Headers

Name:

Major resources :

1. Major Libraries: pandas , numpy , scikit-learn
2. Websites: stack overflow

Programming language: python 2.7

Dataset details

1. pima-indians-diabetes-database

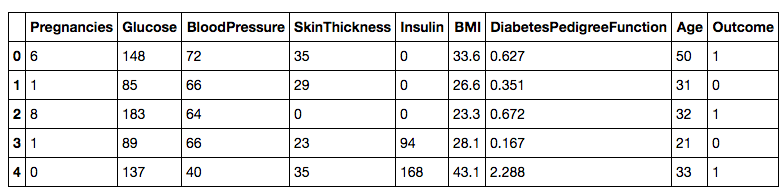
Sample data is like below Fig1. This dataset has eight features and one label. 

Fig1

The distribution of the label is fig2.You can tell from the graph that the ratio between label 1 and label 0 is nearly 0.5.

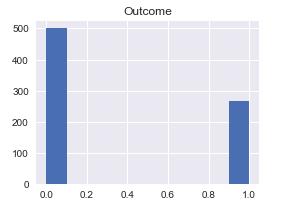


Fig2

The distribution of each feature is listed below in Fig3

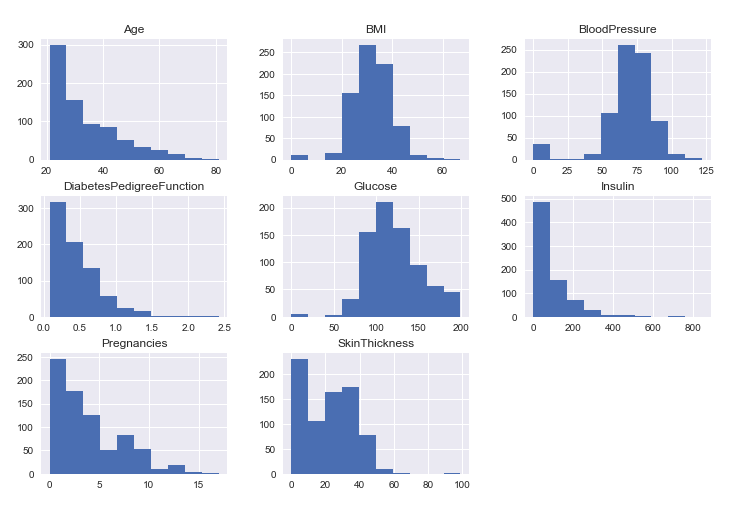


Fig 3

Data split strategy :

The whole dataset has 768 rows which means the number of the entities is 768.

The ratio between training set and testing set is 4 : 1.

According to the slides. It suggests to make the ratio during training set, validation set, test set is 0.8:0.1:0.1. Since this dataset is really small. I skip the validation set and make the training and testing 4:1.

1. The MNIST dataset

This dataset has 784 features and 1 label. And the range of each feature is (0,1). The number of the feature equals to the square of 28 which means each row represents a graph of label number.

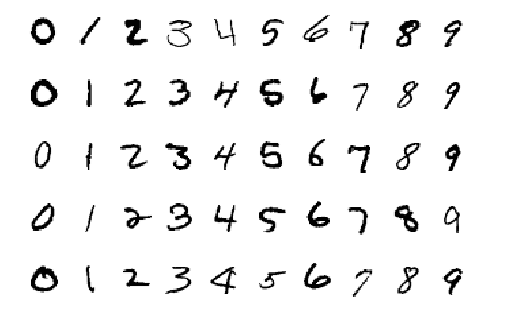


Fig 4

Fig4 is the graph of the data sample from the dataset. And the distribution of the labels is listed below. The number of each kind of the label is nearly equal. So we can take this dataset as the “balanced dataset”.

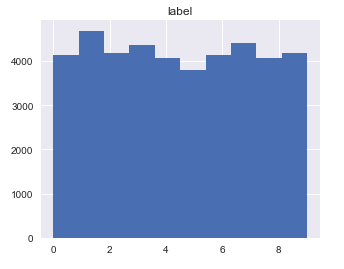


Fig5

Data split strategy :

Since this data has already been split into train set and test set, we won’t split the data anymore.

Algorithm Description:

For dataset ‘pima”, each feature has the different value scope. So I applied “StandardScaler” and “MinMaxScaler” into the features to make every feature has the same range. I import these Scalers from the scikit-learn library. I’ve compared each scaler and found that the

“MinMaxScaler” performed better than “StandardScaler”.

As for the MNIST dataset, each feature has the same range. So I didn’t apply the scaler into them. And I’ve also checked that both datasets didn’t have the missing data.

Algorithm Results:

Pima dataset:

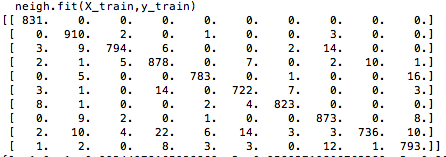
|  |  |  |
| --- | --- | --- |
|  | Predicted True | Predicted False |
| Actual True | 58 | 15 |
| Actual False | 3 | 132 |

Accuracy = (TP+TN) / (TP+NP+TN+FN) = 91.34%

In the code , you can try “MinMaxScaler”, in this small dataset , it can gives 100% accuracy. It’s up to u to decide whether put the 100% result on the table.

MNIST dataset:

Terminal screen shot Remember to delete it



|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| 0 | 831 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 910 | 2 | 0 | 1 | 0 | 0 | 3 | 0 | 0 |
| 2 | 3 | 9 | 794 | 6 | 0 | 0 | 2 | 14 | 0 | 0 |
| 3 | 2 | 1 | 5 | 878 | 0 | 7 | 0 | 2 | 10 | 1 |
| 4 | 0 | 5 | 0 | 0 | 783 | 722 | 7 | 0 | 0 | 3 |
| 5 | 3 | 1 | 0 | 14 | 0 | 722 | 7 | 0 | 0 | 3 |
| 6 | 8 | 1 | 0 | 0 | 2 | 4 | 823 | 873 | 0 | 8 |
| 7 | 0 | 9 | 2 | 0 | 1 | 0 | 0 | 873 | 0 | 8 |
| 8 | 2 | 10 | 4 | 22 | 6 | 14 | 3 | 3 | 736 | 10 |
| 9 | 1 | 2 | 0 | 8 | 3 | 3 | 0 | 12 | 1 | 793 |

Accuracy:



You can pick one as the Accuracy , up to you , remember to delete the screenshot

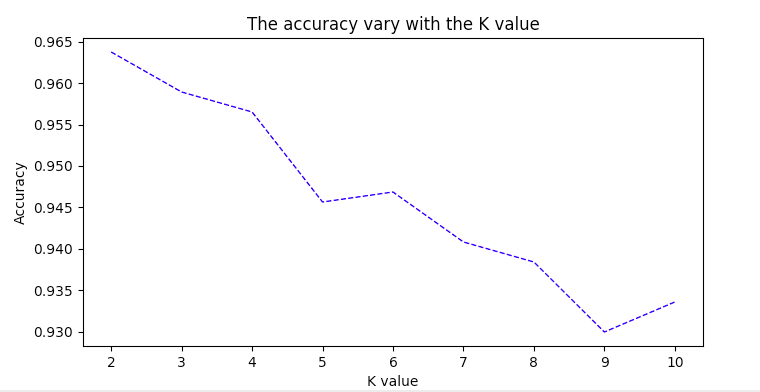


Fig 6

Fig 6 is the graph of the accuracy of number “2” varying with the K value.

Runtime

The key pseudocode is like below

For each item “i” in testing dataset:

Initialize distance\_dict dictionary

For each item “j” in training dataset:

Calculate Euclid distance between item “ I” and item “j”

Store the data in the distance\_dict

Sort the dictionary to get the items “j”s with the top K distance value

Get the major votes from the “K” return values.

Roughly, the running time for this algorithm is O(M\*N). M is the number of training dataset. N is number of testing dataset.

The running time of KNN on Pimi dataset:

1.0974 s

Due to the large number in the training and test dataset , it’s really hard to finish the implemented KNN on the MNIST dataset in couple hours. Instead, I used the KNN algorithm from the scikit-learn library.

The link is

<http://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsRegressor.html>

The running time of KNN on MNIST dataset:

432.439 s