Homework 2: kNN Classifier & Application on Income/Iris Dataset

Yiran Cao.805

Mike Zhang.4660

February 15, 2017

1 Introduction

1.1 Overview

In Homework 1 we used two proximity measures, cosine similarity and Euclidean distance, to find the k nearest neighbors of each record in the training set. In Homework 2, we employed the same proximity measures and compute the similarity between each record in tthe raining set and each record in the test set.

We train the kNN classifier in several steps. First, for a fixed k, we vary the posterior threshold from 0 to 1 to get a combination of TPR and FPR. By saying posterior threshold, we mean the decision rule of the classifier, that if the calculated posterior of being positive is larger than the threshold, we will classify it as positive (in the $Income$ dataset, positive means class = >50k). When the threshold is equal to 0, all records in the test set will be classified as positive, which gives TPR=FPR=1. When the threshold is equal to 1, all records in the test set will be classified as negative, which gives TPR=FPR=0. When presenting the results, unless otherwise indicated, we classify the record to the class with the highest posterior. In binary classification, it's equivalent to choosing a default threshold 0.5 .

Next, we draw the ROC curve for the given k. The ROC curve always passes (0,0), corresponding to threshold equal to 1, and (1,1), corresponding to threshold equal to 0. When we put FPR on x-axis and TPR on the y-axis, the ROC should be above the diagonal line for any reasonable classifier. We calculate the area between the ROC curve and the diagonal line by approximation, which gives us the optimal lift.

We then vary k from 1 to 99, calculate the optimal lift under each k , and pick the k which gives the largest lift.

Finally, we use the optimal k and look into different sets of evaluation measures (error rate, TPR, TFR, FPR, FNR, precision, recall and F-measure) under different posterior thresholds. We pick the posterior threshold that gives the smallest error rate. This optimal k and optimal threshold together make our recommended kNN classifier.

The classifier didn't work well at first, giving an error rate of 24.3% and an FNR of 64.1%. So we modify our proximity measures to improve the performance of the kNN classifier, which indeed gives better results.

1.2 Training Process

In Homework 1, we separated the attributes of the $Income$ dataset into four groups, calculated proximity separatele for each group and combined them together. The four groups of attributes are nominal attributes, ordinal attributes, numerical attributes and asymmetric attributes.

When we first apply the proximity to the kNN classifier with $k = 5$ and posterior threshold 0.5, the performance is not as good as imagined. (Table 1.1)

k	ER	FPR	TPR	FNR	TNR	precision	recall	F-measure
5	0.243	0.122	0.358	0.642	0.878	0.471	0.358	0.407

Table 1.1: First Attempt

So we modify out proximity measure a little to get better performance. With this slight modification, the optimal k is equal to 9 using weighted posterior. The optimal posterior threshold under $k = 9$ is 0.34, and the corresponding error rate ends up with 19.4% and the rest of the evaluations are as follows.(Table 1.2)

k	ER	FPR	TPR	FNR	TNR	Recall	Precision	F-Measure
9	0.194	0.136	0.612	0.388	0.864	0.577	0.612	0.594

Table 1.2: Performance After Modification

This rest of the report is organized as follows: Section 2.1 and Section 2.2 present the classification results and evaluate the performance of the kNN classifier. Section 2.6 gives a detailed analysis of the results and suggests possible further improvements. Secion 2.4 introduced the modification we made of proximity measure. 2.8 shows how the size of training set can affect performance. 2.9 shows the kNN classifier's performance on training set.

2 Results and Evaluation

2.1 Results with different k values

In thie section we presents error rates and confusion matrices under different k values.

Table 2.1 and Table 2.2 show the error rates and confusion matrices of the $Iris$ dataset under different k values. Table 2.3 and Table 2.4 show the same results of the $Income$ dataset. Here we treat the ">50" class as the positive class.

An observation from these results is, in general, the kNN classifier works better on $Iris$ when using Euclidean distance as the proximity measure, while consine similarity works better for $Income$.

$k = 3, ER = 0.243$	P-Setosa	P-Virginica	P-Versicolor
A-Setosa	20	0	0
A-Viriginica	0	19	1
A-Versicolor	3	13	14

$k = 5, ER = 0.271$	P-Setosa	P-Virginica	P-Versicolor
A-Setosa	20	0	0
A-Virginica	0	20	0
A-Versicolor	4	15	11

$k = 19, ER = 0.3$	P-Setosa	P-Virginica	P-Versicolor
A-Setosa	20	0	0
A-Virginica	0	20	0
A-Versicolor	5	16	9

Table 2.1: Error Rates and Confusion Matrices of $Iris$ dataset when $k = 3, 5, 19$ with Cosine Similarity

$k=1, ER=0.143$	P-Setosa	P-Virginica	P-Versicolor
A-Setosa	20	0	0
A-Virginica	0	17	3
A-Versicolor	1	6	23

$k=7, ER=0.229$	P-Setosa	P-Virginica	P-Versicolor
A-Setosa	20	0	0
A-Virginica	0	14	6
A-Versicolor	1	9	20

$k=75, ER=0.1$	P-Setosa	P-Virginica	P-Versicolor
A-Setosa	20	0	0
A-Virginica	0	18	2
A-Versicolor	2	3	25

Table 2.2: Error Rates and Confusion Matrices of $Iris$ dataset when $k = 1,7,75$ with Euclidean Distance

$k=3, ER=0.236$	P-Pos	P-Neg
A-Pos	26	41
A-Neg	27	194

$k=7, ER=0.208$	P-Pos	P-Neg
A-Pos	24	43
A-Neg	17	204

$k=27, ER=0.236$	P-Pos	P-Neg
A-Pos	13	54
A-Neg	14	207

Table 2.3: Error Rates and Confusion Matrices of $Income$ dataset when $k = 3,7,27$ with Cosine Similarity

2.2 Performance Evaluation

We introduced the following evaluation measures to evaluate the kNN classifier:

$$Error\ Rate = \frac{FN+FP}{TP+FN+FP+TN}$$
$$TPR = \frac{TP}{TP+FN}$$
$$TNR = \frac{TN}{TN+FP}$$
$$FPR = \frac{FP}{FP+TN}$$
$$FNR = \frac{FN}{FN+TP}$$

$k=3, ER=0.285$	P-Pos	P-Neg
A-Pos	23	44
A-Neg	38	183

$k=29, ER=0.215$	P-Pos	P-Neg
A-Pos	12	55
A-Neg	7	214

$k=51, ER=0.239$	P-Pos	P-Neg
A-Pos	1	66
A-Neg	3	218

Table 2.4: Error Rates and Confusion Matrices of $Income$ dataset when $k = 3, 7, 27$ with Euclidean Similarity

Since the kNN classifier works better when taking cosine similarity as the proximity measure, in this section all the evalutaions are based on this proximity.

To train the kNN classifier, we varied the value of k from 1 to 99. For each k, we calculated the improvement over random guess and chose a optimal $k = 7$ that gives the largest lift. How we have chosen the optimal k will be explained in detailed in Section 2.7.

With this optimal $k = 7$, by tweaking the decision posterior, we got a combination of TPR and and FPR. We plotted TRP against FPR and got the ROC curve. (Firgure 2.1)

As we know, each point on the ROC curve corresponds to a decision threshold, which is the posterior probabilty $P(C = Pos|X)$ in this case. Here we pick 2 threshold values, one of which gives the largerst improvement over the baseline random guess approach, and the other gives the minimum error rate.We report the evaluation measures including True Positive, False Positive, True Negative and False Negative Rates, Recall, Precision and F-Measure under these two posterior thresholds (Table 2.5)

threshold	FPR	TPR	FNR	TNR	Recall	Precision	F-Measure
0.15	0.281	0.791	0.209	0.719	0.461	0.791	0.582
0.29	0.149	0.582	0.418	0.851	0.542	0.582	0.561

Table 2.5: Evaluation Measures, $k = 7$, $threshold = 0.15, 0.29$

In Table 2.5, when $k = 7$, the threshold 0.29 gives the minimum error rate while 0.15 gives the most significant improvement over random guess.

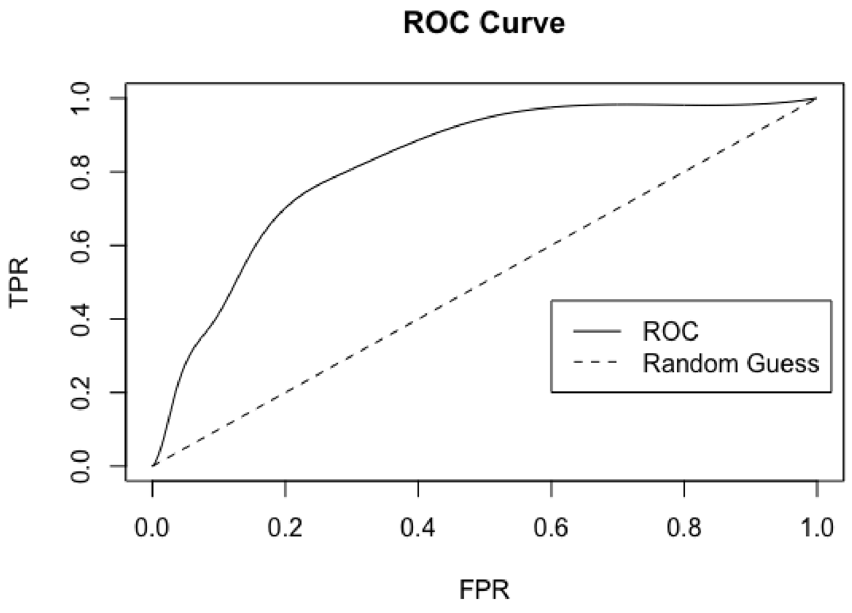

Figure 2.1: ROC Curve, $k = 7$

2.3 kNN with Weighted Posterior

We also implemented the kNN classifier with weighted posterior, which is

$$P(C = Pos|X) = \frac{\sum_{i=1}^{k} similarity(X_i)\mathbb{1}(C_i = Pos)}{\sum_{i=1}^{k} similarity(X_i)}.$$

With weighted posterior, when $k = 9$, the most significant improvement is achieved over random guess. The ROC curve is shown below. (Figire 2.2)

Similarly, we picked two threshold which give the minimum error rate and most significant improvement over random guess, respectively, to evaluate the performance. (Table 2.6)

threshold	FPR	TPR	FNR	TNR	Recall	Precision	F-Measure
0.11	0.434	0.940	0.060	0.566	0.396	0.940	0.558
0.34	0.136	0.612	0.388	0.864	0.577	0.611	0.594

Table 2.6: Evaluation Measures, with Weighted Posterior, $k = 9$, $threshold = 0.11, 0.34$

The results show that our classifier works better with weighted posterior, giving a slightly hight improvement over random guess (0.319 v.s. 0.314) and a smaller error rate (0.19 v.s. 0.21)

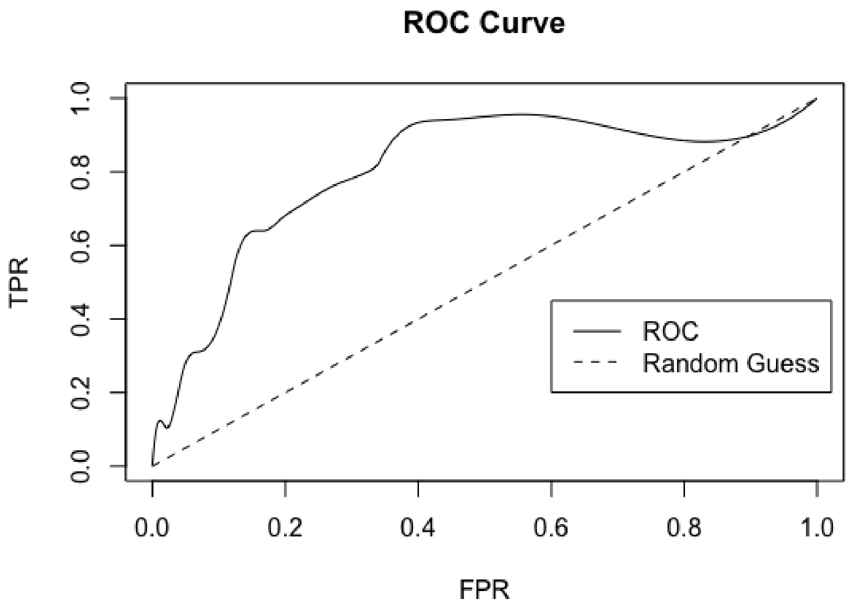

Figure 2.2: ROC Curve, with Weighted Posterior, $k = 9$

as well.

2.4 Adaptation to Proximity measure

In Homework 1 we treated two attributes, $capital\_gain$ and $capital\_loss$, as asymmetric numeric attributes. The motivation was that the distribution of these two attributes are fairly sparse, which can be easily drawn with some simple exploratory method. And it's also numeric. So when calucating proximity, we ignore all 0-0 matches, and for the non 0-0 pairs, we use distance to measure the proximity on these two attributes as if they were numeric values. However, as is shown in Table 1.1, our classifier was not working well this approach.

So we need to redesign our proximity measure to improve the performance. The problem with the $capital\_gain$ and $capital\_loss$ attributes are that, they are not only sparse but also range widely. For example, the maximum value of the $capital\_gain$ is 99999. According to our previous approach, 0 and 10000 has a similarity of 0.9 ($1 - \frac{10000-0}{99999}$) while we ignore all 0-0 matches. In general, the similarity of other attributes are smaller than 0.9 So this implies a 0-10000 match would have larger similarity than a 0-0 match, which we find unreasonable. So we decide to treat those two attributes as asymmetric binary attributes, that is, all non-zero values are treated as 1. A 0-0 match is ignored. A 0-1 match has similarity 0. A 1-1 match has similarity 1. This modified proximity measure proves to perform better.

2.5 Performance with Euclidean Distance

In Homework 1 we also applied euclidean distance as the proximity measure on numeric attributes. As we've mentioned before, for the *income* dataset, the *k*NN classifier gains a better performance when taking cosine similarity as its proximity measure. The detailed performance comparison is given in Table 2.7

Prox. Measure	ER	FPR	TPR	FNR	TNR	Recall	Precision	F-Measure
Cosine Similarity	0.19	0.434	0.940	0.060	0.566	0.396	0.940	0.558
Euclidean Distance	0.22	0.127	0.493	0.507	0.873	0.541	0.493	0.516

Table 2.7: Evaluation Measures with Different Proximity Measures

2.6 Analysis of Results

During the training process, we would always want the *k*NN to be accurate enough and to have a good enough performance according to some evaluation measures(i.e. F-Measure). So we fixed the decision posterior threshold to be 0.5 and plotted Figure 2.3 to see the trends of these two measures with *k* changing.

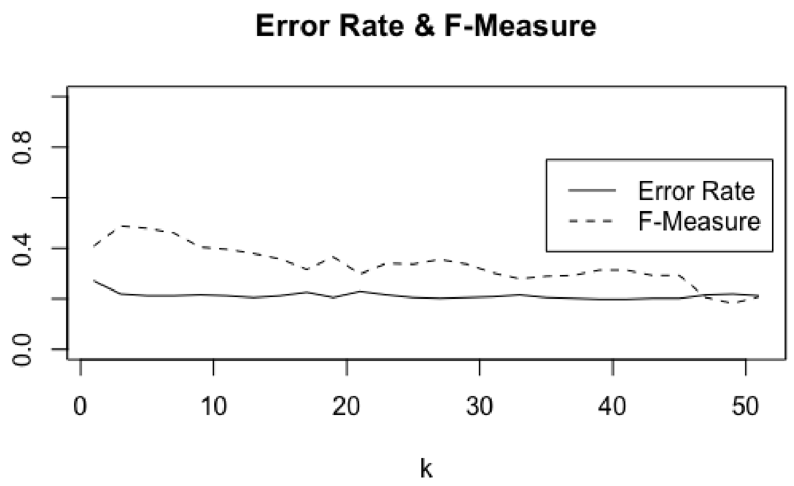

Figure 2.3: Trends of Error Rate and F-Measure with *k* Changing, threshold = 0.5

However, these two measures, error rate and F-measure, don't reach their optimal point at the same k value, which led us to our choice of method discussed in Section 2.7 to evaluate the performance of the classifier under different k values.

Then how do true positive rate and true negative rate vary with k? From Figure 2.4 we could observe that TPR and TNR go oppositely as k increasing. Thus, for this $income$ dataset, we are note able to maximize TPR and TNR at the same time.

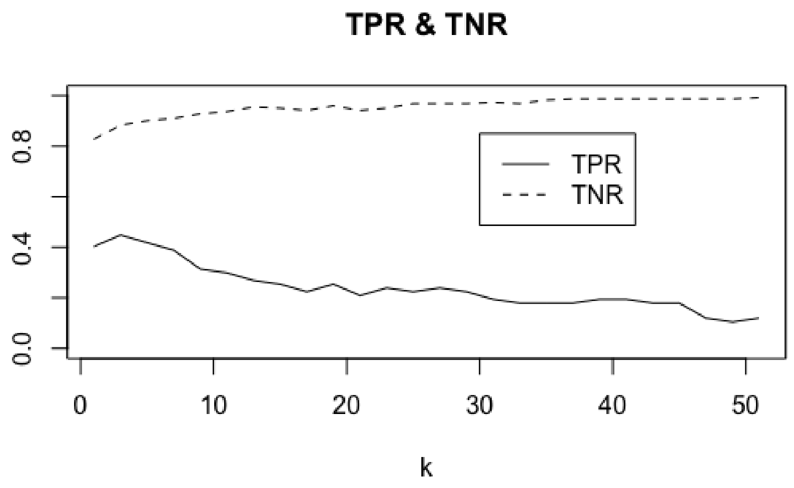

Figure 2.4: Trends of True Positive Rate and True Negative Rate with k Changing, threshold = 0.5

In addition, according to Figure 2.4, TNR is much higher than TPR under any setting of k, which means our classifier classifies records with the negative class labels more efficiently. Our guess is that, this is because there are approximately $1/4$ positive records and $3/4$ negative records in the training set. So the ability of classifying negative records is well-trained while the other is not.

Another observation that supports our guess is that, in Figure 2.3 that with different k, the error rate doesn't vary much, ending up steadily around 0.233 with large k's, which is the proportion of positive records in the training set.

Next, we fixed the k value to be 5, tweaking the thredshold posterior and observing the trends of some evaluation measures. (Figure 2.5 and 2.6)

So, with a fixed k, it's possible to get a satisfying error rate and F-measure with some threshold according to Figure 2.5. However, with k fixed, a high TPR and a high TNR cannot be reached at the same time according to 2.6.

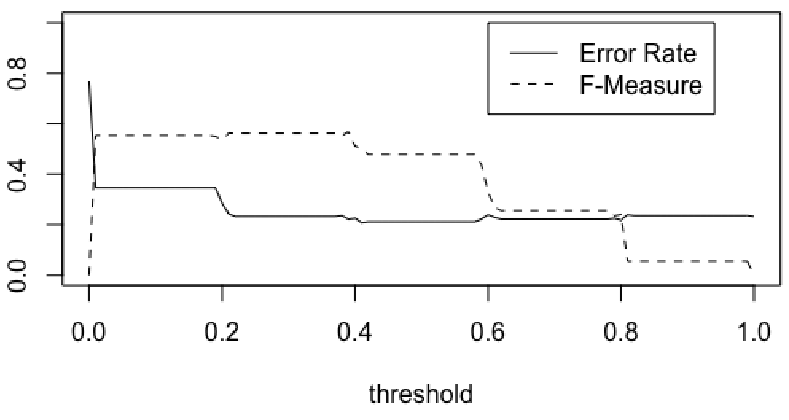

Figure 2.5: Trends of True Positive Rate and True Negative Rate with threshold Changing, $k = 5$

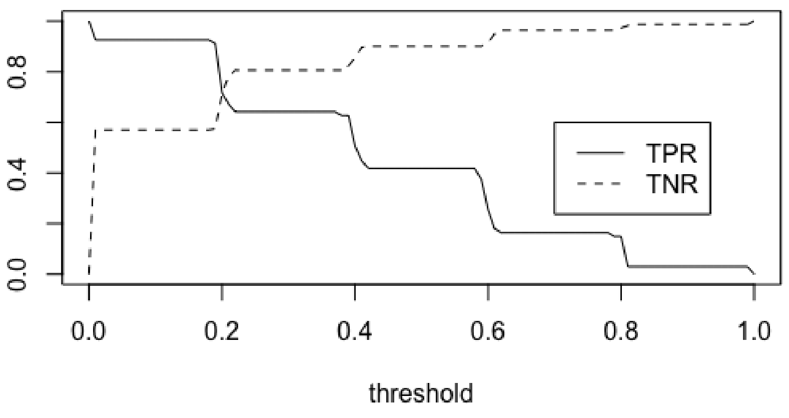

Figure 2.6: Trends of True Positive Rate and True Negative Rate with threshold Changing, $k = 5$

2.7 Training: Choosing the Optimal k Value

The most important goal of training is to find an optimal k that gives the best performance. But how to define the "best performance"?

Given an ROC curve (Figure 2.7), the improvement of the classifies over the default approach, random guess, can be defined as lift, which is the area between the ROC curve of our approach and the ROC curve of random guess (the diagonal line). To calculate the lift, we need in principle integrate the function over the interval [0,1] to calculate the area under the curve. However, it's impossible to know the closed-form function. So we calculate the area using approximation. In brief, we choose uniformly-spaced posterior thresholds from 0 to 1.The TPR and FPR of two consecutive thresholds are two points on the ROC curve. We connect these two points and draw two vertical lines passing the two points. This will make a trapezoid. We calculate the area of each trapezoid and add them up. If we make the space bwtween two consecutive thresholds small enough, this would be a good proximation of the actual area.

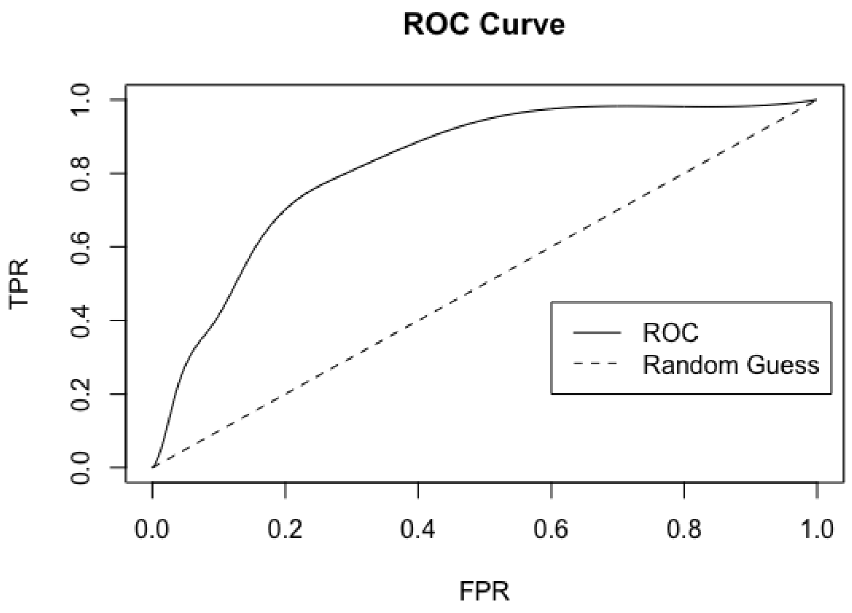

Figure 2.7: An ROC Curve

With lift as our optimization goal, we could get the optimal k by changing the value of k and choose the one that gives the maximum lift.

Figure 2.8 shows the lifts under different setting of k, among which $k = 9$ gives the maximum lift of 0.32. So after training, the classifier will do classification based on the class of the 9 nearest neighbors of a record with unknown class label.

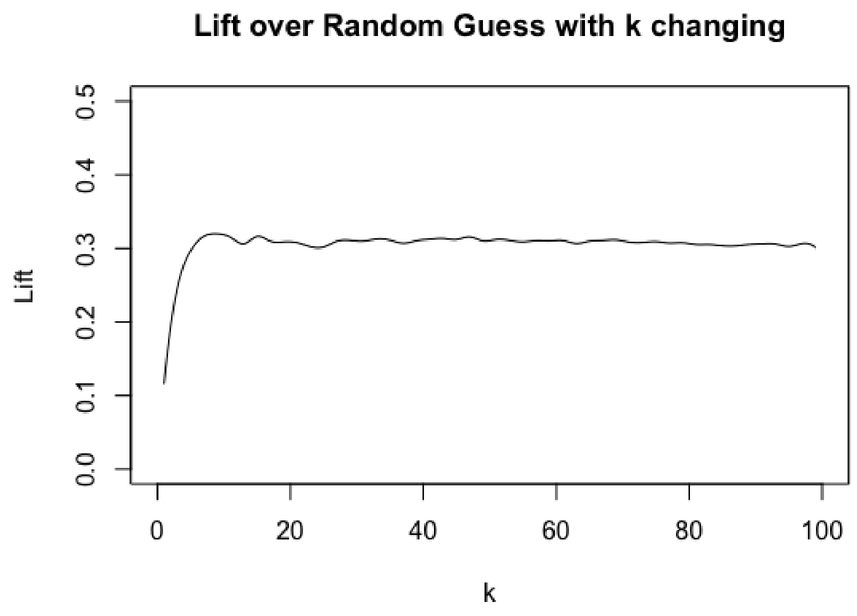

Figure 2.8: Lifts gained with k changing

2.8 Training: Trying Different Sizes of Training Sets

To see how the size of the training set can affect the classifier's performance, we changed the size of training data, randomly choosing 25%, 50%, 75% of the examples to train the classifier. Table 2.8 shows the affect of size of training set on performance of the kNN classifier. The corresponding ROC curves are shown in Figure 2.9. In general, a larger size of training size could guarantee a better performance of the classifier. And as the size of the training set decreasing, a larger k is needed to get the best performance.

size	opt. k	F-measure	Error Rate
100%	9	0.594	0.190
75%	7	0.400	0.219
50%	29	0.252	0.226
25%	63	0.510	0.349

Table 2.8: kNN Performance with Different Size of Training Set

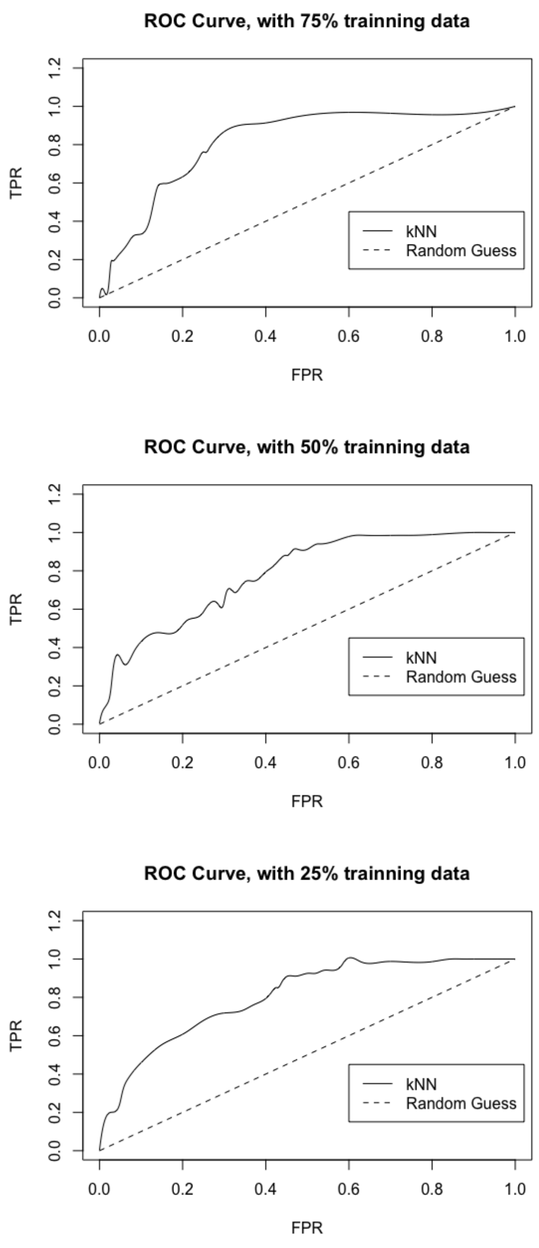

Figure 2.9: ROC Curve under Different Sizes of Training Data

2.9 *k*NN Working on Training Set

If taking training dataset itself as test dataset, we get the perforance shown in Table 2.9.

The performance on training set, however, doesn't show any significant difference in terms of F-Measure and error rate. Which indicates that our model doesn't overfit the training set too much.

Test	ER	FPR	TPR	FNR	TNR	Recall	Precision	F-Measure
income_te	0.19	0.434	0.940	0.060	0.566	0.396	0.940	0.558
income_tr	0.19	0.152	0.660	0.340	0.848	0.508	0.660	0.573

Table 2.9: Evaluation Measures, with Training Set as Test Set

3 Comparison to an Off-the-Shelf kNN Classifier

One of the off-the-shelf implementation of kNN classifier would be $kknn$ package which is created and maintained by Klaus Schliep (https://cran.r-project.org/web/packages/kknn/). It is a weighted kNN classifier, performing k-nearest neighbor classification of a test set using a training set. For each row of the test set, the k nearest training set vectors (according to Minkowski distance) are found, and the classification is done via the maximum of summed kernel densities. In addition, even ordinal and continuous variables can be predicted.

We compared our approach to the $kknn$ package using a dataset of a relatively smaller size. We extract 2/3 data from the training set as our new training set and the rest 1/3 as test set.

The evaluation measures of these two classifier are in Table 3.1

Test	ER	FPR	TPR	FNR	TNR	Recall	Precision	F-Measure
our approach	0.121	0.014	0.407	0.594	0.986	0.867	0.406	0.553
kknn	0.13	0.063	0.563	0.438	0.936	0.667	0.563	0.610

Table 3.1: Comarison to the off-the-shelf $kknn$ Classifier

Our model surprisingly beat the $kknn$ classifier on accuracy! However, the running time of our model is much slower.