Classification of Income using KNN Classifier

Yashwanth Kumar Pamidimukkala

Department of Integrated Systems Engineering, The Ohio State University, Columbus, 43201, pamidimukkala.3@osu.edu

1. Introduction

[2] k-nearest neighboring algorithm is a non – parametric method, means it makes no explicit assumptions about the functional form of the training data used for classification. Also, it is instance based which means that this algorithm explicitly doesn't learn the data, instead it memorizes the training instances which are subsequently used as knowledge for the prediction phase.

2. Implementing the KNN Classifier:

We will implement the Knn classifier following the algorithm. The algorithm is to compute the distance D between a single test instance and a set of training instances. Suppose z = (x,y) is the test instance, we will compute the distance D(x, X) between z and every training example (X,Y). Based on the value of k we choose, we will sort k training examples which are neighbors to z i.e. $D_z \subseteq D$. Once we have our k closest examples, we will determine the class from the nearest neighbors using two methods i.e. 1. We will take majority vote i.e. threshold posterior probability will be greater than 0.5 and 2. we will compute the weighted distances for each class. We will then calculate how accurately both these computations are predicting the test data. We will pick whichever computation provides higher accuracy by varying values of k.

Initially, we used the following distance metrics for the data: 1. Manhattan Distance for continuous attributes i.e. age, hour_per_week and fnlwgt. 2. Nominal attributes will have $if\ p = q => d = 0\ or\ if\ p \neq q, d = 1$ and the nominal attributes in our data set are workclass, marital_status, occupation, relationship, race, gender, native country (In our previous assignment we have used jaccard similarity to calculate their similarities. However, for this assignment to keep the application simple and easy to implement, we will treat them both as nominal attributes which the above metric for nominal data will apply. 3. Ordinal attribute education_cat is being mapped to values in between 0 to 1 with equidistance.

With these metrics, we will calculate the distances, sort them and we will obtain a matrix of size based on the value of k. From this matrix, we can calculate the weighted distances or take the majority vote of a class i.e. if majority of the k nearest neighbors are '<=50' we will predict that the test set belongs to '<=50' class. Also, we implemented the Euclidian distance on three attributes by weighing the attributes equally. This gave a lower accuracy score and the problem we faced with the Euclidian distance was it is computationally slow. So, we decided to drop the Euclidian distance or not based on the ROC curve.

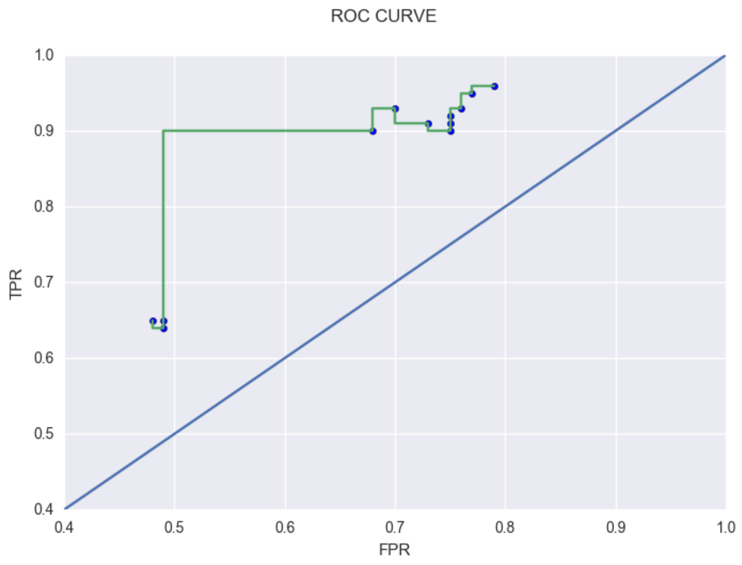
3. Performance of Classifier:

We computed the weighted posterior to determine the class along with the majority vote (probability of class in k-neighbors to be greater than 50%). The value of k which we will initially begin with is k = 23 which is an odd number close to the square root of the total number of training examples. This was a rule of thumb suggested by many online resources.

Starting with k = 23, Manhattan proximity function for continuous attributes, nominal function $if\ p = q => d = 0\ or\ if\ p \neq q, d = 1$ for nominal attributes, and ordinal function for ordinal attributes were applied to the data. Later, we varied the value of k, proximity functions and the class predictor function. For each variation True Positive Rate(TPR), False Positive Rate(FPR), False Negative Rate(FNR), True Negative Rate(TNR), Precision(p), Recall(r) and F-measures(F) are calculated. ROC – curve is then plotted for these various values using the TPR and FPR values.

Table 1: Performance Measures of KNN Classifier

k	TPR	TNR	FPR	FNR	Precision	Recall	F -measure	Method	Proximity
23	0.9	0.31	0.68	0.09	0.81	0.9	0.85	Weighted	Manhattan
23	0.93	0.298	0.701	0.067	0.81	0.93	0.87	Unweighted	Manhattan
33	0.9	0.25	0.75	0.09	0.8	0.9	0.85	Weighted	Manhattan
33	0.95	0.22	0.77	0.05	0.8	0.95	0.87	Unweighted	Manhattan
33	0.64	0.51	0.49	0.36	0.81	0.64	0.71	Weighted	Euclidian
33	0.92	0.25	0.75	0.08	0.8	0.92	0.85	Unweighted	Euclidian
23	0.65	0.52	0.48	0.35	0.82	0.65	0.72	Weighted	Euclidian
23	0.91	0.25	0.75	0.09	0.8	0.92	0.85	Unweighted	Euclidian
43	0.65	0.51	0.49	0.35	0.81	0.65	0.72	Weighted	Euclidian
43	0.93	0.24	0.76	0.07	0.8	0.93	0.86	Unweighted	Euclidian
43	0.91	0.27	0.73	0.09	0.8	0.91	0.85	Weighted	Manhattan
43	0.96	0.2	0.79	0.04	0.8	0.96	0.87	Unweighted	Manhattan



Above is the ROC curve for the various values of TPR and FPR. From this curve, it seems that at FPR = 0.7 and TPR = 0.93, the curve seems to have higher lift. So, we can choose the parameters which helped us achieve that lift.

Since, the above graph does not do well in interpretation, we used the *sklearn* off the shelf graph to plot the ROC curve. The TPR and FPR are obtained from our algorithm and we can plot the ROC curve by varying k and proximity function of the continuous attributes. In table 2, we have area under curve by varying the values of k and the proximity function.

Table 2: Area under curve

k	Proximity	Area Under Curve	Accuracy %(Majority vote)	Accuracy %(Weighted Distance)
11	Manhattan	0.623	78.12	75.6
17	Manhattan	0.623	78.12	75.6
23	Manhattan	0.615	78.4	76.7
29	Manhattan	0.605	78.47	76.3
35	Manhattan	0.592	78.12	75.69
11	Euclidian	0.606	77.77	75.31
17	Euclidian	0.592	77.43	75.24
23	Euclidian	0.584	76.04	75.52
29	Euclidian	0.601	77.08	75.69
35	Euclidian	0.586	76.3	75.6

It appears that our area under curve is maximum when k = 17 and proximity function used is Manhattan distance. At k = 11, area under curve is the same as when k = 17 but I penalised the tie with model which has low sensitivity and when k = 17, the sensitivity is 0.927 which is lower than sensitivity when k = 11. So we will proceed further and do further evaluations using k = 17. ROC curves are shown below for a few different variations and the area under curve is also shown.

ROC Curves for various values of k and proximity function

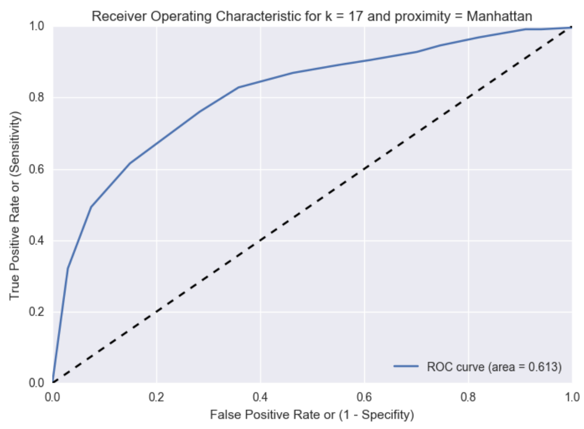


Figure 1

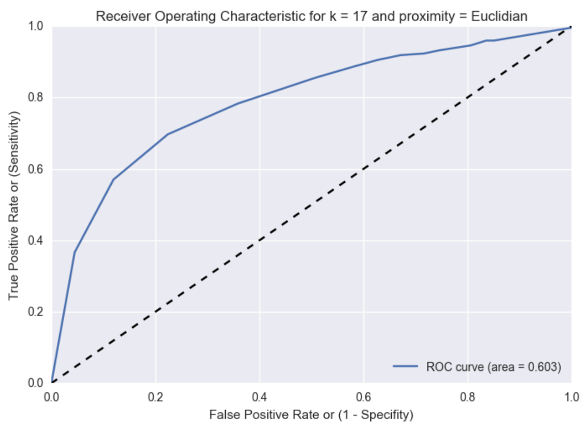


Figure 2

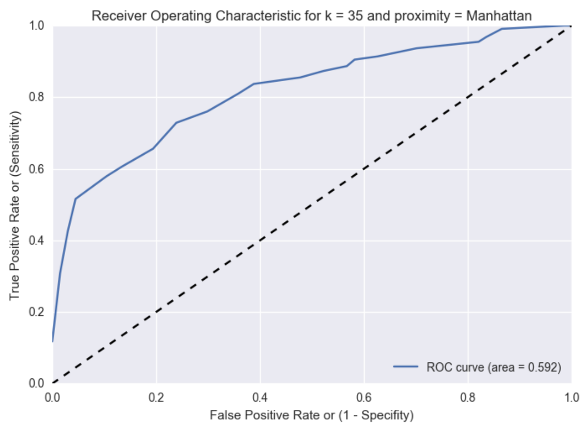


Figure 3

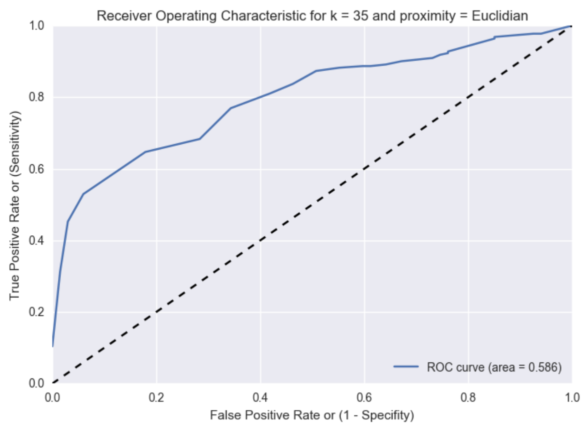


Figure 4

Performance Metrics for knn Classifier, confusion matrix and Misclassification error:

Given below are the performance metrics for a few different k values,

For k = 11:

True Positive Rate(TPR) or Sensitivity: 0.935, True Negative Rate(TNR) or specificity: 0.31, False Positive Rate(FPR): 0.69, False Negative Rate(FNR): 0.065, Precision :0.81, Recall :0.93, F-measure :0.87, Accuracy = 78.8%

$\frac{predicted}{actual}$	**<=50**	**>50**
<=50	206	15
>50	46	21

For k = 11 , missclassification error is 1 – accuracy = 1 – 0.78.8 = 0.212 which is 21.2% misclassification error

For k = 17:

True Positive Rate(TPR) or Sensitivity: 0.927, True Negative Rate(TNR) or specificity: 0.3, False Positive Rate(FPR): 0.70, False Negative Rate(FNR): 0.073, Precision :0.81, Recall :0.93, F-measure :0.87, Accuracy = 78.1%

$\frac{predicted}{actual}$	**<=50**	**>50**
<=50	205	16
>50	47	20

For k = 17 , missclassification error is 1 – accuracy = 1 – 0.781 = 0.218 which is 21.8% misclassification error

For k = 23

True Positive Rate(TPR) or Sensitivity: 0.96, True Negative Rate(TNR) or specificity: 0.2, False Positive Rate(FPR): 0.8, False Negative Rate(FNR): 0.04, Precision :0.8, Recall :0.96, F-measure :0.87, Accuracy = 78.1%

$\frac{predicted}{actual}$	**<=50**	**>50**
<=50	212	9
>50	53	14

For k = 23 , missclassification error is 1 – accuracy = 1 – 0.781 = 0.219 which has 21.9% misclassification error

For k = 103:

We will go little higher with the k value and look for specifics as we don't see any significant changes with low increments.

True Positive Rate(TPR) or Sensitivity: 1, True Negative Rate(TNR) or specificity: 0.01, False Positive Rate(FPR): 0.98, False Negative Rate(FNR): 0.0, Precision :0.7, Recall :1, F-measure :0.87, Accuracy = 77.08%

$\frac{predicted}{actual}$	**<=50**	**>50**
<=50	221	0
>50	66	1

For k = 17 , missclassification error is 1 – accuracy = 1 – 0.770 = 0.229 which is 22.9% misclassification error

As we increased the value of k, it forced all the negatives to shift towards positives i.e. it predicted everything to be positive. From our understanding, we concluded that for higher values of k we will end up predicting wrong values for class '>50'. Basically, it implies that our algorithm is predicting poorly on the income class which is greater than 50. To predict properly, we should have lower values of k.

At k = 11, 17 and 23 the perfomance seems to be stable. Further analysis can be evaluated using ROC curves to penalize between these values. These values can be seen in the graphs and tables above. The changes we made to this homework was we performed distance calculations using simple metrics rather than the complicated metrics used in the 1st assignment. Other than there was no difference between the two home works.

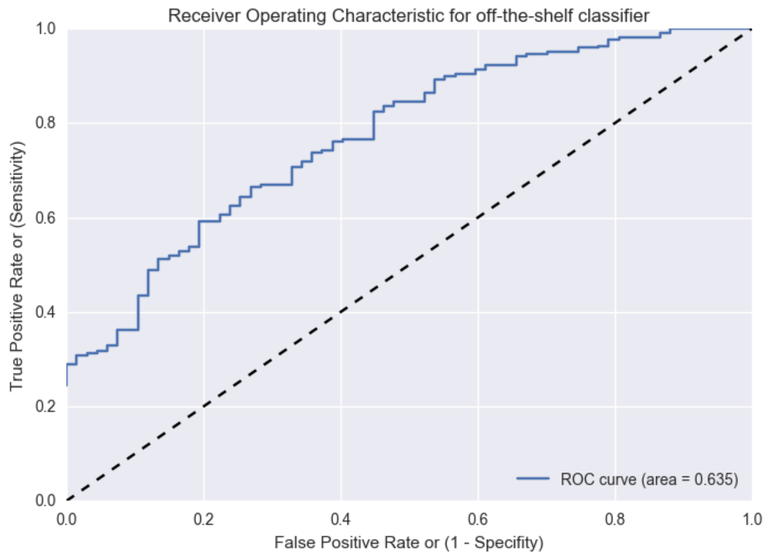
Weighted distances was implemented to predict the income class. This model seemed to be performing the same way as that of prediciting a class using majority vote. There did not seem any significant variation except for few values of k and when the distance metric for continuous attributes was calculated using Euclidian instead of Manhattan. We did most of our calculations and analysis using the majority vote.

Additionally, we ran the alogirthm on training data. This gave us an accuracy of 84.6% which is slightly higher than the accuracy on the test data for value of k = 23

4. Off-the-shelf Implementation:

We used scikit-learn package in python to implement the KNN classifier to predict the income class on test data. This model produced an accuracy of 78.5% on the test data at k = 17, uniform weights and Manhattan distance as our distance metric. We were not able to use the classifier's distance metric specifically for varying types of attributes. To proceed further, we labelled each of the nominal variables to numerical. We then plugged these values into the classifier and predicted the income class.

The accuracy was almost the same as the model we developed and predicted the classes using the majority vote. We calculated the accuracy score and the area under ROC curve for various k values, it seemed that the area under ROC curve was highest for k = 17 and p = 1(implies Manhattan distance) which is shown in the graph below.



Performance metrics:

True Positive Rate(TPR) or Sensitivity: 0.95, True Negative Rate(TNR) or specificity: 0.25, False Positive Rate(FPR): 0.75, False Negative Rate(FNR): 0.05, Precision :0.80, Recall :0.95, F-measure :0.87

Confusion Matrix:

$\frac{predicted}{actual}$	**<=50**	**>50**
<=50	209	12
>50	50	17

5. Conclusion:

In this analysis, we varied the value of k, proximity function over the continuous attributes, implemented two different methods to predict the class in the k-nearest neighbors, evaluated the algorithm on training data and implemented off-the-shelf classifier. From the analysis of ROC curve, we understood that area under ROC curve is maximum when k = 17 which implies we obtain maximum lift at this value of k and the proximity is Manhattan. Accuracy score of the model was better when we used majority vote rule.

Also, if we increase the value of k, our model is forcing all the values into the positives i.e. it is strictly predicting them to be of income class '<=50'. So, we should use lower values of k to find the k-nearest neighbors.

When we applied our algorithm on training data, we got an increased accuracy score on the test data which is 84.6%. On the other hand, off the shelf classifier gave an accuracy score of 79.8% which is close to the accuracy that we received in our model when k = 23.

Room for further improvement: Our model can perhaps be improved by varying the proximity functions used for the attributes.

Note: Only few plots for ROC were presented in this report for the sake of keeping the report simple and precise.