Pattern Recognition Assignment

Siddhartha Singh Bhadauriya(2k18/se/123)

Problem Statement

In this assignment, the problem is basically to understand the concept of K-nn algorithm and as we know that finding the value of K in K-nn algorithm i snot an easy task, so we will have to use grid square method to find optimal number of nearest neighbors.

Objectives:

(1) Load the dataset and perform splitting into training and validation sets with 70:30 ratio. Use tsne plot to visualise the dataset.

(2) Implement the kNN algorithm from scratch. You need to find the optimal number of k using the grid search. You may use sklearn for grid search. Plot the error vs number of neighbors graph (k). Report the optimal number of neighbours.

(3) Report the training and the validation accuracy only with optimal value of k using sklearn kNN function.

(4) Comment on the accuracy obtained for optimal value of k for both the methods i.e, your implementation and the inbuilt sklearn function.

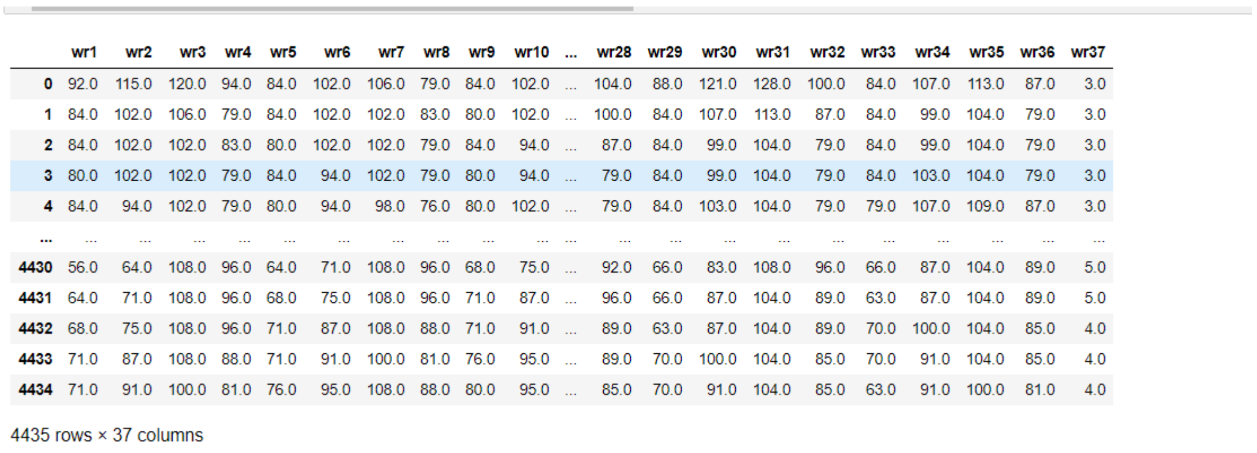
Solution:

Dataset used: We will use Iris dataset to perform this k-nn algorithm.

Dataset Description:
Each pixel is an 8-bit binary word, with 0 corresponding to black and 255 to white. The spatial resolution of a pixel is about 80m x 80m. Each image contains 2340 x 3380 such pixels. Each line contains the pixel values in the four spectral bands (converted to ASCII) of each of the 9 pixels in the 3x3 neighbourhood and a number indicating the classification label of the central pixel. The number is a code for the following classes:

1. red soil
2. cotton crop
3. grey soil
4. damp grey soil
5. soil with vegetation stubble
6. mixture class (all types present)
7. very damp grey soil

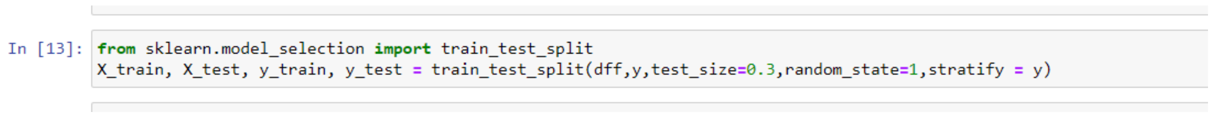
There are no examples with class 6 in this dataset. The data is given in random order and certain lines of data have been removed so you cannot reconstruct the original image from this dataset.
Training Set 4435
Test Set 2000
36 (= 4 spectral bands x 9 pixels in neighbourhood)
The attributes are numerical, in the range 0 to 255. There are 6 decision classes: 1,2,3,4,5 and 7. There are no examples with class 6 in this datasetthey have all been removed because of doubts about the validity of this
Class.



	wr1	wr2	wr3	wr4	wr5	wr6	wr7	wr8	wr9	wr10	...	wr28	wr29	wr30	wr31	wr32	wr33	wr34	wr35	wr36	wr37
0	92.0	115.0	120.0	94.0	84.0	102.0	106.0	79.0	84.0	102.0	...	104.0	88.0	121.0	128.0	100.0	84.0	107.0	113.0	87.0	3.0
1	84.0	102.0	106.0	79.0	84.0	102.0	102.0	83.0	80.0	102.0	...	100.0	84.0	107.0	113.0	87.0	84.0	99.0	104.0	79.0	3.0
2	84.0	102.0	102.0	83.0	80.0	102.0	102.0	79.0	84.0	94.0	...	87.0	84.0	99.0	104.0	79.0	84.0	99.0	104.0	79.0	3.0
3	80.0	102.0	102.0	79.0	84.0	94.0	102.0	79.0	80.0	94.0	...	79.0	84.0	99.0	104.0	79.0	84.0	103.0	104.0	79.0	3.0
4	84.0	94.0	102.0	79.0	80.0	94.0	98.0	76.0	80.0	102.0	...	79.0	84.0	103.0	104.0	79.0	79.0	107.0	109.0	87.0	3.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
4430	56.0	64.0	108.0	96.0	64.0	71.0	108.0	96.0	68.0	75.0	...	92.0	66.0	83.0	108.0	96.0	66.0	87.0	104.0	89.0	5.0
4431	64.0	71.0	108.0	96.0	68.0	75.0	108.0	96.0	71.0	87.0	...	96.0	66.0	87.0	104.0	89.0	63.0	87.0	104.0	89.0	5.0
4432	68.0	75.0	108.0	96.0	71.0	87.0	108.0	88.0	71.0	91.0	...	89.0	63.0	87.0	104.0	89.0	70.0	100.0	104.0	85.0	4.0
4433	71.0	87.0	108.0	88.0	71.0	91.0	100.0	81.0	76.0	95.0	...	89.0	70.0	100.0	104.0	85.0	70.0	91.0	104.0	85.0	4.0
4434	71.0	91.0	100.0	81.0	76.0	95.0	108.0	88.0	80.0	95.0	...	85.0	70.0	91.0	104.0	85.0	63.0	91.0	100.0	81.0	4.0

4435 rows × 37 columns

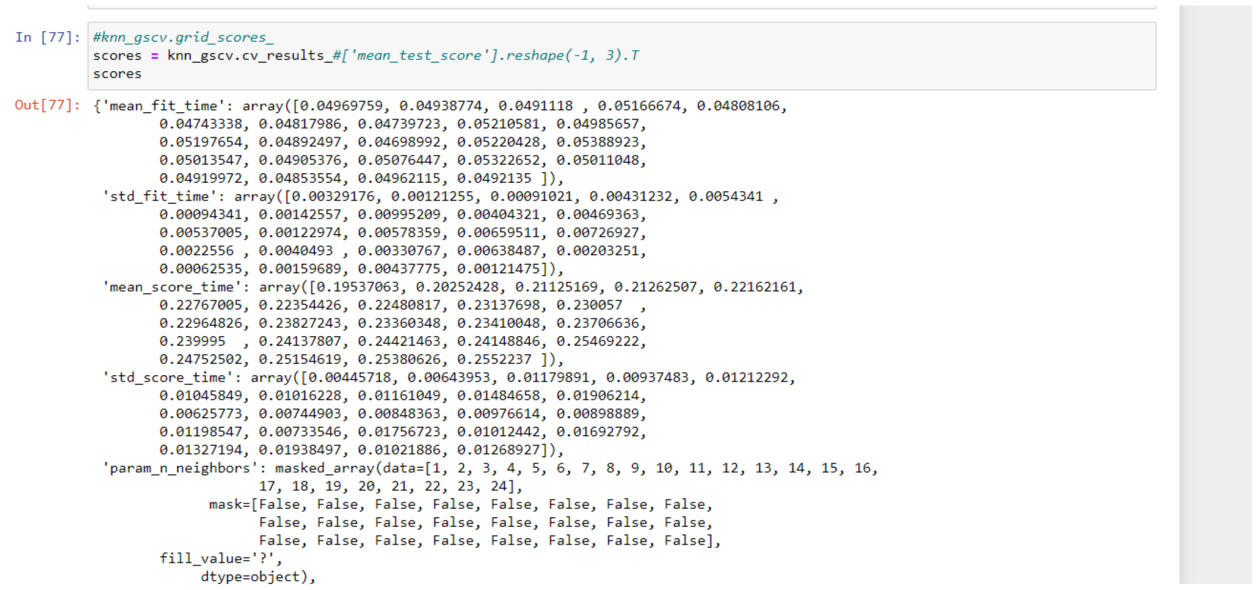
Dataset Preprocessing:
We split the dataset into training and testing using sklearn's train_test_split with the test_size set to 0.3 indicating 30% of the dataset for testing.



```
In [13]: from sklearn.model_selection import train_test_split
         X_train, X_test, y_train, y_test = train_test_split(dff,y,test_size=0.3,random_state=1,stratify = y)
```

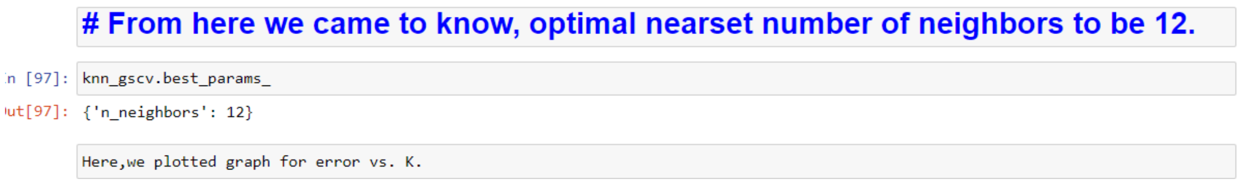
Grid Search Method for determining nearest number of neighbors:

Here, currently we don't know the value of what K is to be taken, so we started trying with K=5, but after applying Grid search method, which basically works on some range of K(given by us) an checks for mean_score_time, mean_fit_time and other things and looking into all those factors it finally tells us the optimal number of k which we can use in our dataset.



```
In [77]: #knn_gscv.grid_scores_
         scores = knn_gscv.cv_results_#['mean_test_score'].reshape(-1, 3).T
         scores

Out[77]: {'mean_fit_time': array([0.04969759, 0.04938774, 0.0491118 , 0.05166674, 0.04808106,
                 0.04743338, 0.04817986, 0.04739723, 0.05210581, 0.04985657,
                 0.05197654, 0.04892497, 0.04698992, 0.05220428, 0.05388923,
                 0.05013547, 0.04905376, 0.05076447, 0.05322652, 0.05011048,
                 0.04919972, 0.04853554, 0.04962115, 0.0492135 ]),
          'std_fit_time': array([0.00329176, 0.00121255, 0.00091021, 0.00431232, 0.0054341 ,
                 0.00094341, 0.00142557, 0.00995209, 0.00404321, 0.00469363,
                 0.00537005, 0.00122974, 0.00578359, 0.00659511, 0.00726927,
                 0.0022556 , 0.0040493 , 0.00330767, 0.00638487, 0.00203251,
                 0.00062535, 0.00159689, 0.00437775, 0.00121475]),
          'mean_score_time': array([0.19537063, 0.20252428, 0.21125169, 0.21262507, 0.22162161,
                 0.22767005, 0.22354426, 0.22480817, 0.23137698, 0.230057  ,
                 0.22964826, 0.23827243, 0.23360348, 0.23410048, 0.23706636,
                 0.239995  , 0.24137807, 0.24421463, 0.24148846, 0.25469222,
                 0.24752502, 0.25154619, 0.25380626, 0.2552237 ]),
          'std_score_time': array([0.00445718, 0.00643953, 0.01179891, 0.00937483, 0.01212292,
                 0.01045849, 0.01016228, 0.01161049, 0.01484658, 0.01906214,
                 0.00625773, 0.00744903, 0.00848363, 0.00976614, 0.00898889,
                 0.01198547, 0.00733546, 0.01756723, 0.01012442, 0.01692792,
                 0.01327194, 0.01938497, 0.01021886, 0.01268927]),
          'param_n_neighbors': masked_array(data=[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16,
                             17, 18, 19, 20, 21, 22, 23, 24],
                       mask=[False, False, False, False, False, False, False, False,
                             False, False, False, False, False, False, False, False,
                             False, False, False, False, False, False, False, False],
                 fill_value='?',
                      dtype=object),
```



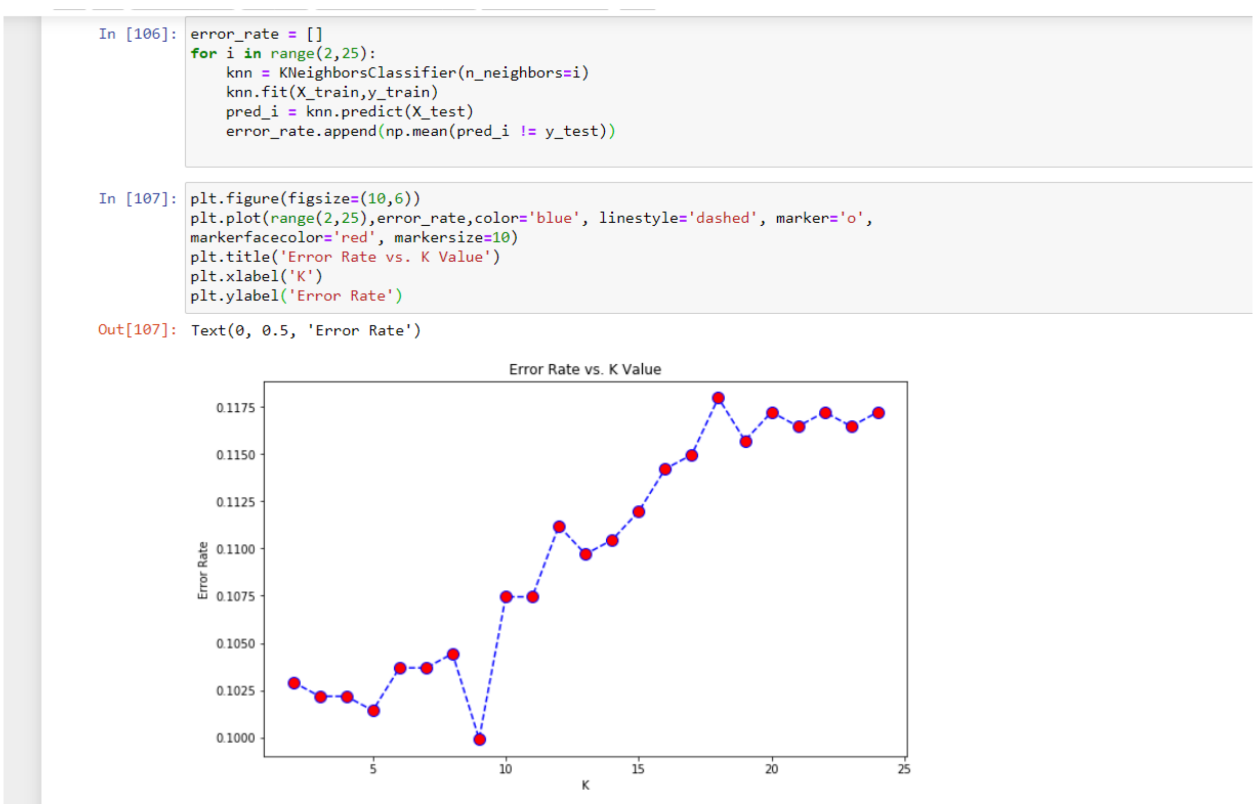
From here we came to know, optimal nearset number of neighbors to be 12.

```
n [97]: knn_gscv.best_params_

ut[97]: {'n_neighbors': 12}
```

```
Here,we plotted graph for error vs. K.
```

After this, we need to print graph for error vs, value of K, so we have to perform iterations for that shown as:



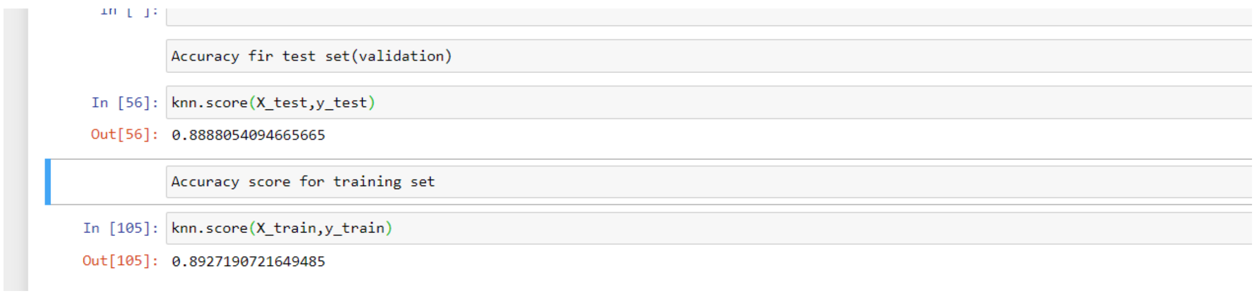
```
In [106]: error_rate = []
          for i in range(2,25):
              knn = KNeighborsClassifier(n_neighbors=i)
              knn.fit(X_train,y_train)
              pred_i = knn.predict(X_test)
              error_rate.append(np.mean(pred_i != y_test))

In [107]: plt.figure(figsize=(10,6))
          plt.plot(range(2,25),error_rate,color='blue', linestyle='dashed', marker='o',
          markerfacecolor='red', markersize=10)
          plt.title('Error Rate vs. K Value')
          plt.xlabel('K')
          plt.ylabel('Error Rate')

Out[107]: Text(0, 0.5, 'Error Rate')
```

Accuracy:

Accuracies for both Training dataset and Validation Dataset with confusion matrix:



```
In [ ]:

          Accuracy fir test set(validation)

In [56]:  knn.score(X_test,y_test)

Out[56]:  0.8888054094665665

          Accuracy score for training set

In [105]: knn.score(X_train,y_train)

Out[105]: 0.8927190721649485
```