Problem 1:

Consider a two category classification problem with one dimensional discrete feature x. Assume that the priors are $P(\omega_1) = \frac{1}{2}$ and $P(\omega_2) = \frac{1}{2}$. Let the domain of x be nonnegative integers $\{0,1,2, \dots\}$. Let the class-conditional probabilities have the form

$$P(x|\omega_i) = \frac{e^{-\lambda_i}\lambda_i^x}{x!}, \qquad x = 0, 1, 2 \dots.$$

where λ_i for $i = 1,2$ are unknown parameters.

(a) Suppose that samples are drawn independently according to $P(x|\omega_1)$ and $P(x|\omega_2)$ and we get dataset $D_1 = \{1, 5\}$ and $D_2 = \{3,9\}$ for ω_1 and ω_2, respectively. Derive the maximum-likelihood estimate for λ_1 and λ_2.

Solution:

Problem 1:

a) We have, $P(x|\omega_i) = e^{-\lambda_i}\frac{\lambda_i^x}{x!}$; $x = 0, 1, \dots$

where, λ_i for $i = 1, 2$ are unknown.

$$L = \prod_{D_1} P(x|\omega_1) \prod_{D_2} P(x|\omega_2) \quad \text{(since independent)}$$

$$l = \ln L = \sum_{j=1}^{2} \ln P(x_j|\omega_1) + \sum_{j=1}^{2} \ln P(x_j|\omega_2)$$

$$= \sum_{j=1}^{2} [e^{-\lambda_1}\lambda$$

$$l = \sum_{j=1}^{2} [-\lambda_1 + x_j \ln\lambda_1 - \ln x_j!] + \sum_{j=1}^{2} [-\lambda_2 + x_j \ln\lambda_2 - \ln x_j!]$$

$$\frac{\partial l}{\partial \lambda_1} = -2 + \sum_{j=1}^{2} x_j/\lambda_1 = 0$$

$$\Rightarrow \hat{\lambda}_1 = \sum_{j=1}^{2} x_j/2 \quad \text{where, } x_j \in D_1$$

$$\frac{\partial l}{\partial \lambda_2} = -2 + \sum_{j=1}^{2} x_j/\lambda_2 = 0$$

$$\Rightarrow \hat{\lambda}_2 = \sum_{j=1}^{2} x_j/2 \quad \text{where, } x_j \in D_2$$

Thus, $\hat{\lambda}_1 = \frac{\sum x_j}{2} = \frac{1+5}{2} = 3$

$\hat{\lambda}_2 = \frac{\sum x_j}{2} = \frac{3+9}{2} = 6$

(b) Let λ_{ij} be the loss incurred for deciding ω_i when the true category is ω_j. Assume $\lambda_{11} = 0,\ \lambda_{12} = 1,\ \lambda_{21} = 3,\ \lambda_{22} = 0$. Derive the Bayes decision rule for the minimum risk classification.

Solution:

b) We have, $\lambda_{11}=0,\ \lambda_{12}=1,\ \lambda_{21}=3,\ \lambda_{22}=0$

Risk function can be written as—

$$R(\alpha_1|x) = \lambda_{11} P(\omega_1|x) + \lambda_{12} P(\omega_2|x) \quad \text{—①}$$

and $$R(\alpha_2|x) = \lambda_{21} P(\omega_1|x) + \lambda_{22} P(\omega_2|x) \quad \text{—②}$$

From ① ⇒

$$R(\alpha_1|x) = 0 + P(\omega_2|x) = P(\omega_2|x)$$

From ② ⇒

$$R(\alpha_2|x) = 3P(\omega_1|x) + 0 = 3P(\omega_1|x)$$

Bayes decision rule based on minimum risk use classification,

$$g(x) = \begin{cases} R(\alpha_1|x) < R(\alpha_2|x) & \Rightarrow \omega_1 \text{ class} \\ \text{Otherwise} & \Rightarrow \omega_2 \text{ "} \end{cases}$$

So,

$$R(\alpha_1|x) < R(\alpha_2|x)$$

$$\Rightarrow P(\omega_2|x) < 3P(\omega_1|x)$$

$$\Rightarrow e^{-\hat{\lambda}_2}\frac{\hat{\lambda}_2^x}{x!} < 3\, e^{-\hat{\lambda}_1}\frac{\hat{\lambda}_1^x}{x!}$$

$$\Rightarrow \left(\frac{\hat{\lambda}_2}{\hat{\lambda}_1}\right)^x < 3\exp(-\hat{\lambda}_1+\hat{\lambda}_2)$$

$$\Rightarrow x < \frac{\ln 3 + (-\hat{\lambda}_1+\hat{\lambda}_2)}{\ln\hat{\lambda}_2 - \ln\hat{\lambda}_1}$$

$$\Rightarrow x < \frac{\ln 3 + (-3+6)}{\ln 2} = 5.9130$$

Thus, the Bayes decision rule for the minimum risk classification-

$$g(x) = \begin{cases} \text{if } x < \dfrac{\ln 3 - \hat{\lambda}_1 + \hat{\lambda}_2}{\ln\left(\dfrac{\hat{\lambda}_2}{\hat{\lambda}_1}\right)} = \dfrac{\ln 3 + 3}{\ln 2} = 5.9130 & \Rightarrow\ \omega_1\ Class \\ Otherwise & \Rightarrow\ \omega_2\ Class \end{cases}$$

Problem 2:

Assume we are given 300 labeled examples to train a decision tree classifier and report its performance. Describe how to use 3-fold cross-validation to estimate the classification accuracy of the learned classifier.

Solution:

The algorithm of the 30fold cross-validation to estimate the classification accuracy of the learned classifier is given below-

i. Randomly split 300 labeled examples into three groups. Each fold will contain 100 examples.
ii. Take fold 1 and 2 and consider it as a training data. Fit a decision tree classifier and evaluate classification accuracy on fold 3 (consider test data) to get the validation accuracy and evaluate classification accuracy on fold 1 and 2 to get the training accuracy.
iii. Similarly, fit decision tree classifier considering training data as fold 2 & 3 and fold 1 & 3 and get validation accuracy on fold 1 and fold 2, respectively. Also, we can calculate training accuracy as well.
iv. We will have three training and validation accuracy considering each fold as a validation data and the other two sets as training data.
v. To calculate the overall training and validation accuracy, we can take average of the three training and validation accuracy obtained in previous step.

Problem 3:

Consider using a neural network for a binary classification problem such that given an input $\vec{x}_k$ the single output of the network $y(\vec{x}_k; W)$ corresponds to the posterior probability $P(\omega_1|\vec{x}_k)$. Assume we have i.i.d data $\{(\vec{x}_k, t_k)\}_{k=1}^n$, where $t_k \in \{0,1\}$, $t_k = 1$ if $\vec{x}_k$ is labeled to category ω_1, and $t_k = 0$ if $\vec{x}_k$ is labeled to category ω_2, but there is a probability ϵ that the class label on a training data point has been incorrectly set. Derive the error function corresponding to the negative log (conditional) likelihood $P(t_1, t_2, \dots, t_n|\vec{x}_1, \dots, \vec{x}_n)$.

Solution:

Problem 3:

Let $P_k = P(t_k = 1 | \vec{x}_k; W)$ for $\epsilon = 0$; and it is given that $t_k \in \{0,1\}$.

If we consider there is no probability, $\epsilon = 0$, that the class label on a training data point has been incorrectly set. then,

$$P(t_k | \vec{x}_k; W) = P_k^{t_k} (1 - P_k)^{1-t_k}$$

Now, since $\epsilon \neq 0 \Rightarrow$

$$\begin{aligned} P(t_k = 1 | \vec{x}_k; \omega) &= P_k (1-\epsilon) + (1-P_k)\epsilon \\ &= P_k - P_k\epsilon + \epsilon - P_k\epsilon \\ &= P_k + \epsilon - 2P_k\epsilon \end{aligned}$$

and $P(t_k = 0 | \vec{x}_k; \omega) = 1 - P_k + \epsilon + 2P_k\epsilon$

We know that the cross-entropy error funcⁿ is

$$\begin{aligned} \ell &= -\sum_{k=1}^{n} \ln P(t_k | \vec{x}_k : \omega) \\ &= -\sum_{k=1}^{n} \ln \left[(P_k + \epsilon - 2P_k\epsilon)^{t_k} (1 - P_k - \epsilon + 2P_k\epsilon)^{1-t_k} \right] \\ &= -\sum_{k=1}^{n} t_k \ln (P_k + \epsilon - 2P_k\epsilon) + (1-t_k) \ln (1 - P_k - \epsilon + 2P_k\epsilon) \end{aligned}$$

Problem 4:

We want to train a neural network to solve a classification problem. Discuss two possible approaches for controlling the overfitting issue.

Solution:

There are different approaches for controlling the overfitting issue. Such that

a) Early Stopping
b) Weight decay
c) Regularization
d) Dropouts
e) Simplify the network structure
f) Increasing training data

Early Stopping:

We know that as the model complexity increases training error/accuracy decreases. For a large network, batch back-propagation performs gradient descent in the error function, if the learning rate is not too high the training error tends to decrease monotonically. However normally, the error on the validation set decreases until some point of model complexity, but then increases for more complex models. This is an indication that the model may be overfitting the training data. So, there will be a point during training when the model will stop generalizing and it will start learning the statistical noise in the training dataset. So, if we can stop learning when it will start to increase generalization error on the validation data, we will get a useful model for making predictions on new data.

After each epoch model will be evaluated on the validation data and if the generalization error on the validation data starts to increase, then the training process is stopped since this is the best indicator of model performance over unseen examples. This procedure is called early stopping.

However, it is important to consider careful technique to stop early because the training of a neural network is stochastic and can be noisy. The performance of a model on a validation dataset may go up and down many times. This means that the first sign of overfitting may not be a good place to stop training.

Some stopping rules include:

- No change in metric over a given number of epochs.
- An absolute change in a metric.
- A decrease in performance observed over a given number of epochs.
- Average change in metric over a given number of epochs.

When we will stop that is not the best model because we continued to learn model to check the stopping rules. So, we must keep track of the best model found earlier.

Weight Decay:

Weight Decay is one of the useful techniques for simplifying network and avoiding over-fitting. It will start a network with large number of weights. After each weight update, every weight is multiplied by a number between 0 and 1 to make it smaller such as

$$w = w(1 - \epsilon) \qquad ; \qquad where\ 0 < \epsilon < 1$$

If after each weight update there is no change in some weights w, those weights will be smaller and smaller, possibly to such a small value that they can be eliminated altogether. However, those weights that are needed to solve the problem will not decay indefinitely.

Problem 5:

Given valid kernels $K_1(\vec{X},\vec{X}')$ and $K_2(\vec{X},\vec{X}')$, prove that the following will also be a valid kernel:

$$K(\vec{X},\vec{X}') = K_1(\vec{X},\vec{X}') + K_2(\vec{X},\vec{X}')$$

Solution:

Kernel Definition:

We say that $K(\vec{X},\vec{X}')$ is a kernel function iff there exists a feature map $\varphi: \vec{X} \to \Re^N$ such that for all $\vec{X}$,

$$K(\vec{X},\vec{X}') = \langle \varphi(\vec{X}), \varphi(\vec{X}') \rangle$$

is symmetric and positive semi-definite then it is called kernel function.

Since $K_1(\vec{X},\vec{X}')$ and $K_2(\vec{X},\vec{X}')$ are valid kernels then

$$K_1(\vec{X},\vec{X}') = \langle \varphi_1(\vec{X}), \varphi_1(\vec{X}') \rangle \quad \text{and} \quad K_2(\vec{X},\vec{X}') = \langle \varphi_2(\vec{X}), \varphi_2(\vec{X}') \rangle$$

for any $\varphi_1: \vec{X} \to \Re^N$ and $\varphi_2: \vec{X} \to \Re^N$ and they are symmetric and positive semi-definite.

Now

$$\begin{aligned} K(\vec{X},\vec{X}') &= K_1(\vec{X},\vec{X}') + K_2(\vec{X},\vec{X}') \\ &= \langle \varphi_1(\vec{X}), \varphi_1(\vec{X}') \rangle + \langle \varphi_2(\vec{X}), \varphi_2(\vec{X}') \rangle \\ &= \langle [\varphi_1(\vec{X}), \varphi_2(\vec{X})], [\varphi_1(\vec{X}'), \varphi_2(\vec{X})] \rangle \end{aligned}$$

which is symmetric and positive semi-definite.

Thus, linear combination of kernel function generates new kernel function. It can be proved using mercer's theorem as well.

$$\begin{aligned} &\int_{X\times X} K(\vec{X},\vec{X}') f(\vec{X}) f(\vec{X}') d\vec{X} d\vec{X}' \\ &= \int_{X\times X} K_1(\vec{X},\vec{X}') f(\vec{X}) f(\vec{X}') d\vec{X} d\vec{X}' + \int_{X\times X} K_2(\vec{X},\vec{X}') f(\vec{X}) f(\vec{X}') d\vec{X} d\vec{X}' \\ &\geq 0 \end{aligned}$$

Since both $K_1(\vec{X},\vec{X}')$ and $K_2(\vec{X},\vec{X}')$ are valid kernel.

Problem 6:

Both Bagging and Boosting work by collecting a set of base classifiers and using a (weighted) majority voting to classify new examples. What are the key differences between Bagging and Boosting methods?

Solution:

Bagging:

Multiple random with replacement subsets of data (bootstrap samples) from training sample are chosen and for each subset of data is used to train a model (classifier). For new examples, majority voting (ensemble) is used on the multiple predictions obtained from multiple models fitted on multiple random subsets of data. Bagging is used to reduce the variance of a prediction classifier and reduces over-fitting of the model.

Boosting:

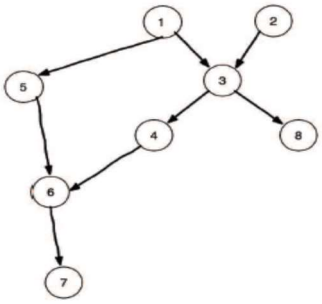
Boosting also manipulate the training set. The base classifiers are trained in sequence. Each base classifier is trained using a weighted data set and update weights by placing more weight on training examples that were misclassified and less weight on examples that were correctly classified that is it puts more weights on portions of data space not previously well predicted and finally the predictions of base classifiers are combined through a weighted majority vote. The objective of boosting methods is to reduce variance as well as bias so that at the end of the process we can obtain a strong learner with lower variance and bias compare to a single weak learner.

Thus, the main difference between bagging and boosting are-

- Both models generate multiple random samples from the training sample but Boosting method puts more weights on portions of data space not previously well predicted.
- Both models predict new examples based on majority voting from multiple learners, but bagging puts equal weights and boosting uses weighted majority voting and puts more weight to those with better performance on training data.
- Bagging and Boosting both try to reduce variance. However, Boosting also tries to reduce bias.
- Bagging can reduce the over-fitting problem. However, Boosting can increase over-fitting problem.

Problem 7:

Consider the following Bayesian network



(a) List the local Markovian assumptions asserted by the network structure (that is, the assumptions under which the graph is a Bayesian network).

Solution:

Local Markovian assumptions are given below-

- $I(1, \emptyset, 2)$
- $I(2, \emptyset, \{1,5\})$
- $I(3, \{1,2\}, 5)$
- $I(4, 3, \{1,2,5,8\})$
- $I(5, 1, \{2,3,4,8\})$
- $I(6, \{4,5\}, \{1,2,3,8\})$
- $I(7, 6, \{1,2,3,4,5,8\})$
- $I(8,3, \{1,2,4,5,6,8\})$

(b) True or false? Please explain if false.

Solution:

i. dsep(2,4,5)

b) (i) dsep(2, 4, 5)

There are two paths.

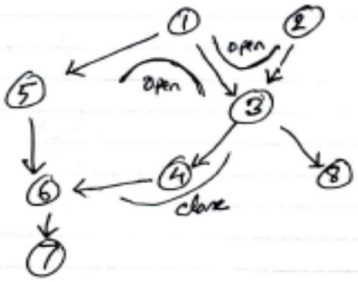
(a) $2 \rightarrow 3 \leftarrow 1 \rightarrow 5$

(b) $2 \rightarrow 3 \rightarrow 4 \rightarrow 6 \leftarrow 5$

Path (b) is closed because $3 \rightarrow 4 \rightarrow 6$ is a sequential valve and it is closed because 4 is given.

Now, path (a) is open because $2 \rightarrow 3 \leftarrow 1$ is a convergent valve and $2 \rightarrow 3 \leftarrow 1$ is open because the descendants (4) of 3 is given.

Thus dsep(2, 4, 5): False



ii. dsep(2,{1,8},5)

(ii) dsep(2, {1,8}, 5)

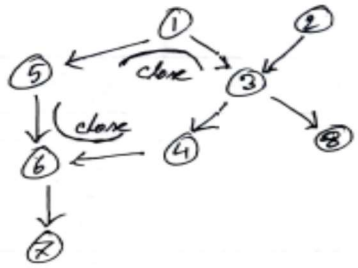
Paths. (a) 2 → 3 ← 1 → 5

(b) 2 → 3 → 4 → 6 ← 5

Path (a) is close because 3 ← 1 → 5 is a divergent valve and it is close because 1 is given.

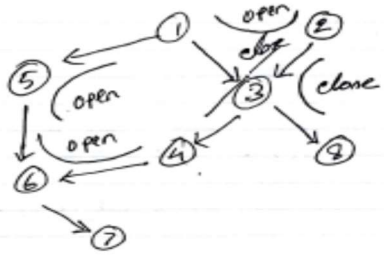
Path (b) is also close because 4 → 6 ← 5 is a convergent valve and it is close because neither 6 nor any of its descendants are given.

Thus dsep(2, {1,8}, 5) : True



iii. dsep(2,{3, 7},{4,8})

iii) dsep(2, {3,7}, {4,8})



Paths. (a) 2 → 3 → 4 (c) 2 → 3 ← 1 → 5 → 6 ← 4

(b) 2 → 3 → 8

Since, (3) is given ⇒ both sequencial valve (a) and (b) are close since (3) is given.

~~Thus dsep(2, {3,7}, {4,8}) : True.~~

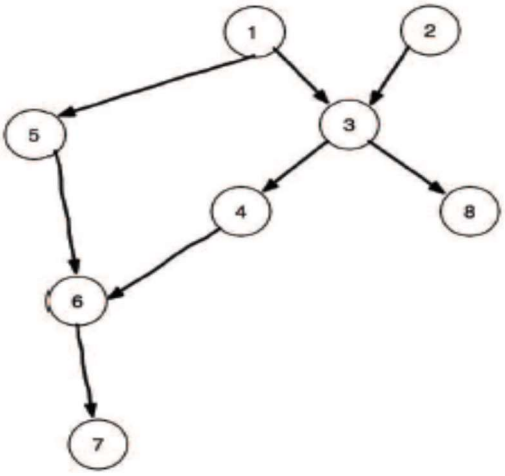
Now, Since, 3 and 7 is given ⇒ all valves are open.

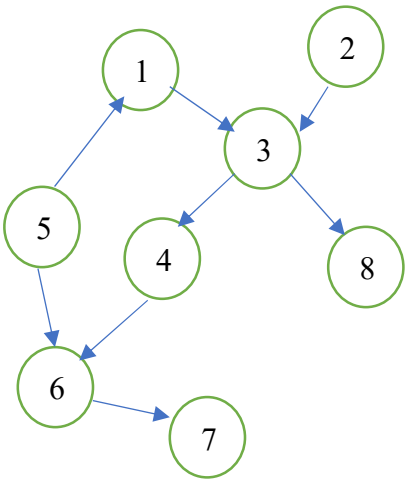
Thus, dsep(2, {3,7}, {4,8}) : False

(c) Which edge directions could be reversed in DAGs that are independence (d-separation) equivalent to the above DAG?

Solution:

We can reverse the edge $1 \rightarrow 5$ to obtain independence equivalent DAG.



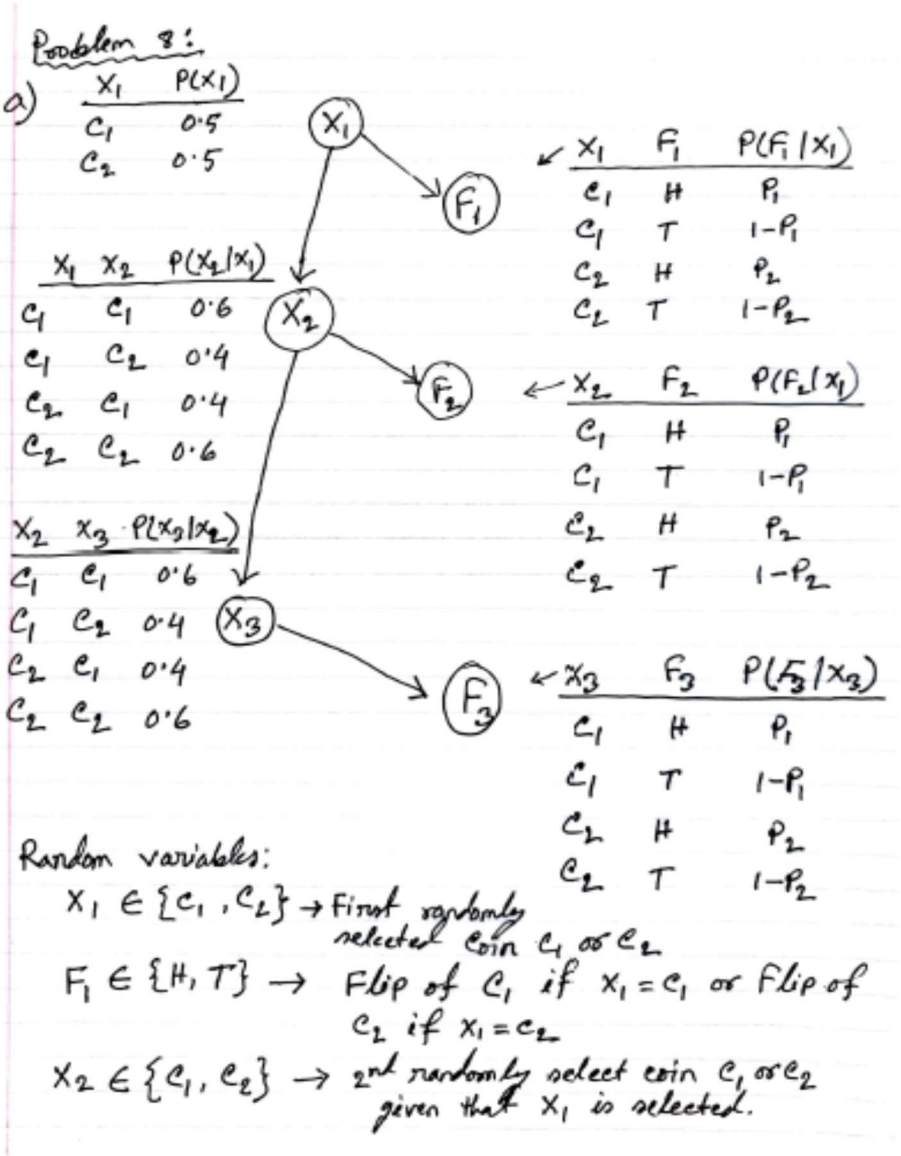


Problem 8:

Jack has two coins C_1 and C_2 with p_1 and p_2 as their corresponding probabilities of landing heads. First Jack randomly picks a coin (50-50 chance) and flips it. From then on he decides which coin to flip next as follows: 60% chance he will flip the same coin, with 40% chance of flipping the other coin. Jack performed the flip three times with given outcomes, e.g. (tail, head, tail). We want to determine which three coins were most likely flipped in turn (e.g., (C_1, C_1, C_2)).

(a) Describe a Bayesian network to solve this problem (i.e., to help determining which three coins were most likely flipped in turn given the outcomes of the three flips). Please specify the variables, their domains, the structure of the model, model parameters, and so on.

Solution:



$F_2 \in \{H, T\}$ → Flip of C_1 if $X_2 = C_1$ or flip of C_2 if $X_2 = C_2$

$X_3 \in \{C_1, C_2\}$ → 3rd randomly selected coin C_1 or C_2 given that X_2 is selected.

$F_3 \in \{H, T\}$ → Flip of C_1 if $X_3 = C_1$ or flip of C_2 if $X_3 = C_2$

Local Markovian assumption:

- $I(X_1, \emptyset, , \emptyset)$
- $I(F_1, X_1, \{X_2, X_3, F_2, F_3\})$
- $I(X_2, X_1, F_1)$
- $I(F_2, X_2, \{X_1, X_3, F_1, F_3\})$
- $I(X_3, X_2, \{X_1, F_1, F_2\})$
- $I(F_3, X_3, \{X_1, X_2, F_1, F_2\})$

(b) What is the probabilistic query to ask the model (to determine most likely which three coins were flipped in turn given the three outcomes)?

Solution:

b) Query : $P(X_3 = C_i, X_2 = C_j, X_1 = C_k \mid F_3 = T, F_2 = H, F_1 = T)$?

where, $i = 1, 2$; $j = 1, 2$; $k = 1, 2$

That is,

$P(X_3 = C_1, X_2 = C_1, X_1 = C_1 \mid F_3 = T, F_2 = H, F_1 = T)$?

$P(X_3 = C_2, X_2 = C_1, X_1 = C_1 \mid F_3 = T, F_2 = H, F_1 = T)$?

$P(X_3 = C_1, X_2 = C_2, X_1 = C_1 \mid F_3 = T, F_2 = H, F_1 = T)$?

$P(X_3 = C_2, X_2 = C_2, X_1 = C_1 \mid F_3 = T, F_2 = H, F_1 = T)$?

$P(X_3 = C_1, X_2 = C_1, X_1 = C_2 \mid F_3 = T, F_2 = H, F_1 = T)$?

$P(X_3 = C_2, X_2 = C_1, X_1 = C_2 \mid F_3 = T, F_2 = H, F_1 = T)$?

$P(X_3 = C_1, X_2 = C_2, X_1 = C_2 \mid F_3 = T, F_2 = H, F_1 = T)$?

$P(X_3 = C_2, X_2 = C_2, X_1 = C_2 \mid F_3 = T, F_2 = H, F_1 = T)$?

Problem 9:

K-means is an iterative algorithm that initializes the cluster centers randomly. Is K-means algorithm guaranteed to converge regardless of how the cluster centers are initialized? Why?

Solution:

K-means algorithm guaranteed to converge to the local minimum, not necessarily to the global minimum of the given loss function regardless of how the cluster centers are initialized. Also, K-means does not guarantee unique clustering because we can get different results with randomly chosen initial clusters. It means that K-means does not necessarily converge to the same point regardless of initial cluster centers. Because, when we assign the initial centers, it tries to assign all example close to that initial centers and update centers based on the assigned examples and loss function tries to minimize around the initial centers if there are more than one minimum (local minimum). That is why, we try to choose different random initial cluster center and obtain the minimum loss functions for each initial cluster center and consider the best cluster with lowest minimum loss.

K-means algorithm minimizes within cluster sum of squares loss function

$$J = \sum_{i=1}^{n} \sum_{j=1}^{k} r_{ij} \left|\left| \boldsymbol{x}_i - \boldsymbol{\mu}_j \right|\right|^2$$

where $r_{ij} = 1$ if x_i is assigned to cluster j and $r_{ij} = 0$ for $j \neq k$ and $\boldsymbol{\mu}_j$ is the mean of data points in cluster j, C_j,

$$\boldsymbol{\mu}_j = \frac{1}{n_j} \sum_{x_i \in C_j} x_i$$

It is an EM algorithm where it minimizes J in the estimation step for a given $\boldsymbol{\mu}_j$ and recompute $\boldsymbol{\mu}_j$ that minimizes J for given r_{ij}. Each phase of the algorithm reduces the value of the function J, which assures the convergence of the algorithm at least to a local minimum.

However, it cannot adapt to different cluster densities, even when the clusters are spherical, have equal radii and are well-separated. It also sensitive to outlier and it is insensitive to the differing cluster density. K-means does not work well when the clusters are non-convex.

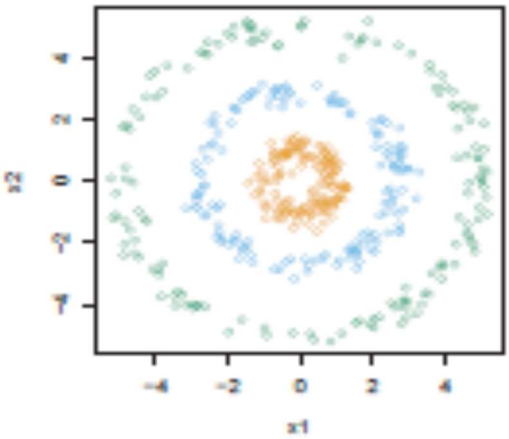
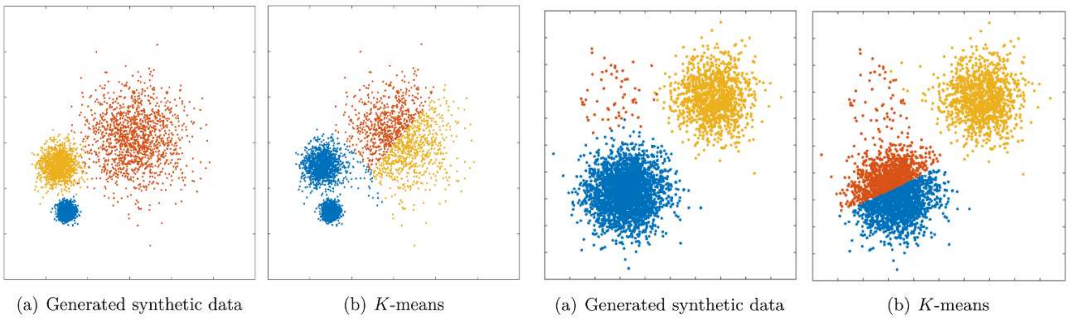


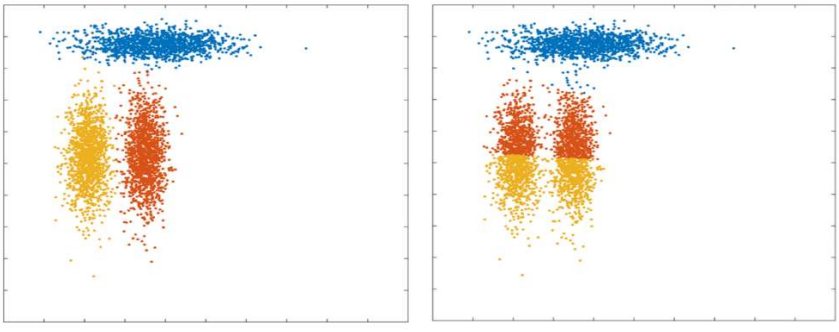
Figure 1: Cluster data on a sphere shape

For example, in figure 1, clearly there are 3 cluster in this two dimensional (x_1, x_2) space. Whatever random starting point we try it will not be able to find any spheres that can separate this three cluster. We will get different results with randomly chosen initial clusters for data given in figure 1. Following plots are taken from [2].



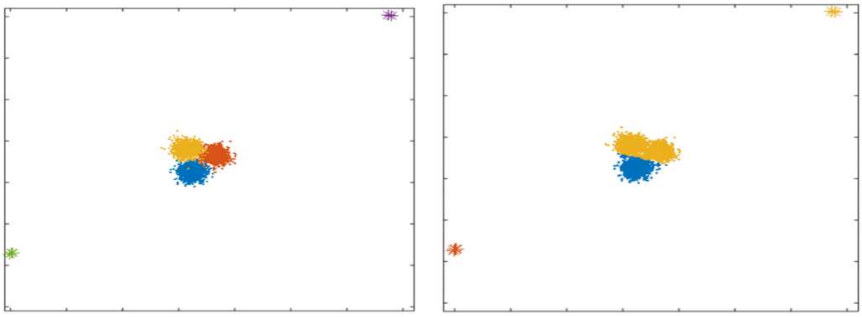
(a) Generated synthetic data (b) K-means (a) Generated synthetic data (b) K-means

Figure 2: Three cluster with different density of data. In general, K-means assumes equal density within each cluster as a result fails to identify correct cluster but converge to a local minimum of the loss function.



(a) Generated synthetic data (b) K-means

Figure 3: All clusters share the same volume and density, but one is rotated relative to the others and K-means fails to converge to global minimum of the loss function.



(a) Generated synthetic data (b) K-means

Figure 4: K-means suffers from outlier. Due to outlier, K-means converges to the local minimum of the loss function.

Reference:

1. lecture14-Clustering-2up.pdf
2. https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0162259