ProgrammingSolution_Sheet3_NayanManSinghPradhan

March 2, 2021

1 Assignment Sheet 3

1.1 Done by Nayan Man Singh Pradhan

1.1.1 My Approach Explained:

- I downloaded all the documents from website: https://archive.ics.uci.edu/ml/datasets/spambase.
- I loaded the data from the "spambase.data" file.
- I used Linear Regression, SVC, and KNN to create a Pipeline that predicts whether the email is Spam or Not Spam based on the loaded data from the "spambase.data" file.
- I picked the classifier with the best accuracy (SVC) in order to predict any test email as Spam or Not Spam based on the trained input data.
- I loaded the example spam emails.
- I extracted the required 57 attributes.
- I printed the 57 attributes into the file: "only_all_attributes.txt".
- I predicted whether the example was Spam or Not Spam based on the trained pipeline (SVC).
- I printed the total vector or 57 attributes + spam or not spam class = 58 into the file: "attributes+prediction_output.txt" file.
- I tested the example mail + an extra test email (just for testing purpose).

1.1.2 Importing libraries

[1]:
```
## importing libraries
import numpy as np
import pandas as pd

import sklearn
from sklearn.pipeline import Pipeline
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report,␣
↪confusion_matrix
from sklearn.svm import LinearSVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LinearRegression
import re
```

1.1.3 Loading data

[2]:
```python
## loading data
raw_data = pd.read_csv('spambase.data', sep=',', header=None)
raw_data.head()
```

[2]:
```
     0     1     2    3     4     5     6     7     8     9   ...    48  \
0  0.00  0.64  0.64  0.0  0.32  0.00  0.00  0.00  0.00  0.00  ...  0.00
1  0.21  0.28  0.50  0.0  0.14  0.28  0.21  0.07  0.00  0.94  ...  0.00
2  0.06  0.00  0.71  0.0  1.23  0.19  0.19  0.12  0.64  0.25  ...  0.01
3  0.00  0.00  0.00  0.0  0.63  0.00  0.31  0.63  0.31  0.63  ...  0.00
4  0.00  0.00  0.00  0.0  0.63  0.00  0.31  0.63  0.31  0.63  ...  0.00

      49   50     51     52     53     54   55    56  57
0  0.000  0.0  0.778  0.000  0.000  3.756   61   278   1
1  0.132  0.0  0.372  0.180  0.048  5.114  101  1028   1
2  0.143  0.0  0.276  0.184  0.010  9.821  485  2259   1
3  0.137  0.0  0.137  0.000  0.000  3.537   40   191   1
4  0.135  0.0  0.135  0.000  0.000  3.537   40   191   1

[5 rows x 58 columns]
```

1.1.4 Training and Testing Data using different classifiers

[3]:
```python
## function that turns float prediction to binary (only for linear regression)
def predict_binary(raw_out):
    if (raw_out < 0.5):
        return "0"
    else:
        return "1"
```

[4]:
```python
## X and y
X = np.array(raw_data.drop(raw_data.columns[57], 1))
y = np.array(raw_data[57])
```

[5]:
```python
## test examples

# example = [0,0,0,0,0,0,0,0,0,0,0,0,0,0.85,0,0,0,0,0.
↪85,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0.
↪126,0,0,0,0,3.925,51,106]
example = [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1.57,0,0,0,0,0.
↪78,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0.184,0,8.
↪161,31,253]
# print(len(example))
```

[6]:
```python
## Linear Reg classifier
```

```
classifier = LinearRegression()
x_train, x_test, y_train, y_test = train_test_split(X,y,test_size=0.1,␣
 ↪random_state=50)
linear_clf = Pipeline([('clf', classifier)])

linear_clf.fit(x_train, y_train)
y_pred = linear_clf.predict(x_test)

acc_Linreg = linear_clf.score(x_test, y_test)
print("Accuracy Using Linear Regression:", acc_Linreg)

print(linear_clf.predict([example]))
print(predict_binary(linear_clf.predict([example])))
```

```
Accuracy Using Linear Regression: 0.5723034491347803
[0.47398099]
0
```

[7]:
```
## SVC classifier

classifier = LinearSVC(dual=False)
x_train, x_test, y_train, y_test = train_test_split(X, y, test_size = 0.1,␣
 ↪random_state=50)
SVC_clf = Pipeline([('clf', classifier)])

SVC_clf.fit(x_train, y_train)
y_pred = SVC_clf.predict(x_test)

acc_SVC = SVC_clf.score(x_test, y_test)
print("Accuracy using SVC:", acc_SVC)

print(SVC_clf.predict([example]))
print(predict_binary(SVC_clf.predict([example])))
```

```
Accuracy using SVC: 0.9501084598698482
[1]
1
```

[8]:
```
## KNN classifier

classifier = KNeighborsClassifier(n_neighbors=2)
x_train, x_test, y_train, y_test = train_test_split(X, y, test_size = 0.1,␣
 ↪random_state=50)

KNN_clf = Pipeline([('clf', classifier)])

KNN_clf.fit(x_train, y_train)
```

```
y_pred = KNN_clf.predict(x_test)

acc_KNN = KNN_clf.score(x_test, y_test)
print("Accuracy using KNN:", acc_KNN)

print(KNN_clf.predict([example]))
print(predict_binary(KNN_clf.predict([example])))
```

```
Accuracy using KNN: 0.7982646420824295
[1]
1
```

1.1.5 Loading sample emails

[9]:
```
## loading samples

f1 = open('Sample_Emails/spam_or_no_spam.txt', 'r')
sample_mail_1 = f1.read()

f2 = open('Sample_Emails/spam_or_no_spam_2.txt', 'r')
sample_mail_2 = f2.read()

f3 = open('Sample_Emails/spam_or_no_spam_3.txt', 'r')
sample_mail_3 = f3.read()

# print(sample_mail_1)
# print(sample_mail_2)
# print(sample_mail_3)
```

1.1.6 Extracting attributes

[10]:
```
words = ['make', 'address', 'all', '3d', 'our', 'over', 'remove', 'internet',␣
 ↪'order', 'mail', 'receive', 'will', 'people', 'report', 'addresses', 'free',␣
 ↪'business', 'email', 'you', 'credit', 'your', 'font', '000', 'money', 'hp',␣
 ↪'hpl', 'george', '650', 'lab', 'labs', 'telnet', '857', 'data', '415', '85',␣
 ↪'technology', '1999', 'parts', 'pm', 'direct', 'cs', 'meeting', 'original',␣
 ↪'project', 're', 'edu', 'table', 'conference']
char = [';', '(', '[', '!', '$', '#']

# print(len(words))
# print(len(char))
```

1.1.7 Function returning array of attributes (without last prediction class)

[11]:
```
## function returning array of attributes (without last prediction attribute)

def return_attributes(name_of_input_file):

    output_arr = np.zeros(0)

    tokens = [t for t in name_of_input_file.split()]
    total_no_tokens = (len(tokens))

    total_no_chars = len(name_of_input_file)

    ## For 48 attributes
    words_freq = np.zeros(len(words)) # creating numpy array for storing freq␣
↪of words
    words_freq_perc = np.zeros(len(words)) # creating numpy array for stroing␣
↪percentage of frequency of words

    for token in tokens:
        if token.lower() in words:
            words_freq[words.index(token.lower())] += 1 # add 1 to freq

    for idx, single_word_freq in enumerate(words_freq):
        words_freq_perc[idx] = (100*single_word_freq)/total_no_tokens #### OUT␣
↪1 ####

    ## For 6 attributes
    char_freq = np.zeros(len(char))
    char_freq_perc = np.zeros(len(char))
    for token in tokens:
        for ind_char in token:
            if ind_char in char:
                char_freq[char.index(ind_char)] += 1

    for idx, single_char_freq in enumerate(char_freq):
        char_freq_perc[idx] = (100*single_char_freq)/total_no_chars #### OUT 2␣
↪####


    ## For Capital Letters
    all_capital_letters = re.findall(r"[A-Z]+", name_of_input_file)

    sum_len_capital_arr = np.zeros(1)
    sum_len_capital = 0
    for capital_letter in all_capital_letters:
        temp = (len(capital_letter))
```

```
        sum_len_capital += temp
    sum_len_capital_arr[0] = sum_len_capital #### OUT 5 ####

    len_longest_seq_capital = np.zeros(1)
    longest_seq_capital = max(all_capital_letters, key=len)
    len_longest_seq_capital[0] = len(longest_seq_capital) #### OUT 4 ###

    avg_leng = np.zeros(1)
    avg_leng[0] = sum_len_capital/len(all_capital_letters) #### OUT 3 ###

    output_arr = np.zeros(0)
    output_arr = np.concatenate([words_freq_perc, char_freq_perc, avg_leng,␣
 ↪len_longest_seq_capital, sum_len_capital_arr], axis=0)

#     last_attr = np.zeros(1) #### OUT 6 ####
#     last_attr[0] = predict_binary(SVC_clf.predict([temp_output_arr]))
#     output_arr = np.concatenate([words_freq_perc, char_freq_perc, avg_leng,␣
 ↪len_longest_seq_capital, sum_len_capital_arr, last_attr], axis=0)

    return output_arr
```

1.1.8 Extracting all Attributes (57 columns)

[12]:
```
## for 'spam_or_no_spam.txt' doc

only_attributes_1 = return_attributes(sample_mail_1)
print(only_attributes_1)
print("len =",len(only_attributes_1))
```

```
[ 0.          0.          0.          0.          0.          0.
  0.          0.          0.          0.          0.          1.09289617
  0.          0.          0.          0.          0.          0.
  0.54644809  0.          0.54644809  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.          0.1980198   0.1980198   0.0660066   0.          0.0660066
  1.16666667  2.         56.        ]
len = 57
```

[13]:
```
## for 'spam_or_no_spam_2.txt' doc

only_attributes_2 = return_attributes(sample_mail_2)
print(only_attributes_2)
print("len =",len(only_attributes_2))
```

```
[ 0.          0.          0.          0.          0.          0.
  0.          0.          0.          0.          0.          0.
  0.          0.          0.          0.          0.          3.44827586
  1.72413793  0.          1.72413793  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.
  1.36363636  5.         45.        ]
len = 57
```

[14]:
```
## for 'spam_or_no_spam_3.txt' doc

only_attributes_3 = return_attributes(sample_mail_3)
print(only_attributes_3)
print("len =",len(only_attributes_3))
```

```
[ 0.          0.          0.          0.          0.81967213  0.
  0.          0.          0.          0.          0.          0.
  0.          0.          0.          0.          3.27868852  0.
  0.81967213  0.          1.63934426  0.          0.          0.
  0.          0.          0.          0.          0.          0.
  0.          0.          0.          0.          0.          0.
  0.          0.          0.          0.          0.          0.
  0.          0.81967213  0.          0.          0.          0.
  0.12210012  0.12210012  0.          0.          0.          0.
  1.65714286 10.         58.        ]
len = 57
```

1.1.9 Writing Extracted Attributes + prediction spam or not spam to Output File

[15]:
```
out_file_1 = open("only_all_attributes.txt", "w+")
out_file_1.write(str(only_attributes_1))
out_file_1.write('\n')
out_file_1.write(str(only_attributes_2))
out_file_1.write('\n')
out_file_1.write(str(only_attributes_3))
out_file_1.write('\n')
out_file_1.close()
```

1.1.10 Function Returning Array of Attributes (including last spam or not spam prediction class)

[16]:
```
## function returning array of complete attributes + spam or not spam prediction class

def return_comp_attributes(name_of_input_file):

    output_arr = np.zeros(0)

    tokens = [t for t in name_of_input_file.split()]
    total_no_tokens = (len(tokens))

    total_no_chars = len(name_of_input_file)

    ## For 48 attributes
    words_freq = np.zeros(len(words)) # creating numpy array for storing freq of words
    words_freq_perc = np.zeros(len(words)) # creating numpy array for stroing percentage of frequency of words

    for token in tokens:
        if token.lower() in words:
            words_freq[words.index(token.lower())] += 1 # add 1 to freq

    for idx, single_word_freq in enumerate(words_freq):
        words_freq_perc[idx] = (100*single_word_freq)/total_no_tokens #### OUT 1 ####

    ## For 6 attributes
    char_freq = np.zeros(len(char))
    char_freq_perc = np.zeros(len(char))
    for token in tokens:
        for ind_char in token:
            if ind_char in char:
                char_freq[char.index(ind_char)] += 1

    for idx, single_char_freq in enumerate(char_freq):
        char_freq_perc[idx] = (100*single_char_freq)/total_no_chars #### OUT 2 ####


    ## For Capital Letters
    all_capital_letters = re.findall(r"[A-Z]+", name_of_input_file)

    sum_len_capital_arr = np.zeros(1)
    sum_len_capital = 0
```

```
    for capital_letter in all_capital_letters:
        temp = (len(capital_letter))
        sum_len_capital += temp
    sum_len_capital_arr[0] = sum_len_capital #### OUT 5 ####

    len_longest_seq_capital = np.zeros(1)
    longest_seq_capital = max(all_capital_letters, key=len)
    len_longest_seq_capital[0] = len(longest_seq_capital) #### OUT 4 ###

    avg_leng = np.zeros(1)
    avg_leng[0] = sum_len_capital/len(all_capital_letters) #### OUT 3 ###

    temp_output_arr = np.zeros(0)
    temp_output_arr = np.concatenate([words_freq_perc, char_freq_perc,␣
↪avg_leng, len_longest_seq_capital, sum_len_capital_arr], axis=0)

    ## Prediction Attribute (final attribute) using SVC classifier
    last_attr = np.zeros(1) #### OUT 6 ####
    last_attr[0] = predict_binary(SVC_clf.predict([temp_output_arr]))
    output_arr = np.concatenate([words_freq_perc, char_freq_perc, avg_leng,␣
↪len_longest_seq_capital, sum_len_capital_arr, last_attr], axis=0)

    return output_arr
```

1.1.11 Extracting all Attributes + last column of Spam or Not Spam prediction (57+1=58 columns)

[17]:
```
## for 'spam_or_no_spam.txt' doc

attributes_1 = return_comp_attributes(sample_mail_1)
print(attributes_1)
print("len =",len(attributes_1))
```

```
[ 0.          0.          0.          0.          0.          0.
  0.          0.          0.          0.          0.          1.09289617
  0.          0.          0.          0.          0.          0.
  0.54644809  0.          0.54644809  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.          0.1980198   0.1980198   0.0660066   0.          0.0660066
  1.16666667  2.         56.          0.        ]
len = 58
```

[18]:
```
## for 'spam_or_no_spam_2.txt' doc

attributes_2 = return_comp_attributes(sample_mail_2)
print(attributes_2)
print("len =",len(attributes_2))
```

```
[ 0.          0.          0.          0.          0.          0.
  0.          0.          0.          0.          0.          0.
  0.          0.          0.          0.          0.          3.44827586
  1.72413793  0.          1.72413793  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.
  1.36363636  5.         45.          0.        ]
len = 58
```

[19]:
```
## for 'spam_or_no_spam_3.txt' doc

attributes_3 = return_comp_attributes(sample_mail_3)
print(attributes_3)
print("len =",len(attributes_3))
```

```
[ 0.          0.          0.          0.          0.81967213  0.
  0.          0.          0.          0.          0.          0.
  0.          0.          0.          0.          3.27868852  0.
  0.81967213  0.          1.63934426  0.          0.          0.
  0.          0.          0.          0.          0.          0.
  0.          0.          0.          0.          0.          0.
  0.          0.          0.          0.          0.          0.
  0.          0.81967213  0.          0.          0.          0.
  0.12210012  0.12210012  0.          0.          0.          0.
  1.65714286 10.         58.          1.        ]
len = 58
```

1.1.12 Writing Extracted Attributes + prediction spam or not spam to Output File

[20]:
```
out_file_2 = open("attributes+prediction_output.txt", "w+")
out_file_2.write(str(attributes_1))
out_file_2.write('\n')
out_file_2.write(str(attributes_2))
out_file_2.write('\n')
out_file_2.write(str(attributes_3))
out_file_2.write('\n')
out_file_2.close()
```

1.1.13 Testing Different Classifiers on Given Example Emails

[21]:
```
print("For 'spam_or_no_spam.txt':")
print("Using Linear Regression Classifier:", predict_binary(linear_clf.
 ↪predict([return_attributes(sample_mail_1)])))
print("Using SVC Classifier:", predict_binary(SVC_clf.
 ↪predict([return_attributes(sample_mail_1)])))
print("Using KNN Classifier:", predict_binary(KNN_clf.
 ↪predict([return_attributes(sample_mail_1)])))
```

```
For 'spam_or_no_spam.txt':
Using Linear Regression Classifier: 0
Using SVC Classifier: 0
Using KNN Classifier: 0
```

[22]:
```
print("For 'spam_or_no_spam_2.txt':")
print("Using Linear Regression Classifier:", predict_binary(linear_clf.
 ↪predict([return_attributes(sample_mail_2)])))
print("Using SVC Classifier:", predict_binary(SVC_clf.
 ↪predict([return_attributes(sample_mail_2)])))
print("Using KNN Classifier:", predict_binary(KNN_clf.
 ↪predict([return_attributes(sample_mail_2)])))
```

```
For 'spam_or_no_spam_2.txt':
Using Linear Regression Classifier: 0
Using SVC Classifier: 0
Using KNN Classifier: 0
```

[23]:
```
print("For 'spam_or_no_spam_3.txt':")
print("Using Linear Regression Classifier:", predict_binary(linear_clf.
 ↪predict([return_attributes(sample_mail_3)])))
print("Using SVC Classifier:", predict_binary(SVC_clf.
 ↪predict([return_attributes(sample_mail_3)])))
print("Using KNN Classifier:", predict_binary(KNN_clf.
 ↪predict([return_attributes(sample_mail_3)])))
```

```
For 'spam_or_no_spam_3.txt':
Using Linear Regression Classifier: 0
Using SVC Classifier: 1
Using KNN Classifier: 0
```

1.1.14 Testing Random Mail file: 'spam_try.txt'

[24]:
```
f4 = open('Sample_Emails/spam_try.txt', 'r')
sample_mail_4 = f4.read()
attributes_4 = return_attributes(sample_mail_4)
```

```
print("Using Linear Regression Classifier:", predict_binary(linear_clf.
 ↪predict([attributes_4])))
print("Using SVC Classifier:", predict_binary(SVC_clf.predict([attributes_4])))
print("Using KNN Classifier:", predict_binary(KNN_clf.predict([attributes_4])))
```

```
Using Linear Regression Classifier: 1
Using SVC Classifier: 1
Using KNN Classifier: 0
```

[ ]: