Week 9 Quiz - Classification and Clustering - Solutions

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.

Question 1

Build a Naive Bayes classifier.

We have 1000 fruits which could be either ‘banana’, ‘orange’ or ‘other’. These are the 3 possible classes of the Y variable. For the following X variables (Long, Sweet, Yellow), all of data are binary (1 or 0). The first few rows of the training dataset look like this:

Fruit	Long(x_1)	Sweet(x_2)	Yellow(x_3)
Orange	0	1	0
Banana	1	0	1
Banana	1	1	1
Other	1	1	0
....	....	....	....

For the sake of computing the probabilities, let’s aggregate the training data to form a counts table like this:

Type	Long	Not Long	Sweet	Not Sweet	Yellow	Not Yellow	Total
Banana	400	100	350	150	450	50	500
Orange	0	300	150	150	300	0	300
Other	100	100	150	50	50	150	200
Total	500	500	650	350	800	200	1000

So the objective of the classifier is to predict if a given fruit is a 'Banana' or ‘Orange’ or ‘Other’ when only the 3 features (long, sweet and yellow) are known. Being given a fruit that is: Long, Sweet and Yellow, can you predict what fruit it is?

Answer 1

First you need to look up the probability prediction formula for the naive Bayes classifier in the 'Classification and Clustering' section of the formula sheet to see what probabilities you need to compute. We'll start with computing the probabilities of the numerator on the right hand side (RHS) of the formula, as you can always get the marginal distribution in the denominator if you compute the numerator for the different values of the target (i.e. fruit is 'Banana', 'Orange' or 'Other') and sum over these probabilities to get the marginal probability.

Step 1: Compute the 'Prior' probabilities for each of the class of fruits. That is, the proportion of each fruit class out of all the fruits from the population. You can provide the 'Priors' from prior information about the population. Otherwise, it can be computed from the training data.

For this case, let's compute from the training data. Out of 1000 records in training data, we have 500 Bananas, 300 Oranges and 200 Others. So the respective priors are 0.5, 0.3 and 0.2.

$$P[Y = Banana] = 500/1000 = 0.5$$
$$P[Y = Orange] = 300/1000 = 0.3$$
$$P[Y = Other] = 200/1000 = 0.2$$

Step 2: Compute the conditional probabilities in the numerator required to compute $P(Banana|Long, Sweet, Yellow)$**.** To compute this we need the product of conditional probabilities of the 3 features conditioned on 'Banana'. Refer to the naive Bayes formula, it says $P(x_j|y)$. Here we can treat x_1 as 'Long' and y as 'Banana'. That means the probability the fruit is 'Long' given that it is a 'Banana'. In the above table, we have 500 Bananas. Out of that 400 are long. So, $P(Long|Banana) = 400/500 = 0.8$. Based on the table for the 3 features we can say:

$$P[x_1 = Long|Y = Banana] = 400/500 = 0.8$$
$$p[x_2 = Sweet|Y = Banana] = 350/500 = 0.7$$
$$P[x_3 = Yellow|Y = Banana] = 450/500 = 0.9$$

Step 3: Substitute the above probabilities into the numerator on the RHS of the Naive Bayes formula, to get the joint probability $P(Banana, Long, Sweet, Yellow)$**.**

$$P(Banana, Long, Sweet, Yellow) = P(Long|Banana) * P(Sweet|Banana) * P(Yellow|Banana) \times P(Banana)$$
$$= 0.8 \times 0.7 \times 0.9 \times 0.5 = 0.252$$

Step 4: Apply the same procedure from steps 2 and 3 to get $P(orange, Long, Sweet, Yellow)$ **and** $P(other, Long, Sweet, Yellow)$**.**

Applying similar steps we can see

$$P(Orange, Long, Sweet, Yellow) = 0, \text{because } P(Long|Orange) = 0$$
$$P(Other, Long, Sweet, Yellow) = 0.01875$$

Step 5: Now lets compute the marginal probability on the denominator on the RHS of the naive Bayes formula $P(Long, Sweet, Yellow)$**.**

We can see

$$P(Long, Sweet, Yellow) = P(banana, Long, Sweet, Yellow) + P(orange, Long, Sweet, Yellow) + P(other, Long, Sweet, Yellow)$$
$$= 0.252 + 0 + 0.01875 = 0.27075.$$

Step 6: Now lets compute the exact probabilities of the target given the predictor ($P(Banana|Long, Sweet, Yellow)$, $P(Orange|Long, Sweet, Yellow)$ **and** $P(other|Long, Sweet, Yellow)$**) to see which fruit is more likely.**

$$P(Banana|Long, Sweet, Yellow) = P(banana, Long, Sweet, Yellow)/P(Long, Sweet, Yellow)$$
$$= 0.252/0.27075 = 0.93$$
$$P(Orange|Long, Sweet, Yellow) = P(orange, Long, Sweet, Yellow)/P(Long, Sweet, Yellow)$$
$$= 0/0.27075 = 0$$
$$P(Other|Long, Sweet, Yellow) = P(other, Long, Sweet, Yellow)/P(Long, Sweet, Yellow)$$
$$= 0.01875/0.27075 = 0.07$$

So we can see 'Banana' has the highest probability and so is the most likely fruit if the fruit is 'Long', 'Sweet' and 'Yellow'. Note if a question doesn't ask you to explicitly compute the exact probabilities you can ignore calculation of the demoniminator, i.e. the marginal probability $P(Long, Sweet, Yellow)$, as it does not effect the final classification decision.

Question 2

Logistic Regression Process and Intuition.

In the lecture on logistic regression we defined the negative log-likelihood expression that is minimised to estimate the model parameters but we didn't go into the details of the underlying estimation algorithm. Although we wouldn't ask you about the estimation algorithm in the exam, here we summarise the algorithm to give you some deeper intuition to help you understand what machine learning is. The steps of training a logistic regression classifier are described below;

A data set consists of vectors of dependent variables (predictors) and a vector of labels. Vectors of p predictors will be represented by $(x_{i,1}, x_{i,2}, \ldots, x_{i,j}, \ldots, x_{i,p})$ where j indexes the predictors and i indexes the n data observations. Therefore, we have n data points to train the algorithm with. y_i is the class label for the ith data point.

Given above the information about the data, the first step of logistic regression is random initialisation of the weight values of each predictor (also known as regression coefficients) then multiplying them with predictors as follows;

$$\alpha_i = w_0 + w_1 x_{i,1} + w_2 x_{i,2} \ldots + w_p x_{i,p}$$

Note the w_j and α_i are the same as the β_j and η_i considered in the lecture, respectively. We are just changing the symbols used since people don't always use the same notation (Often people refer to the parameters as weights in neural networks and use the symbol w_j. Logistic regression can be thought of as a single neuron model). Then we input the output α_i into the sigmoid function and get the probability predictions as follows;

$$\hat{y_i} = \frac{1}{1 + e^{-\alpha_i}}$$

The value of the cost function i.e. log-loss (another way of saying negative log-likelihood divided by the number of observations in this case) is calculated for current values of weights (w) using the below equation;

$$cost(w) = \frac{-1}{n} \sum_{i=1}^{i=n} y_i log(\hat{y_i}) + (1 - y_i) log(1 - \hat{y_i})$$

Then the derivative of the cost function i.e gradient is calculated (remember in the simple linear regression lecture, how we take the partial deriviative with respect to β_0 and β_1, that's basically what we are doing here with the w_j although we don't bother you with the math used to derive these algorithmic calculations, we only show the final form of the algorithmic calculations here):

$$dw_j = \sum_{k=1}^{k=p} (\hat{y_i} - y_i) x_{i,k}.$$

Finally the weight parameters are updated using the gradients:

$$w_j = w_j - dw_j * \eta$$

where η is a hyperparameter called learning rate. Hyperparameters are parameters of the algorithm selected by the user, not parameters of the model optimised by the algorithm, like w_j.

This iterative process can be summarised as;

(1) Calculating α.

(2) Using the sigmoid function to turn α into probability predictions i.e $\hat{y}$.

(3) Calculating the cost value for values of w in the current iteration. -- This step is not necessary for training as gradients can directly be calculated using the derivative of cost equation. Knowing the cost value in different iterations provides extra information about training process.

(4) Calculating gradients i.e dw_j.

(5) Updating the weights i.e w.

Considering those 5 steps above;

a-) In what step(s) of the algorithm do you think "learning" happens?

b-) What is the learning step trying to optimise?

c-) What is the purpose of step 2? Why do we need to use the sigmoid function to turn the output of the linear equation into probability predictions?

d-) What does α_i represent?

e-) What does the gradient (dw_j) represent?

This question is adapted from the medium post: here (https://medium.com/analytics-vidhya/logistic-regression-b35d2801a29c)

Warning: Includes spoiler if you haven't attempted the questions.

Answer 2

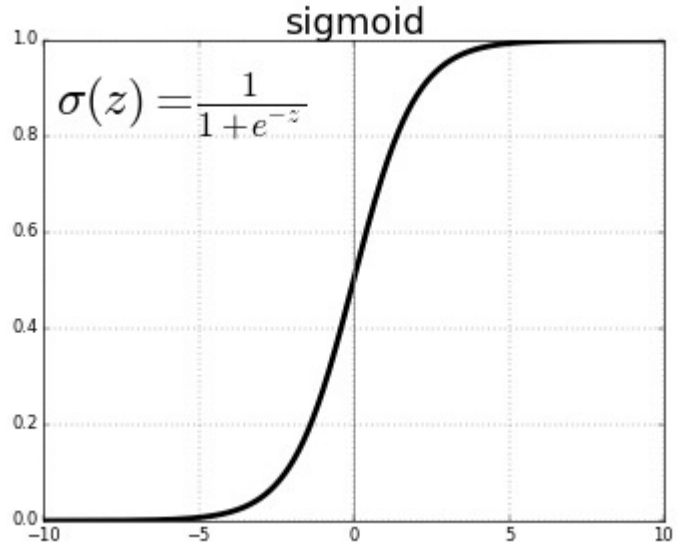
a-) Our model is defined by its weights (w) -- coefficients of the linear equation. Once we update them, we change our model and we do this update in a way that new weights gives a lower cost value -- output of the log-loss cost function. This process of updating the weights of the model is where the learning happens.

Therefore, learning happens at step 5.

b-) The learning step updates the weights in a way that new weights give a lower cost value. Therefore, learning tries to optimise/minimize the cost function i.e log-loss/negative log-likelihood.

c-) The goal of the logistic regression algorithm is to create a linear decision boundary separating two classes from one another and calculate the conditional class probabilities $P(y=1|x;w)$ and $P(y=0|x;w)$ for the given data point x and given weights w.

The value of α potentially can be any value from negative infinity to positive infinity. We need to narrow it down to a score that is in between zero and one as probabilities always are in that range and logistic regression is interested in probabilities in order to decide which of $y=1$ or $y=0$ is more likely given the data x. The Sigmoid function makes this transformation on the output of the linear equation i.e α.



Looking the figure above, the sigmoid function maps any input between $-\inf$ and $\inf$ to between 0 and 1. This helps transforming the output of the linear equation into class probability predictions.

d-) We know that we turn the output of the linear equation into class probabilities by applying the sigmoid transformation. If we express α in terms of $\hat{y}$, we can understand what does α tell us about class probabilities.

Starting with;

$$\hat{y_i} = \frac{1}{1 + e^{-\alpha_i}}$$

$$e^{-\alpha_i} = \frac{1}{\hat{y_i}} - 1$$

$$e^{-\alpha_i} = \frac{1 - \hat{y_i}}{\hat{y_i}}$$

$$\alpha_i = \log \frac{\hat{y_i}}{1 - \hat{y_i}}$$

We know that $\hat{y_i}$ is the probability of being in one class and $1 - \hat{y_i}$ is the probability of being in the other class. The Log of the division of those two probabilities are called the \textbf{log-odds ratio}. Hence, the value of α_i is the log-odds for an input data x. The log-odds ratio is used commonly in statistics as it offers advantages in certain cases. Logistic regression uses the log-odds ratio in its definition as follows:

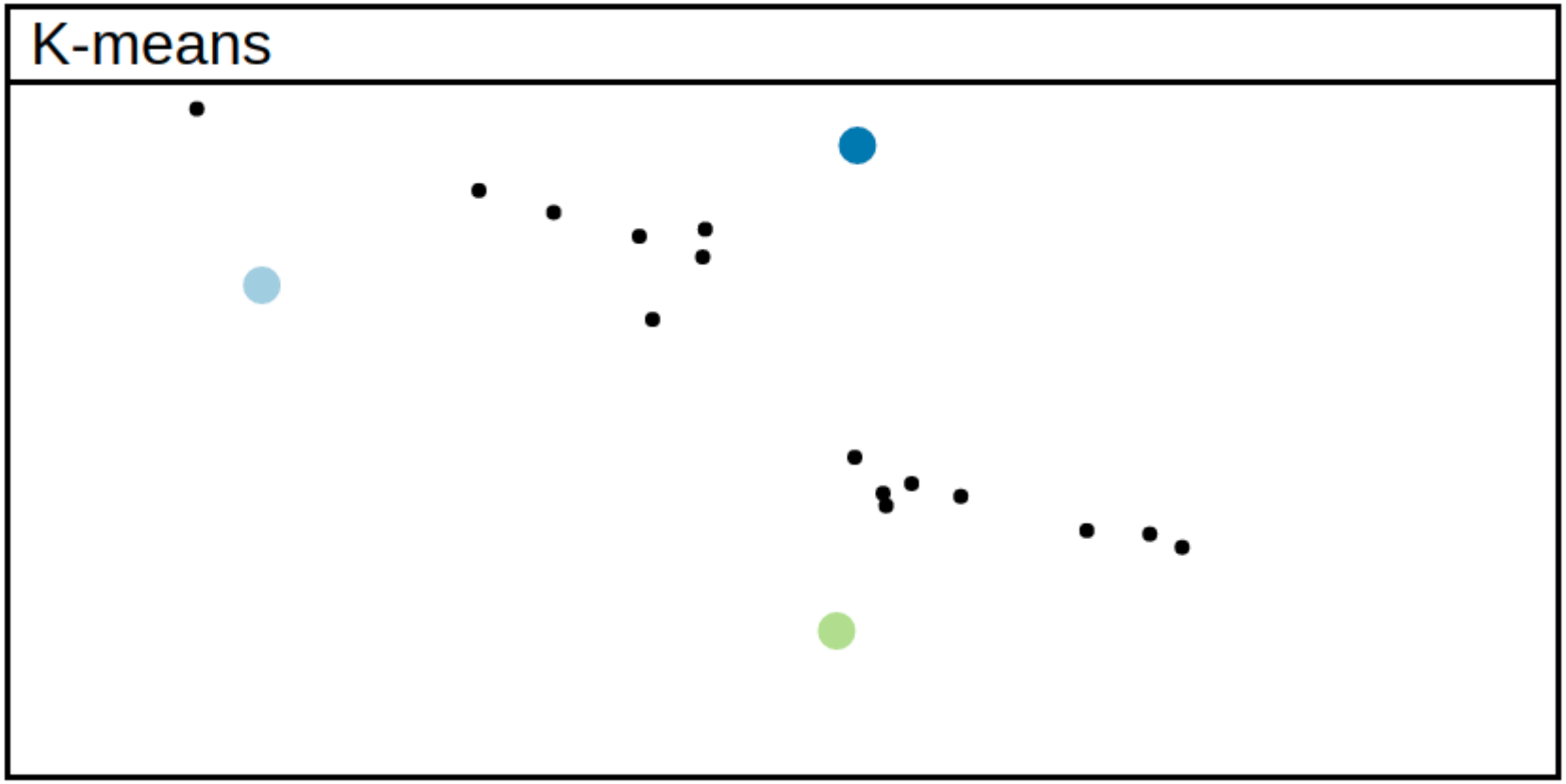
$$\log\left(\frac{P(Y_i = 1|x_{i,1}, \dots, x_{i,p})}{P(Y_i = 0|x_{i,1}, \dots, x_{i,p})}\right) = w_0 + \sum_{j=1}^{p} w_j x_{i,j} \equiv \alpha_i$$

As can be seen in the figure of the sigmoid above, using the log-odds ratio leads to symmetry of the sigmoid around the value of $z = \alpha_i = 0$. This gives the advantage that when $w_0 = 0$, it is only possible for the predictors to bias prediction towards $y_i = 0$ or $y_i = 1$ depending on their values. Therefore the parameter w_0 can be treated as a bias parameter if we wish to bias our predictions regardless of the values of the predictors, i.e. if our predictors are all zero, $w_0 < 0$ and $w_0 > 0$ will bias predictions towards $y_i = 0$ and $y_i = 1$, respectively.

e-) Gradients are the first derivative of the cost function. Hence, they describe increasing/decreasing behaviour (slope) of the cost function. Keeping in mind that the purpose of the learning process is to minimise the cost function, negative gradients define the directions that we need to update the weights towards to decrease in the value of the cost function. This way of using the derivative of the cost function to minimise the cost function is generally called a gradient descent algorithm.

Question 3

The figure below shows some points in 2-D that we wish to model with the K-means algorithm. The 3 larger round dots are the proposed initial cluster centers. The distance measure used in this case is the geometric distance between the two points.



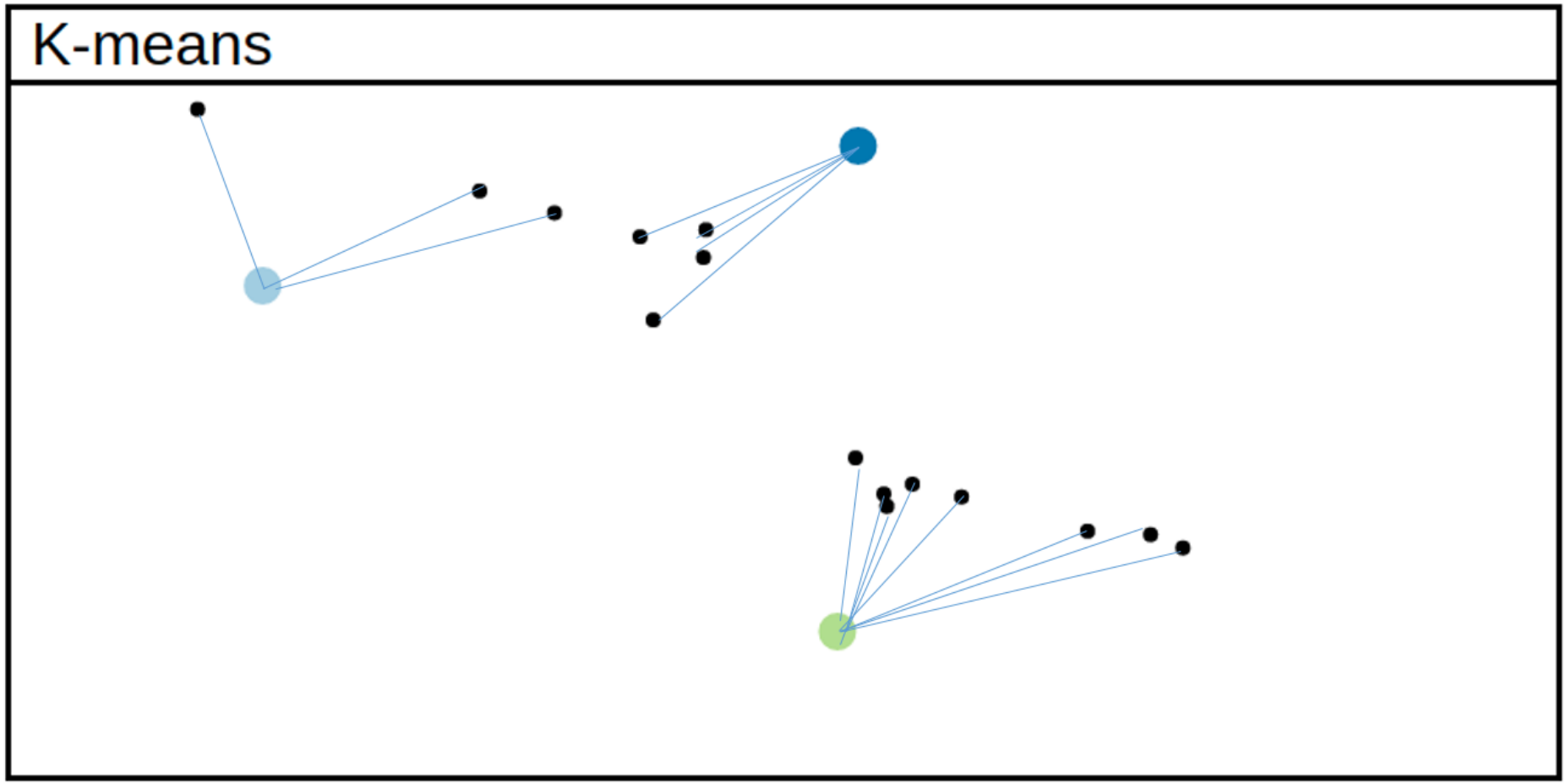
(a) Draw on the figure the results of the first step of the k-means algorithm: assigning points to cluster centers. Connect each point to its appropriate cluster center with a straight line.

(b) Draw on the figure the results of the second step of the k-means algorithm, recomputing the cluster centers.

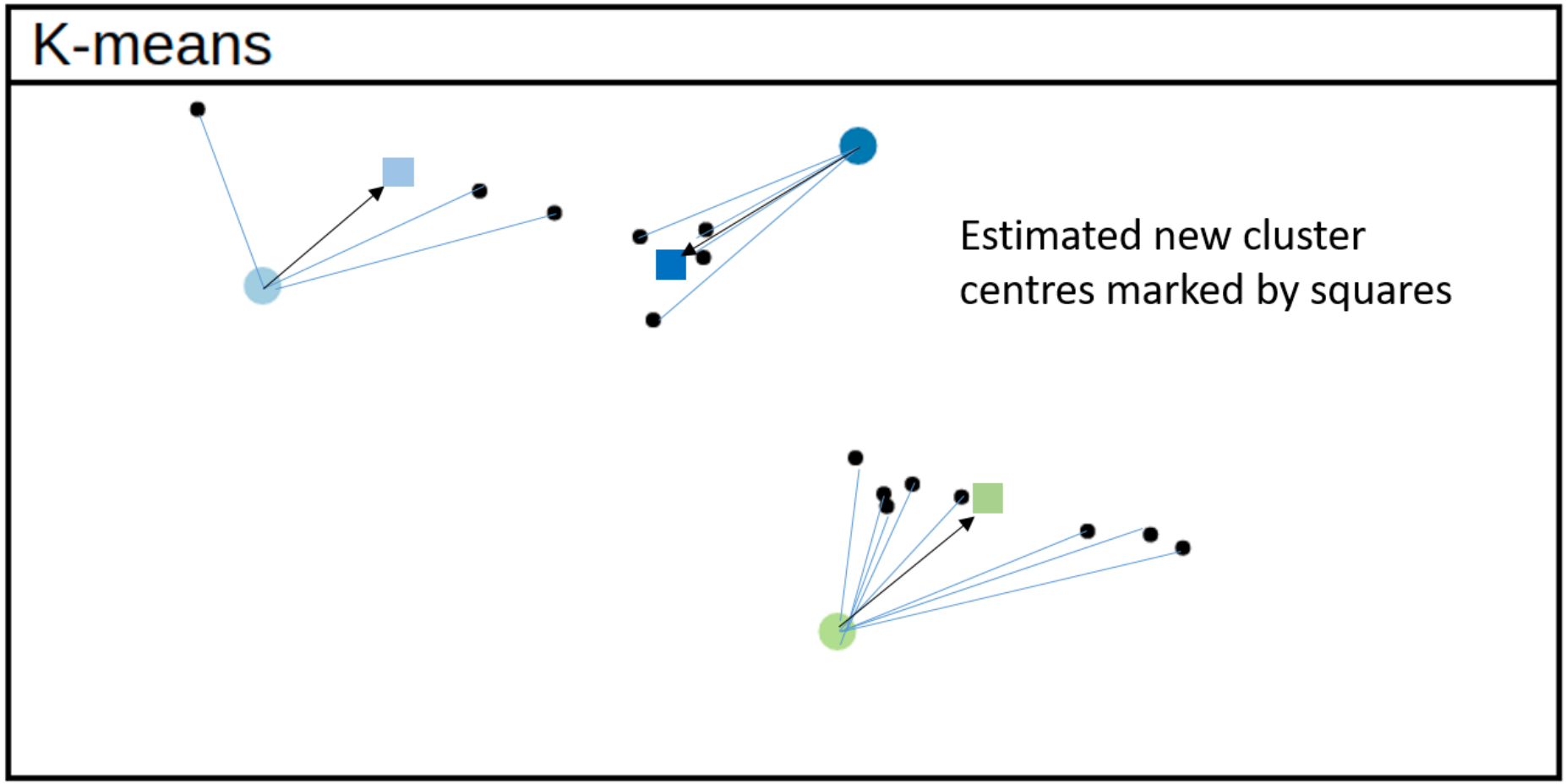
(c) Give an intuitive argument as to why the k-means algorithm is guaranteed to converge in a finite number of steps.

Answer 3

(a) Your drawing should connect each point with a straight line to its CLOSEST centroid.



(b) The new cluster centres are the centre point for each cluster, your drawing should depict these new cluster centre positions approximately.



(c) After each of the two main algorithm steps (step 1: reassign points to clusters, step 2: recenter clusters), the total distance of all points to their assigned cluster center is decreasing. Thus we can expect the algorithm to converge in a finite number of steps if the number of data points is finite.

R hackers bone-crunching challenge

In this problem you will implement naive Bayes and logistic regression classifiers. Use the 2 csv files: $train.csv$ for training your models and $test.csv$ for testing your models on data they were not trained on. The files are available here (https://www.dropbox.com/sh/j4drj1bx5qk9fjw/AADmhYZYm0uII-IT8um0BH6za?dl=0). These files have 3 variables: $x1$ and $x2$(these two variables are the input/predictors), $label$(the output/target).

a) Plot the train and test data on separate 2-dimensional graphs with axis labels, a title, and color coding of the class labels

b) Use the $naiveBayes$ function in R to get a naive Bayes classifier to learn using train.csv and then predict the class labels for test.csv. Compute accuracy of the trained model on the train and 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.

c) Use the glm function in R to get a logistic regression classifier to learn using train.csv and then predict the class labels for test.csv. Compute accuracy of the trained model on the train and test data.

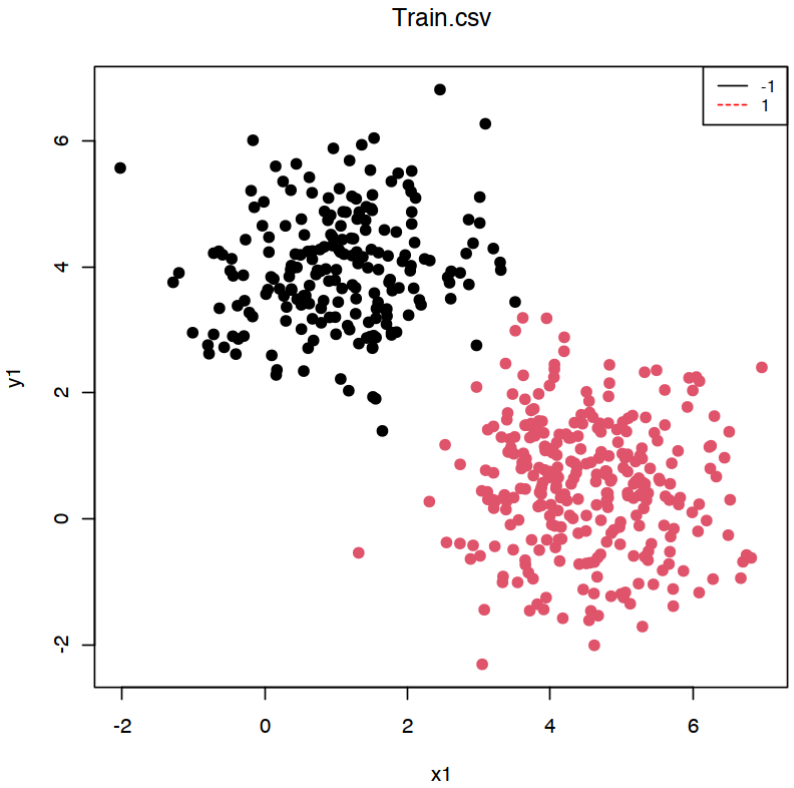
d) Compare the results of both the models. Based on the results, lecture notes and internet research, explain why you might see the observed similarities/differences in performance (accuracy) of logistic regression and naive Bayes classifier. Which algorithm was faster to run and why do you think this was so?

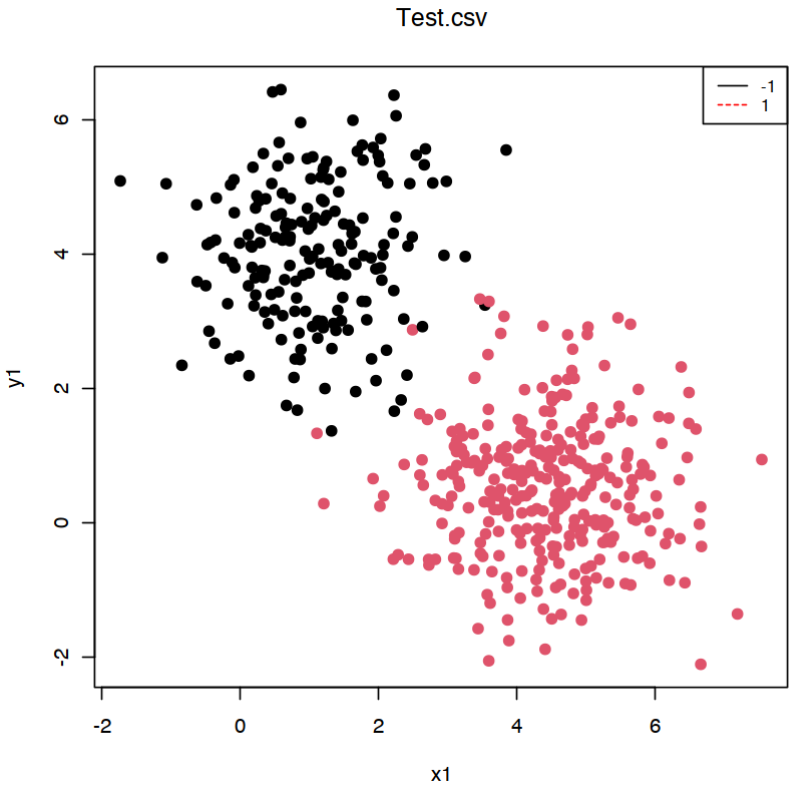
Answer

Warning: this solution applies a naive Bayes classifier designed for continous predictors and using a Gaussian parameterisation. Note we coded it ourselves to show you the inner workings of such an algorithm. For this task we asked you to use the naiveBayes() R function that is not part of the R base package set, but you should get similar results to what we get. Our naive Bayes classifier implementation still relies on the same formula covered in the formula sheet, however, the conditional probabilities $P(predictor|target)$ are assumed to follow a Gaussian distribution. Exam math questions only ask you to apply naive Bayes classification to categorical predictors like covered in the example above where Gaussian parameterisation is not relevant.

a)

In [1]:
```
train <- read.csv("train.csv")
test <- read.csv("test.csv")
train$label <- as.factor(train$label)
test$label <- as.factor(test$label)
plot(x=train$x1, y=train$x2, col=train$label, pch=19, main="Train.csv", xlab="x1", ylab="y1")
legend("topright", legend=c("-1", "1"), col=c("black", "red"), lty=1:2, cex=0.8)
plot(x=test$x1, y=test$x2, col=test$label, pch=19, main="Test.csv", xlab="x1", ylab="y1")
legend("topright", legend=c("-1", "1"), col=c("black", "red"), lty=1:2, cex=0.8)
```




Test.csv
-1
1
y1
x1
-2
0
2
4
6

b) We use the multivariate Gaussian Distribution formula to calculate the likelihood and posterior probabilities of the data points from the estimated means and covariance matrices:

$$f(\mathbf{x}) = \frac{1}{(2\pi)^{p/2}|\mathbf{\Sigma}|^{1/2}} \exp\left\{ -\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})'\mathbf{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu}) \right\}$$

For bivariate normal distribution,

$$f(x,y) = \frac{\exp\left\{ -\frac{1}{2(1-\rho^2)} \left[\frac{(x-\mu_x)^2}{\sigma_x^2} + \frac{(y-\mu_y)^2}{\sigma_y^2} - \frac{2\rho(x-\mu_x)(y-\mu_y)}{\sigma_x\sigma_y} \right] \right\}}{2\pi\sigma_x\sigma_y\sqrt{1-\rho^2}}$$

In [2]:
```
#The data has 2 labels: +1 and -1.
c0 <- 1
c1 <- -1

#This function returns probability density for a data point given the mu and sigma of the multivariate gaussian
#distribution.
multi_norm_dens <- function(data, mu, sigma){
    mu_x1 <- mu[["x1"]]
    mu_x2 <- mu[["x2"]]
    sigma_x1 <- sigma[1,1]
    sigma_x2 <- sigma[2,2]
    rho_x1_x2 <- sigma[1,2]

    denom <- 2 * pi * sigma_x1 * sigma_x2 * sqrt(1-rho_x1_x2^2)
    power <- (-1/(2*(1-rho_x1_x2^2))) * (((data$x1-mu_x1)^2/sigma_x1^2) + ((data$x2-mu_x2)^2/sigma_x2^2) -
                                         (2*rho_x1_x2*(data$x1-mu_x1)*(data$x2-mu_x2)/(sigma_x1*sigma_x2)))
    return (exp(power)/denom)
}
```

In [3]:
```
naiveBayes <- function(train, test, c0, c1){
    #calculating prior probabilities from the training data for both the classes
    p0.hat <- sum(train$label==c0)/nrow(train)
    p1.hat <- sum(train$label==c1)/nrow(train)

    #calculating means of input variables for both the classes.
    mu0.hat <- colMeans(train[train$label==c0,c("x1","x2")])
    mu1.hat <- colMeans(train[train$label==c1,c("x1","x2")])

    #calculating variance of input variables for both the classes.
    sigma0.hat <- var(train[train$label==c0,c("x1", "x2")])
    sigma1.hat <- var(train[train$label==c1,c("x1", "x2")])
    sigma.hat <- p0.hat * sigma0.hat + p1.hat * sigma1.hat

    #We use bayes theorem to calculate posterior probability from prior and likelihood.
    posterior0 <- p0.hat*multi_norm_dens(train, mu0.hat, sigma.hat)
    posterior1 <- p1.hat*multi_norm_dens(train, mu1.hat, sigma.hat)

    #We assign label to the training data based on the probabilities calculation.
    train.predict <- ifelse(posterior0 > posterior1, c0, c1)

    #We use the estimated parameters to calculate labels of the test data.
    prob0 <- p0.hat*multi_norm_dens(test, mu0.hat, sigma.hat)
    prob1 <- p1.hat*multi_norm_dens(test, mu1.hat, sigma.hat)

    test.predict <- ifelse(prob0 > prob1, c0, c1)

    return(data.frame(train=train.predict, test=test.predict))

}
predictions <- naiveBayes(train, test, c0, c1)

train.predict <- predictions[["train"]]
test.predict <- predictions[["test"]]

#Confusion matrix for training data
table(train.predict, train$label)
#Confustion matrix for testing data
table(test.predict, test$label)

cat(sprintf('\nTraining accuracy:\t%.2f%%', sum(train$label==train.predict)/nrow(train)*100))
```

```
cat(sprintf('\nTesting accuracy:\t%.2f%%', sum(test$label==test.predict)/nrow(test)*100))
```

```
  train.predict  -1   1
             -1 206   0
             1    0 294


  test.predict  -1   1
            -1 185   4
            1    2 309


  Training accuracy:      100.00%
  Testing accuracy:       98.80%
```

c)

In [4]:
```
logisticRegression <- function(train, test, c0, c1){
    #We use glm function in R to calculate logistic regression model fitted on the training data set.
    glm.fit <- glm(label ~ x1 + x2, data = train, family = "binomial")

    train.probs <- predict(glm.fit, train[,c("x1","x2")], type = "response")
    train.predict <- ifelse(train.probs > 0.5, c0, c1)

    test.probs <- predict(glm.fit, test[,c("x1","x2")], type = "response")
    test.predict <- ifelse(test.probs > 0.5, c0, c1)

    return(data.frame(train=train.predict, test=test.predict))
}
predictions <- logisticRegression(train, test, c0, c1)

train.predict <- predictions[["train"]]
test.predict <- predictions[["test"]]

#Confustion matrix for training data
table(train.predict, train$label)
#Confustion matrix for test data
table(test.predict, test$label)

cat(sprintf('\nTraining accuracy:\t%.2f%%', sum(train$label==train.predict)/nrow(train)*100))
cat(sprintf('\nTesting accuracy:\t%.2f%%', sum(test$label==test.predict)/nrow(test)*100))
```

```
Warning message:
“glm.fit: algorithm did not converge”
Warning message:
“glm.fit: fitted probabilities numerically 0 or 1 occurred”


train.predict  -1   1
           -1 206   0
           1    0 294


test.predict  -1   1
          -1 186   4
          1    1 309
```

```
Training accuracy:      100.00%
Testing accuracy:       99.00%
```

d) For this dataset there is not much difference in the performance (accuracy) of the classifiers, both are very good. Naive Bayes seemed to run faster but we didn't actually perform a precise run-time test. Based on the literature we can further note:

Naive Bayes Classifier

- It is easy and fast to predict the class of test data set.
- When the assumption of independence holds, a Naive Bayes classifier performs better compared to other models like logistic regression with less training data.
- It will converge faster than logistic regression.
- It is more complex model as it has more parameters to learn compared to logistic regression. Thus, there can be overfitting. More training data can prevent this.
- If a categorical variable has a category (in test data set), which was not observed in training data set, then the model will assign a 0 (zero) probability and will be unable to make a prediction. This is often known as “Zero Frequency”.
- In real life, it is quite unusual to get a set of predictors which are completely independent.

Logistic Regression

- Logistic regression directly models the conditional probabilities.
- In practise, with sufficient data, logistic regression (being a discriminative model) performs better than a NB classifier.
- It has a lesser number of parameters to learn. Logistic Regression is therefore a more robust model (less prone to overfitting).
- It is based on linear modelling. So, it has the advantages of the linear model like variable transformation and regularisation (to prevent overfitting).
- It makes no assumption about the distribution of the classes in feature space.
- Unless nonlinear variable transformations are applied to predictors, non-linear problems can’t be solved with logistic regression because it has a linear decision surface. Linearly separable data is rarely found in real-world scenarios, and often it is also hard to figure out what nonlinear variable transformations might be appropriate for a given problem.