K-Nearest Neighbors

Seong-Eun Cho

November 8, 2019

Introduction

Nearest Neighbor algorithm makes one of the most basic assumptions, that is if the input features are alike or similar, then the output will most likely be similar as well. The approach is simply to choose K number of the closest points to the new input we are trying to predict, and each of the points gets to "vote" on what the new input should be based on their labels. The closeness of each point is defined by some distance metric and K is typically chosen to be 3 or 5. We can also weigh each of the votes differently according to how close the voter is to the point we are trying to predict.

The main advantage of this method is that its pretty straightforward and intuitive, so it is easy to implement. However, the method does not work very well if the relationship between different points of the same class is not well determined by a common distance metric, typically the Euclidean distance or the Manhattan distance. In this case, it might be necessary to learn a new distance metric which is a different machine learning problem itself. Also, this method is very susceptible bad features that doesn't give much information towards its output class, since the nearest neighbors approach weighs each of the input features equally. It is also possible to learn to weigh each features from the data, but that is also a whole new problem in itself. Also, the nearest neighbor approach is usually fast at train time and slow at test time. There isn't a lot going on when training the model, so most of the computation is done during test time.

1 Code Implementation

The model is initialized with the type of task (classification or regression), types of input features (real or nominal), an integer K, whether to normalize the input data between 0 and 1, and the type of weight we will incorporate for the votes. As mentioned above, there isn't much that goes on in the training time of the Nearest Neighbor model. For my implementation, the model simply takes in the data points and stores it. If the input has both nominal and continuous features, then I split the input into two separate features. Also, it is usually a good idea to normalize the continuous input features. The reason for this will be discussed in later sections.

At test time, we take our current input and calculate the distance between all of the points in the training set. For the purpose of this lab, my implementation only uses the Euclidean distance. For classification task, we find the K closest points in the training set. If we do not incorporate weights for the votes, the K value should be an odd number to prevent ties (unless the number of output classes is greater than 2, in which case it is a bit harder to prevent ties). We simply choose the class with more votes and use it as our prediction. If we do incorporate weights for the votes,

we can use the inverse of the distance so that the votes of the points that are closer are weighed more heavily.

The regression task is near identical to the classification task. We find the K closest points and we can simply average their values as our prediction. If we incorporate weights, then we can also use the inverse distance to weigh the closer values more heavily. We then sum these values and normalize by dividing by the sum of the weights. This can be summarized as:

$$\hat{f}(x_t) = \frac{\sum_{i=1}^{k} w_i f(x_i)}{\sum_{i=1}^{k} w_i}$$

$$w_i = \frac{1}{||x_t - x_i||_2}$$

where $x_1, ..., x_k$ are the K of the nearest points from x_t using the Euclidean distance.

1.1 Debug and Evaluation

The debug dataset trained on seismic-bumps_train.arff was tested on the seismic-bumps_test.arff. The model was initialized with K=3 and used inverse distance for weights and scored 93.57% which is the expected accuracy. The predictions also matched with the seismic-bump-prediction.csv file provided by the TAs. For the evaluation data, the model was trained on diabetes.arff and tested on diabetes_test.arff with the same initialization. The score was 86.59% and the resulting predictions were saved in a .csv file that is included in the folder with code.

2 Magic Telescope Dataset

The Magic Telescope data is a classification task where all of the input features are real. This is ideal for KNN models. I first tried to train and test without normalizing the data with the KNN model initialized with the specified parameters (K=3, no distance weight) and got an accuracy of 80.83%. After normalizing all of the input features to be between 0 and 1, I got 83.06% with the same parameters. For this task and in most cases, normalizing the dataset will improve the performance of the model. Without normalization, some of the features can be in a much greater (or lower) scale which can have too much (or negligible) effect when the distance is calculated. For example, if one of the feature has values ranging between 0 to 1 while another feature has values ranging between 0 to 1000, then the latter feature will have about 1000 times more effect than the former.

In order to find the best K for this task, I trained independent models with odd K values ranging between 1 and 15. The table below shows the results obtained for each value of K.

Best K value for the Magic Telescope Data

K	1	3	5	7	9	11	13	15
accuracy	81.14	83.06	84.44	84.38	84.46	84.62	84.65	84.29

The best result was obtained when K=13 with accuracy of 84.65%.

3 Housing Dataset

The Housing data is a regression task. I followed the same procedure as was done for the Magic Telescope data, except for the fact that it's a regression problem, and because it is, we can't use accuracy to measure the score of a model. Instead, for classification tasks, we use the MSE error to calculate the scores. We normalize the input values, but we don't normalize the labels. This is mainly because normalizing the output values doesn't add anything when searching for the nearest neighbors since the outputs are not accounted for when calculating how far away each of the points are. The baseline score with K=3 was 16.60. We searched for the best K by following the same procedure as above. The below table shows the MSE for each K.

Best K value for the Housing Data

K	1	3	5	7	9	11	13	15
MSE	24.61	16.60	15.90	18.97	20.66	23.50	24.51	24.28

The best result was obtained when K=5 with accuracy of 15.90.

4 Distance Weight

I repeated the experiment for the Telescope and Housing data but this time using inverse distance for weights. This greatly improved the performance of the Housing data but less so with the Magic Telescope data. The below table shows the results for each value of K for both datasets.

Best K value for the Housing Data

K	1	3	5	7	9	11	13	15
Magic Telescope accuracy	81.14	83.11	84.56	84.91	84.85	85.28	85.16	85.07
Housing MSE	24.61	16.36	12.12	10.74	11.35	11.55	11.66	11.88

The best K for Magic Telescope is 11 with 85.28% accuracy and 7 with 10.74 MSE error for Housing.

5 Credit Approval Dataset

The Credit Approval is a binary classification task that introduces more challenge for the nearest neighbor algorithm; the input features are mixed with both continuous and nominal features, making it difficult to have a good distance measure that takes into consideration of both types of features. There are also many missing values which has to be handled in some way when calculating the distance.

First, in order to deal nominal features, we simply add up the number of features that does not match. In other words, if a nominal feature of the new input matches that of a training data, then we would assign 0 as the distance for that feature. If not, we would assign 1. There are a total of 9 nominal features in the data, so the maximum distance of the nominal features would be 9. We then calculate the Euclidean distance of the continuous features the same way, except we don't

take the square root of it yet. We add the square Euclidean distance with the nominal distance and then take the square root. Mathematically, if x and y are two instances of this data and x_c, y_c is only the continuous values of x, y and x_n, y_n is the nominal values of x, y the distance between these two points is calculated as:

$$\delta(x, y) = \sqrt{||x_c - y_c||_2^2 + \delta_n(x_n, y_n)}$$

$$\delta_n(x_n, y_n) = \text{the number of dissimilar features between } x_n \text{ and } y_n$$

In order to deal with the missing values, instead of imputing new values for these data, we handle them at test time when measuring the distance. We impose that if a feature of a data point is missing, then the distance between that data and any other data point w.r.t. that feature will always be the maximal distance, which is 1 when the data is normalized.

I split the data into train and test sets using Scikit-learn's implementation using 70-30 splits. I searched for the best K using the same technique for the above two sections and found that K=13 worked best for this task. The score I got using this K is 86.47%.

6 Scikit-learn's Implementation

Using the Scikit-learn's implementation of K-Nearest Neighbor Classifier and Regressor, I repeated the same experiment for the Magic Telescope and Housing data. To my surprise, both of the tasks did better when using the Manhattan distance rather than Euclidean distance. The computation was much faster than my implementation but the performance didn't see much improvement. Similar values for K was the best using their implementation as well. For the Magic Telescope data, the best score I got was 85.18% using K=11 and Manhattan distance. This is worse than the best result I got from my own implementation. For the Housing regression task, the scoring metric was different so I was not able to compare the performance with my model. They use R^2 value for their metric which ranges between 0 and 1 instead of the MSE error. The best score I got was 0.8579 (higher is better) with K=5 and also using the Manhattan distance.

7 Human Activity Recognition Using Smartphones Data Set

The dataset I used for the final task is the Human Activity Recognition task. I obtained the dataset from the UCI dataset repository. This is a high dimensional classification task with over 500 features, which seemed appropriate for this part. There are slightly over 10,000 data points split into train-test sets with around 70-30 ratio. Each of the input feature is real-valued and is already normalized to be between -1 and 1. Each represent some sort of a movement or position that can be detected by Smartphones. There are 6 output labels, each label representing a certain task. The best accuracy when trained on the full input features is 90.63%.

In order to reduce the dimension of the data, I first used PCA. After obtaining the singular values of the data, I can determine how many components would appropriately contain enough percentage of the variance. I concluded that 34 components would be sufficient since 34 principle components contains just over 90% of the total variance of the data. After transforming the data to be 34 dimensional, I tried training the model again using KNN and got a score of 89.14%.

I then tried using greedy wrapper approach to reduce the dimensionality. The approach is very simple: I first start with empty set of features and add 1 feature at a time that improves my

score the most. In order to do this, I first have to define how I want my model to look. I used 11 neighbors with distance weight and tried using both the Manhattan distance and the Euclidean distance to see which gave me a better result. Both of the models performed very well, better than using all 500 features. This was surprising to me at first, but it made sense since many of the features probably aren't adding any information for determining the activity being performed and are simply adding noise to the data. This comes back to one of the fundamental weaknesses of the KNN approach, which is that it will always weigh every feature equally, which can be a very poor choice at times. By only using the most relevant features, I am able to obtain a much better accuracy. The best score I got was by using Euclidean distance with 30 features which resulted in a score of 95.01%, which is nearly 5% better than using all of the features.