CS 5525
Solutions to Homework Assignment 3

Meghendra Singh

October 18, 2016

[10] 1.

The given Confusion Matrix:

	Predicted Class		
		+	-
Actual Class	+	100	25
	-	50	145

(a) Accuracy:

$$\begin{aligned} Accuracy &= \frac{TP + TN}{TP + TN + FP + FN} \\ &= \frac{100 + 145}{100 + 145 + 50 + 25} \\ &= \frac{245}{320} \\ &= 0.765 \textbf{ or } \textbf{76.56\%} \end{aligned}$$

(b) Precision:

$$\begin{aligned} Precision &= \frac{TP}{TP + FP} \\ &= \frac{100}{100 + 50} \\ &= \frac{100}{150} \\ &= 0.666 \textbf{ or } \textbf{66.66\%} \end{aligned}$$

(c) F-measure:

$$\begin{aligned} F - measure &= \frac{2 * TP}{2 * TP + FP + FN} \\ &= \frac{200}{200 + 50 + 25} \\ &= \frac{200}{275} \\ &= 0.727 \textbf{ or } \textbf{72.72\%} \end{aligned}$$

(d) Sensitivity:

$$\begin{aligned} \textit{Sensitivity} \textbf{ or } TPR &= \frac{TP}{TP+FN} \\ &= \frac{100}{100+25} \\ &= \frac{100}{125} \\ &= 0.80 \textbf{ or } \textbf{80\%} \end{aligned}$$

(e) Specificity:

$$\begin{aligned} \textit{Specificty} \textbf{ or } (1-FPR) &= 1 - \frac{FP}{FP+TN} \\ &= 1 - \frac{50}{50+145} \\ &= 1 - \frac{50}{195} \\ &= 0.7435 \textbf{ or } \textbf{74.35\%} \end{aligned}$$

(f) Cost (classification cost is -1 and misclassification cost is 5):

$$\begin{aligned} \textit{Cost} &= -1 * (TP+TN) + 5 * (FP+FN) \\ &= -245 + 375 \\ &= \textbf{130} \end{aligned}$$

---

[15] 2.

(a) Precision at 40% Recall:

Search engine A	**Search engine B**
Here, $TP = 4$ and $FP = 4$ Therefore, $\textit{Precision} = \frac{4}{4+4} = \frac{4}{8}$ $= 0.5$ **or 50%**	Here, $TP = 4$ and $FP = 8$ Therefore, $\textit{Precision} = \frac{4}{4+8} = \frac{4}{12}$ $= 0.333$ **or 33.33%**

(b) Precision, Recall and F-measure if only first 15 documents are displayed:

Measures	**Search engine A**	**Search engine B**
Precision (p) $= \frac{TP}{TP+FP}$	$\frac{6}{6+9} = 0.4$ **or 40%**	$\frac{5}{5+10} = 0.333$ **or 33.33%**
Recall (r)$= \frac{TP}{TP+FN}$	$\frac{6}{6+4} = 0.6$ **or 60%**	$\frac{5}{5+5} = 0.5$ **or 50%**
F-Measure $= \frac{2*r*p}{r+p}$	$\frac{2*0.6*0.4}{0.6+0.4} =$ **0.48**	$\frac{2*0.5*0.33}{0.5+0.33} =$ **0.396**

---

[20] 3.

PART I: The performance of models M1 and M2 for the given conditions are as follows:

(a) Recall and F-measure for the models at threshold value of 0.5:

(1) Contingency tables:

(i) M1:

		Predicted Class	
		+	-
Actual Class	+	3	2
	-	1	4

(ii) M2:

		Predicted Class	
		+	-
Actual Class	+	1	4
	-	1	4

(2) Recall and F-Measure:

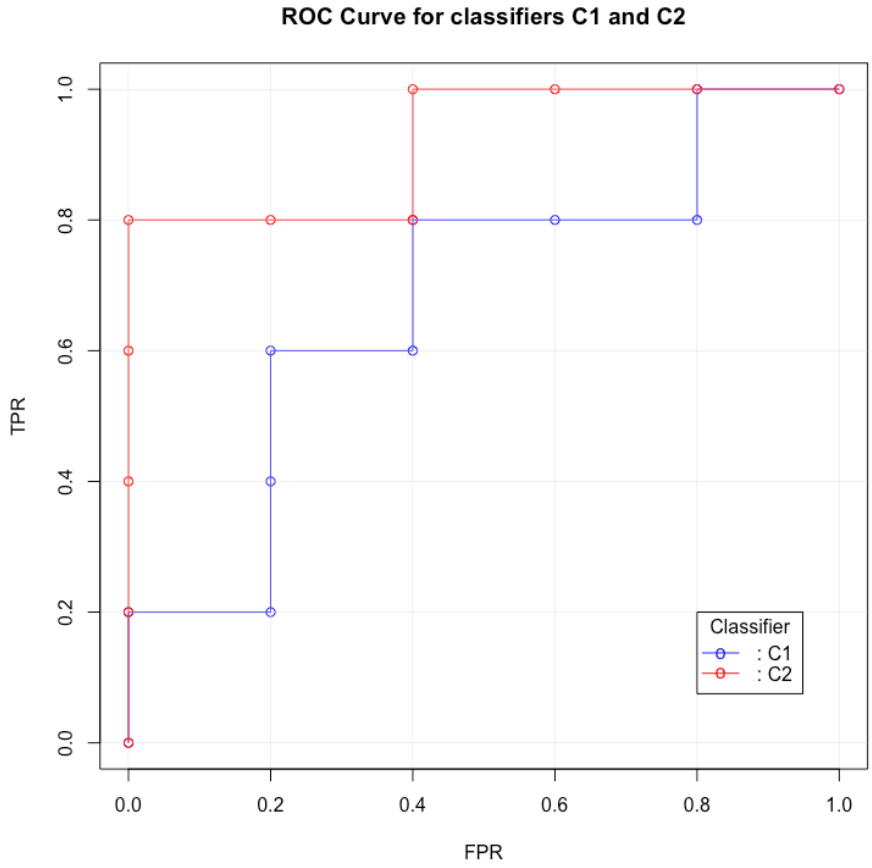
Measures	**M1**	**M2**
Recall = $\frac{TP}{TP+FN}$	$\frac{3}{3+2} = 0.6$ **or 60%**	$\frac{1}{1+4} = 0.2$ **or 20%**
F-Measure = $\frac{2*TP}{2*TP+FP+FN}$	$\frac{2*3}{2*3+1+2} =$ **0.66**	$\frac{2*1}{2*1+1+4} =$ **0.285**

(b) Precision when Recall = 60%:

Precision = $\frac{TP}{TP+FP}$	**M1**	**M2**
Best case	$\frac{3}{3+0} = 1$ **or 100%**	$\frac{3}{3+3} = 0.5$ **or 50%**
Worst case	$\frac{3}{3+1} = 0.75$ **or 75%**	$\frac{3}{3+5} = 0.375$ **or 37.5%**

PART II: ROC curves and Wilcoxon Mann Whitney statistic for classifiers C1 and C2:

(a) The ROC Curves are plotted below:



Classifier C2 (red plot line) has a larger area under the ROC curve ($Area(C2) = \mathbf{0.92}$) as compared to C1 (blue plot line, $Area(C1) = \mathbf{0.68}$).

(b) Wilcoxon Mann Whitney (WMW) statistic for C1 and C2:

WMW	**C1**	**C2**
$WMW = \frac{\sum_{i=1}^{m-1}\sum_{i=1}^{n-1} I(x_i, y_i)}{mn}$	$\frac{1+3+4+4+5}{5*5} = \frac{17}{25} = \mathbf{0.68}$	$\frac{3+5+5+5+5}{5*5} = \frac{23}{25} = \mathbf{0.92}$

Classifier **C2** has a larger WMW value as compared to classifier **C1**.

---

[30] 4.

PART I: Rule-based Classification:

(a) FOIL's Information Gain:

$$Gain(R0, R1) = t * [\log(\frac{p1}{p1 + n1}) - \log(\frac{p0}{p0 + n0})]$$

The following table gives the FOIL's Information Gain for the three rules R1, R2 and R3:

Rule	**FOIL's Info. Gain**
R1	$12 * [\log(\frac{12}{12+3}) - \log(\frac{29}{29+21})] = \mathbf{5.59}$
R2	$7 * [\log(\frac{7}{7+3}) - \log(\frac{29}{29+21})] = \mathbf{1.892}$
R3	$8 * [\log(\frac{8}{8+4}) - \log(\frac{29}{29+21})] = \mathbf{1.60}$

Rule R1 has the largest FOIL's Information Gain followed by R2 and R3. Therefore, R1 is the best rule and R3 is the worst rule.

(b) M-estimate measure (with $k = 2$ and $p+ = 0.58$):

$$M - estimate = \frac{n_c + kp}{n + k}$$

The following table gives the M-estimate for the three rules R1, R2 and R3:

Rule	**M-estimate**
R1	$\frac{12+2*0.58}{15+2} = \mathbf{0.774}$
R2	$\frac{7+2*0.58}{10+2} = \mathbf{0.68}$
R3	$\frac{8+2*0.58}{15+2} = \mathbf{0.65}$

Rule R1 has the largest M-estimate followed by R2 and R3. Therefore, R1 is the best rule and R3 is the worst rule.

(c) Rule accuracy of R2 and R3 after R1 has been discovered and positive examples covered by R1 have been discarded:

$$Accuracy = \frac{|A\&\&Y|}{|A|}$$

The following table gives the Accuracies for rules R2 and R3 after R1 has been discovered:

Rule	**Accuracy**
R2	$\frac{7}{10} = \mathbf{0.7}$
R3	$\frac{6}{10} = \mathbf{0.6}$

Rule R2 has greater accuracy than R3. Therefore, R2 is the best rule and R3 is the worst rule.

PART II: Coverage, Accuracy and FOIL's Information Gain for R1, R2 and R3 given the initial rule R0: $\phi \rightarrow +$.

Rule	**Coverage**	**Accuracy**	**FOIL's Info. Gain**
R1	$\frac{15}{300} = \mathbf{0.05}$	$\frac{12}{15} = \mathbf{0.8}$	$12 * [\log(\frac{12}{15}) - \log(\frac{100}{300})] = \mathbf{15.17}$
R2	$\frac{30}{300} = \mathbf{0.1}$	$\frac{20}{30} = \mathbf{0.666}$	$20 * [\log(\frac{20}{30}) - \log(\frac{100}{300})] = \mathbf{20.09}$
R3	$\frac{180}{300} = \mathbf{0.6}$	$\frac{100}{180} = \mathbf{0.555}$	$100 * [\log(\frac{100}{180}) - \log(\frac{100}{300})] = \mathbf{73.6}$

---

[25] 5.

PART I: Nearest-Neighbor Classification:

(a) The 5-nearest neighbors for the test point:$P = (5, 4)$ on the basis of euclidean distance are:

Point	**Distance from P**	**Class (+ or −)**
$Q1 = (4, 4)$	1.0	−
$Q2 = (6, 5)$	1.41	+
$Q3 = (7, 3)$	2.23	+
$Q4 = (5, 1)$	3.0	−
$Q5 = (4, 1)$	3.162	−

As the majority of the nearest neighbors (**3** out of **5**) belong to '−' class, the test point P is assigned to class '−'.

(b) The 3-nearest neighbors to the test point:$P = (5, 4)$ are:

Point	**Euclidean distance to P**	**Class (+ or −)**	**Manhattan distance to P**
$Q1 = (4, 4)$	1.0	−	1
$Q2 = (6, 5)$	1.41	+	2
$Q3 = (7, 3)$	2.23	+	3

For Manhattan distance weighted 3-nearest neighbor (with $weight = \frac{1}{d^2}$) the weights computed for the two classes '+' and '−' are as follows:

Class	**Weight =** $\frac{1}{d^2}$; d =Manhattan distance to P
−	$\frac{1}{1^2} = \mathbf{1}$
+	$\frac{1}{2^2} + \frac{1}{3^2} = \mathbf{0.361}$

As the '−' class has a greater weight, the test point P is assigned to class '−'.

PART II: Nearest-Neighbor Classification programming exercise:

(a) Unweighted 3-NN actual classes, predicted classes and classification probabilities for different normalization schemes:

kNN		No Normalization		Min-Max		Z-Score	
Point	**Act. Class**	**Pred. Class**	**Prob.**	**Pred. Class**	**Prob.**	**Pred. Class**	**Prob.**
1	1	1	1	1	1	1	1
2	0	0	0.667	0	1	1	0.667
3	0	0	1	0	1	0	1
4	1	1	1	1	1	1	1
5	0	0	1	0	1	0	1
6	1	1	1	1	1	1	1
7	0	0	1	0	1	0	1
8	0	0	1	0	1	0	1
9	0	0	1	0	1	0	1
10	0	0	1	0	1	0	1
11	1	1	1	1	0.667	1	1
12	0	0	1	0	1	0	1
13	0	0	1	0	1	0	1
14	0	0	1	0	1	0	1
15	0	1	0.667	0	0.667	1	1
16	1	1	1	1	1	1	1
17	1	1	1	1	1	1	1
18	0	0	1	0	1	0	1
19	0	0	1	0	1	0	1
20	0	0	1	0	1	0	1

Here, we observe that as we change the normalization scheme used to scale the training and test data, the classification probabilities for some test data points changes. Also, the classification of points 2 and 15 change if we change the normalization scheme.

(b) Weighted 3-NN (Weight= $\frac{1}{d^2}$) actual classes, predicted classes and classification probabilities for different normalization schemes:

WkNN		No Normalization		Min-Max		Z-Score	
Point	**Act. Class**	**Pred. Class**	**Prob.**	**Pred. Class**	**Prob.**	**Pred. Class**	**Prob.**
1	1	1	1	1	1	1	1
2	0	0	0.712	0	1	1	0.949
3	0	0	1	0	1	0	1
4	1	1	1	1	1	1	1
5	0	0	1	0	1	0	1
6	1	1	1	1	1	1	1
7	0	0	1	0	1	0	1
8	0	0	1	0	1	0	1
9	0	0	1	0	1	0	1
10	0	0	1	0	1	0	1
11	1	1	1	1	0.993	1	1
12	0	0	1	0	1	0	1
13	0	0	1	0	1	0	1
14	0	0	1	0	1	0	1
15	0	1	0.689	0	0.995	1	1
16	1	1	1	1	1	1	1
17	1	1	1	1	1	1	1
18	0	0	1	0	1	0	1
19	0	0	1	0	1	0	1
20	0	0	1	0	1	0	1

Once again we observe that the normalization scheme used to scale the training and test data changes, the classification probabilities for some test data points changes. But, if we compare individual sequences, weighted k-NN does not change the outcome of the classification. Although, the classification probabilities change because of the weighting function, we get the same predicted class label for each point when we use $\frac{1}{d^2}$ weighting function in the k-NN.

(c) **(1)** Confusion matrix, Accuracy, Precision and F-measure for the two cases without normalization of the data:

(i) Unweighted k-NN:

	Predicted Class		
		0	1
Actual Class	0	13	0
	1	1	6

$$\begin{aligned} Accuracy &= \frac{13+6}{13+6+1+0} \\ &= \frac{19}{20} \\ &= 0.95 \textbf{ or } \textbf{95\%} \end{aligned}$$

$$\begin{aligned} Precision &= \frac{13}{13+1} \\ &= \frac{13}{14} \\ &= 0.928 \textbf{ or } \textbf{92.8\%} \end{aligned}$$

$$\begin{aligned} F-measure &= \frac{2*13}{2*13+1+0} \\ &= \frac{26}{26+1+0} \\ &= \frac{26}{27} \\ &= 0.962 \textbf{ or } \textbf{96.2\%} \end{aligned}$$

(ii) Weighted k-NN:

	Predicted Class		
Actual Class		0	1
	0	13	0
	1	1	6

$$\begin{aligned} Accuracy &= \frac{13+6}{13+6+1+0} \\ &= \frac{19}{20} \\ &= 0.95 \textbf{ or } \textbf{95\%} \end{aligned}$$

$$\begin{aligned} Precision &= \frac{13}{13+1} \\ &= \frac{13}{14} \\ &= 0.928 \textbf{ or } \textbf{92.8\%} \end{aligned}$$

$$\begin{aligned} F-measure &= \frac{2*13}{2*13+1+0} \\ &= \frac{26}{26+1+0} \\ &= \frac{26}{27} \\ &= 0.962 \textbf{ or } \textbf{96.2\%} \end{aligned}$$

We get the same results from both the methods.

(2) Confusion matrix, Accuracy, Precision and F-measure for the two cases for the *Min-Max* normalization scheme:

(i) Unweighted k-NN:

	Predicted Class		
		0	1
Actual Class	0	14	0
	1	0	6

$$\begin{aligned} Accuracy &= \frac{14+6}{14+6+0+0} \\ &= \frac{20}{20} \\ &= 1 \textbf{ or } \textbf{100\%} \end{aligned}$$

$$\begin{aligned} Precision &= \frac{14}{14+0} \\ &= \frac{14}{14} \\ &= 1 \textbf{ or } \textbf{100\%} \end{aligned}$$

$$\begin{aligned} F-measure &= \frac{2*14}{2*14+0+0} \\ &= \frac{28}{28+0+0} \\ &= \frac{28}{28} \\ &= 1 \textbf{ or } \textbf{100\%} \end{aligned}$$

(ii) Weighted k-NN:

	Predicted Class		
		0	1
Actual Class	0	14	0
	1	0	6

$$\begin{aligned} Accuracy &= \frac{14+6}{14+6+0+0} \\ &= \frac{20}{20} \\ &= 1 \textbf{ or } \textbf{100\%} \end{aligned}$$

$$\begin{aligned} Precision &= \frac{14}{14+0} \\ &= \frac{14}{14} \\ &= 1 \textbf{ or } \textbf{100\%} \end{aligned}$$

$$\begin{aligned} F - measure &= \frac{2 * 14}{2 * 14 + 0 + 0} \\ &= \frac{28}{28 + 0 + 0} \\ &= \frac{28}{28} \\ &= 1 \textbf{ or } \textbf{100\%} \end{aligned}$$

We get the same results from both the methods.

(d) Source code in R for both the methods, with and without normalizations:

```
## utility functions used in the rest of the code
# normalize the data with min max
normalize <- function(x) {
  nx <- ((x - min(x)) / (max(x) - min(x)))
  return(nx)
}
# normalize the data with z-score
zScoreNormalize <- function(x) {
  nx <- (x - mean(x)) / sd(x)
  return(nx)
}
# find euclidean distance between test and training data
distance <- function(x, y) {
  dxy <- sqrt(sum(( x - y )^2))
  return(dxy)
}
# find k nearest neighbors by distance
getKnnByDistance <- function(dxy, k) {
  knnbd <- order(dxy) [1:k]
  return (knnbd)
}
# find weights for the votes of k nearest neighbors by distance
getKnnWeightsByDistance <- function(dxy, k) {
  knnbd <- order(dxy) [1:k]
  knnwbd <- 1/(dxy[knnbd])^2
  return (knnwbd)
}
# find the majority vote / weighted majority vote for each label
findVotes <- function(neighborIds, weightVector=rep(1, length(neighborIds))) {
  result <- matrix(0, nrow=2, ncol = 3)
  knnLabels <- train_labels[neighborIds,1]
  for (i in 1 : length(knnLabels))
  {
    if (knnLabels[i] == 0) {
```

```
      result[1,] <- c(0, (sum (1, result[1,2])) * weightVector[i], 0)
    }
    if (knnLabels[i] == 1) {
      result[2,] <- c(1, (sum (1, result[2,2])) * weightVector[i], 0)
    }
  }
  result[,3] <- result[,2] / sum(result[,2])
  return(result)
}

#load data
train <- read.csv("train.csv")
test <- read.csv("test.csv")
head(train)
head(test)
# test.csv has an additional column ID which is useless hence discarded
## clean and normalize the data by min-max normalization
train_n <- as.data.frame(lapply(train[,c(1,2,3)], normalize))
test_n <- as.data.frame(lapply(test[,c(2,3,4)], normalize))
train_labels <- as.data.frame(train[,c(4)])
test_labels <- as.data.frame(test[,c(5)])

## OR
## normalize the data using z-score normalization
# train_n <- as.data.frame(lapply(train[,c(1,2,3)], zScoreNormalize))
# test_n <- as.data.frame(lapply(test[,c(2,3,4)], zScoreNormalize))
# train_labels <- as.data.frame(train[,c(4)])
# test_labels <- as.data.frame(test[,c(5)])

# OR
## dont normalize the data
# train_n <- as.data.frame(train[,c(1,2,3)])
# test_n <- as.data.frame(test[,c(2,3,4)])
# train_labels <- as.data.frame(train[,c(4)])
# test_labels <- as.data.frame(test[,c(5)])

## compute distance matrix d
d <- matrix(NA, nrow = nrow(test_n), ncol = nrow(train_n))
for (i in 1:nrow(test_n)) {
  d[i,] = apply(train_n[,1:3], 1, distance, x = test_n[i,1:3])
}
d <- t(d)
##############################################
## For unweighted kNN with k=3
## Compute ids of k-NN with k = 3
```

```
knns <- apply(d, 2, getKnnByDistance, k=3)
predictedLabelAggr <- vector()
predictedProbAggr <- vector()
for (i in 1:ncol(knns)){
  votes <- findVotes(knns[,i])
  predictedLabelAggr[i] <- votes[which.max(votes[,3]), 1]
  predictedProbAggr[i] <- votes[which.max(votes[,3]), 3]
}
result <- cbind(predictedLabelAggr, predictedProbAggr)
# write confusion matrix
table(predictedLabelAggr, test_labels[,1])

################################################
## For weighted k-NN with k = 3 and w=(1/d^2)
## determine the weightvectors of the kNNs
weightVectors <- apply(d, 2, getKnnWeightsByDistance, k=3)
predictedLabelAggr <- vector()
predictedProbAggr <- vector()
for (i in 1:ncol(knns)){
  votes <- findVotes(knns[,i], weightVector = weightVectors[,i])
  predictedLabelAggr[i] <- votes[which.max(votes[,3]), 1]
  predictedProbAggr[i] <- votes[which.max(votes[,3]), 3]
}
resultWeighted <- cbind(predictedLabelAggr, predictedProbAggr)
# write confusion matrix
table(predictedLabelAggr, test_labels[,1])
```