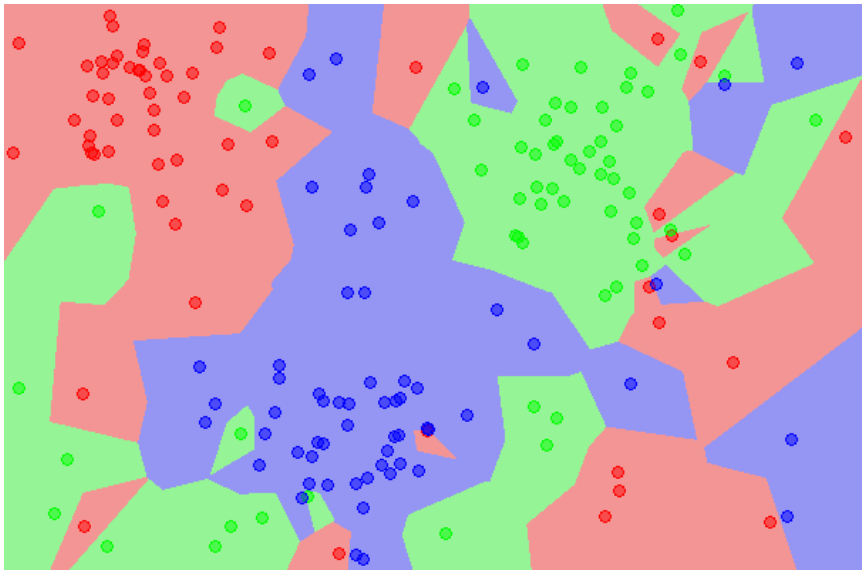
K-Nearest Neighbour (kNN) for classification and prediction

COMP9417 Machine Learning Project

Tianwei Zhu, z5140081; Haoxiang Zhao, z5084093

Introduction

The k-Nearest Neighbour (kNN) is one of the most popular classification and prediction algorithm for machine learning and data mining. In k-NN classification, the output is a class membership. A sample is classified by a majority vote of its neighbors, with the object being assigned to the class most common among its k nearest neighbors. KNN is very easy to understand and implement, which makes it been widely used in scientific study and commercial management. It is especially suitable for multi-model classification and does a better job than SVM in this case.



1-KNN graph from Wikipedia

Related Work

Classification of radar returns from the ionosphere using k-Nearest Neighbour algorithm. This radar data was collected by a system in Goose Bay, Labrador. The targets were free electrons in the ionosphere. "Good" radar returns are those showing evidence of some type of structure in the ionosphere; "Bad" returns are those that do not. Hence, we are here to implement kNN to classify unknown result into "g" or "b" using current dataset.

Instance-based prediction of real-valued attributes. There are 26 different attributes for each car and we need to predicted price of car using all numeric and Boolean attributes. As it is very normal that some attributes can be lost due to some reasons, in this situation, predicting missing data can be very useful in real world.

Implementation

The main idea of kNN is very simple. First, calculate the distance between the samples to be classified and the training samples of known categories, and find k neighbors closest to the data of the samples to be classified. Then, the category of the sample data to be classified is determined according to the category of these neighbors.

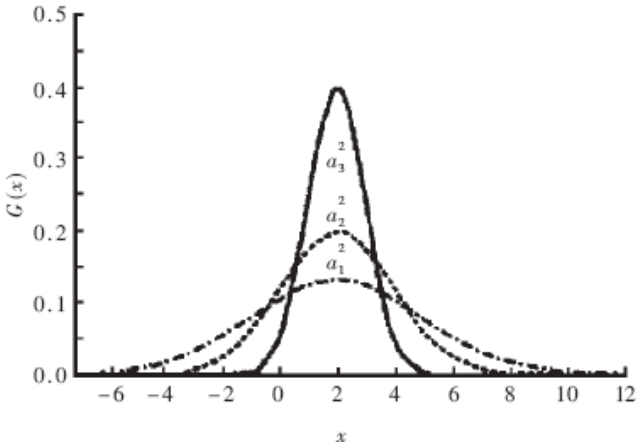
When measuring the distance between two points in space, the greater the distance, the less similar the two points are. There are many options for distance such as Minkowski-Distance, Euclidean-Metric and Manhattan-Distance. In this project, we used Euclidean-Distance to classify and predict test cases:

$$d_{euc}(x,y) = \sqrt{\sum_{k=1}^{d}(x_j - y_j)^2}$$

For the first part of classification, we use the standard UCI data set "ionosphere" to test our program and also evaluate the system by leave-one-out cross-validation. We have designed both non-weighted classify and weighted one to produce results:

- In non-weighted mode, count each class in the k nearest points, then pick the class which has highest votes. It is possible that two classes have same number of votes when k is set to even number. In this situation, we simply choose "good" for the result. We should notice that classification without weight doesn’ t consider values of distance, this will influence the accuracy of result.
- In weighted mode, we used Gaussian Function instead of Inverse Function because the latter allocates much higher weight for the nearer points, which means the weight will fall fast as distance becomes larger. This characteristic may case the system be sensitive with noise. On the other hand, Gaussian Function also give high weights to the close points, but distant points can still get acceptable weights. Below is the Gaussian Function formula and graph:

$$G(x) = ae^{-(x-b)^2/2c^2}$$



After classifying last 151 test data sets with first 200 train data sets, leave-one-out cross-validation is used to verify the result. By splitting data set into 199 training-data and one testing-data, the system

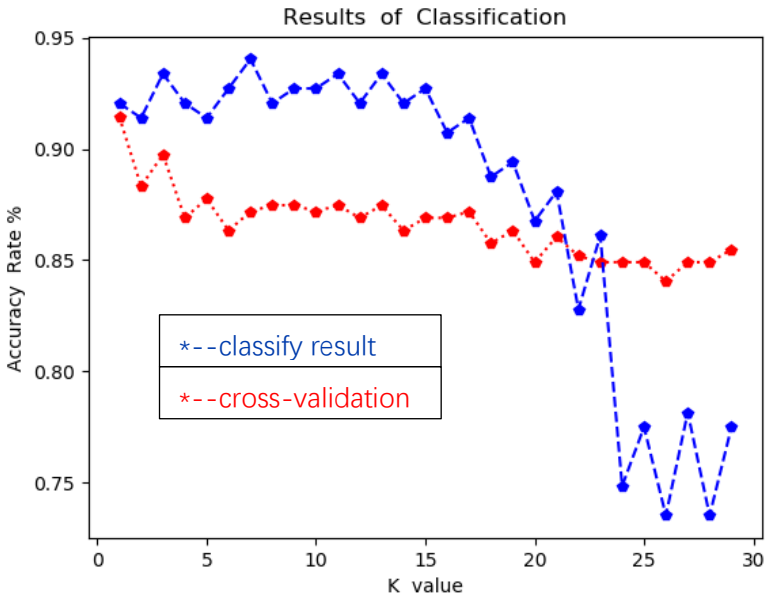
can get 200 independent classification results. We can get accuracy rate by comparing the system results with original classifications.

Part two is a KNN question for price prediction. As same as the first part, we still need to calculate "distance" between train-set points and test-set points. We have some unknown attributes in some special sample, so that we need to remove these instances from the set we want to use as training set. Then we need to assign the price attribute from each usable instance as the training class which means we select other 15 attributes as the training set and use the price to calculate mean price from first Kth nearest points to the test point (mean price here is because this is not a binary classification). We do predictions in both non-weighted mode and weighted mode as well.

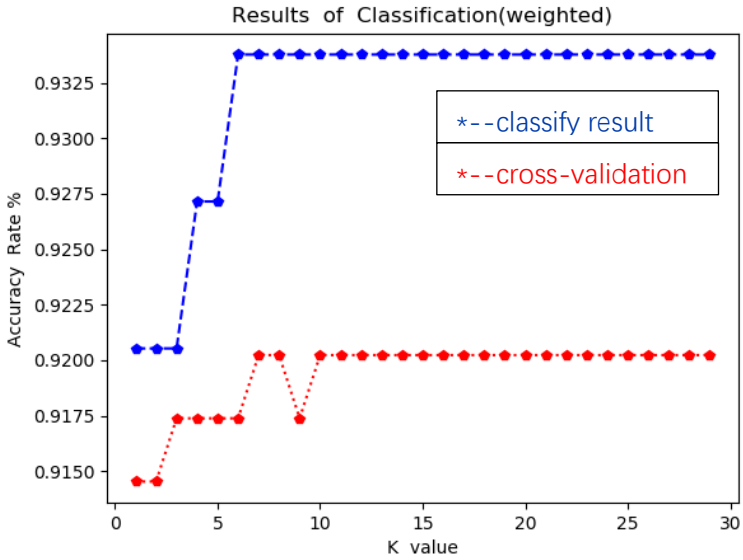
- In non-weighted part, we just calculate the mean price of all first kth nearest nodes. After calculating, we derive a predicted price without weight which is our task in this section. Percent Average Deviation Error(PADE) of prediction is a very import standard to justify the accuracy, thus in next step (leave-one-out cross-validation), we calculate PADE. However, we cannot judge the best K only depending on PADE, in this case we also calculate root-mean-square error(RMSE) which illustrates dispersion degree.
- In some circumstances, non-weighted mode would get a lower accuracy for result because of effect of a far distance point, thus there we provide a weighted mode for prediction. In weighted mode, we also calculate the PADE and RMSE. Different from classification, the price here is not the mean price from all train nodes. We choose the first nearest point as the datum point, calculate the proportion of rest k-1 points depending on this basic node, then use each price value from Kth training nodes times their proportion and then add them up, at last, we use previous result divide the sum of proportions. The result we derived is the prediction of price.

Result

Let's first look at the non-weighted classification result as below. Although the test result for 151 testing-data sets fluctuates around 0.93% rate and reaches top at k = 7, the accuracy of cross-validation shows highest value at k = 1.

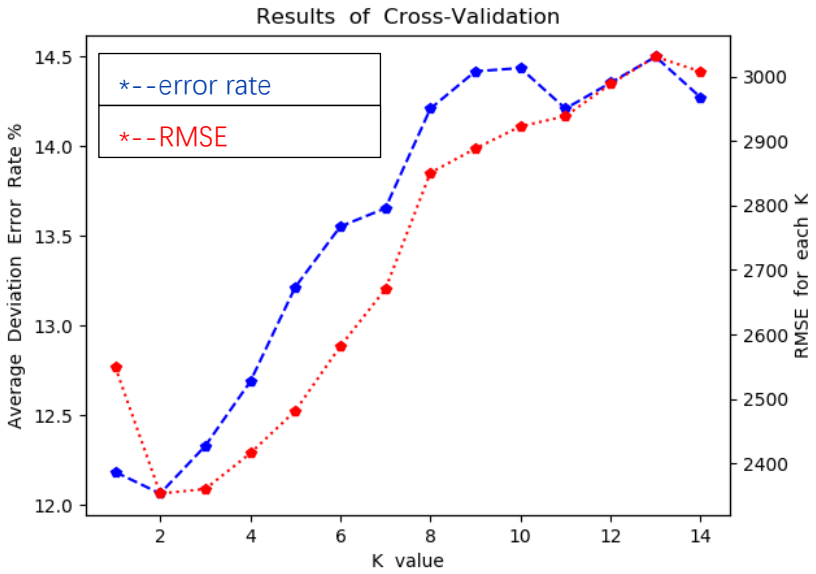


Then running the same program with weight of Gaussian Function (a=1, b=0, c=0.3) and we can get a different graph below:

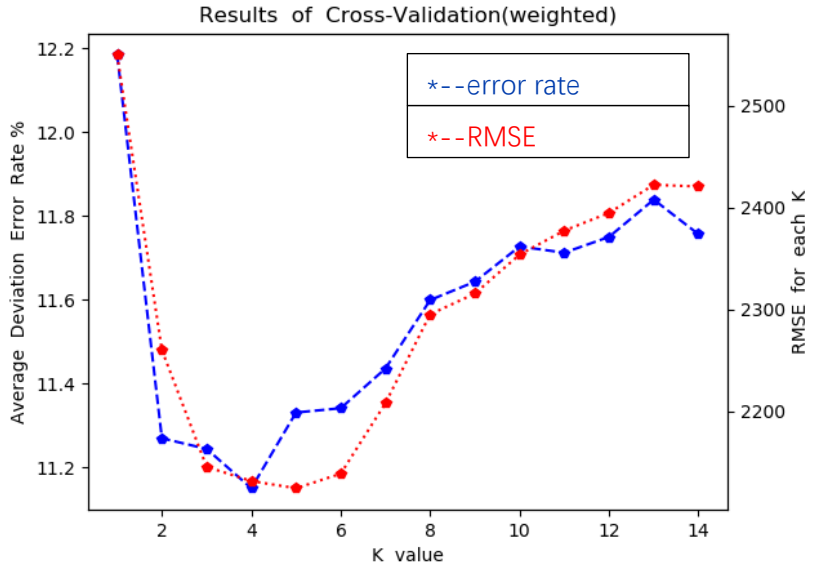


It is obvious that the classification accuracy bring into correspondence with cross-validation as k raises. On the one hand, weighted classifying dose a better job in classification accuracy than non-weighted classifying. Cross-validation is designed to pick a best k for a fixed training. The graph shows that both classification result and cross-validation reach their highest values at k = 7 (0.9337% and 0.9202%), which verified the system can do a best job by setting k to 7 (This k is only suitable for the first 200 training data set using weighted classifying).

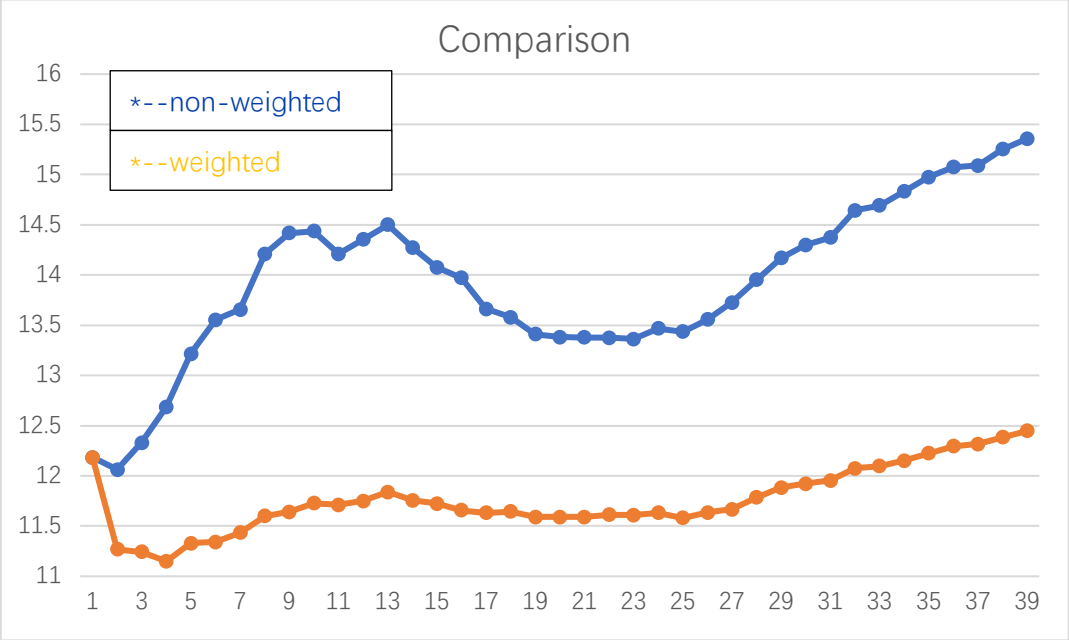
In part 2 for prediction, the graph below is the result of non-weighted mode. From the graph we can see that when k = 2, the percent average deviation rate (12.06%) and the root-mean-square error (2353.17) are both lowest, which means we get a good prediction of price which has minor deviation error.



The graph below is weighted mode result. We can find that the feasible k here is 4. This result is more reasonable than non-weighted one because k should not be too small or big as lower k means lower prediction accuracy and higher k would include some non-related data. In other words, weighted mode could improve the accuracy of classification and prediction. Also, we get a lower error rate (11.15%) and root-mean-square error (2131.29).



In order to have a clearer comparison, we have put results of these two modes into one graph as below:



Conclusion

Although k-Nearest Neighbour is one of the simplest machine learning algorithms, it still requires us to do some necessary adjustment according to the practical samples. The results above show the importance of selecting a propriate weight method in order to get a better outcome. Furthermore, we are also expected to filtrate attributes from data sets, which will reduce time complexity and raise the accuracy. To sum up, kNN is popular on classifying and predicting field, and we can compare several different machine learning algorithms to get an ideal result.