, 1)                 101
=================================================================
Total params: 407,821
Trainable params: 407,821
Non-trainable params: 0
_________________________________________________________________
None
```

[21]:
```
early_stop = EarlyStopping(monitor='val_loss', mode='min', verbose=1,␣
 ↪patience=10)

# fit the model
model.fit(x=padded_train,
          y=y_train,
          epochs=50,
          validation_data=(padded_test, y_test), verbose=1,
          callbacks=[early_stop]
          )
```

```
Epoch 1/50
```

```
140/140 [==============================] - 1s 4ms/step - loss: 0.2034 -
accuracy: 0.9195 - val_loss: 0.1061 - val_accuracy: 0.9758
Epoch 2/50
140/140 [==============================] - 0s 3ms/step - loss: 0.0447 -
accuracy: 0.9865 - val_loss: 0.0840 - val_accuracy: 0.9821
Epoch 3/50
140/140 [==============================] - 0s 3ms/step - loss: 0.0136 -
accuracy: 0.9969 - val_loss: 0.0997 - val_accuracy: 0.9839
Epoch 4/50
140/140 [==============================] - 0s 3ms/step - loss: 6.0631e-04 -
accuracy: 0.9998 - val_loss: 0.2119 - val_accuracy: 0.9830
Epoch 5/50
140/140 [==============================] - 0s 3ms/step - loss: 1.2411e-06 -
accuracy: 1.0000 - val_loss: 0.2899 - val_accuracy: 0.9803
Epoch 6/50
140/140 [==============================] - 0s 3ms/step - loss: 3.1918e-08 -
accuracy: 1.0000 - val_loss: 0.2903 - val_accuracy: 0.9821
Epoch 7/50
140/140 [==============================] - 0s 3ms/step - loss: 4.8863e-09 -
accuracy: 1.0000 - val_loss: 0.2921 - val_accuracy: 0.9830
Epoch 8/50
140/140 [==============================] - 0s 2ms/step - loss: 9.7544e-10 -
accuracy: 1.0000 - val_loss: 0.2946 - val_accuracy: 0.9830
Epoch 9/50
140/140 [==============================] - 0s 3ms/step - loss: 1.3770e-09 -
accuracy: 1.0000 - val_loss: 0.3048 - val_accuracy: 0.9821
Epoch 10/50
140/140 [==============================] - 0s 3ms/step - loss: 1.3219e-09 -
accuracy: 1.0000 - val_loss: 0.3032 - val_accuracy: 0.9812
Epoch 11/50
140/140 [==============================] - 0s 3ms/step - loss: 1.1548e-09 -
accuracy: 1.0000 - val_loss: 0.3015 - val_accuracy: 0.9830
Epoch 12/50
140/140 [==============================] - 0s 3ms/step - loss: 8.7392e-10 -
accuracy: 1.0000 - val_loss: 0.3087 - val_accuracy: 0.9830
Epoch 00012: early stopping
```

[21]:
```
<tensorflow.python.keras.callbacks.History at 0x7f98a9070d50>
```

[24]:
```
# evaluate the model
loss, accuracy = model.evaluate(padded_test, y_test, verbose=0)
print('Accuracy: %f' % (accuracy*100))
```

```
Accuracy: 98.295963
```

[25]:
```
preds = (model.predict(padded_test) > 0.5).astype("int32")
```

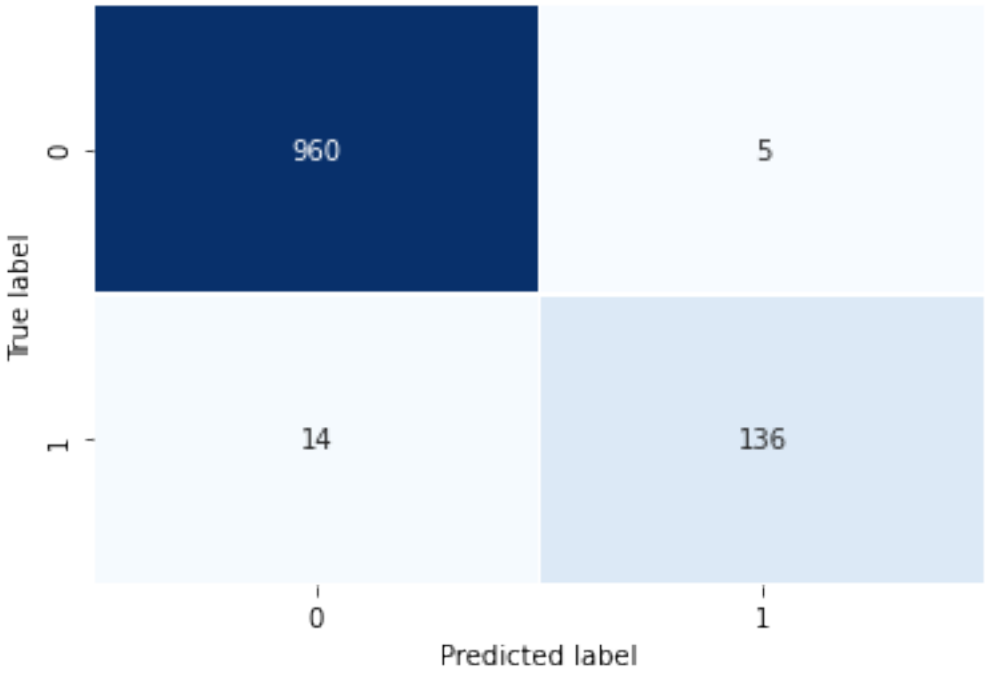
[26]:
```
from sklearn.metrics import classification_report, confusion_matrix,␣
↪accuracy_score


def c_report(y_true, y_pred):
    print("Classification Report")
    print(classification_report(y_true, y_pred))
    acc_sc = accuracy_score(y_true, y_pred)
    print("Accuracy : "+ str(acc_sc))
    return acc_sc


def plot_confusion_matrix(y_true, y_pred):
    mtx = confusion_matrix(y_true, y_pred)
    sns.heatmap(mtx, annot=True, fmt='d', linewidths=.5,
                cmap="Blues", cbar=False)
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
    plt.savefig("cf.png")
```

[27]:
```
c_report(y_test, preds)
```

```
Classification Report
              precision    recall  f1-score   support

           0       0.99      0.99      0.99       965
           1       0.96      0.91      0.93       150

    accuracy                           0.98      1115
   macro avg       0.98      0.95      0.96      1115
weighted avg       0.98      0.98      0.98      1115

Accuracy : 0.9829596412556054
```

[27]:
```
0.9829596412556054
```

[28]:
```
plot_confusion_matrix(y_test, preds)
```



1.2 SAVING MODEL AND TOKENIZER

[29]:
```
model.save("spam_model")
```

```
WARNING:tensorflow:From
/Users/mac/opt/anaconda3/envs/deeplearning/lib/python3.7/site-
packages/tensorflow/python/training/tracking/tracking.py:111:
Model.state_updates (from tensorflow.python.keras.engine.training) is deprecated
and will be removed in a future version.
Instructions for updating:
This property should not be used in TensorFlow 2.0, as updates are applied
automatically.
WARNING:tensorflow:From
/Users/mac/opt/anaconda3/envs/deeplearning/lib/python3.7/site-
packages/tensorflow/python/training/tracking/tracking.py:111: Layer.updates
(from tensorflow.python.keras.engine.base_layer) is deprecated and will be
removed in a future version.
Instructions for updating:
This property should not be used in TensorFlow 2.0, as updates are applied
automatically.
INFO:tensorflow:Assets written to: spam_model/assets
```

[30]:
```
with open('spam_model/tokenizer.pkl', 'wb') as output:
    pickle.dump(t, output, pickle.HIGHEST_PROTOCOL)
```

1.3 LOADING AND EVALUATING MODEL

[31]:
```
s_model = tf.keras.models.load_model("spam_model")

with open('spam_model/tokenizer.pkl', 'rb') as input:
    tokener = pickle.load(input)

# s_model.summary()
```

[38]:
```
sms_spam = ["We know someone who you know that fancies you. Call 09058097218 to␣
 ↪find out who. POBox 6, LS15HB "]
sms_ham = ["I'll text Tanya when I get home, hang on"]

sms_proc = tokener.texts_to_sequences(sms_ham)
sms_proc = pad_sequences(sms_proc, maxlen=max_length, padding='post')

pred = (model.predict(sms_proc) > 0.5).astype("int32").item()
pred
```

[39]:
```
pred = (model.predict(sms_proc) > 0.5).astype("int32").item()
pred
```

[39]:
```
0
```

[33]:
```
X_test[5]
```

[33]:
```
"I'll text carlos and let you know, hang on"
```

[ ]:
```

```