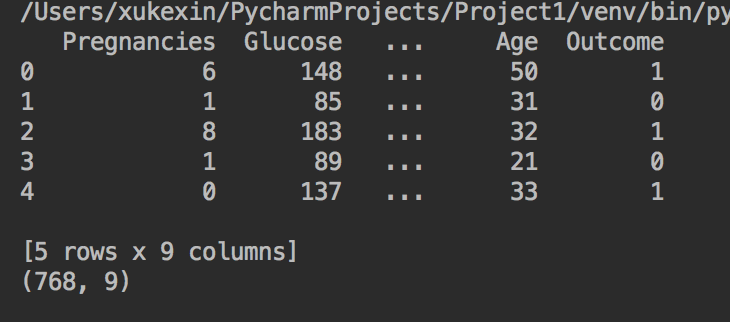
Project 1 : KNN

KNN with PIMA Indians Diabetes Dataset

# Dataset details:

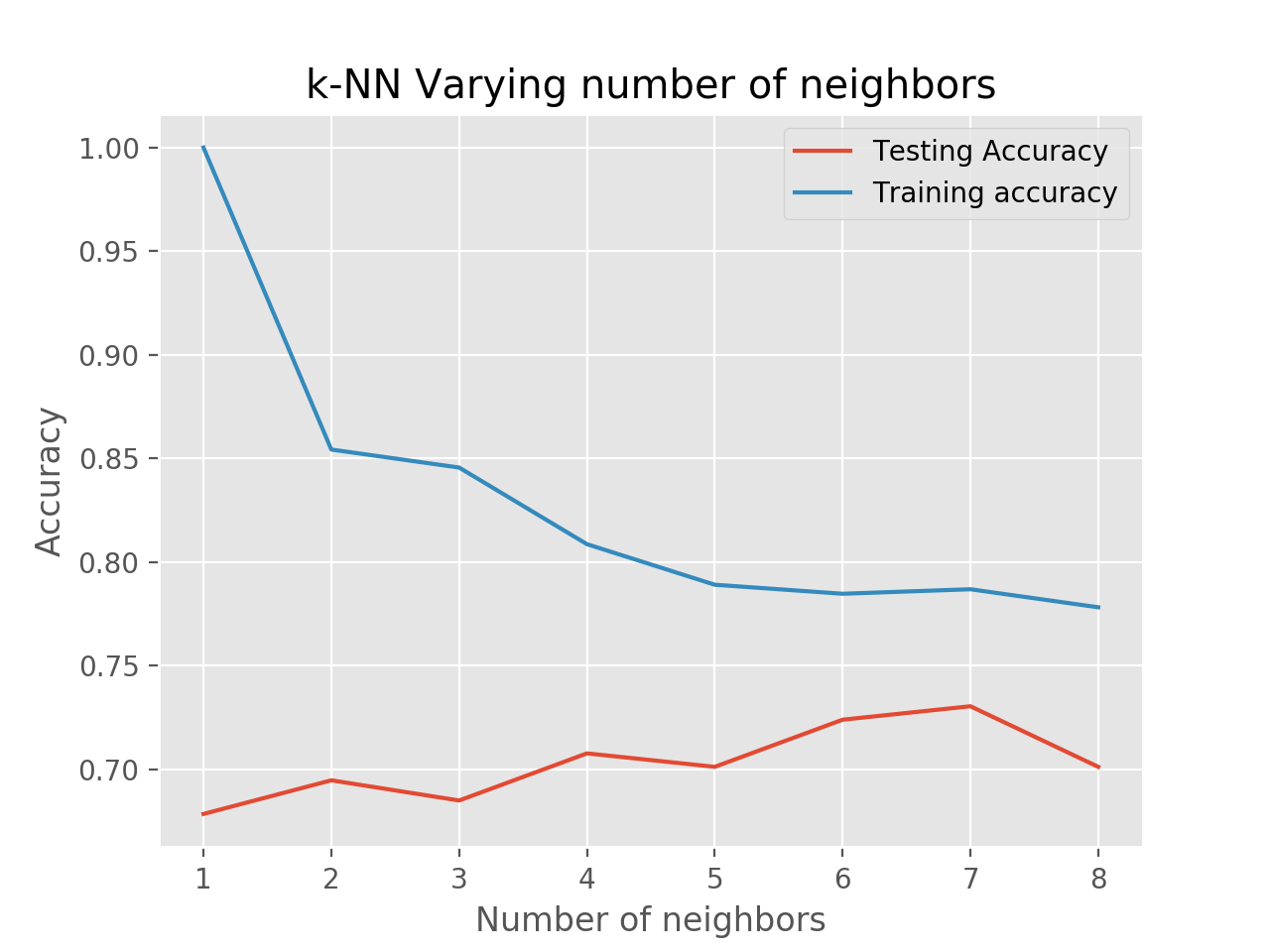
First, after we read the data from “diabetes.csv”, we got a dataset with 768 rows and 9 columns. From our obervation, the first 8 columns represent the features and the last column represent the label(have diabetes or not).



We split dataset with scikit-learn's train\_test\_split method. 40% of the dataset for the test set.

# Algorithm Description:

Then we created the classifier. After Compared accuracies with different values of K we set k =7 because from image we can tell that we get maximum test accuracy when k =7.



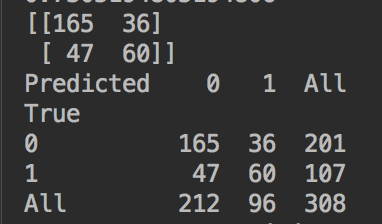
# Algorithm Description:

Accurancy: 

Also, we want tables showing false positive, false negative, true positive and true negatives.

Method 1: We built a Confusion Matrix with Scikit-learn's confusion\_matrix method. It is very handy.

Method 2: Pandas also has crosstab method which we can use to create the confusion matrix.



# Runtime:

Because we use the simplest enumerate algorithms to fix this problem, the time complexity is O(DN),

If we want to optimize it, we can construct the training data into K-dimensional tree, then the time will reduce to O(D\*log(N)).

“Wall-Clock” is about 2 seconds.

**Handwritten numbers**

**I switched to use anaconda to compile because there are some functions cannot work under PyCharm.**

# Dataset details:

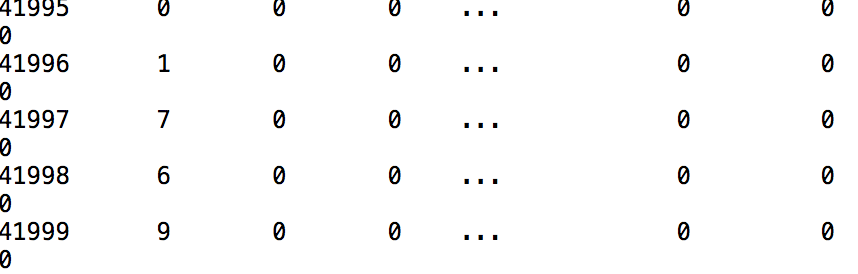
The data files contain gray-scale images of hand-drawn digits, from zero through nine.

Pre-Processing: From my point of view, this image’s background noise is really low, so I didn’t do the pre-processing. We still can do background noise remove and sharpening steps if we need with ImageMagick Library, but it will be cost-efficiently.

Train.csv has pixel intensity values as flattened vectors for 42000 images and their corresponding labels.

Test.csv has pixel intensity values for 28000 unlabeled images.

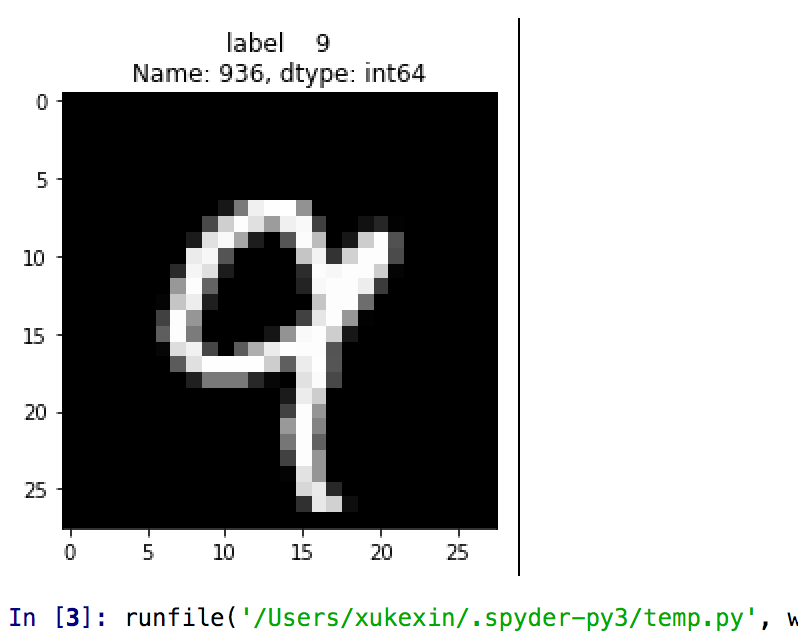
We open the train.csv and as we thought there are 42000 groups of data.



We can see the first column is the label column, and the remaining 784 columns are the intensity of each pixels of 28 X 28 machines.

We separate the training and testing dataset. I prefer 20% for testing and 80% for training. Still we used Train\_test\_split method.

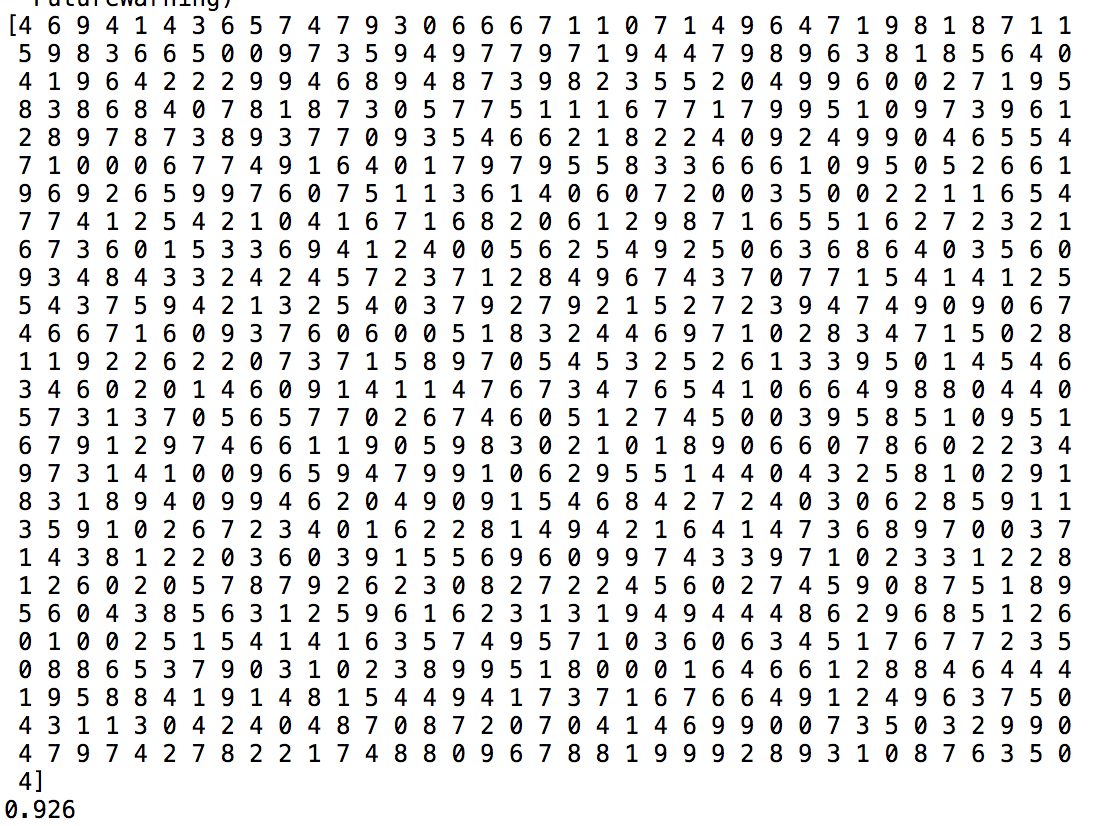
This one sample of the training dataset.



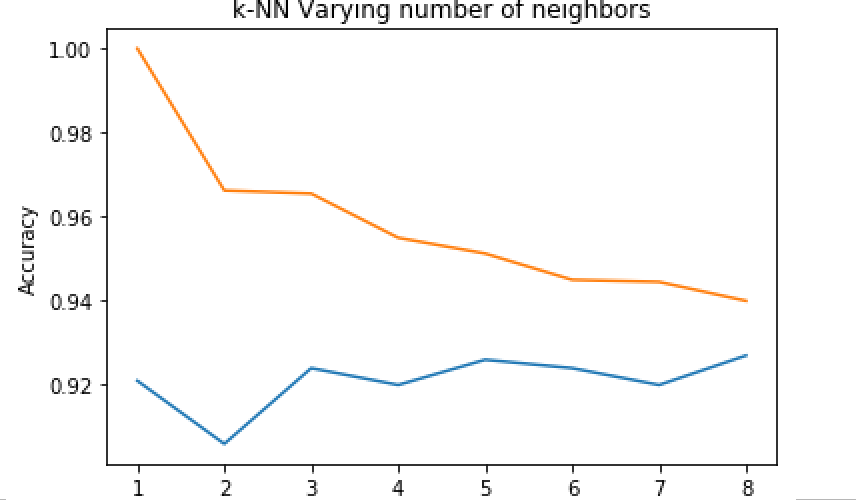
# Algorithm Description

Then， we imported KNeighborsClassifier from sklearn library. (the easiest way I found) Also， it’s accuracy is really high, even though it is time-costing.

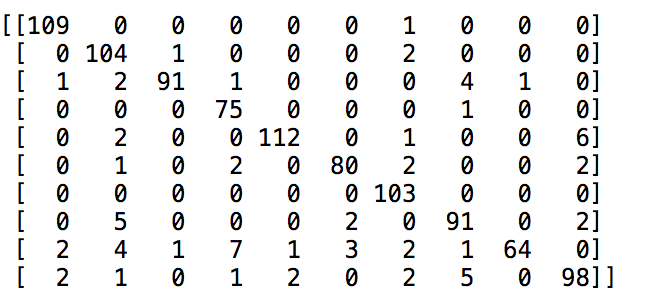
# Algorithm Result



Use the same method , we get the accuracy varies as a function of K with label 9 .



Confusion matrix :



Runtime: O(D\*log(N))

Wall-Clock : each K will cost 15 seconds to get the prediction. Others will cost about 5 seconds.