Updated KNN

December 14, 2020

[1]:
```
%matplotlib inline
import numpy as np
import pandas as pd
import sklearn
from sklearn import datasets, metrics
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
from termcolor import colored
from sklearn.neighbors import KNeighborsClassifier


# The training data has 42000 entries and 785 features
train_digits = pd.read_csv("train.csv")
```

[2]:
```
# Creating sections for training and testing data
# Creating training and test sets
# Splitting the data into train and test
A_matrix = train_digits.iloc[:, 1:]
Labels_A = train_digits.iloc[:, 0]
```

[ ]:
```
# Processing the images before analysis using scale function from sklearn.
 ↪preprocessing
# Scale is used to standardise the pixels(features) in the image. This needs to␣
 ↪be done because our dataset is sparse/widely soread out.
# Which can cause issues with our model being statistically skewed. Scaling the␣
 ↪data in this manner
# makes  the matrix entries have 0 mean and the standard variation of 1 in each␣
 ↪field.
A_matrix = sklearn.preprocessing.scale(A_matrix)
```

[ ]:
```
# Splitting the data with 42k datapoints into 80% training data and 20% testing␣
 ↪data
# The stratify property was used to ensure the data is split with even label␣
 ↪distribution
X_train, X_test, b_train, b_test = train_test_split(A_matrix, Labels_A,␣
 ↪train_size=0.80, random_state=10, stratify=Labels_A)
print("Total data points in the A matrix : ", len(b_train))
```

```
# Taking 10% of the training data and setting aside for validation
(X_train, X_validation, b_train, b_validation) = train_test_split(X_train,␣
 ↪b_train, test_size=0.10)
print("Points from previously assigned training data points to be used for␣
 ↪VALIDATION:", len(b_validation))
print("All X matrix data points for TESTING: ", len(b_test))
print("All X matrix data points for TRAINING:", len(b_train))
print("\n")
```

[ ]:
```
#For loop to test accuracy for different k values ranging from 1 to 20
for k in range(1, 21, 1):
    accuracy_model = sklearn.neighbors.KNeighborsClassifier(n_neighbors=k)
    #Training KNN Algo
    accuracy_model.fit(X_train, b_train)
    #Calculating Model Accuracy
    print("k=%d, Classification Accuracy =%.2f%%" % (k, accuracy_model.
 ↪score(X_validation, b_validation) * 100))
print(colored("k = 1 gives us the maximum accuracy, and so k = 2 is the best␣
 ↪parameter that we will choose to test the KNN model on the test data set",␣
 ↪'green', attrs=['bold']))

#Training KNN using k = 1
knn_model_new = sklearn.neighbors.KNeighborsClassifier(n_neighbors=1)
#Training KNN Algo
knn_model_new.fit(X_train, b_train)
#Estimating model accuracy on Test data set
accuracy_predict = knn_model_new.predict(X_test)
print(colored("\n\n\nClassification Report for KNN Classification", 'red'))
#Generates a classification report analysing the accuracy for each digit's␣
 ↪classification
report_c = sklearn.metrics.classification_report(b_test, accuracy_predict)
print(report_c)
```

0.0.1 Confusion Matrix

[ ]:
```
confusion_MAT = sklearn.metrics.plot_confusion_matrix(knn_model_new, X_test,␣
 ↪b_test)
confusion_MAT.ax_.set_title('Confusion Matrix for KNN', color = 'white')
plt.xlabel("Predicted Label", color = 'red')
plt.ylabel("True Label", color = 'red')
plt.gcf().set_size_inches(15,8)
print(colored("CONFUSION MATRIX", 'red', attrs=['bold']))
```