51. How to handle categorical variables in KNN?
Ans: Create dummy variables out of a categorical variable and include them instead of the original categorical variable. Unlike regression, create k dummies instead of (k-1).
For example, a categorical variable named "Department" has 5 unique levels/categories. So we will create 5 dummy variables. Each dummy variable has 1 against its department and else 0.
52. Can KNN be used for Regression? How to use KNN for Regression?
Ans: Yes, K-nearest neighbour can be used for regression. In other words, the K-nearest neighbour algorithm can be applied when the dependent variable is continuous. In this case, the predicted value is the average of the values of its k nearest neighbours.
53. Discuss the difference between KNN and K Means Algorithms.
Ans: KNN and k-means clustering both are very different algorithms that solve different problems and have their own meanings of what the variable 'k' is. KNN is a supervised classification algorithm that will label new data points based on the 'k' number of nearest data points and k-means clustering is an unsupervised clustering algorithm that groups the data into 'k' number of clusters.
54. How to reduce the increased variance of the model other than changing k?
Ans: By using bagging-based decision boundaries. If not restricted in the number of times, one can draw samples from the original dataset, a sample variance reduction method would be to sample, many times, and then simply take a majority vote of the kNN models to fit each of these samples to classify each test data point. This variance reduction method is called bagging.
55. What is the effect of sampling on KNN?
Ans: Sampling does several things from the perspective of a single data point since kNN works on a point-by-point basis.

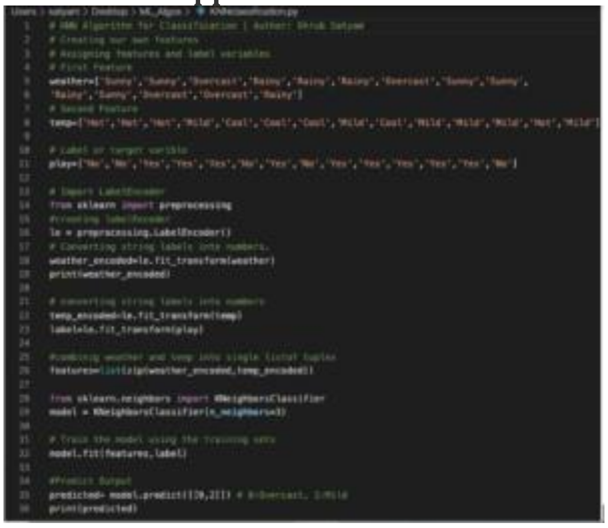
1. The average distance to the k nearest neighbours increases due to increased sparsity in the dataset.
2. Consequently, the area covered by k-nearest neighbours increases in size and covers a larger area of the feature space.
3. The sample variance increases.

A consequence of this change in input is an increase in variance. When we talk of variance, we refer to the variability in the predictions given different samples from the population. Why would the immediate effects of sampling lead to the increased variance of the model? Notice that now a larger area of the feature space is represented by the same k data points. While our sample size has not grown, the population space that it represents has increased in size. This will result in higher variance in the proportion of classes in the k nearest data points, and consequently a higher variance in the classification of each data point.
56. What happens when we change the value of K in KNN?
Ans: Short Answer: The class boundaries of the predictions become more smooth as k increases.
Long Answer: What really is the significance of these effects? First, it gives hints that a lower k value makes the KNN model more "sensitive." That is, it is more sensitive to the local changes in the dataset. The "sensitivity" of the model directly translates to its variance. All of these examples point to an inverse relationship between variance and k. Additionally, consider how KNN operates when k reaches its maximum value, k=n, where n is the number of points in the training set) In this case, the majority class in the training set will always dominate the predictions. It will simply pick the most abundant class in the data, and never deviate, effectively resulting in zero variance. Therefore, it seems to reduce variance, k must be increased.
Final Verdict: In order to offset the increased variance due to sampling, k can be increased to decrease model variance.
57. What is the thumb rule to approach the KNN problem?

Ans:

1.
    1. Load the data
    2. Initialize the value of k
        - Calculate the distance between test data and each row of training data. Here we will use Euclidean distance as our distance metric since it’s the most popular method. The other metrics that can be used are Chebyshev, cosine, etc.
        - Sort the calculated distances in ascending order based on distance values
        - Get top k rows from the sorted array
        - Get the most frequent class of these rows
        - Return the predicted class for getting the predicted class, iterate from 1 to the total number of training data points.

KNN Code Snippet:

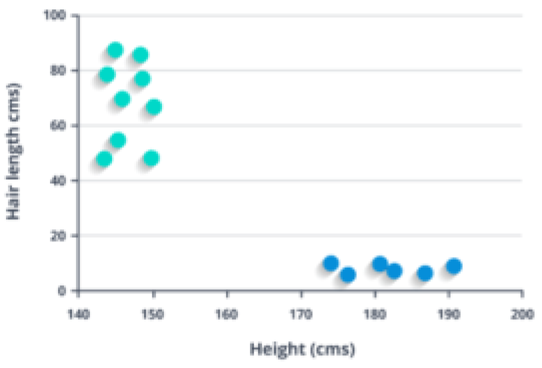


```
Users > satyam > Desktop > ML_Algos > KNNclassification.py
# KNN Algorithm for Classification | Author: Shubh Satyam
# Creating our own features
# Assigning features and label variables
# First Feature
weather=['Sunny','Sunny','Overcast','Rainy','Rainy','Rainy','Overcast','Sunny','Sunny',
'Rainy','Sunny','Overcast','Overcast','Rainy']
# Second Feature
temp=['Hot','Hot','Hot','Mild','Cool','Cool','Cool','Mild','Cool','Mild','Mild','Mild','Hot','Mild']

# Label or target varible
play=['No','No','Yes','Yes','Yes','No','Yes','No','Yes','Yes','Yes','Yes','Yes','No']

# Import LabelEncoder
from sklearn import preprocessing
#creating labelEncoder
le = preprocessing.LabelEncoder()
# Converting string labels into numbers.
weather_encoded=le.fit_transform(weather)
print(weather_encoded)

# converting string labels into numbers
temp_encoded=le.fit_transform(temp)
label=le.fit_transform(play)

#combinig weather and temp into single listof tuples
features=list(zip(weather_encoded,temp_encoded))

from sklearn.neighbors import KNeighborsClassifier
model = KNeighborsClassifier(n_neighbors=3)

# Train the model using the training sets
model.fit(features,label)

#Predict Output
predicted= model.predict([[0,2]]) # 0:Overcast, 2:Mild
print(predicted)
```

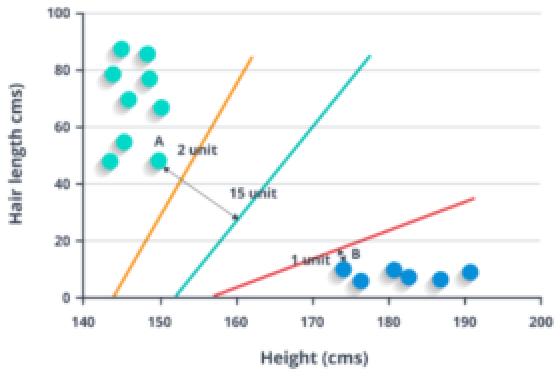
58. What is SVM Algorithm?

Ans: SVM stands for support vector machine, it is a supervised machine learning algorithm that can be used for both Regression and Classification. In this algorithm, we plot each data item as a point in n-dimensional space (where n is a number of features you have) with the value of each feature being the value of a particular coordinate.

For example, if we only had two features like Height and Hair length of an individual, we’d first plot these two variables in two-dimensional space where each point has two coordinates (these co-ordinates are known as Support Vectors)



Now, we will find some line that splits the data between the two differently classified groups of data. This will be the line such that the distances from the closest point in each of the two groups will be farthest away.



In the example shown above, the line which splits the data into two differently classified groups is the black line, since the two closest points are the farthest apart from the line. This line is our classifier. Then, depending on where the testing data lands on either side of the line, that's what class we can classify the new data as.

59. What are support Vectors?

Ans: A support vector machine attempts to find the line that "best" separates two classes of points. By "best", we mean the line that results in the largest margin between the two classes. The points that lie on this margin are the support vectors.

The vectors that define the hyperplane are the support vectors.

60. What is the purpose of the Support Vector in SVM?

Ans: A Support Vector Machine (SVM) performs classification by finding the hyperplane that maximizes the distance margin between the two classes. The extreme points in the data sets that define the hyperplane are the support vectors.

61. What are kernels?

Ans: SVM algorithms use a set of mathematical functions that are defined as the kernel. The function of the kernel is to take data as input and transform it into the required form. Different SVM algorithms use different types of kernel functions. These functions can be of different types.

There are four types of kernels in SVM.

1. Linear Kernel
2. Polynomial kernel
3. Radial basis kernel
4. Sigmoid kernel

62. What is Kernel Trick?

Ans: Short Answer: It allows us to operate in the original feature space without computing the coordinates of the data in a higher-dimensional space.

Long Answer:

1. For a dataset with n features (~n-dimensional), SVMs find an n-1-dimensional hyperplane to separate it (let us say for classification)
2. Thus, SVMs perform very badly with datasets that are not linearly separable
3. SVM can now do well with datasets that are not linearly separable
4. But, quite often, it's possible to transform our not-linearly-separable dataset into a higher-dimensional dataset where it becomes linearly separable, so that SVMs can do a good job
5. Unfortunately, quite often, the number of dimensions you have to add (via transformations) depends on the number of dimensions you already have (and not linearly)
    1. For datasets with a lot of features, it becomes next to impossible to try out all the interesting transformations

6. Enter the Kernel Trick
    1.
        - Thankfully, the only thing SVMs need to do in the (higher-dimensional) feature space (while training) is computing the pair-wise dot products
        - For a given pair of vectors (in lower-dimensional feature space) and a transformation into a higher-dimensional space, there exists a function (The Kernel Function) which can compute the dot product in the higher-dimensional space without explicitly transforming the vectors into the higher-dimensional space first
        - We are saved!

63. Why is SVM called as Large Margin Classifier?

Ans: Short Answer: Because it places the decision boundary such that it maximizes the distance between two clusters.

Long Answer: choosing the best hyperplane is to choose one in which the distance from the training points is the maximum. This is formalized by the geometric margin. Without getting into the details of the derivation, the geometric margin is given by:

$$\gamma^{(i)} = y^{(i)}\left(\frac{w^T x^{(i)} + b}{||w||}\right)$$

Which is simply the functional margin normalized. So, these intuitions lead to the maximum margin classifier which is a precursor to the SVM.

64. What is the difference between Logistics Regression and SVM? When to use which model?

Ans:

1. SVM tries to find the "best" margin (distance between the line and the support vectors) that separates the classes and this reduces the risk of error on the data, while logistic regression does not, instead it can have different decision boundaries with different weights that are near the optimal point.
2. SVM works well with unstructured and semi-structured data like text and images while logistic regression works with already identified independent variables.
3. SVM is based on the geometrical properties of the data while logistic regression is based on statistical approaches.
4. Logistic Regression can't be applied to a nonlinearly separable dataset whereas SVM can be applied.
5. The risk of overfitting is less in SVM, while Logistic regression is vulnerable to overfitting.

65. When to Use Logistic Regression vs Support Vector Machine?

Ans: Depending on the number of training sets (data)/features that you have, you can choose to use either logistic regression or support vector machine.
Let's take these as an example where:
n = number of features,
m = number of training examples

1. If *n is large (1–10,000) and m is small (10–1000)*: use logistic regression or SVM with a linear kernel.
2. If *n is small (1–1000) and m is intermediate (10–10,000)*: use SVM with (Gaussian, polynomial, etc) kernel
3. If *n is small (1–100), m is large* (50,000–1,000,000+): first, manually add more features and then use logistic regression or SVM with a linear kernel

66. What does c and gamma parameter in SVM signify?

Ans: Short Answer:
Cost and Gamma are the hyper-parameters that decide the performance of an SVM model. There should be a fine balance between Variance and Bias for any ML model. (this is a science and an art – as we call it in empirical studies)
For SVM, a High value of Gamma leads to more accuracy but biased results and vice-versa. Similarly, a large value of Cost parameter (C) indicates poor accuracy but low bias and vice-versa.
Following table summarizes the above explanation –
The art is to choose a model with optimum variance and bias. Therefore, you need to choose the values of C and Gamma accordingly.
Optimum values of C and Gamma can be found by using methods like Grid search.
Long Answer:
The C parameter tells the SVM optimization how much you want to avoid misclassifying each training example. For large values of C, the optimization will choose a smaller-margin hyperplane if that hyperplane does a better job of getting all the training points classified correctly. Conversely, a very small value of C will cause the optimizer to look for a larger margin separating hyperplane, even if that hyperplane misclassifies more points. For very tiny values of C, you should get misclassified examples, often even if your training data is linearly separable.
The gamma parameter defines how far the influence of a single training example reaches, with low values meaning 'far' and high values meaning 'close'.
The gamma parameters can be seen as the inverse of the radius of influence of samples selected by the model as support vectors. If gamma is too large, the radius of the area of influence of the support vectors only includes the support vector itself and no amount of regularization with C will be able to prevent overfitting.
When gamma is very small, the model is too constrained and cannot capture the complexity or "shape" of the data. The region of influence of any selected support vector would include the whole training set. The resulting model will behave similarly to a linear model with a set of hyperplanes that separate the centers of the high density of any pair of two classes.

67. What are the Advantages and Disadvantages of SVM?

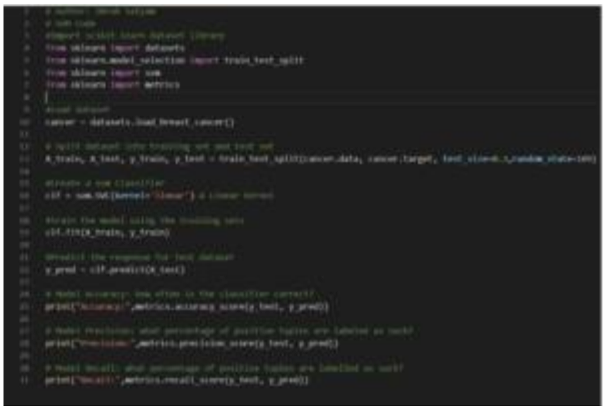
Ans: SVM Advantages

- SVM's are very good when we have no idea about the data.
- Works well with even unstructured and semi-structured data like text, Images, and trees.
- The kernel trick is a real strength of SVM. With an appropriate kernel function, we can solve any complex problem.
- Unlike in neural networks, SVM is not solved for local optima.
- It scales relatively well to high dimensional data.
- SVM models have generalization in practice, the risk of over-fitting is less in SVM.
- SVM is always compared with ANN. When compared to ANN models, SVMs give better results.

SVM Disadvantages

- Choosing a "good" kernel function is not easy.
- Long training time for large datasets.
- Difficult to understand and interpret the final model, variable weights, and individual impact.
- Since the final model is not so easy to see, we can not do small calibrations to the model hence its tough to incorporate our business logic.
- The SVM hyperparameters are Cost -C and gamma. It is not that easy to fine-tune these hyper-parameters. It is hard to visualize their impact.

SVM code snippet:



```
# Author: [illegible]
# SVM Code
#Import scikit-learn dataset library
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn import svm
from sklearn import metrics

#Load dataset
cancer = datasets.load_breast_cancer()

# Split dataset into training set and test set
X_train, X_test, y_train, y_test = train_test_split(cancer.data, cancer.target, test_size=0.3,random_state=109)

#Create a svm Classifier
clf = svm.SVC(kernel='linear') # Linear Kernel

#Train the model using the training sets
clf.fit(X_train, y_train)

#Predict the response for test dataset
y_pred = clf.predict(X_test)

# Model Accuracy: how often is the classifier correct?
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))

# Model Precision: what percentage of positive tuples are labeled as such?
print("Precision:",metrics.precision_score(y_test, y_pred))

# Model Recall: what percentage of positive tuples are labelled as such?
print("Recall:",metrics.recall_score(y_test, y_pred))
```

68. What is Naïve Bayes Algorithm?

Ans: It is a classification algorithm that predicts the probability of each data point belonging to a class and then classifies the point as the class with the highest probability.

Discuss Bayes Theorem.

Bayes' Theorem gives us the probability of an event actually happening by combining the conditional probability given some result and the prior knowledge of an event happening.
Conditional probability is the probability that something will happen, given that something has occurred. In other words, the conditional probability is the probability of X given a test result or P(X|Test). For example, what is the probability an e-mail is spam given that my spam filter classified it as spam.
The prior probability is based on previous experience or the percentage of previous samples. For example, what is the probability that any email is spam?
Formally

- P(A|B) = Posterior probability = Probability of A given B happened
- P(B|A) = Conditional probability = Probability of B happening if A is true
- P(A) = Prior probability = Probability of A happening in general
- P(B) = Evidence probability = Probability of getting a positive test

69. Why is Naïve Bayes Naïve?

Ans: In Layman's Term: The simple meaning of Naive is willing to believe that that life is simple and fair, which is not true. Naive Bayes is naive because it assumes that the features that are going into the model are not related to each other anyhow Change in one variable will not affect the other variable directly.
Long Answer: Naive Bayes (NB) is 'naive' because it makes the assumption that features of measurement are independent of each other. This is naive because it is (almost) never true. Here is how it works even then – NB is a very intuitive classification algorithm. It asks the question, "Given these features, does this measurement belong to class A or B?", and answers it by taking the proportion of all previous measurements with the same features belonging to class A multiplied by the proportion of all measurements in class A. If this number is bigger than the corresponding calculation for class B then we say the measurement belongs in class A.

70. What are feature matrix and response vectors?

Ans: Feature matrix:- The feature matrix contains all the vectors(rows) of the dataset in which each vector consists of the value of dependent features.
Response vectors:- The response vector contains the value of the class variable (prediction or output) for each row of the feature matrix.

71 Applications of Naïve Bayes Classification Algorithms?

Ans: Some of the real-world examples are as given below

- To mark an email as spam, or not spam?
- Classify a news article about technology, politics, or sports?
- Check a piece of text expressing positive emotions, or negative emotions?
- Also used for face recognition software.

72. What are the Advantages and Disadvantages of using the Naïve Bayes Algorithm?

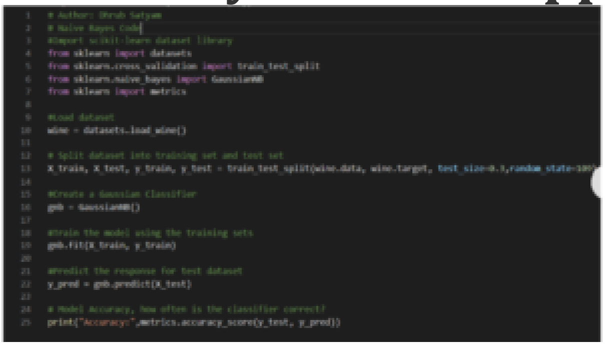
Ans: Advantages

1. Fast
2. Highly scalable.
3. Used for binary and Multiclass Classification.
4. Great Choice for text classification.
5. It can easily train smaller data sets.

Disadvantages

Naive Bayes considers that the features are independent of each other. However, in the real-world, features depend on each other.

Naïve Bayes Code Snippet:



```
# Author: Dhrub Satyam
# Naive Bayes Code
#Import scikit-learn dataset library
from sklearn import datasets
from sklearn.cross_validation import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn import metrics

#Load dataset
wine = datasets.load_wine()

# Split dataset into training set and test set
X_train, X_test, y_train, y_test = train_test_split(wine.data, wine.target, test_size=0.3,random_state=10

#Create a Gaussian Classifier
gnb = GaussianNB()

#Train the model using the training sets
gnb.fit(X_train, y_train)

#Predict the response for test dataset
y_pred = gnb.predict(X_test)

# Model Accuracy, how often is the classifier correct?
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

73. What is K-Means Clustering? What are the steps for it?

Ans: K-means (Macqueen, 1967) is one of the simplest unsupervised learning algorithms that solve the well-known clustering problem. K-means clustering is a method of vector quantization, original from signal processing, that is popular for cluster analysis in data mining.

If k is given, the K-means algorithm can be executed in the following steps:

- Partition of objects into k non-empty subsets
- Identifying the cluster centroids (mean point) of the current partition.
- Assigning each point to a specific cluster
- Compute the distances from each point and allot points to the cluster where the distance from the centroid is minimum.
- After re-allotting the points, find the centroid of the new cluster formed.

74. Why is the word “means” associated with the name of the K-Means algorithm?

Ans: The ‘means’ in the K-means refers to **averaging of the data**; that is, finding the centroid.

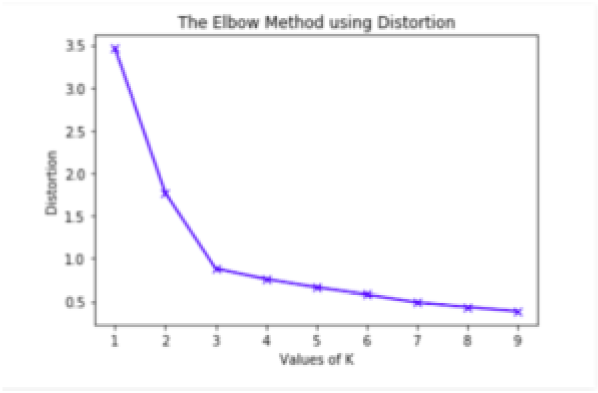
There are k-medoids and k-medians algorithms as well.

k-medoids minimizes the sum of dissimilarities between points labeled to be in a cluster and a point designated as the center of that cluster. In contrast to the k-means algorithm, k-medoids choose datapoints as centers (medoids or exemplars).

k-medians is a variation of k-means clustering where instead of calculating the mean for each cluster to determine its centroid, one instead calculates the median.

75. How to find the optimum number of clusters in K-Means? Discuss the elbow curve/elbow method?

Ans: The basic idea behind partitioning methods, such as k-means clustering, is to define clusters such that the total intra-cluster variation [or total within-cluster sum of square (WSS)] is minimized. The total WSS measures the compactness of the clustering and we want it to be as small as possible.
The Elbow method looks at the total WSS as a function of the number of clusters: One should choose a number of clusters so that adding another cluster doesn't improve much better the total WSS.



Notice the elbow at k =3.
The optimal number of clusters can be defined as follow:

1. Compute clustering algorithm (e.g., k-means clustering) for different values of k. For instance, by varying k from 1 to 10 clusters.
2. For each k, calculate the total within-cluster sum of square (WSS).
3. Plot the curve of WSS according to the number of clusters k.
4. The location of a bend (knee) in the plot is generally considered as an indicator of the appropriate number of clusters.

76. What is the difference between K-Means and Hierarchical Clustering? When to use which?

Ans: Hierarchical Clustering and k-means clustering complement each other. In hierarchical clustering, the researcher is not aware of the number of clusters to be made whereas, in k-means clustering, the number of clusters to be made is specified before-hand.
Advice- If unaware of the number of clusters to be formed, use hierarchical clustering to determine the number and then use k-means clustering to make more stable clusters as hierarchical clustering is a single-pass exercise whereas k-means is an iterative process.

77. What are the advantages and disadvantages of using K-Means Algorithms?

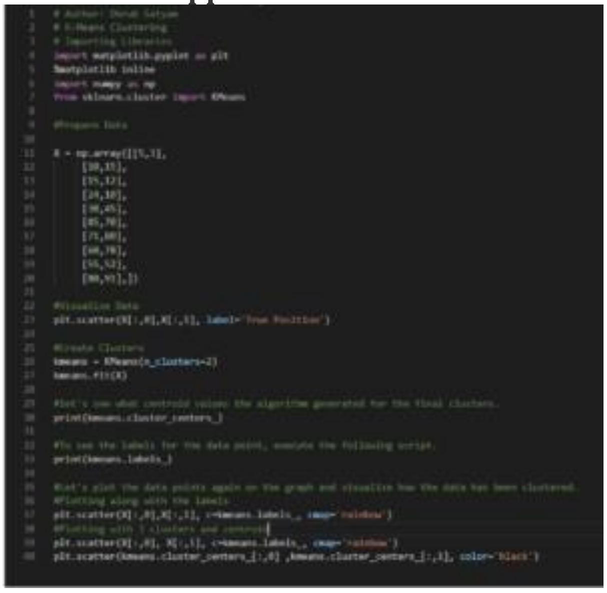
Ans: K-Means Advantages :

1) If variables are huge, then K-Means most of the times computationally faster than hierarchical clustering, if we keep k smalls.

2) K-Means produce tighter clusters than hierarchical clustering, especially if the clusters are globular.

K-Means Disadvantages:

1) Difficult to predict K-Value.
2) With a global cluster, it didn't work well.

3) Different initial partitions can result in different final clusters.
4) It does not work well with clusters (in the original data) of Different sizes and Different density.
KNN code snippet:


```
# Author: [illegible]
# K-Means Clustering
# Importing Libraries
import matplotlib.pyplot as plt
%matplotlib inline
import numpy as np
from sklearn.cluster import KMeans

#Prepare Data

X = np.array([[5,3],
     [10,15],
     [15,12],
     [24,10],
     [30,45],
     [85,70],
     [71,80],
     [60,78],
     [55,52],
     [80,91],])

#Visualize Data
plt.scatter(X[:,0],X[:,1], label='True Position')

#Create Clusters
kmeans = KMeans(n_clusters=2)
kmeans.fit(X)

#Let's see what centroid values the algorithm generated for the final clusters.
print(kmeans.cluster_centers_)

#To see the labels for the data point, execute the following script.
print(kmeans.labels_)

#Let's plot the data points again on the graph and visualize how the data has been clustered.
#Plotting along with the labels
plt.scatter(X[:,0],X[:,1], c=kmeans.labels_, cmap='rainbow')
#Plotting with 2 clusters and centroids
plt.scatter(X[:,0], X[:,1], c=kmeans.labels_, cmap='rainbow')
plt.scatter(kmeans.cluster_centers_[:,0] ,kmeans.cluster_centers_[:,1], color='black')
```

78. What is Hierarchical Clustering?
Ans: Hierarchical clustering is another unsupervised learning algorithm that is used to group together the unlabelled data points having similar characteristics. Hierarchical clustering algorithms fall into the following two categories.
Agglomerative hierarchical algorithms – In agglomerative hierarchical algorithms, each data point is treated as a single cluster and then successively merge or agglomerate (bottom-up approach) the pairs of clusters. The hierarchy of the clusters is represented as a dendrogram or tree structure.
Divisive hierarchical algorithms – On the other hand, in divisive hierarchical algorithms, all the data points are treated as one big cluster and the process of clustering involves dividing (Top-down approach) the one big cluster into various small clusters

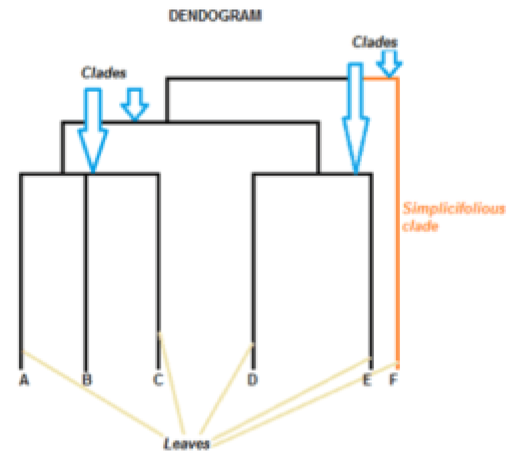
79. What are the steps to perform Agglomerative Hierarchical Clustering?
Ans: Most used and important Hierarchical clustering i.e. agglomerative. The steps to perform the same is as follows –

- **Step 1** – Treat each data point as a single cluster. Hence, we will be having, say K clusters at the start. The number of data points will also be K at the start.
- **Step 2** – Now, in this step we need to form a big cluster by joining two closet datapoints. This will result in a total of K-1 clusters.
- **Step 3** – Now, to form more clusters we need to join two closet clusters. This will result in a total of K-2 clusters.
- **Step 4** – Now, to form one big cluster repeat the above three steps until K would become 0 i.e. no more data points left to join.
- **Step 5** – At last, after making one single big cluster, dendrograms will be used to divide into multiple clusters depending upon the problem.

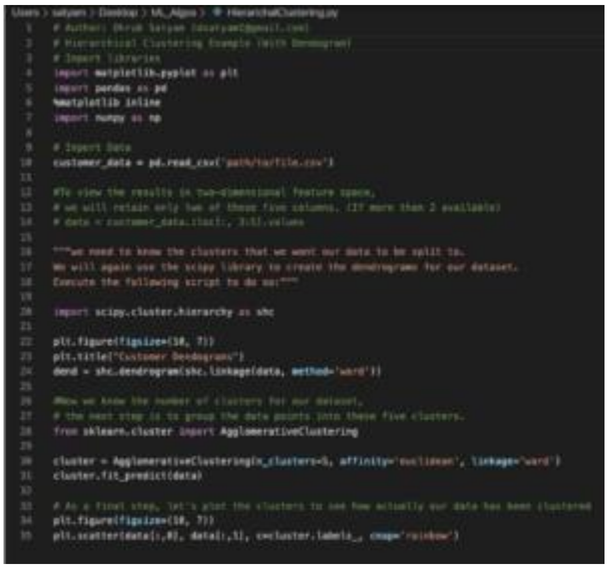
80. What is Dendrogram and what is its importance in Hierarchical Clustering?
Ans: A dendrogram is a type of Tree Diagram showing hierarchical clustering — relationships between similar sets of data. They are frequently used in biology to show clustering between genes or samples, but they can represent any type of grouped data.

The role of the dendrogram starts once the big cluster is formed. Dendrogram will be used to split the clusters into multiple clusters of related data points depending upon our problem.

Parts of Dendrogram:



Hierarchical Clustering Code Snippet:



```
Users > satyam > Desktop > ML_Algos > HierarchicalClustering.py
# Author: Dhruv Satyam (dsatyam@gmail.com)
# Hierarchical Clustering Example (With Dendrogram)
# Import libraries
import matplotlib.pyplot as plt
import pandas as pd
%matplotlib inline
import numpy as np

# Import Data
customer_data = pd.read_csv('path/to/file.csv')

#To view the results in two-dimensional feature space,
# we will retain only two of these five columns. (If more than 2 available)
# data = customer_data.iloc[:, 3:5].values

"""we need to know the clusters that we want our data to be split to.
We will again use the scipy library to create the dendrograms for our dataset.
Execute the following script to do so:"""

import scipy.cluster.hierarchy as shc

plt.figure(figsize=(10, 7))
plt.title("Customer Dendograms")
dend = shc.dendrogram(shc.linkage(data, method='ward'))

#Now we know the number of clusters for our dataset,
# the next step is to group the data points into these five clusters.
from sklearn.cluster import AgglomerativeClustering

cluster = AgglomerativeClustering(n_clusters=5, affinity='euclidean', linkage='ward')
cluster.fit_predict(data)

# As a final step, let's plot the clusters to see how actually our data has been clustered
plt.figure(figsize=(10, 7))
plt.scatter(data[:,0], data[:,1], c=cluster.labels_, cmap='rainbow')
```

81. What is Boosting?

Ans: Boosting is a method of converting weak learners into strong learners. In boosting, each new tree is a fit on a modified version of the original data set.

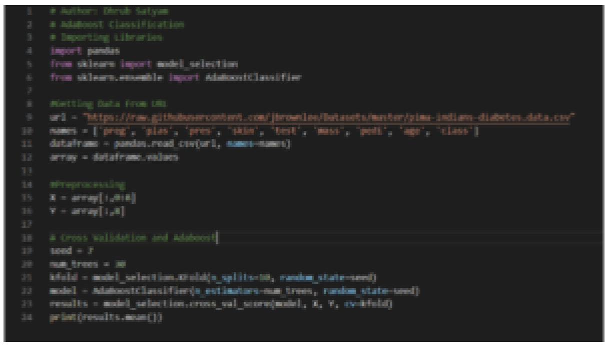
Purpose of Boosting: It helps the weak learner to be modified to become better.

How it evolved: The first Boosting Algorithm gained popularity was AdaBoost or Adaptive Boosting. Further it evolved and generalized as Gradient Boosting.

82. What is Adaboost?

Ans: Adaboost combines multiple weak learners into a single strong learner. The weak learners in AdaBoost are decision trees with a single split, called decision stumps. When AdaBoost creates its first decision stump, all observations are weighted equally. To correct the previous error, the observations that were incorrectly classified now carry more weight than the observations that were correctly classified. AdaBoost algorithms can be used for both classification and regression problems.

Adaboost Code Snippet:



```python
# Author: Dhrub Satyam
# AdaBoost Classification
# Importing Libraries
import pandas
from sklearn import model_selection
from sklearn.ensemble import AdaBoostClassifier

#Getting Data from URL
url = "https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-indians-diabetes.data.csv"
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dataframe = pandas.read_csv(url, names=names)
array = dataframe.values

#Preprocessing
X = array[:,0:8]
Y = array[:,8]

# Cross Validation and Adaboost
seed = 7
num_trees = 30
kfold = model_selection.KFold(n_splits=10, random_state=seed)
model = AdaBoostClassifier(n_estimators=num_trees, random_state=seed)
results = model_selection.cross_val_score(model, X, Y, cv=kfold)
print(results.mean())
```

83. What is Gradient Boosting Method (GBM)?

Ans: Gradient Boosting works by sequentially adding predictors to an ensemble, each one correcting its predecessor. However, instead of changing the weights for every incorrect classified observation at every iteration like AdaBoost, the Gradient Boosting method tries to fit the new predictor to the residual errors made by the previous predictor.
GBM uses Gradient Descent to find the shortcomings in the previous learner's predictions. The GBM algorithm can be given in the following steps.
Fit a model to the data, F1(x) = y
Create a new model, F2(x) = F1(x) + h1(x)
By combining weak learners after weak learners, our final model is able to account for a lot of the error from the original model and reduces this error over time.
Gradient Boosting Code Snippet:

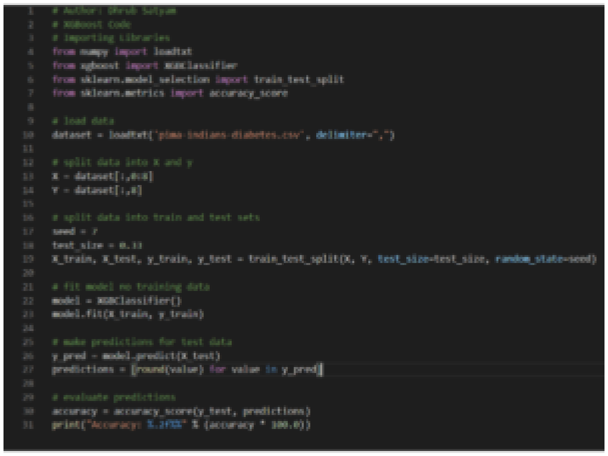
[illegible]

84. What is XGBoost?

Ans: XGBoost stands for e**X**treme **G**radient **B**oosting. XGBoost is an implementation of gradient boosted decision trees designed for speed and performance. Gradient boosting machines are generally very slow in implementation because of sequential model training. Hence, they are not very scalable. Thus, XGBoost is focused on computational speed and model performance. XGBoost provides:

-
    - **Parallelization** of tree construction using all of your CPU cores during training.
    - **Distributed Computing** for training very large models using a cluster of machines.
    - **Out-of-Core Computing** for very large datasets that don't fit into memory.
    - **Cache Optimization** of data structures and algorithm to make the best use of hardware.

XGBoost Code Snippet:



```python
# Author: Dhrub Satyam
# XGBoost Code
# importing Libraries
from numpy import loadtxt
from xgboost import XGBClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

# load data
dataset = loadtxt('pima-indians-diabetes.csv', delimiter=",")

# split data into X and y
X = dataset[:,0:8]
Y = dataset[:,8]

# split data into train and test sets
seed = 7
test_size = 0.33
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=test_size, random_state=seed)

# fit model no training data
model = XGBClassifier()
model.fit(X_train, y_train)

# make predictions for test data
y_pred = model.predict(X_test)
predictions = [round(value) for value in y_pred]

# evaluate predictions
accuracy = accuracy_score(y_test, predictions)
print("Accuracy: %.2f%%" % (accuracy * 100.0))
```

85. What are the basic enhancements done to Gradient Boosting?

Ans: Gradient boosting is a greedy algorithm and can overfit a training dataset quickly. It can benefit from regularization methods that penalize various parts of the algorithm and generally improve the performance of the algorithm by reducing overfitting.
We will look at 4 enhancements to basic gradient boosting:

1. Tree Constraints
2. Shrinkage
3. Random sampling
4. Penalized Learning

1. **Tree Constraints**: A good general heuristic is that the more constrained tree creation is, the more trees you will need in the model, and the reverse, where less constrained individual trees, the fewer trees that will be required.

Below are some constraints that can be imposed on the construction of decision trees:

- **The number of trees**, generally adding more trees to the model can be very slow to overfit. The advice is to keep adding trees until no further improvement is observed.
- **Tree depth**, deeper trees are more complex trees, and shorter trees are preferred. Generally, better results are seen with 4-8 levels.
- **The number of nodes** or number of leaves, like depth, can constrain the size of the tree but is not constrained to a symmetrical structure if other constraints are used.
- **Number of observations per split** imposes a minimum constraint on the amount of training data at a training node before a split can be considered
- Minimum improvement to loss is a constraint on the improvement of any split added to a tree.

2. **Penalized Gradient Boosting:** Additional constraints can be imposed on the parameterized trees in addition to their structure. Classical decision trees like CART are not used as weak learners, instead, a modified form called a regression tree is used that has numeric values in the leaf nodes (also called terminal nodes). The values in the leaves of the trees can be called weights in some literature. As such, the leaf weight values of the trees can be regularized using popular regularization functions, such as **L1 regularization of weights and L2 regularization of weights.** The additional regularization term helps to smooth the final learned weights to avoid over-fitting. Intuitively, the regularized objective will tend to select a model employing simple and predictive functions.
3. **Weighted Updates**: The predictions of each tree are added together sequentially. The contribution of each tree to this sum can be weighted to slow down the learning by the algorithm. This weighting is called a shrinkage or a learning rate.
4. **Stochastic Gradient Boosting:** A big insight into bagging ensembles and the random forest was allowing trees to be greedily created from subsamples of the training dataset. This same benefit can be used to reduce the correlation between the trees in the sequence in gradient boosting models. This variation of boosting is called stochastic gradient boosting. At each iteration a subsample of the training data is drawn at random (without

replacement) from the full training dataset. The randomly selected subsample is then used, instead of the full sample, to fit the base learner.

86. What is Dimensionality Reduction? Why is it used?

Ans: Dimensionality reduction refers to the process of converting a set of data. That data needs to having vast dimensions into data with lesser dimensions. Also, it needs to ensure that it conveys similar information concisely.

Although, we use these techniques to solve machine learning problems. And the problem is to obtain better features for a classification or regression task.

87. What are the commonly used Dimensionality Reduction Techniques?

Ans: The various methods used for dimensionality reduction include:

- Principal Component Analysis (PCA)
- Linear Discriminant Analysis (LDA)
- Generalized Discriminant Analysis (GDA)

88. How does PCA work? When to use?

Ans: Short Answer: Principal Component Analysis (PCA) is an unsupervised, non-parametric statistical technique primarily used for dimensionality reduction in machine learning.

High dimensionality means that the dataset has a large number of features. The primary problem associated with high dimensionality in the machine learning field in model overfitting, which reduces the ability to generalize beyond the examples in the training set.

PCA in Layman's Term: Consider the 2D XY plane.

For the sake of intuition, let us consider variance as the spread of data – the distance between the two farthest points.

Assumption:

Typically, it is believed, that if the variance of data is large, it offers more information, than data that has a small variance. (This may or may not be true). This is the assumption which PCA intends to exploit.

I give you 4 points – {(1,1), (2,2), (3,3), (4,4)}

(all lie on the line X=Y)

What is the variance on X-axis?

Variance(X) = 4-1 = 3

What is the variance on Y-axis?

Variance(Y) = 4-1 = 3

Can we obtain new data with higher variance in some manner?

Rotate your XY system by 45 degrees anticlockwise. What happens? The line X=Y has now become the X(new)-axis. And, X = -Y is now the Y(new)-axis. Let's compute the variance again (in the form of distance)

Variance(X(new)) = distance ((4,4), (1,1)) = sqrt(18) = 4.24

Variance(Y(new)) =requires some calculations.

89. What did we get by doing this rotation?

Ans: Original data – had the highest variance on any axis as 3. This rotation gave us a variance of 4.24

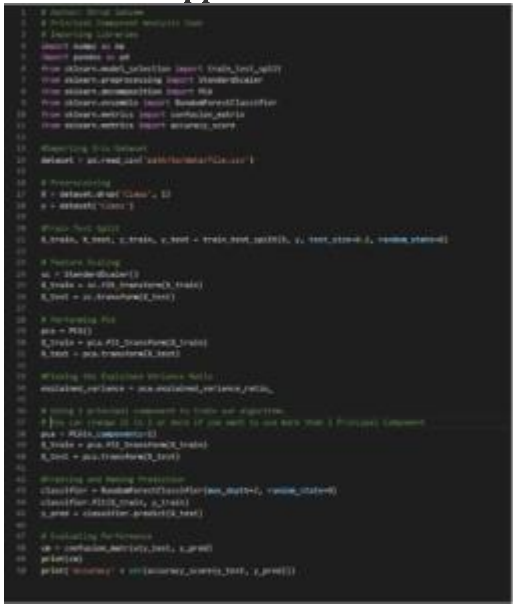
That was the intuitive explanation of what PCA does. Just for further clarification

Eigenvalues = variance of the data along a particular axis in the new coordinate system. In above example, Eigenvalue(X(new)) = 4.24.

Eigenvectors = the vectors which represent the new coordinate system. In above example, vector [1,1], would be an eigenvector for X(new), and [1,-1] eigenvector for Y(new). Since they are just directions – solvers typically give us unit vectors.

Getting transformed data
Once you have the eigenvectors, a dot product of the eigenvector with the original point will give you the new point in the new coordinate system.
Diagonalization: This is the part where you equate covariance to lambda*I. This is basically trying to find an eigenvector, such that all points would lie on the same line, and thus it will have only elements of variance, and covariance terms would be zero.
Steps of PCA:

1. Calculate the covariance matrix X of data points.
2. Calculate eigenvectors and correspond eigenvalues.
3. Sort eigenvectors accordingly to their given value in decrease order.
4. Choose first k eigenvectors and that will be the new k dimensions.
5. Transform the original n-dimensional data points into k-dimensions

PCA code snippet:


```
# [illegible]
# Principal Component Analysis [illegible]
# Importing Libraries
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score

#Importing Iris Dataset
dataset = pd.read_csv('[illegible].csv')

# Preprocessing
X = dataset.drop('Class', 1)
y = dataset['Class']

#Train Test Split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)

# Feature Scaling
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

# Performing PCA
pca = PCA()
X_train = pca.fit_transform(X_train)
X_test = pca.transform(X_test)

#Fixing the Explained Variance Ratio
explained_variance = pca.explained_variance_ratio_

# Using 1 principal component to train our algorithm.
# You can change it to 2 or more if you want to use more than 1 Principal Component
pca = PCA(n_components=1)
X_train = pca.fit_transform(X_train)
X_test = pca.transform(X_test)

#Training and Making Predictions
classifier = RandomForestClassifier(max_depth=2, random_state=0)
classifier.fit(X_train, y_train)
y_pred = classifier.predict(X_test)

# Evaluating Performance
cm = confusion_matrix(y_test, y_pred)
print(cm)
print('Accuracy' + str(accuracy_score(y_test, y_pred)))
```

90. How does LDA work? When to use?
Ans: LDA is a way to reduce 'dimensionality' while at the same time preserving as much of the class discrimination information as possible.
How does it work?
Basically, LDA helps you find the 'boundaries' around clusters of classes. It projects your data points on a line so that your clusters 'are as separated as possible', with each cluster having a relative (close) distance to a centroid.
What was that stuff about dimensionality?
Let's say you have a group of data points in 2 dimensions, and you want to group them into 2 groups. LDA reduces the dimensionality of your settings like so:
K(Groups) = 2. 2-1 = 1.
Why? Because "The K centroids lie in an at most K-1-dimensional affine subspace". What is the affine subspace? It's a geometric concept or *structure* that says, "I am going to generalize the affine properties of Euclidean space". What are those affine properties of the Euclidean space? Basically, it's the fact that we can represent a point with 3 coordinates in a 3-dimensional space (with a nod toward the fact that there may be more than 3 dimensions that we are ultimately dealing with).
So, we should be able to represent a point with 2 coordinates in 2-dimensional space and represent a point with 1 coordinate in a 1-dimensional space. LDA reduced the dimensionality of our 2-dimension problem down to one dimension. So now we can get down to the serious business of listening to the data. We now have 2 groups, and 2 points in any dimension can be joined by a line. How many dimensions does a line have? 1! Now we

are cooking with Crisco!

So we get a bunch of these data points, represented by their 2d representation (x,y). We are going to use LDA to group these points into either group 1 or group 2.

91. What are the Steps for LDA?

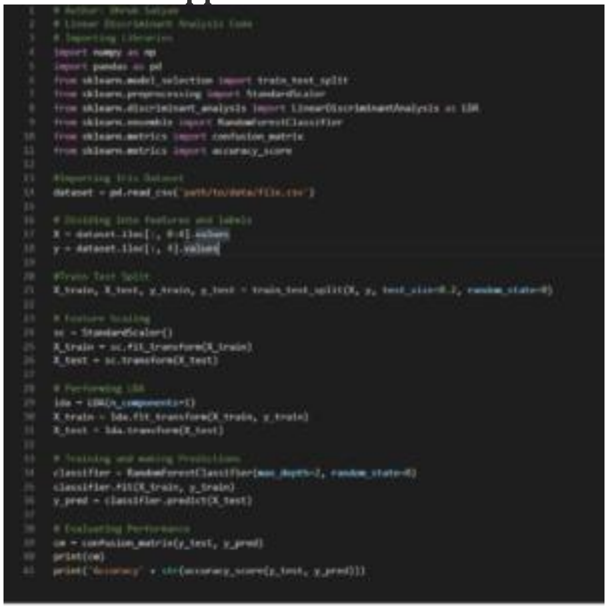
Ans: Steps of LDA:

1. 1. Compute the d-dimensional mean vector for the different classes from the dataset.
2. Compute the Scatter matrix (in between class and within the class scatter matrix)
3. Sort the Eigen Vector by decrease Eigen Value and choose k eigenvector with the largest eigenvalue to from a d x k dimensional matrix w (where every column represents an eigenvector)
4. Used **d * k** eigenvector matrix to transform the sample onto the new subspace.

This can be summarized by the matrix multiplication.

Y = X x W (where X is an **n * d** dimension matrix representing the n samples and **you** are transformed **n * k** dimensional samples in the new subspace.

LDA code snippet:



```
# [illegible]
# Linear Discriminant Analysis Code
# Importing Libraries
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score

#Importing Iris Dataset
dataset = pd.read_csv('path/to/data/file.csv')

# Dividing into features and labels
X = dataset.iloc[:, 0:4].values
y = dataset.iloc[:, 4].values

#Train Test Split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)

# Feature Scaling
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

# Performing LDA
lda = LDA(n_components=1)
X_train = lda.fit_transform(X_train, y_train)
X_test = lda.transform(X_test)

# Training and making Predictions
classifier = RandomForestClassifier(max_depth=2, random_state=0)
classifier.fit(X_train, y_train)
y_pred = classifier.predict(X_test)

# Evaluating Performance
cm = confusion_matrix(y_test, y_pred)
print(cm)
print('Accuracy' + str(accuracy_score(y_test, y_pred)))
```

92. What is GDA?

Ans: When we have a classification problem in which the input features are continuous random variable, we can use GDA, it’s a generative learning algorithm in which we assume p(x|y) is distributed according to a **multivariate normal distribution** and p(y) is distributed according to **Bernoulli**.

Gaussian discriminant analysis (GDA) is a generative model for classification where the distribution of each class is modeled as a multivariate Gaussian.

93. What are the advantages and disadvantages of Dimensionality Reduction?

Ans: Advantages:

- Dimensionality Reduction helps in data compression, and hence reduced storage space.
- It reduces computation time.
- It also helps remove redundant features, if any.
- Dimensionality Reduction helps in data compressing and reducing the storage space required
- It fastens the time required for performing the same computations.
- If there present fewer dimensions then it leads to less computing. Also, dimensions can allow the usage of algorithms unfit for a large number of dimensions.
- It takes care of multicollinearity that improves model performance. It removes redundant features. For example, there is no point in storing a value in two different units (meters and inches).

- Reducing the dimensions of data to 2D or 3D may allow us to plot and visualize it precisely. You can then observe patterns more clearly.

Disadvantages:

- Basically, it may lead to some amount of data loss.
- Although, PCA tends to find linear correlations between variables, which is sometimes undesirable.
- Also, PCA fails in cases where mean and covariance are not enough to define datasets.
- Further, we may not know how many principal components to keep- in practice, some thumb rules are applied.