Week 10 Quiz - More classification - Answers

FIT5197 teaching team

Note you might need to use the unit Formula Sheet (https://lms.monash.edu/mod/resource/view.php?id=7439150) to help you answer the following questions.

If you can't calculate $y = log_2(x)$ on your calculator you can use the formula $log_2(x) = log_c(x)/log_c(2)$ where c is any constant. For example your calculator can probably do $log_2(x) = log_{10}(x)/log_{10}(2)$ or $log_2(x) = log_e(x)/log_e(2)$.

Question 1

An Autonomous Vehicle (AV) needs to establish some rules about whether to drive fast or slow under various road conditions. The following table records the optimal choices:

SPEED	Slow	Slow	Slow	Slow	Slow	Fast	Fast	Fast
Spped Limit	Yes	Yes	No	Yes	Yes	No	Yes	No
Road Surface	Bumpy	Smooth	Bumpy	Bumpy	Smooth	Smooth	Smooth	Smooth
Grade	Steep	Steep	Steep	Flat	Steep	Flat	Flat	Steep

Using infomation gain (i.e. difference in entropy) as a metric, determine which of the 3 predictor variables; Speed Limit, Road Surface or Grade should be used as the first data split in the AVs decision tree. (Please show all workings).

Answer 1

Now we need to determine which predictor, $SpeedLimit$, $RoadSurface$ or $Grade$, when split, gives the most information gain in terms of predicting the target, $SPEED$. Formally we can express the information gain a $predictor$ adds to $knowledge$ about the $target$ as:

$$I.G.(target, predictor) = H(target) - H(target|predictor)$$

where $H(target)$ is the entropy of the $target$ random variable and $H(target|predictor)$ is the entropy of the $target$ random variable conditional on knowledge of the $predictor$ random variable. Although we need $H(target)$ to calculate information gain exactly, we don't need to calculate $H(target)$ to determine which predictor will give us the best split. We only need to calculate $H(target|predictor)$ for each predictor, since the

predictor with the smallest $H(target|predictor)$ value is guaranteed to have the largest information gain based on the formula above.

Another way you can think about this is that entropy is a measure of uncertainty. The higher the entropy the higher the uncertainty. For example, a uniformly distributed random variable has the highest uncertainty/entropy because all possible values of the random variable are equally likely, whereas a random variable which only has one possible event which occurs with probably equal to 1 has zero uncertainty/entropy. So then we can ask, does knowledge of my $predictor$ reduce the uncertainty in what my $target$ will be. We can think of $H(target)$ as our uncertainty in the $target$ without any other knowledge except for knowledge about the $target$. Then we can also think of the $H(target|predictor)$ as the uncertainty in the $target$ when we have knowledge about the $predictor$. So if $H(target|predictor)$ is less than $H(target)$ we know our $predictor$ is decreasing uncertainty in the $target$, which means we can predict the $target$ better. The bigger the magnitude of the uncertainty drop (i.e. information gain), the better the prediction.

Now back to the question. Based on the table above assign the following labels (because it is easier to use a symbol than to write the whole word):

(S) SPEED - (Fa) Fast, (Sl) Slow

(L) Speed Limit - (N) No, (Y) Yes

(R) Road Surface - (Sm) Smooth, (B) Bumpy

(G) Grade - (Fl) Flat, (St) Steep

Using our short-hand symbols from above we basically need to calculate $H(S|L)$, $H(S|R)$ and $H(S|G)$, determine which of the three is the smallest, and choose the corresponding predictor as the one to perform the first data split on. From the unit formula sheet we can see entropy (to base 2 by default) is defined as:

$$H(X) = E\left[\log_2 1/p(X)\right] = \sum_x p(X = x) \log_2 1/p(X = x),$$

and conditional entropy (to base 2 by default) is defined as:

$$H(X|Y) = \sum_y p(Y = y) H(X|Y = y)$$

where $H(X|Y = y)$ is the entropy of the conditional distribution $p(X|Y = y)$, i.e.

$$H(X|Y = y) = E\left[\log_2 1/p(X|Y = y)\right] = \sum_x p(X = x|Y = y) \log_2 1/p(X = x|Y = y).$$

So we see we need to calculate $p(Y = y)$ and $p(X = x|Y = y)$ to determine conditional entropy.

For the given question our target corresponds to $SPEED$ which we referred to above as S.

Based on the table we see: $P(S = Fa) = \frac{3}{8}$ and $P(S = Sl) = \frac{5}{8}$.

So the information we have about S without any knowledge of the predictors is

$$\begin{aligned} H(S) &= \sum_s p(S = s) \log_2 1/p(S = s) \\ &= p(S = Fa) \log_2 1/p(S = Fa) + p(S = sl) \log_2 1/p(S = sl) \\ &= 3/8 \log_2 8/3 + 5/8 \log_2 8/5 \\ &= 3/8.(1.4150374993) + 5/8.(0.67807190511) = 0.9544 \end{aligned}$$

Now in an exam for a question like this you wouldn't need to calculate $H(S)$ unless explicitly asked to calculate $H(S)$ or the exact information gain, but we have shown you here just so you know how to do it.

Now let's calculate $H(S|L)$. To do this we need the following which we calculate from the table: $p(L = Y) = 5/8$, $p(L = N) = 3/8$, $p(S = Fa|L = Y) = 1/5$, $p(S = Sl|L = Y) = 4/5$, $p(S = Fa|L = N) = 2/3$, and $p(S = Sl|L = N) = 1/3$. So we can note

$$\begin{aligned} H(S|L = N) &= \sum_s p(S = s|L = N) \log_2 1/p(S = s|L = N) \\ &= p(S = Fa|L = N) \log_2 1/p(S = Fa|L = N) + p(S = Sl|L = N) \log_2 1/p(S = Sl|L = N) \\ &= 2/3 log_2 3/2 + 1/3 log_2 3 = 0.9183 \\ H(S|L = Y) &= \sum_s p(S = s|L = Y) \log_2 1/p(S = s|L = Y) \\ &= p(S = Fa|L = Y) \log_2 1/p(S = Fa|L = Y) + p(S = Sl|L = Y) \log_2 1/p(S = Sl|L = Y) \\ &= 1/5 log_2 5 + 4/5 log_2 5/4 = 0.7219 \\ H(S|L) &= \sum_l p(L = l) H(S|L = l) \\ &= p(L = N) H(S|L = N) + p(L = Y) H(S|L = Y) \\ &= (3/8).0.9183 + (5/8).0.7219 = 0.7955. \end{aligned}$$

Now let's calculate $H(S|R)$. To do this we need the following which we calculate from the table: $p(R = Sm) = 5/8$, $p(R = B) = 3/8$, $p(S = Fa|R = Sm) = 3/5$, $p(S = Sl|R = Sm) = 2/5$, $p(S = Fa|R = B) = 0$, and $p(S = Sl|R = B) = 1$. So we can note

$$
\begin{aligned}
H(S|R=Sm) &= \sum_s p(S=s|R=Sm)\log_2 1/p(S=s|R=Sm) \\
&= p(S=Fa|R=Sm)\log_2 1/p(S=Fa|R=Sm) + p(S=Sl|R=Sm)\log_2 1/p(S=Sl|R=Sm) \\
&= 3/5 log_2 5/3 + 2/5 log_2 5/2 = 0.9710 \\
H(S|R=B) &= \sum_s p(S=s|R=B)\log_2 1/p(S=s|R=B) \\
&= p(S=Fa|R=B)\log_2 1/p(S=Fa|R=B) + p(S=Sl|R=B)\log_2 1/p(S=Sl|R=B) \\
&= 0 log_2 0 + 1 log_2 1 = 0 \\
H(S|R) &= \sum_r p(R=r)H(S|R=r) \\
&= p(R=Sm)H(S|R=Sm) + p(R=B)H(S|R=B) \\
&= (5/8).0.9710 + (3/8).0 = 0.6069.
\end{aligned}
$$

Now let's calculate $H(S|G)$. To do this we need the following which we calculate from the table: $p(G=Fl) = 3/8$, $p(G=St) = 5/8$, $p(S=Fa|G=Fl) = 2/3$, $p(S=Sl|G=Fl) = 1/3$, $p(S=Fa|G=St) = 1/5$, and $p(S=Sl|G=St) = 4/5$. So we can note

$$
\begin{aligned}
H(S|G=Fl) &= \sum_s p(S=s|G=Fl)\log_2 1/p(S=s|G=Fl) \\
&= p(S=Fa|G=Fl)\log_2 1/p(S=Fa|G=Fl) + p(S=Sl|G=Fl)\log_2 1/p(S=Sl|G=Fl) \\
&= 2/3 log_2 2/3 + 1/3 log_2 3 = 0.9183 \\
H(S|G=St) &= \sum_s p(S=s|G=St)\log_2 1/p(S=s|G=St) \\
&= p(S=Fa|G=St)\log_2 1/p(S=Fa|G=St) + p(S=Sl|G=St)\log_2 1/p(S=Sl|G=St) \\
&= 1/5 log_2 5 + 4/5 log_2 5/4 = 0.7219 \\
H(S|G) &= \sum_g p(G=g)H(S|G=g) \\
&= p(G=Fl)H(S|G=Fl) + p(G=St)H(S|G=St) \\
&= (3/8).0.9183 + (5/8).0.7219 = 0.7955.
\end{aligned}
$$

Therefore since $H(S|R)$ is less than $H(S|G)$ and $H(S|L)$ we should split first on R, Road Surface! Now in case you didn't notice there is a quicker way to arrive at this answer. First we can see that G and L give rise to the same probabilities prior to the entropy calculations so we can note they will perform equivalently (evidenced by the entropy calculations). Second we can see that $R = B$ predicts $S = Sl$ 100 percent of the time and $R = Sm$ is not a great predictor of S but the probability of $R = Sm$ is not that high. Therefore we can expect R to be a better predictor than the noisier predictors G and L. Therefore splitting on R will lead to more information gain.

Question 2

Consider the following table of observations,

Y	**1**	**1**	**0**	**1**	**0**	**0**	**0**	**1**	**1**	**0**
X_1	0	0	0	0	0	1	1	1	1	1
X_2	1	1	0	1	1	0	0	0	1	0

Given Y is the target variable and X_1, X_2 are predictors in the data above. Find which predictor will be used to make the first split in a decision tree classifier (also known as a classification tree).

Answer 2

Decision trees search all the predictors to find the split that maximizes the information gain using the formula below;

$$IG(Y, X_i) = H(Y) - H(Y|X_i)$$

H is entropy function in the above equation and $H(Y)$ is the initial entropy of the target variable before making the split.

$H(Y|X_i)$ is the entropy of the leaves after the split using predictor X_i. Therefore, information gain is basically the change in the entropy due to split and it is maximised when the entropy after split i.e $H(Y|X_i)$ is minimised. Hence, the first split will be done using the predictor X_i that minimises $H(Y|X_i)$.

Because the entropy is decreasing after split, It will create a more pure leaf -- This can be examined by comparing the entropy of a pure leaf and non-pure leaf. We want to end up with pure leaves which means that all/most of the examples falling into that leave are in the same class. This will make our model's predictions more accurate as all/most of the training examples satisfying the conditions leading to that leaves have the same target class y_i. This can be observed in the given image below. By splitting the data into groups-- i.e 3 groups obtained-- based on the values of predictors, we end up with more pure groups. In the figure, we have 3 leaves and those leaves are more pure than the pre-split state of the data.

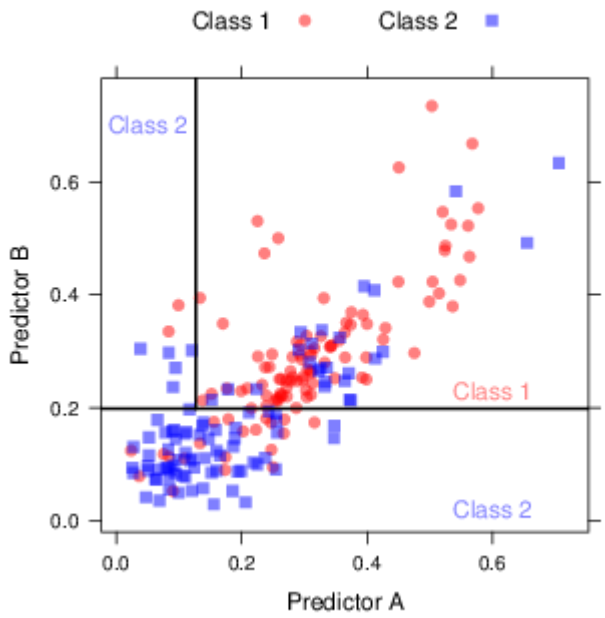


Figure 1. Predictor space of a classification tree with continuous predictors

As you can see in the Figure 1 decision trees splits the predictor space into sub-spaces. Each subspace is a leaf in the decision tree. Purpose of split is creating a more pure sub-spaces and this purity is measured using entropy. The lower entropy means a more pure sub space.

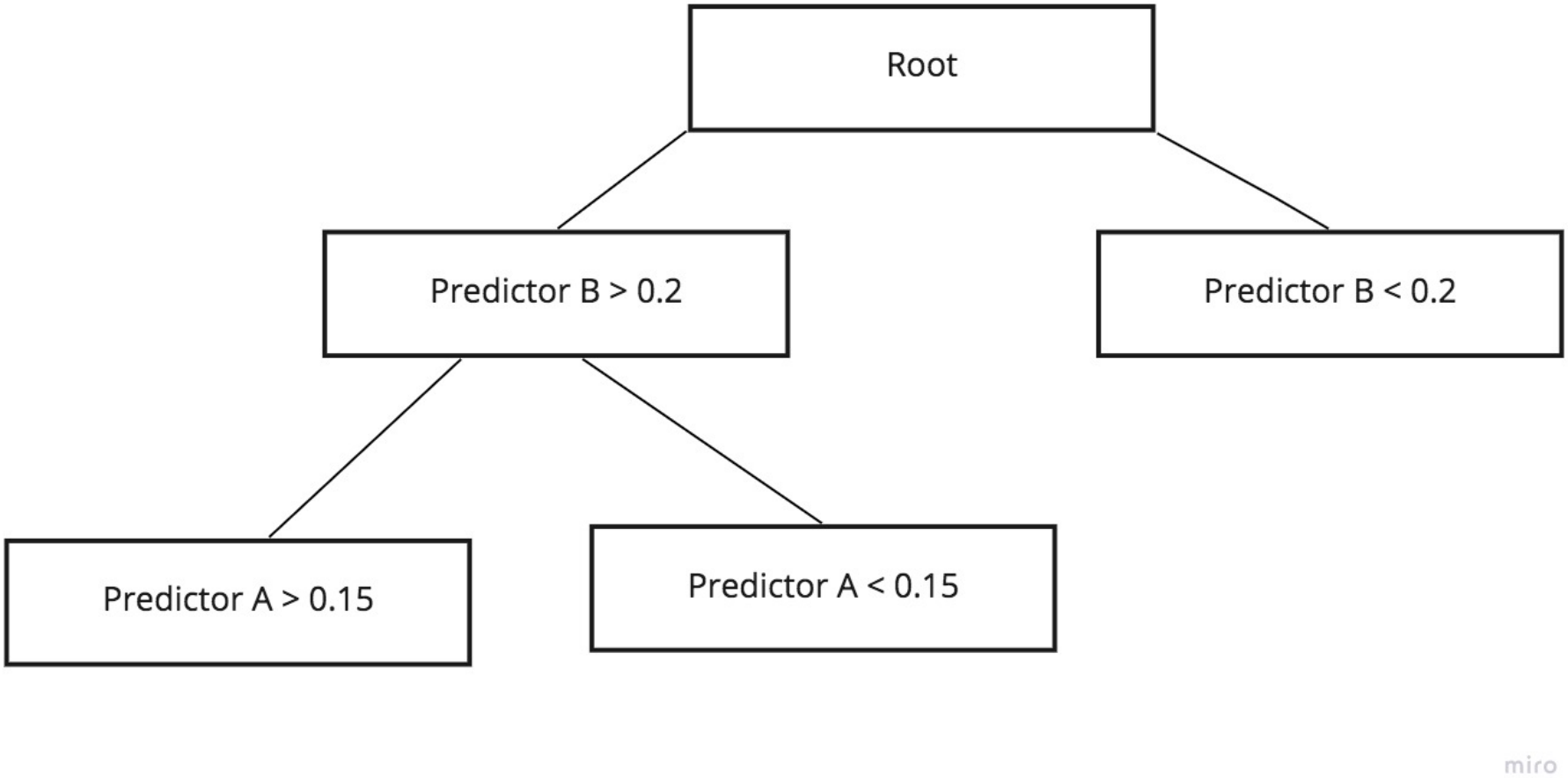


Figure 2. Splits of Predictors

Given the above information, we can start the calculations for the given dataset in the question by writing the formula for entropy.

$$H(Y) = \mathbf{E}\left[\log_2 \frac{1}{P(Y)}\right]$$

$$H(Y) = \sum_{i=1}^{K} P(Y = y_i) \log_2 \frac{1}{P(Y = y_i)}$$

In the given dataset $Y \subseteq \{0, 1\}$. Hence;

$$H(Y) = P(Y = 0) \log_2 \frac{1}{P(Y = 0)} + P(Y = 1) \log_2 \frac{1}{P(Y = 1)}$$

$$P(Y = 0) = \frac{5}{10} = 0.5$$
$$P(Y = 1) = \frac{5}{10} = 0.5$$

$$H(Y) = 0.5 \log_2 \frac{1}{0.5} + 0.5 \log_2 \frac{1}{0.5} = 1$$

According to above finding, initial entropy 1. Now entropy of the leaves after splitting on X_1 and X_2 will be calculated.

$$H(Y|X_j) = \sum_{k=1}^{N} P(X_j = x_k) H(Y|X_j = x_k)$$

Above equation tells us to take the expected value--can thought as weighted average operation as well-- of the entropies of the new leaves -- weights are the percentage of the examples falling into that leaf after split which is $P(X_j = x_k)$.

Also;

$$H(Y|X_j = x_k) = \sum_{i=1}^{K} P(Y = y_i|X_j = x_k) \log_2 \frac{1}{P(Y = y_i|X_j = x_k)}$$

$H(Y|X_j = x_k)$ is the entropy of the leaf that satisfies $X_j = x_k$ condition.

Below calculating the required probabilities from the given data to use in the above equations; $X_1, X_2 \subseteq \{0, 1\}$. Hence;

$$P(X_1 = 0) = \frac{5}{10} = 0.5$$
$$P(X_1 = 1) = \frac{5}{10} = 0.5$$

$$P(X_2 = 0) = \frac{5}{10} = 0.5$$
$$P(X_2 = 1) = \frac{5}{10} = 0.5$$

$$P(Y=1|X_1=0) = \frac{3}{5} = 0.6$$
$$P(Y=0|X_1=0) = \frac{2}{5} = 0.4$$
$$P(Y=1|X_1=1) = \frac{2}{5} = 0.4$$
$$P(Y=0|X_1=1) = \frac{3}{5} = 0.6$$

$$P(Y=1|X_2=0) = \frac{1}{5} = 0.2$$
$$P(Y=0|X_2=0) = \frac{4}{5} = 0.8$$
$$P(Y=1|X_2=1) = \frac{4}{5} = 0.8$$
$$P(Y=0|X_2=1) = \frac{1}{5} = 0.2$$

Now, calculating the entropy for the leaf gain for X_1 using the equation (4);

Entropy of $X_1 = 0$ leaf;

$$H(Y|X_1=0) = P(Y=0|X_1=0)\log_2 \frac{1}{P(Y=0|X_1=0)} + P(Y=1|X_1=0)\log_2 \frac{1}{P(Y=1|X_1=0)}$$

$$H(Y|X_1=0) = 0.4\log_2 \frac{1}{0.4} + 0.6\log_2 \frac{1}{0.6} \approx 0.9710$$

Entropy of $X_1 = 1$ leaf;

$$H(Y|X_1=1) = P(Y=0|X_1=1)\log_2 \frac{1}{P(Y=0|X_1=1)} + P(Y=1|X_1=1)\log_2 \frac{1}{P(Y=1|X_1=1)}$$

$$H(Y|X_1=1) = 0.6\log_2 \frac{1}{0.6} + 0.4\log_2 \frac{1}{0.4} \approx 0.9710$$

Then taking the expected value of those 2 entropies -- using equation (3) -- as follows;

$$H(Y|X_1) = P(X_1=0)H(Y|X_1=0) + P(X_1=1)H(Y|X_1=1)$$

$$H(Y|X_1) = 0.5.0.9710 + 0.5.0.9710 = 0.9710$$

Repeating the same steps for X_2 we find;

$$H(Y|X_2) = 0.7219$$

$$H(Y|X_2) < H(Y|X_1)$$

so we decide to split on X_2.

Question 3

Consider the data,

$\mathbf{X_1}$ =Acid Durability(sec)	$\mathbf{X_2}$ =Strength(kg/m^2)	$\mathbf{Y}$
7	7	"Bad"
7	4	"Bad"
3	4	"Good"
1	4	"Good"

We have the above data of a survey that asks people's opinion for different tissues. Predictors are acid durability X_1 and strength X_2 of a special paper tissue. Target variable Y is the people's opinion about the tissue i.e "Good" or "Bad".

Given the above data, now the factory produces a new paper tissue that passes laboratory tests with $X_1 = 3$ and $X_2 = 7$. Using the K-nearest neighbors (KNN) algorithm, find the classification of this new tissue, i.e is it "Good" or "Bad"?

Answer 3

Let's start by explaining the steps of KNN algorithm.

1-) We need to select a value for the parameter K=Number of nearest neighbours to consider while classifying each example.

2-) Calculate the distance between the test data and all the training points. This is required as we need to find out which training examples are the nearest to the test example.

3-) Sort distances calculated in step 2 to determine K number of training points that are nearest to test.

4-) Gather the values of target variable for those K nearest neighbours.

5-) Find the most common target variable value for those K nearest neighbours and this becomes the KNN's prediction for test data point. This is called majority voting. This step would be slightly different if the KNN is used for a regression problem though previous steps are same.

Given above steps, now we can apply those steps on our dataset.

1-) Selecting K

Ideally, we need to experiment with different K values to find the one that leads to the best predictive performance. This process is called hyperparameter tuning as K is a hyperparameter of KNN. For the purpose of this example we will skip this step and just assume that K=3 and do the remeaning calculations accordingly.

2-) Calculating distance between test example and the training examples

Because our purpose is determining the nearest examples, we can calculate the squared distance instead of distance as they both will determine the same training examples as the k-nearest ones. Purpose in here is simplifying the calculation.

Coordinates of the test instance in predictor space is (3,7). Hence;

X_1 =Acid Durability(sec)	X_2 =Strength(kg/m^2)	Square distance to test instance (3,7)
7	7	$(7-3)^2+(7-7)^2=16$
7	4	$(7-3)^2+(7-4)^2=25$
3	4	$(3-3)^2+(4-7)^2=9$
1	4	$(1-3)^2+(4-7)^2=13$

3-) Sort the distances and determine K nearest training instances

X_1 =Acid Durability(sec)	X_2 =Strength(kg/m^2)	Square distance to test instance (3,7)	Rank according to minimum distance	Is it included in 3 nearest neighbours?
7	7	$(7-3)^2+(7-7)^2=16$	3	Yes
7	4	$(7-3)^2+(7-4)^2=25$	4	No
3	4	$(3-3)^2+(4-7)^2=9$	1	Yes
1	4	$(1-3)^2+(4-7)^2=13$	2	Yes

4-) Gather the values of target variable for those K nearest neighbours

X_1 ≡Acid Durability(sec)	X_2 ≡Strength(kg/m^2)	Square distance to test instance (3,7)	Rank according to minimum distance	Is it included in 3 nearest neighbours?	Y ≡ Category of nearest neighbour
7	7	$(7-3)^2+(7-7)^2=16$	3	Yes	Bad
7	4	$(7-3)^2+(7-4)^2=25$	4	No	-
3	4	$(3-3)^2+(4-7)^2=9$	1	Yes	Good
1	4	$(1-3)^2+(4-7)^2=13$	2	Yes	Good

Notice in the second row, value of Y is not included because the rank of the data is bigger than 3(=K).

5-) Apply majority vote

We got 2 "Good" and 1 "Bad" then KNN's prediction with K=3 becomes "Good" as majority of K nearest examples have "Good" as label.

R hackers bone-crunching challenge

Using the 2 attached csv files (same files as the week 9 quiz): train.csv for training your models, and test.csv for testing your models on data they were not trained on, implement Decision Tree and KNN classifiers with R code.

The provided files consists of 3 variables: $x1$ and $x2$ (these two variables are the input/predictors), $label$ (the output/target).

a) Use the $rpart$ function in R with default settings.

- Learn using train.csv to generate the classification model of a Decision Tree
- Plot the resulting decision tree
- Predict the class labels for test.csv
- Compute accuracy of the trained model on the test data. (Accuracy is defined as the percentage of observations correctly classified. i.e. the percentage of observations for which the predicted class label matches the true class label)

Note $rpart$ is not part of the R base packages so you may need to use the following syntax to install it:

```
install.packages("some package", repos='http://cran.us.r-project.org')
```

```
library("some package")
```

b) Use the knn function in R

- Let the number of nearest neighbors be k=1, use train.csv and test.csv to get the accuracy of KNN classfier
- Let k=5, use train.csv and test.csv to get the accuracy of KNN classfier
- Plot and compare the accuracies for different values of k within the range [1,100], try to understand the overfitting and underfitting tradeoff when choosing k.

Answer

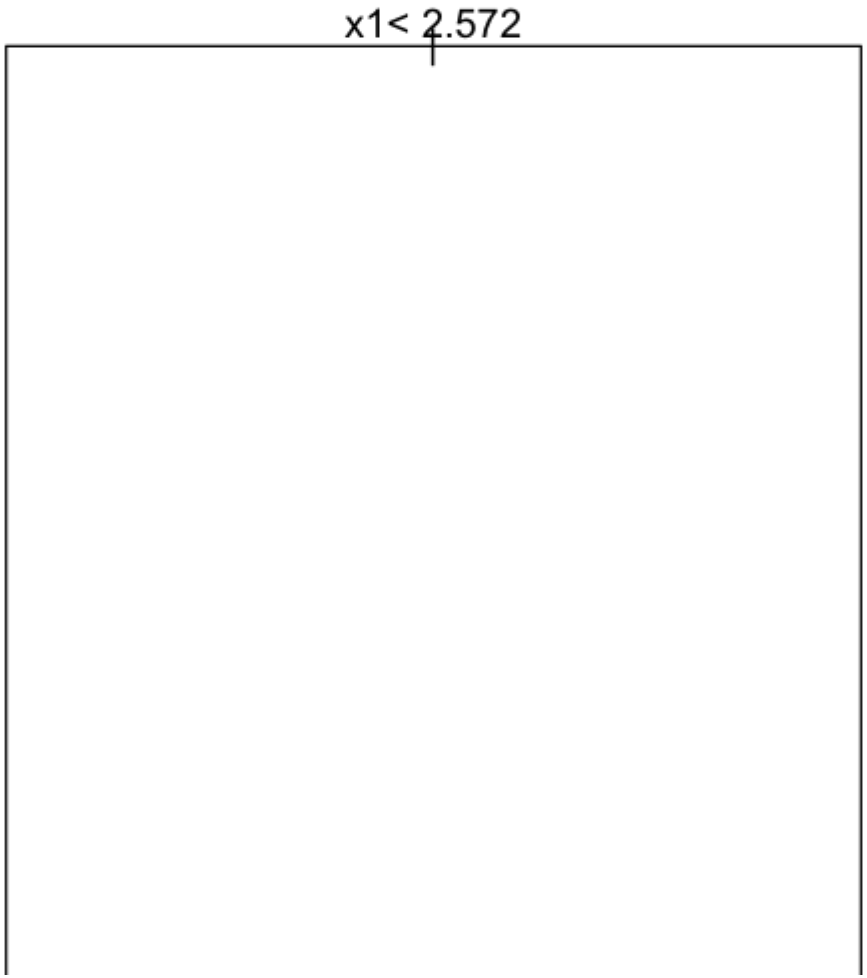
a) Decision Tree

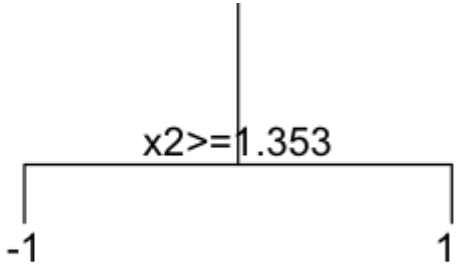
In [1]:
```
# Decision Tree
# Read the data from CSV files
train_X <- read.csv('train.csv',sep=',',header=TRUE)[2:4]
test_X <- read.csv('test.csv',sep=',',header=TRUE)[2:4]
```

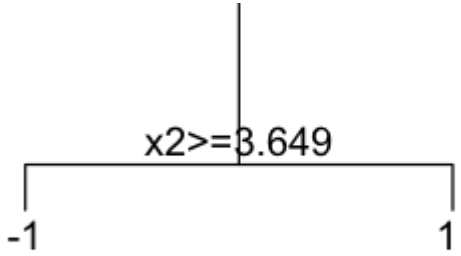
In [2]:
```
# Change the data types of the column lable into the format of characters.
# Otherwise, when do the classification, the data will be treated as the type of doubles
# leading to the values in the tree calculated into decimal rather than 1 or -1
train_X$label <- as.character(train_X$label)
test_X$label <- as.character(test_X$label)
# Check the data types for each column
str(train_X)
```

```
'data.frame':   500 obs. of  3 variables:
 $ x1   : num  0.827 0.79 3.477 2.222 3.597 ...
 $ x2   : num  1.6792 3.1508 -0.4517 3.4584 0.0173 ...
 $ label: chr  "-1" "-1" "1" "-1" ...
```

In [3]:
```
# Load the necessary lib rpart to generate the tree
library(rpart)
train_tree <- rpart(label ~., data = train_X)
# Plot the tree model into the diagram with the values displayed
plot(train_tree)
text(train_tree, pretty = 0)
```







In [4]:
```
#Show the tree in the textual format with different levels
train_tree
```

```
n= 500

node), split, n, loss, yval, (yprob)
      * denotes terminal node

1) root 500 187 1 (0.374000000 0.626000000)
  2) x1< 2.571841 188  10 -1 (0.946808511 0.053191489)
    4) x2>=1.352538 179   1 -1 (0.994413408 0.005586592) *
    5) x2< 1.352538 9   0 1 (0.000000000 1.000000000) *
  3) x1>=2.571841 312   9 1 (0.028846154 0.971153846)
    6) x2>=3.648816 7   0 -1 (1.000000000 0.000000000) *
    7) x2< 3.648816 305   2 1 (0.006557377 0.993442623) *
```

In [5]:
```
#Predict the test data using trained tree model
# 1st parameter in the predict function will be the trained tree model
# 2nd parameter should be the data frame storing the test data
# 3rd parameter is used to declare the model will be a classification model
test_pred = predict(train_tree, test_X, type = "class")
# Save the result of prediction into the table, and compare the values with actual values in the test dataset
pred_table <- table(Predicted_Class = test_pred, Actual_Class = test_X$label)
print(pred_table)
# Based on the table, 203 records are predicted to be -1 and 294 records are predicted to be 1 correctly.
# 3 records are predicted to be 1 while the actual values are -1
```

```
                Actual_Class
  Predicted_Class  -1   1
               -1 203   0
               1    3 294
```

In [6]:
```
#Calculate the testing accuracy of the trained tree model
tree_accuracy <- round((pred_table[1,1]+pred_table[2,2])/(pred_table[1,1]+pred_table[1,2]+pred_table[2,1]
                                                          +pred_table[2,2])*100,2)
cat("The accuracy of this tree model is ", tree_accuracy)
```

```
  The accuracy of this tree model is  99.4
```

b) K-nearest Neighbor

In [7]:
```
# Read the data from csv files, and save into the format of data frame
train_Knn <- read.csv('train.csv',sep=',',header=TRUE)[2:4]
test_Knn <- read.csv('test.csv',sep=',',header=TRUE)[2:4]
```

For the algorithm where distance plays an important role for classification, normalising the variables is necessary to avoid miss classification. When we are using KNN, the distances between paris of samples are measured. While these distances are influenced by the measurement units of different variables. Without normalisaton on all of the features, some features may have larger effect on the prediction capability. So in order to make all the features equally accountable for the prediction capability of algorithm KNN, we will use Min-Max to complete the normalisation.

In [8]:
```
#Define the function to do the normalisation
nor <- function(x){(x -min(x))/(max(x)-min(x))}
```

In [9]:
```
# Apply the defined function to normalise the training and testing data. Convert the values of lable into characters
train_df <- as.data.frame(lapply(train_Knn, nor))
train_df$label <- as.character(train_df$label)
test_df <- as.data.frame(lapply(test_Knn, nor))
test_df$label <- as.character(test_df$label)
str(train_df)
```

```
'data.frame':   500 obs. of  3 variables:
 $ x1   : num  0.276 0.272 0.561 0.426 0.574 ...
 $ x2   : num  0.442 0.614 0.193 0.65 0.248 ...
 $ label: chr  "0" "0" "1" "0" ...
```

In [10]:
```
train_KnnX <- train_df[1:2]
test_KnnX <- test_df[1:2]
test_label <- test_df$label
# Check the length of the training and testing data to make sure the lengths are the same so that in can be applied
# in the knn function to do the prediction
dim(train_KnnX)
dim(test_KnnX)
length(train_df$label)
# Load the required lib of class to make use the function: knn
library(class)
```

```
500  2
```

```
500  2
```

```
500
```

K=1

In [11]:
```
# We set a random seed before we apply knn() because if several observations are tied as nearest neighbors,
# then R will randomly break the tie. Therefore, a seed must be set in order to ensure reproducibility of results
et.seed(1)
# knn() is used to predict the values for test data using the training data directly
knn_pred1 = knn(train_KnnX, test_KnnX, train_df$label, k=1)
table(knn_pred1, test_label)
```

```
           test_label
  knn_pred1   0   1
          0 200   0
          1   6 294
```

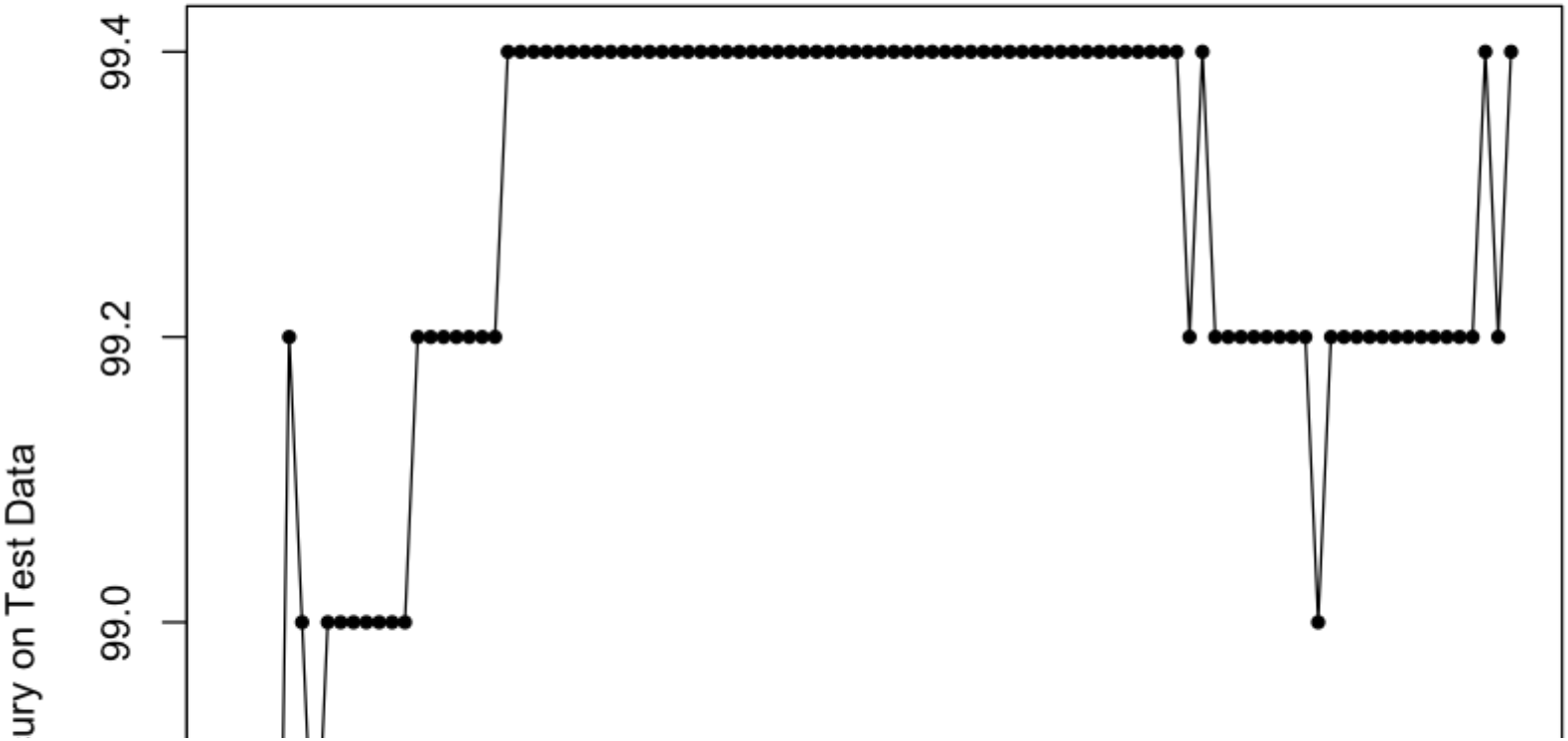
K=5

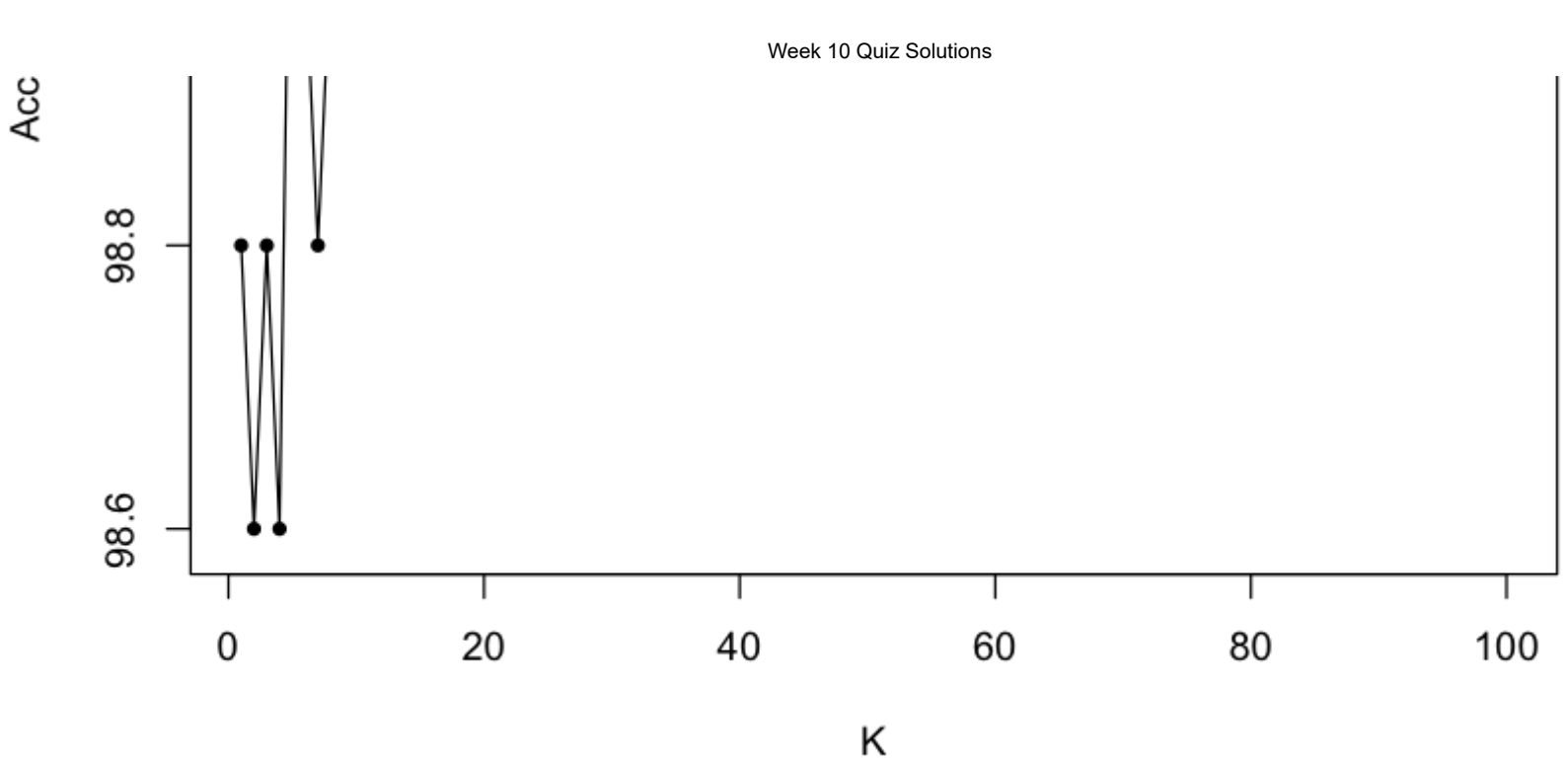
In [12]:
```
set.seed(1)
knn_pred3 = knn(train_KnnX, test_KnnX, train_df$label, k=5)
table(knn_pred3, test_label)
```

```
           test_label
  knn_pred3   0   1
          0 202   0
          1   4 294
```

Compare the accuracies for the value of K within [1,100]

In [13]:
```
# Save the pairs of k and accuracy into the data frame to plot the graph
# declare the name of data frame and the varibale
df <- NULL
correct <- rep(0,10)
# Use for loop to run the knn() with different values ranging from 1 to 100
for(i in 1:100){
    set.seed(1)
    fit_pre <- knn(train_KnnX, test_KnnX, train_df$label,k=i)
    t <- table(fit_pre, test_label)
    correct[i] <- round((t[1,1]+t[2,2])/(t[1,1]+t[1,2]+t[2,1]+t[2,2])*100,2)
    df <- rbind(df, data.frame(i, correct[i]))
}
plot(df, type="o", pch=20, xlab = "K",ylab = 'Accuracy on Test Data')
```





Usually, underfitting implies high bias and low variance, which means the model does not predict on the training data very well. While overfitting implies low bias and high variance, which means the model predicts the training data too well, but the prediction may be wrong for new data points in the test data.

For the KNN classifier, when K=1, this is the extreme case where the training data will be perfectly predicted (i.e. overfit). However, for the test data, it will have a higher chance to produce an error. When increasing K, the training error will increase, while the change of the test error varies for different data sets. Generally, the accuracy of the test data may increase first and then decline.

According to the plotted graph, we can determine the K to prevent the model from underfitting or overfitting.