Omgamganapathayenamaha

Omgamganapathayenamaha

Omgamganapathayenamaha

Omgamganapathayenamaha

Omgamganapathayenamaha

Good Things learnt during prep

1. Saving the Model

# Saving the Model

import pickle

with open('model.sav', 'wb') as fw:

pickle.dump(regressor, fw)

1. Loading the Model

# Loading the Model

with open('model.sav', 'rb') as fh:

regressor = pickle.load(fh)

1. X\_grid = np.arange(min(X), max(X), 0.01)
2. X\_grid = np.reshape(X\_grid, [len(X\_grid), 1])
3. LabelEncoding should be done column by column
4. While doing Feature Scaling with StandardScaler class, your data should be in a Matrix (if it is in a vector, you have to reshape it using np.reshape)
5. Feature Scaling has to be done for SVR, Logistic Regression, KNN
6. Mean, Variance and Standard Deviation

observations = [23, 40, 6, 74, 38, 1, 70]  
mean = 36  
difference\_from\_the\_mean = [13, 4, 30, 38, 2, 35, 34]  
square\_of\_the\_differences = [169, 16, 900, 1444, 4, 1225, 1156]  
variance = (169+16+900+1444+4+1225+1156)/7 = 4914/7 = 702

Standard Deviation = Square\_root(702) = 26.49

1. Standard deviation is an excellent way to identify outliers. Data points that lie more than one standard deviation from the mean can be considered unusual. In many cases, data points that are more than two standard deviations away from the mean are not considered in analysis. We can talk about how extreme a data point is by asking the question “how many sigmas away from the mean is this?”
   1. <https://medium.com/technology-nineleaps/basics-of-statistics-for-machine-learning-engineers-ii-d25c5a5dac67>
2. Performance of different Models can be compared using below methods:
   1. Accuracy Rate and Failure Rate using ‘Confusion Matrix’
   2. CAP Curve (Cumulative Accuracy Profile Curve)
   3. K-fold cross validation
3. 2 things to remember when running Clustering Algorithms
   1. No need to split the data into Training and Test Data (because this is UnSupervised Learning. No Training needed)
   2. No need to split the data into X and y (There is no y here because we don’t know how many clusters are there and what are those Clusters?). Infact, we have to find vector ‘y’ with Clustering algorithms

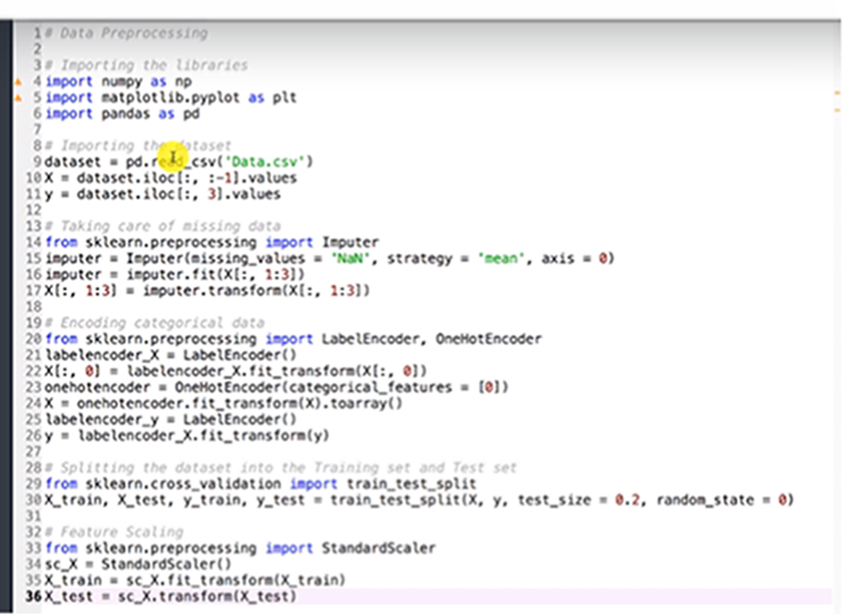
Points learnt from the Video

1. Applications of Machine Learning
   1. Face Recognition on Facebook (Supervised Learning based on the previous tagging) (Classification Algorithm) (Convolution Neural Networks)
   2. Kinect Sports (uses Random Forest Algorithm) (Supervised Learning) (I think it classifies to which part of the body is the bat closer to based on training set)
   3. Virtual Reality Headset
   4. Voice recognition on the Phone (Supervised Learning)
   5. Robot Dogs (They learn how to walk using Re-inforcement Learning)
   6. Facebook Ads
   7. Recommender Systems (Amazon, Netflix, audible, flipkart use apriori like algorithms and clustering algorithms with unsupervised learning)
   8. Medical Field
   9. In Space Field to “recognize certain areas of the world through maps”
   10. Explore new territories such as Mars
   11. Spam Filters (Binary Classifier) (Supervised Learning)
2. Data Exhaust
   1. Data exhaust refers to the data generated as trails or information byproducts resulting from all digital or online activities. These consist of storable choices, actions and preferences such as log files, cookies, temporary files and even information that is generated for every process or transaction done digitally. This data can be very revealing about an individual, so it is very valuable to researchers and especially to marketers and business entities
   2. This data is used to target advertisements to specific demographics and for market research, which tells businesses the online preferences, behaviors and habits of potential customers, giving them insight into how to mold their business into something that the people will consume. This is called behavioral targeting
   3. In more legitimate science, this can be used to improve digital and online processes based on the behaviors of users. We can look for shortcuts to minimize required actions and find areas to optimize, improve or change outright. Data exhaust is widely used in data mining and big data analytics
3. You can fit in a Human Genome (DNA) in 725MB (if not in 725MB, say 1 GB)
4. Bytes – KB – MB – GB – TB – Peta Bytes – EXA Bytes
5. Anaconda Distribution, Jupyter Notebook, Spyder
6. Get the dataset from https://[www.superdatascience.com/machine-learning](http://www.superdatascience.com/machine-learning)
7. In Spyder IDE, if you are not able see the full array in iPython Console, please do as below:
8. 
9. Data Pre-processing Steps
   1. Libraries we use in Data pre-processing
      1. import pandas # For managing datasets
      2. import numpy # For playing with Matrices
      3. import matplotlib.pyplot # For plotting
   2. Importing the Data
      1. dataset = pandas.read\_csv
      2. dataset.iloc
   3. Handling Missing Data
      1. from sklearn.preprocessing import Imputer
      2. DeprecationWarning: Class Imputer is deprecated; Imputer was deprecated in version 0.20 and will be removed in 0.22. Import impute.SimpleImputer from sklearn instead.

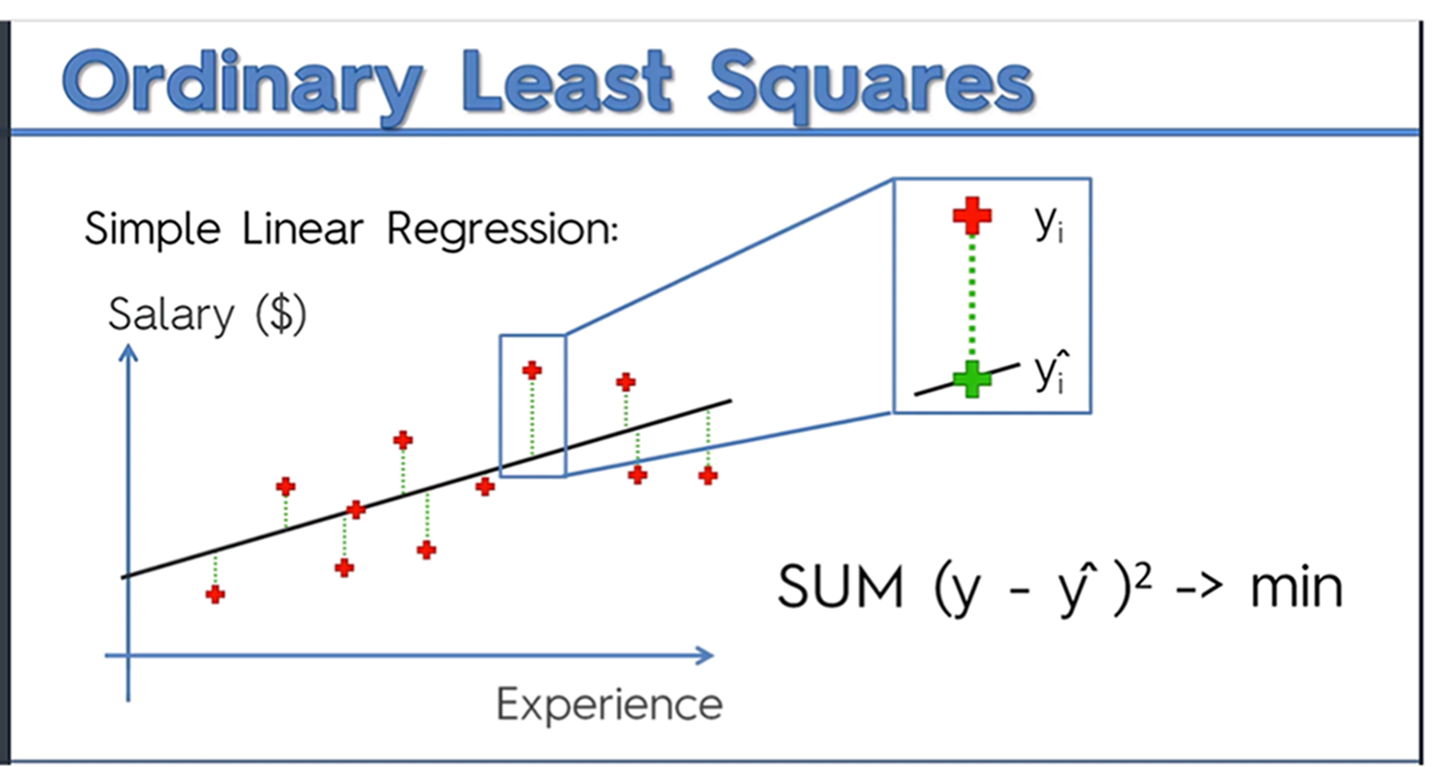
warnings.warn(msg, category=DeprecationWarning)

* + 1. numpy.nan
  1. Converting Categorical Data to Quantitative Data
     1. from sklearn.preprocessing import LabelEncoder
     2. Label Encoding should be done column by column (Not all at once)
  2. Introducing Dummy Variables
     1. from sklearn.preprocessing import OneHotEncoder
  3. Splitting the Data in to Training Set and Test Set
     1. from sklearn.model\_selection import train\_test\_split
  4. Feature Scaling
     1. from sklearn.preprocessing import StandardScaler

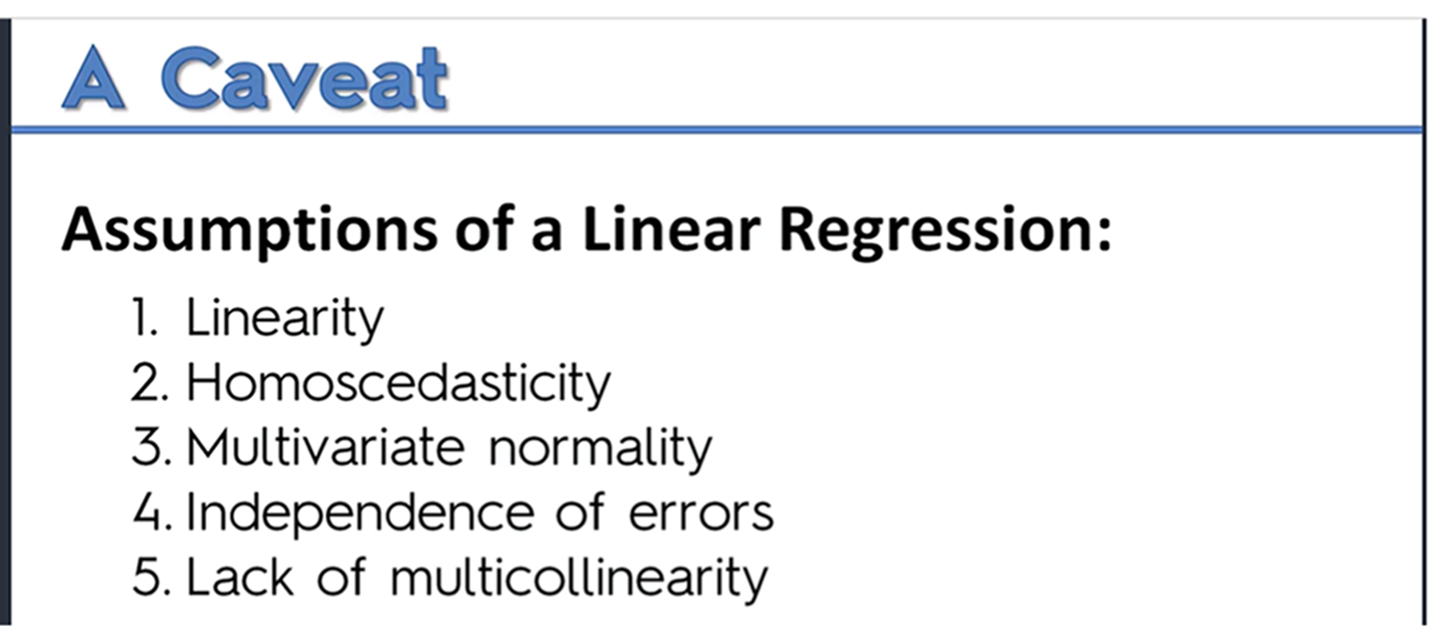
1. Data Pre-processing Template



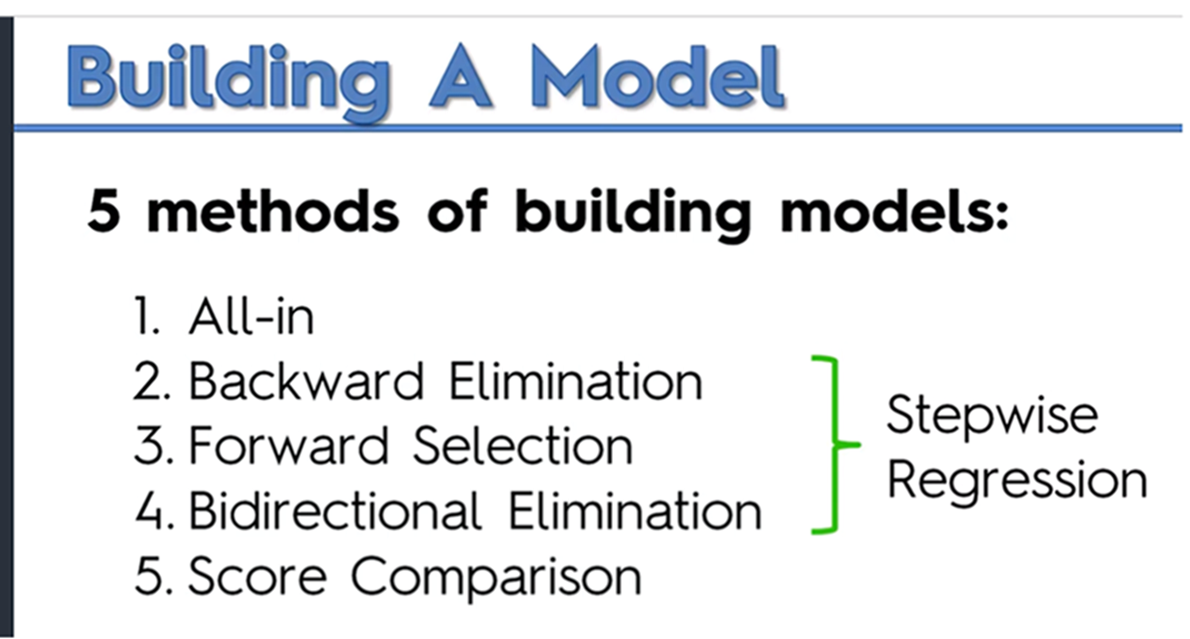
1. Simple Linear Regression
   1. y = mx + c (For example, Salary = m \* Experience + c)
   2. Given Training Data, the Idea is to find a best fitting line through Ordinary Least Squares where SUM((yi – yhat) ^ 2) is minimum (SS-residual)



1. What does it mean by creating a model?
   1. Using the Training Data, understand the correlation between the independent variables and dependent variables and come up with an equation
   2. Apply this equation on the test data to check whether the results are good
   3. If they are good, move the model to production for predicting on the live data
   4. If the model is not predicting correctly, use learning curves to see whether there is any bias (or) variance. Try to optimize the data, optimize the features and fix the issues. Move to point (a)
2. To create a Linear regression model, we should check whether the below things are satisfied:



1. Dummy Variable Trap
   1. If you have a column with categories, we have to convert that column in to multiple columns which are called ‘Dummy Variables’
   2. If we have ‘n’ dummy variable columns, we should include only ‘n-1’ dummy variable columns ignoring one
   3. Why we should ignore that one column? If a column can derive other column, then that is called ‘Multicollinearity’ which we should avoid as per the assumption above
   4. If we consider all ‘n’ columns, then we are in a ‘Dummy Variable Trap’
2. What is P-Value?
3. Methods of Building Models



1. All-in: Add all the variables you have without any filtering (May be because you know ahead all these are significant from your domain knowledge (or) may be your mgmt. asked you to do so)
2. Backward Elimination: Eliminate variables with p-value > SL(Significance Level) and continue until you find a maximum p-value which is less than or equal to SL (we have to create at max O(n) models)
3. Forward Selection: we have to create at max O(n ^ 2) models i.e., (n + n-1 + n-2 + …. + 1)
   1. k = n
   2. Start with ‘k’ variables and create ‘k’ linear regression models
   3. Select the variable with lowest p-value and add this to the previous model if p-value < SL. If p-value > SL, goto step e
   4. Make k = k – 1 and move to step b
   5. Consider your previous model and that is the final one
4. Bi-directional elimination:
   1. Add a new variable to the model based on Forward Selection
   2. Apply Backward-elimination to see how many variables we can eliminate
   3. Continue steps a and b until there are no variables to add using forward-selection and no variables to delete using backward-elimination
5. Score Comparison:
   1. Build all models with all the combinations i.e., if you have ‘n’ variables, build all combinations

nc1 + nc2 + nc3 + … + ncn = (2 ^ n) – 1

* 1. Extremely resource consuming and it is not practical

1. Multiple Linear Regression Model

y = b0 + b1\*x1 + b2\*x2 + b3\*x3 + … + bn\*xn

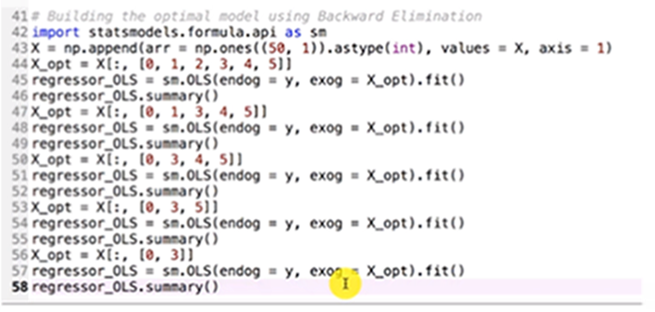
1. Optimizing the built model through backward elimination (statsmodel library helps us in getting the statistics about our independent variables)

import statsmodel.formula.api as sm

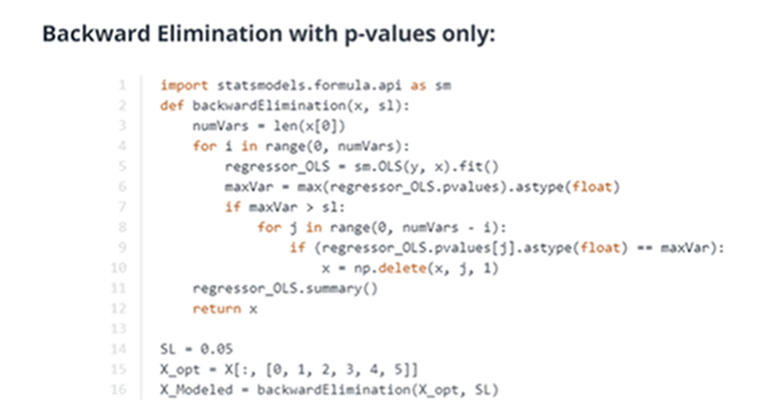
use np.append and np.ones to add column of ones with respect to b0\*x0 (column of ones correspond to x0)

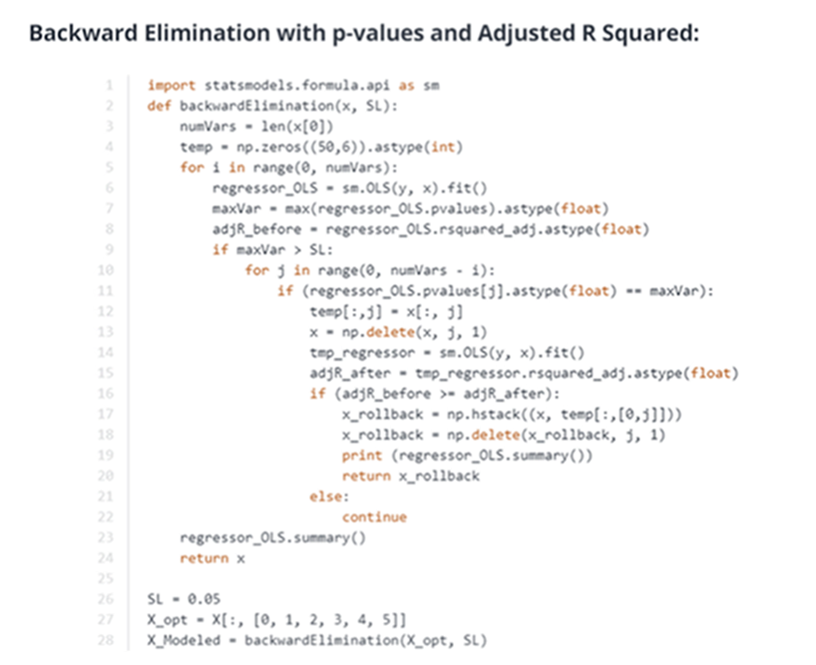
regressor\_OLS = sm.OLS() # Fitting the OLS class to our data

regressor\_OLS.summary() # will give us the r-squared, adjusted r-squared, p-values of all independent variables etc.

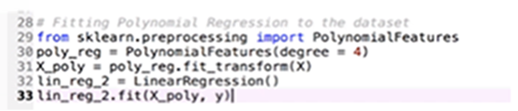


1. Automatic Backward Elimination using p-values





1. Polynomial Regression
   1. y = b0 + b1\*x1 + b2\*x1^2 + b3 \* x1 ^ 3 + … + bn \* x1 ^ n
   2. We only have one independent variable (x1) and one dependent variable (y)
   3. We will add more variables i.e., x1 ^ 2, x1 ^ 3 etc. as say x2, x3 and will call Multiple Linear Regression on the data



1. To convert 10 points in X to say 100 points to obtain more smoother curve, use ‘np.arange’ and ‘np.reshape’
   1. X\_grid = np.arange(min(X), max(X), 0.01)
   2. X\_grid = np.reshape(X\_grid, [len(X\_grid), 1])
2. SVR (Support Vector Regression): SVR has a different goal than Linear Regression. In linear regression, we are trying to minimize the error between the prediction and the actual data. In SVR, our goal is to make sure that errors do not exceed the threshold (i.e., epsilon)
   1. We have linear kernel as well as non-linear kernels to use with SVR
   2. Best one among “Non-linear Kernels” is “Gaussian Kernel”
   3. You have to “Feature Scale” the data while using SVR
   4. Non-linear, continuous model
   5. Library and the Class to use

from sklearn.svm import SVR

regressor = SVR(kernel=’rbf’)

regressor.fit(X,y)

* 1. What are Support Vectors?
  2. SVR and SVC work well on corner cases. For example, we are regressing or classifying Apples and Oranges. Normal models classify all the normal cases properly as long as the orange is in orange color and the apple is in red color. SVR and SVC can classify properly the cases where we have red oranges and orange apples. That’s the power of SVR and SVC

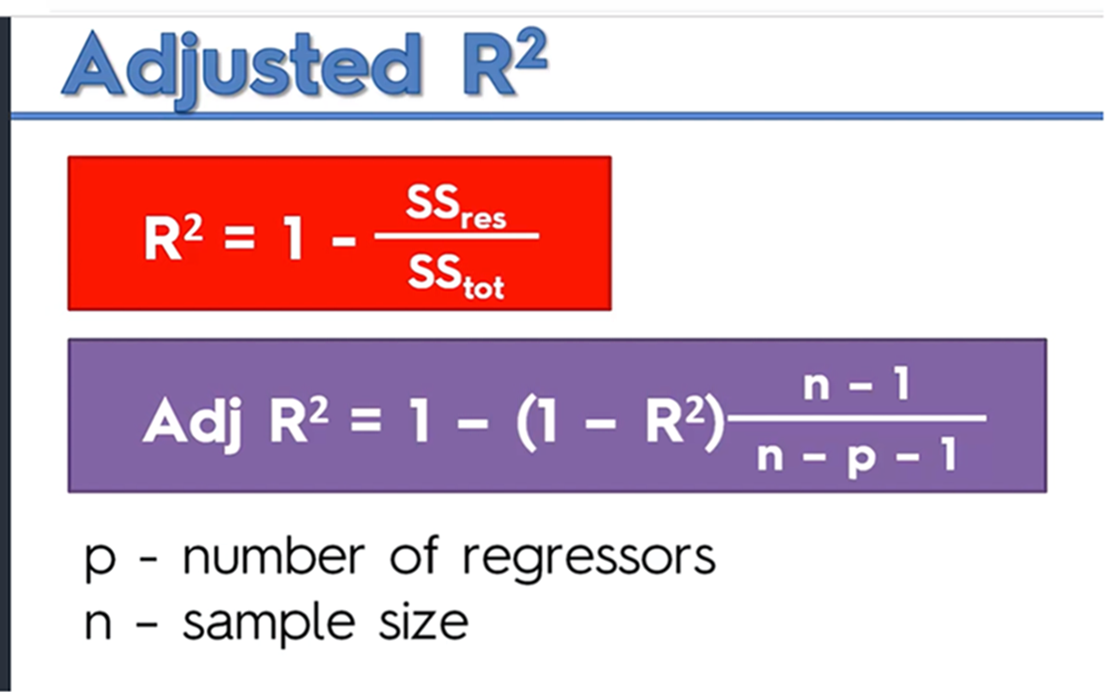
1. Decision Trees
   1. CART: Classification and Regression Trees
   2. Given data will be split based on factors like ‘Information Entropy’, ‘Gini Impurity’ etc. and will create few groups
      1. Entropy = Sigma of -pi logpi
   3. Algorithm will stop splitting the data based on factors like ‘number of records in the leaves’ or when the split is not improving the information
   4. Non-linear, Non-continuous model
   5. Library and the Class to use

from sklearn.tree import DecisionTreeRegressor

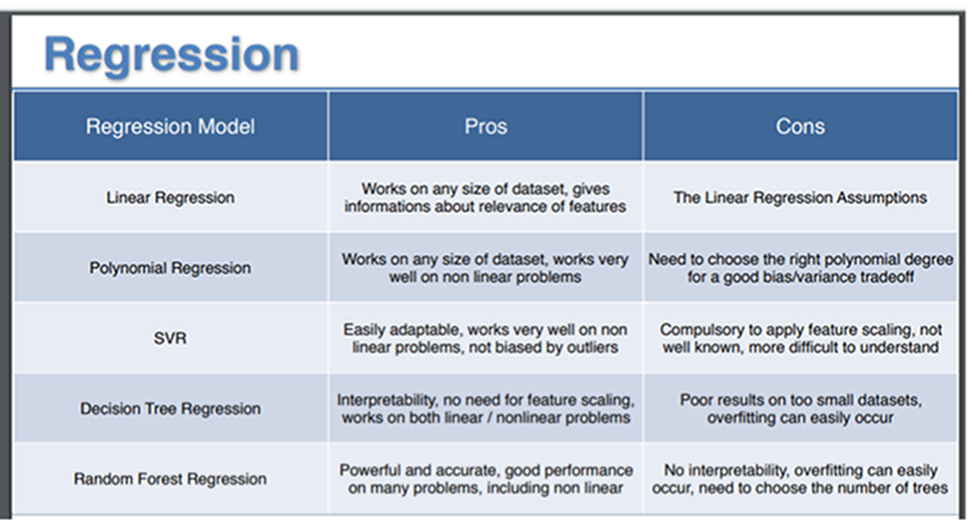
regressor = DecisionTreeRegressor(random\_state = 0)

regressor.fit(X, y)

1. Ensemble Learning: Take multiple algorithms (or) the same algorithm multiple times and make the results better. These algorithms are more stable and are not affected by the change in the data
   1. Random Forest
   2. Gradient Boosting
2. Random Forest
   1. Non-linear, ensemble, non-continuous model
   2. Library and the Class to use
      1. from sklearn.ensemble import RandomForestRegressor
      2. regressor = RandomForestRegressor(n\_estimators=300, random\_state=0)
      3. regressor.fit(X, y)
3. We need more than “p-values of the independent variables” to optimize our models
   1. R-Squared : With the addition of independent variables, R-Squared always increases. So, we need a better parameter
   2. Adjusted R-Squared : This is a best parameter to be used along with p-values as this has a penalization factor and uses R-Squared in its equation
   3. R-Squared and Adjusted R-Squared should be closer to 1 to be better. The more closer to 1, much better
   4. SS-res = SUM ( yi – yhat ) ^ 2 where
      1. yi = the real observation
      2. yhat = the predicted value
   5. To come up with a Model, Algorithm finds a line with minimum SS-res
   6. SS-tot = Sum (yi – yavg) ^ 2



1. Pros and cons of the Regression Algos we have seen:



1. How to select a proper Regression Algo?
   1. Find out whether the problem is linear (or) non-linear
   2. If linear: One Independent Variable – Simple Linear Regression, More Independent variables – Multiple Linear Regression
   3. Else, choose non-linear regression algos (Polynomial Regression, SVR, Decision Tree, Random Forest)
      1. Based on k-fold cross validation, you can choose the best among the above 4
2. How to evaluate your Model?
   1. Learning Curves
   2. K-fold cross validation
3. How to improve the performance of your models?
   1. Tune your Hyper Parameters (Hyper Parameters are non-learnable parameters such as regularization parameter lambda (or) the penalty parameter C)
   2. Apply Model optimization techniques like Backward Elimination
4. Classification Models: Instead of trying to predict a number, unlike regression, classification models aim at predicting a category
5. Logistic Regression (Linear Classifier)
   1. Just like Linear regression where it uses y = mx + c equation to find out the best line that reduces the error, Logistic regression uses “ln (p / (1 – p)) = mx + c” (sigmoid function) to find out the best sigmoid curve that reduces the overall error
   2. After finding out the best sigmoid curve, it uses that curve to predict the probabilities (i.e., values b/n 0 and 1) for the given data
   3. Now, it is up to you to segregate that predicted probabilities in to different categories. For example,
      1. Binary Classifier: Category False: 0 to 0.5 and Category True: 0.5 to 1
      2. More than 2 categories: Category1: 0 t 0.25 category2: 0.25 to 0.5 Category3: 0.5 to 0.75 Category4: 0.75 to 1
   4. Library and Class to use

from sklearn.linear\_model import LogisticRegression

classifier = LogisticRegression(random\_state = 0)

classifier.fit(X\_train, y\_train)

* 1. Making the Confusion Matrix

from sklearn.metrics import confusion\_matrix

cm = confusion\_matrix(y\_test, y\_pred)

* 1. We have to Feature Scale the data when using Logistic Regression

1. Code to plot the classification regions, classification boundary and the data points



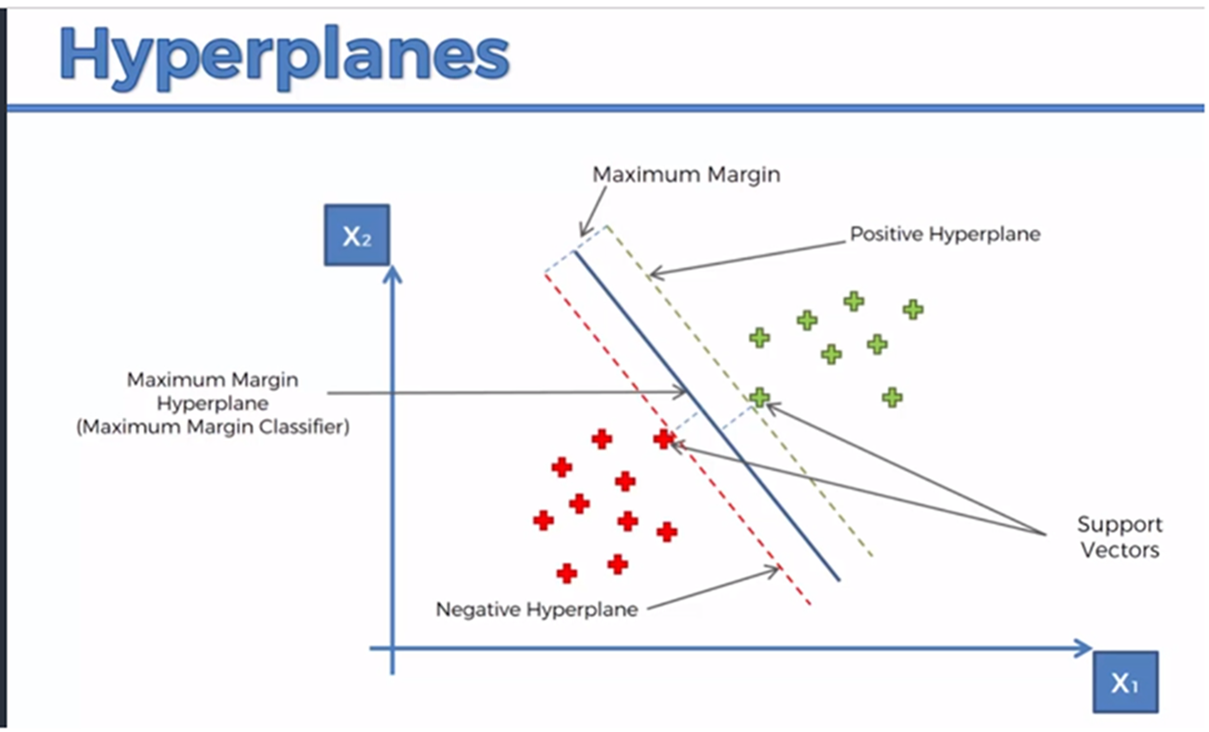
1. K-NN Classifier
   1. Classifies data points based on k nearest neighbors
   2. Nearest points are identified based on Euclidean or Manhattan Distance and the model assigns category which dominates the k nearest points
   3. K-NN is a non-linear classifier. If the data is not linearly separable, we can use this classifier
   4. Library and Class to use

from sklearn.neighbors import KNeighborsClassifier

classifier = KNeighborsClassifier(n\_neighbors = 5, metric = ‘minkowski’, p = 2)

classifier.fit(X\_train, y\_train)

1. SVM (Support Vector Machine)
   1. Assume there are 2 categories of data points: positive category and negative category
   2. In these 2 sets of points, find out 1 extreme point each from each category which are closest to each other
   3. These 2 points are called ‘Support Vectors’
   4. Equidistant to these 2 points, draw a line (hyperplane if it is more than 2 axes) perpendicular to both the points
   5. That line is called ‘Maximum Margin Hyperplane’ and that is the linear decision boundary



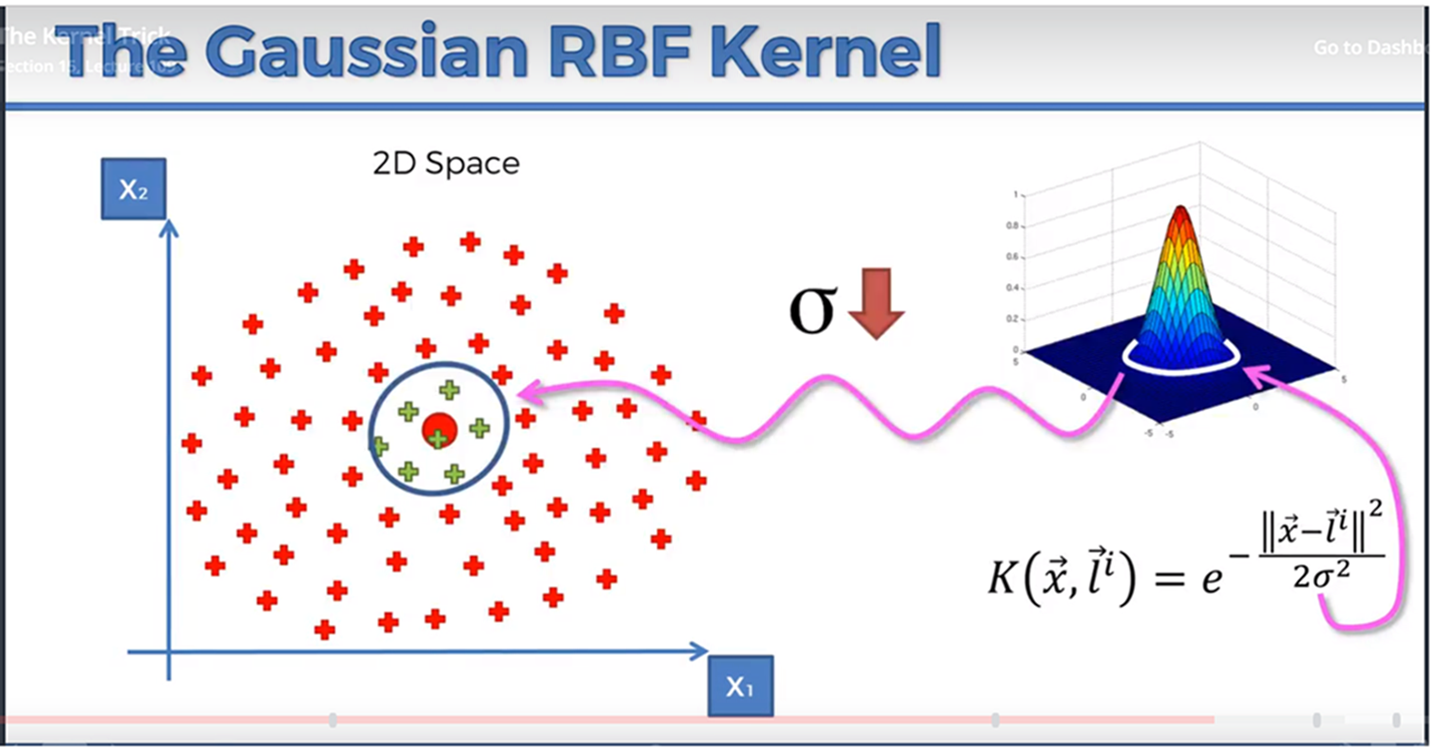
1. SVMs are powerful because they consider the extreme cases rather than stock-standard cases considered by other algorithms
2. SVM Classifiers can be linear and non-linear based on the kernel we use
3. SVM Linear Kernel: Library and Class to use

from sklearn.svm import SVC

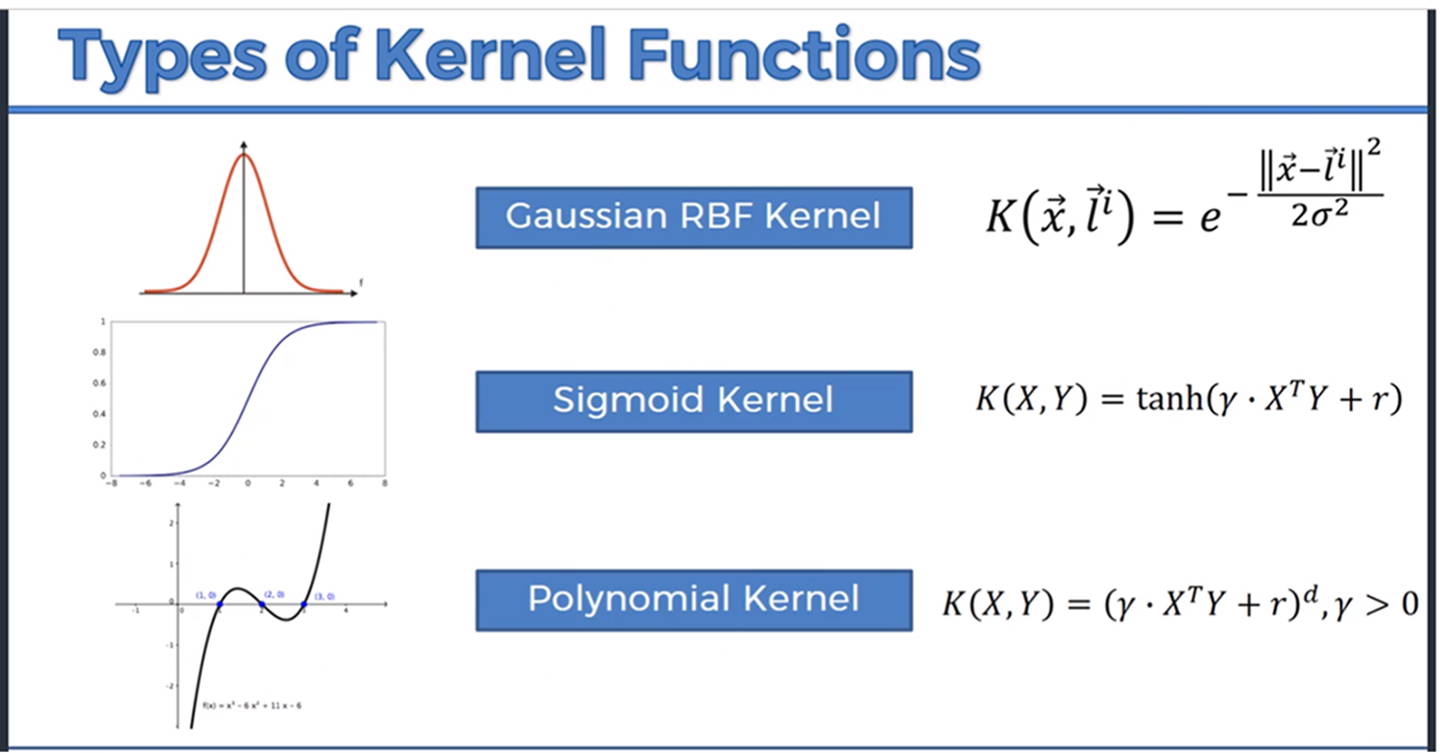
classifier = SVC(kernel=’linear’, random\_state = 0)

classifier.fit(X\_train, y\_train)

1. If the data is linearly separable, we can directly apply the SVM Classifier
2. If the data is not linearly separable, we can take the data to the “higher dimension space” using “Mapping Functions” and then we can separate the data linearly (it’s possible)
3. Mapping to a higher dimensional space is computationally intensive and so we don’t want to follow this way. Instead we apply the kernel trick
4. Gaussian RBF Kernel Trick
   1. Without going in to the higher dimensional space, using an exponential function and Sigma, convert the data points to something between 0 and 1
   2. If the output is much greater than 0, classify that data point in green category (if it is nearer to 0, it is red)
   3. With this Gaussian Kernel, we can classify much complex data sets which have complex decision boundaries



1. Gaussian kernel is not the only kernel available. There are other popular choices as well



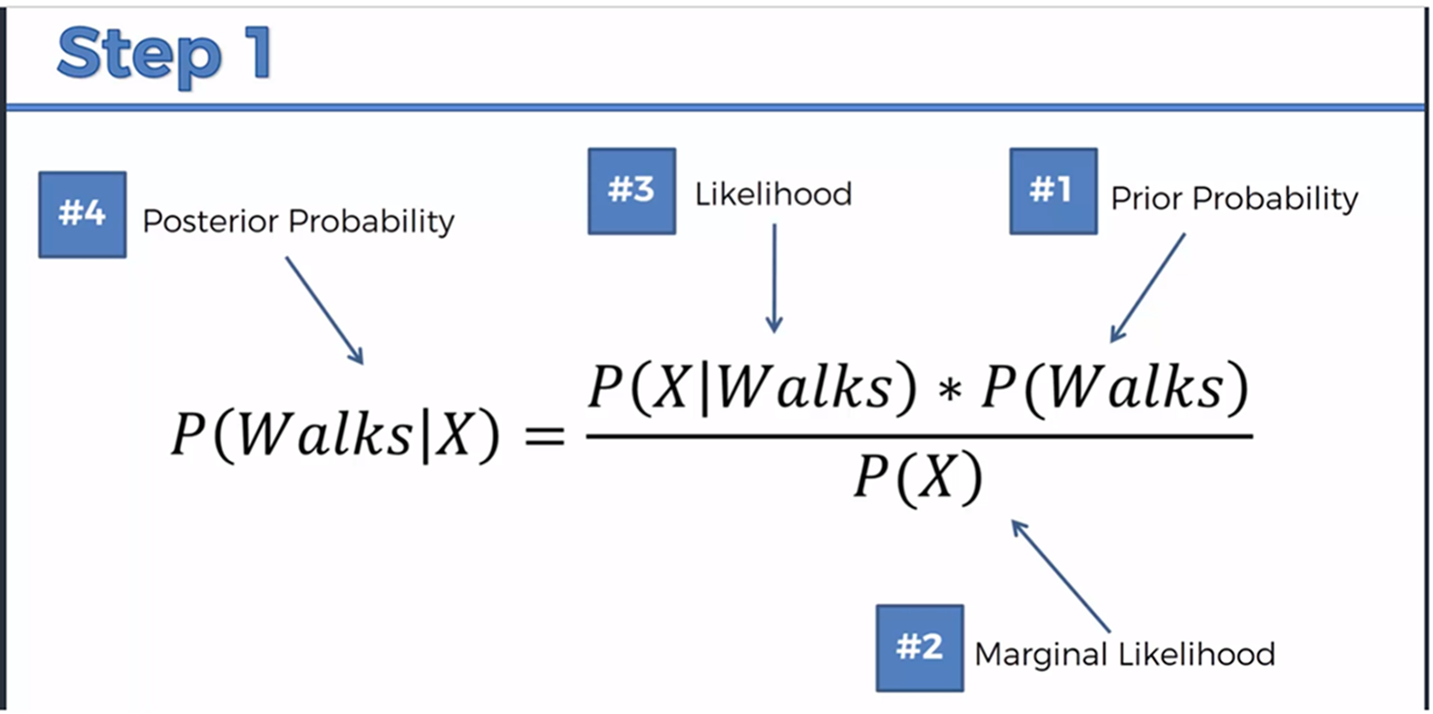
1. SVM Gaussian Kernel: Library and Class to use

from sklearn.svm import SVC

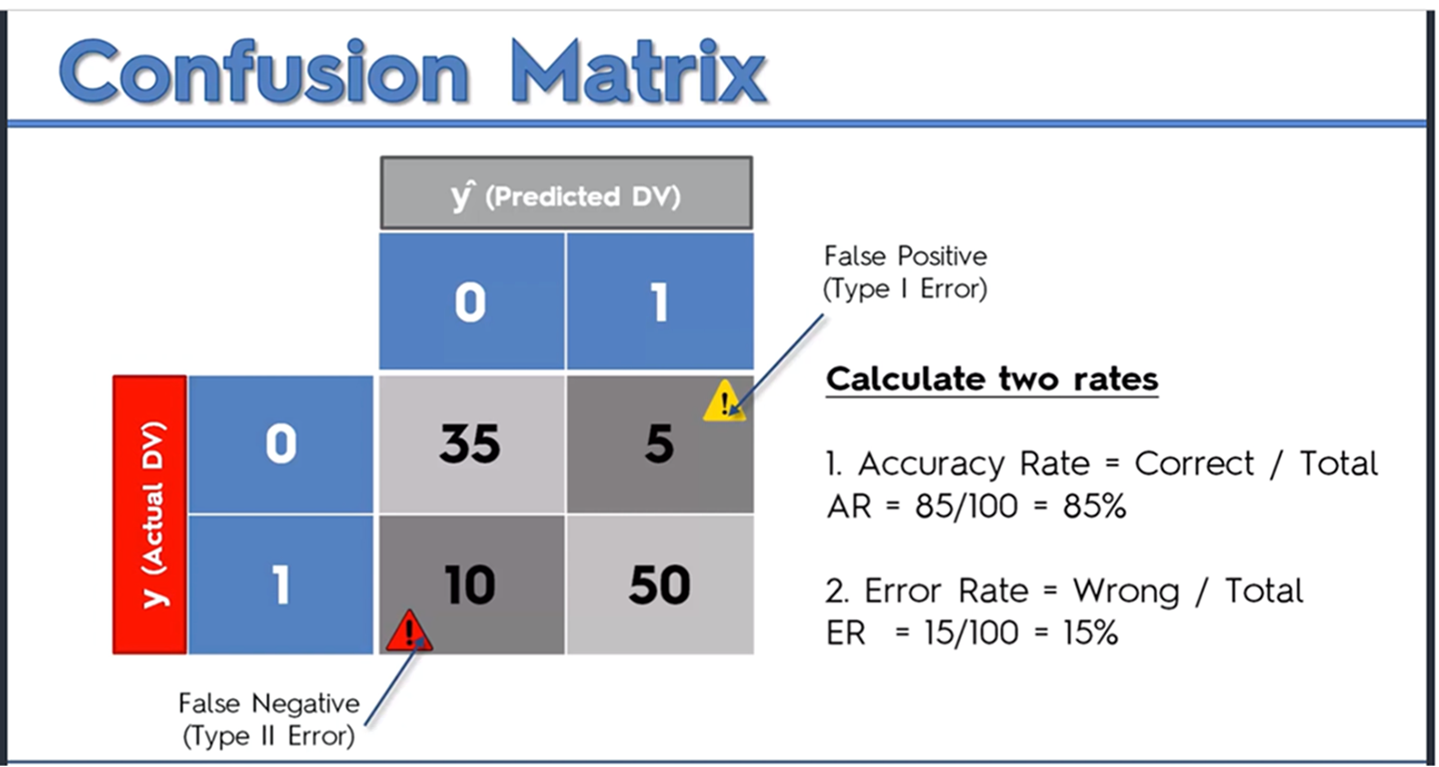
classifier = SVC(kernel=’rbf’, random\_state = 0)

classifier.fit (X\_train, y\_train)

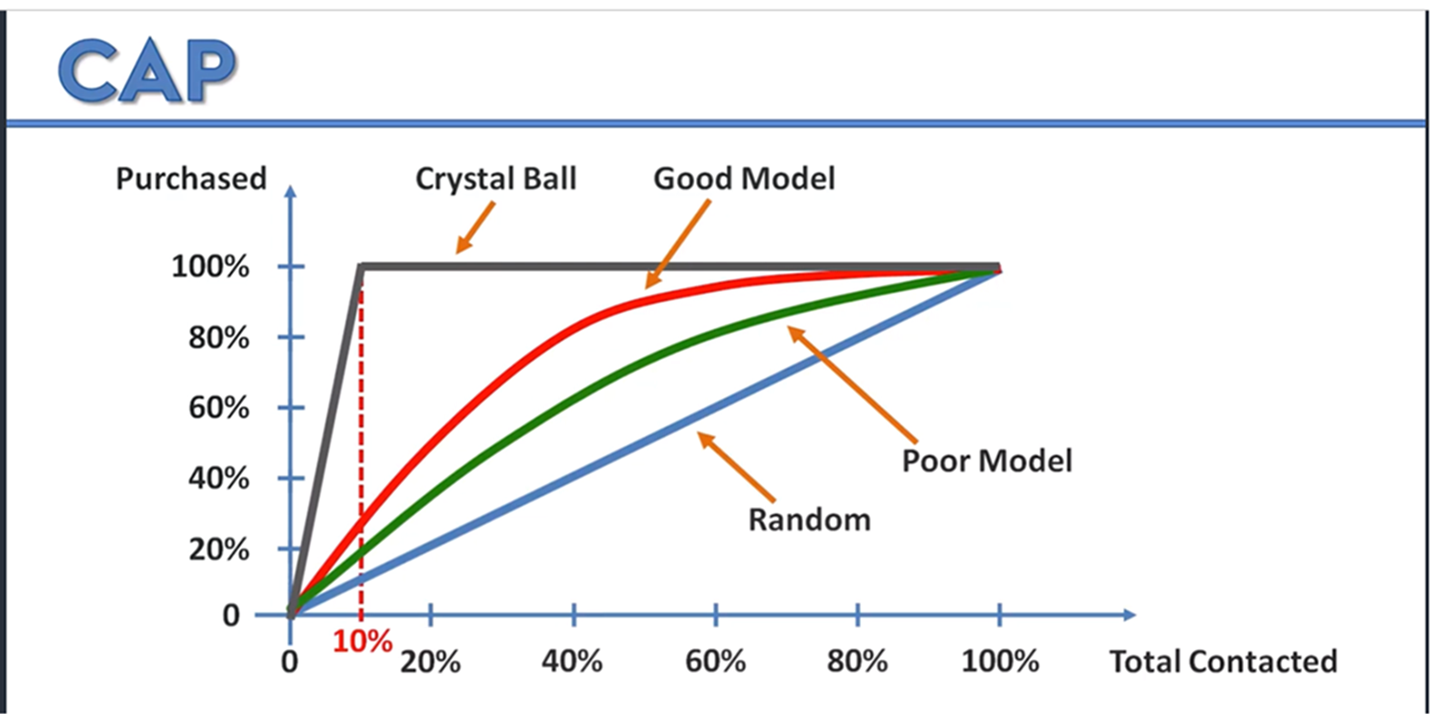
1. Naïve Bayes classifier
   1. This classifier is based on the Bayes Theorem which is based on conditional probability
   2. P(category | data\_point\_X) = P(data\_point\_X | category) \* P(category) / P(data\_point\_X)
   3. If there are 3 categories, we will run the above formula for all the 3 categories and see which category the data\_point\_X is falling in to based on the highest probability number
   4. If the data point X has say 4 features, Bayes theorem assumes all these 4 features to be independent of each other
   5. Generally, it will not be the case and mostly all the features are related. Still, we will go ahead and apply Bayes theorem and that’s why it is called Naïve Bayes



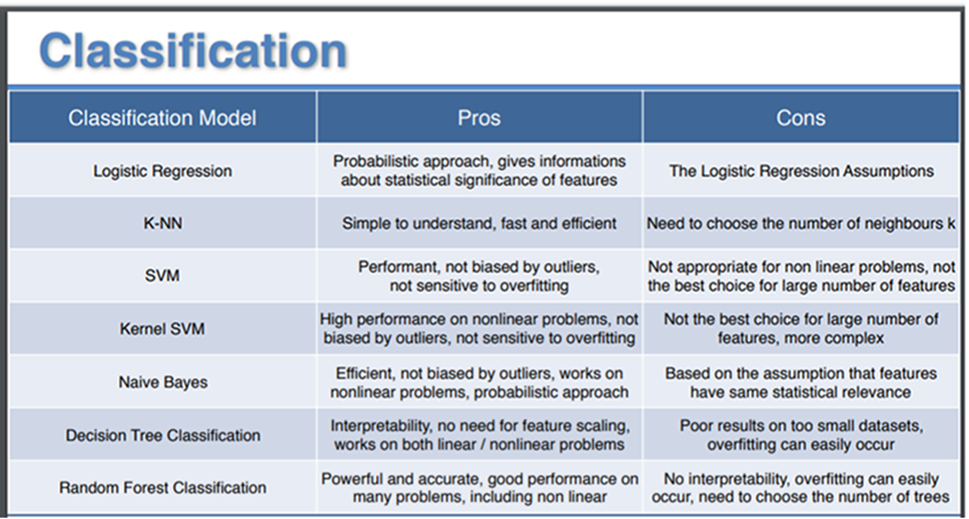
1. Library and Class to use
   1. from sklearn.naive\_bayes import GaussianNB
   2. classifier = GaussianNB()
   3. classifier.fit(X\_train, y\_train)
2. Decision Tree Classifier
   1. Decision Tree splits the data in to different buckets based on “entropy” (information gain), “gini impurity” etc.
      1. Entropy = Sigma –pi logpi
   2. Whenever a split is made, the idea is, we should have “information gain” i.e., the split nodes should be more homogeneous
   3. The difference b/n the homogeneous factor before and after split is called the “information gain”
   4. Split should increase the “information gain” and reduce the “entropy” (entropy = 0 means the node contains all the data points of same category)
   5. Library and class to use
      1. from sklearn.tree import DecisionTreeClassifier
      2. classifier = DecisionTreeClassifier(criterion = ‘entropy’, random\_state = 0)
      3. classifier.fit(X\_train, y\_train)
3. Random Forest Classifier
   1. It is one of the ensemble learning methods
   2. It builds large number of Decision Trees. Each tree is built by taking randomly K out of N data points in the training set
   3. For each record in the test data set, outcome is predicted by taking the outcome of all the trees and the maximum vote wins
   4. Library and Class to use
      1. from sklearn.ensemble import RandomForestClassifier
      2. classifier = RandomForestClassifier(n\_estimators = 300, criterion = ‘entropy’, random\_state = 0)
      3. classifier.fit(X\_train, y\_train)
4. Confusion Matrix, False positives and False Negatives
   1. False positives (Type 1 Errors): Model predicted the outcome as positive but actually it is not
   2. False Negatives (Type 2 Errors): Model predicted the outcome as negative but actually it occurred
   3. False Negatives are more dangerous than False Positives especially considering the eg., Earthquakes, Medical Field situations etc.



1. Accuracy Paradox
   1. Sometimes, by completely stopping using any ML model and saying eg., our prediction is always 0, accuracy rate might go up (this is called accuracy paradox)
   2. For example, our model’s Confusion Matrix is [9700, 150, 50, 100] … Accuracy rate = 9800 / 10000 = 98%
   3. We said we are stopping the model usage and always our prediction will be ZERO … with this our confusion matrix becomes [9850, 0, 150, 0] … Accuracy rate = 9850 / 10000 = 98.5%
   4. So, we have to use better attributes than “Accuracy Rate” such as “CAP”
2. CAP (Cumulative Accuracy Profile) Curve
   1. CAP Curve is used to compare models’ performance based on the Area under the Curve



1. CAP is not the same as ROC (Receiver Operating Characteristic)
2. CAP Analysis: At x-axis 50% mark, draw a vertical line on to your model and then from there a horizontal line on to the y-axis
   1. If the y-axis mark is < 60%, your model is rubbish
   2. If the y-axis mark is between 60% and 70%, your model is poor
   3. If the y-axis mark is between 70% and 80%, your model is good
   4. If the y-axis mark is between 80% and 90%, your model is too good
   5. Above that, you may have to check for over-fitting in your model ☺
3. Pros and Cons of classification models



1. Clustering
   1. Clustering basically falls in to Unsupervised Learning
   2. We don’t know ahead how many clusters (or) Segments will be present in our data
   3. Using a method, we will come up with a number K suggesting the number of clusters
      1. Elbow method
      2. Dendrograms
2. K-Means Clustering
   1. We will start with a number K which tells us there are K clusters in the data we are dealing with (This K will be found out by a method called Elbow method)
   2. Start with K centroids (you can almost start randomly or better use ‘k-means++’)
   3. Assign each data point to one of K centroids based on the Distance (can be Euclidean Distance or any other Distance selected)
   4. Now we have K clusters
   5. Adjust the centroids by calculating the average of the points in each cluster
   6. Data points might get re-assigned because of the new centroids. If there is any re-assignment of data points, go to step e. Else, we are done
   7. Library and Class to use
      1. from sklearn.cluster import KMeans
      2. km = KMeans(n\_clusters = 5, init = ‘k-means++’, n\_init = 10, max\_iter = 300)
      3. y\_pred = km.fit\_predict(X)
3. Random Centroid Initialization Trap
   1. With K-Means, selection of centroids randomly at different places will yield different clusters. This is a bit bad and is called Random Initialization Trap
   2. We can overcome this by using K-Means++ algorithm through which we can combat the Random Initialization Trap
4. Coming up with optimal number of clusters (K)
   1. The attribute we use for finding the optimal number of clusters is WCSS (Within Cluster Sum of Squares)
   2. Paste the formula here
   3. We plot a graph with Number of Clusters (from 1 to #Data Points) on the X-axis and WCSS value on the y-axis
   4. By looking at the plotted graph, we can identify the optimal number of clusters (Number where the WCSS value slows down on the reduction)
   5. This method is called Elbow Method (Paste the graph)
   6. Library and Class to use
      1. from sklearn.cluster import KMeans
      2. km = KMeans(n\_clusters = i, init = ‘k-means++’, n\_init = 10, max\_iter = 300)
      3. km.fit(X)
      4. wcss.append(km.inertia\_) # intertia\_ is the WCSS
5. Hierarchical Clustering
   1. We will start with a number K which is the optimal number of clusters (K is found using Dendrograms)
   2. We will start with ‘n’ clusters corresponding to ‘n’ data points
   3. We will merge 2 closest clusters at a time based on the minimum cluster distance (from n clusters, we will come to n-1, n-2, … K)
   4. We can see that all the data points are assigned to one of these K clusters. We are done
   5. Library and Class to use
      1. from sklearn.cluster import AgglomerativeClustering
      2. hc = AgglomerativeClustering(n\_clusters = 5, linkage = ‘ward’, affinity = ‘euclidean’)
      3. y\_pred = hc.fit\_predict(X)
6. Coming up with optimal number of clusters (K)
   1. Start with ‘n’ clusters corresponding to ‘n’ data points
   2. Merge 2 clusters at a time in each step based on the minimum cluster distance between clusters
   3. So, from n clusters, we will come to n-1, n-2, … , 1
   4. In this process of reducing from ‘n’ clusters to ‘1’ cluster, store the entire process of which cluster got merged with which cluster
   5. When we represent this merging process, we have a Dendrogram
   6. In the Dendrogram, find the longest vertical line which is not bisected by any horizontal line
   7. If we draw a horizontal line across this ‘longest vertical line’, the number of vertical lines that got cut by this horizontal line gives us the ‘optimal number of clusters’ i.e., K
   8. Library and Class to use
      1. import scipy.cluster.hierarchy as sch
      2. dendrogram = sch.dendrogram(sch.linkage(X, method = ‘ward’))
7. Association Rule Learning. Using ARL, we can solve problems like
   1. People who bought this also bought this …
   2. People who watched this also watched this …
   3. People who did this also did this …
8. Apriori
   1. Support
      1. Movie Recommendation: Support(M) = Total Number of users watched Movie M / Total Number of Users (M can be more than one movie here)
      2. Market basket Optimization: Support(I) = Total Number of transactions containing I / Total Number of Transactions
   2. Confidence
      1. Movie Recommendation: Confidence(M1 -> M2) = Total Number of Users who watched both M1 and M2 / Total Number of Users who watched M1
      2. Market Basket Optimization: Confidence(I1 -> I2) = Total Number of Transactions containing both I1 and I2 / Total Number of Transactions containing I1
   3. Lift
      1. Movie Recommendation: Lift(M1 -> M2) = Confidence(M1 -> M2) / Support(M2)
      2. Market Basket Optimization: Lift(I1 -> I2) = Confidence(I1 -> I2) / Support(I2)
9. Apriori Example
   * 1. Support(Movie X-Machina) = # of users who watched X-Machina / Total # of users = 10 / 100 = 10%
     2. Confidence(Interstellar -> X-Machina) = # of users who watched both Interstellar and X-Machina / # of users who watched Interstellar = 7 / 40 = 17.5%
     3. Lift(Interstellar -> X-Machina) = Confidence(Interstellar -> X-Machina) / Support(Movie X-Machina) = 17.5% / 10% = 1.75
10. Eclat
    1. Just uses Support
       1. Movie Recommendation: Support(M) = Total Number of users watched Movie M / Total Number of Users (M can be more than one movie here)
11. Learning Curves are one way to find out whether our model is good (or) suffering from bias (or) suffering from variance (We can plot curves and check)
12. One more way of evaluating the model performance is through “k-fold cross validation”
    1. This method divides the Training set in to k-folds (generally k = 10)
    2. The model will be run k times. In each execution, one fold will be test set and other k-1 folds will be training set
    3. We will get k accuracies of the model
    4. Calculate the Mean and Standard Deviation from the accuracies
    5. Our Model accuracies should be within (Mean + Standard Deviation) and (Mean – Standard Deviation). If it is the case, then we don’t have much variance
    6. If the accuracies are good, we don’t have much bias. Otherwise, we have bias in the data and we can say model is suffering from bias
13. Implementing k-fold cross validation

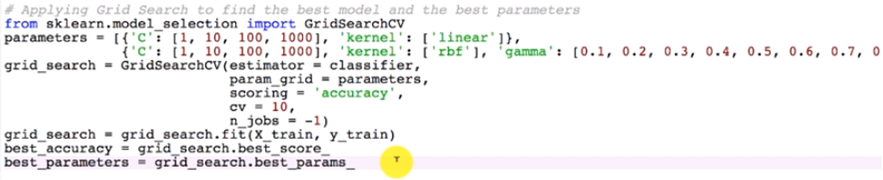
from sklearn.model\_selection import cross\_val\_score

accuracies = cross\_val\_score(estimator = classifier, X = X\_train, y= y\_train, cv = 10) # cv is the number of folds

print(accuracies.mean())

print(accuracies.std())

1. Improving the model performance. We can improve the model performance by
   1. Choosing the optimal values for ‘Hyper Parameters’
      1. We can do this by ‘Grid Search’
   2. Choosing between ‘linear’ or ‘non-linear’ models
      1. We can do this by checking whether the data is linearly separable
      2. We can also do this by ‘Grid Search’
2. Grid Search internally uses ‘k-fold cross validation’ to come up with the optimal values for hyper parameters
3. Implementing GridSearch



1. ‘C’ is the penalty parameter which is used to reduce/eliminate the over-fitting of the model with the data
2. ‘n\_jobs’ is used to use your hardware to the maximum extent when you have huge data
3. XGBoost is one of the best ensemble algorithms

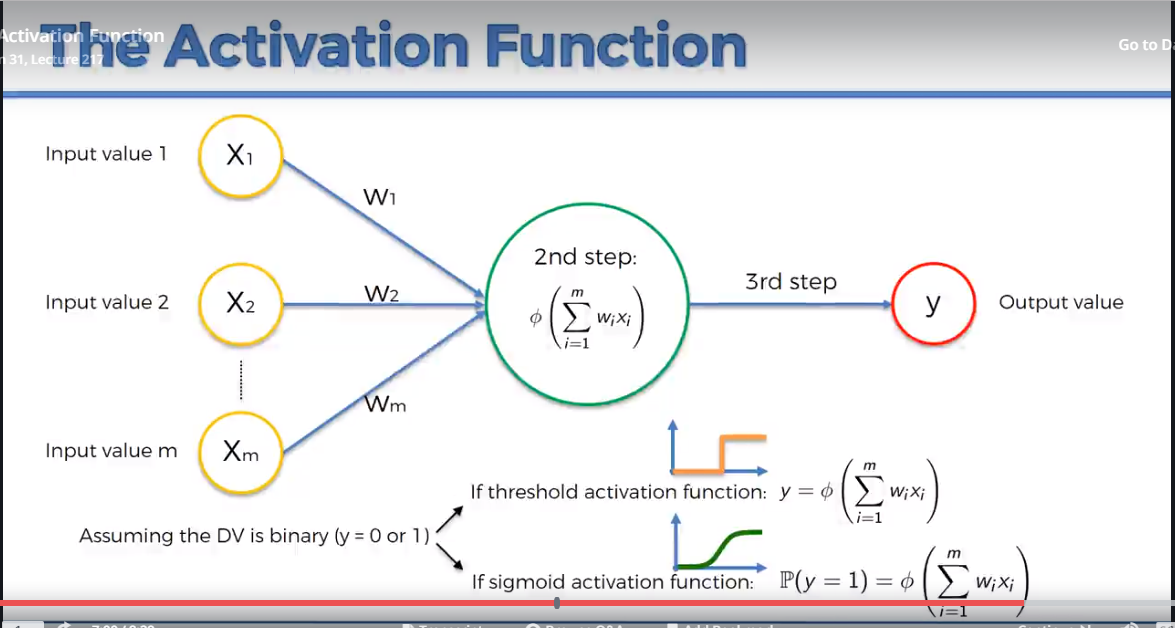


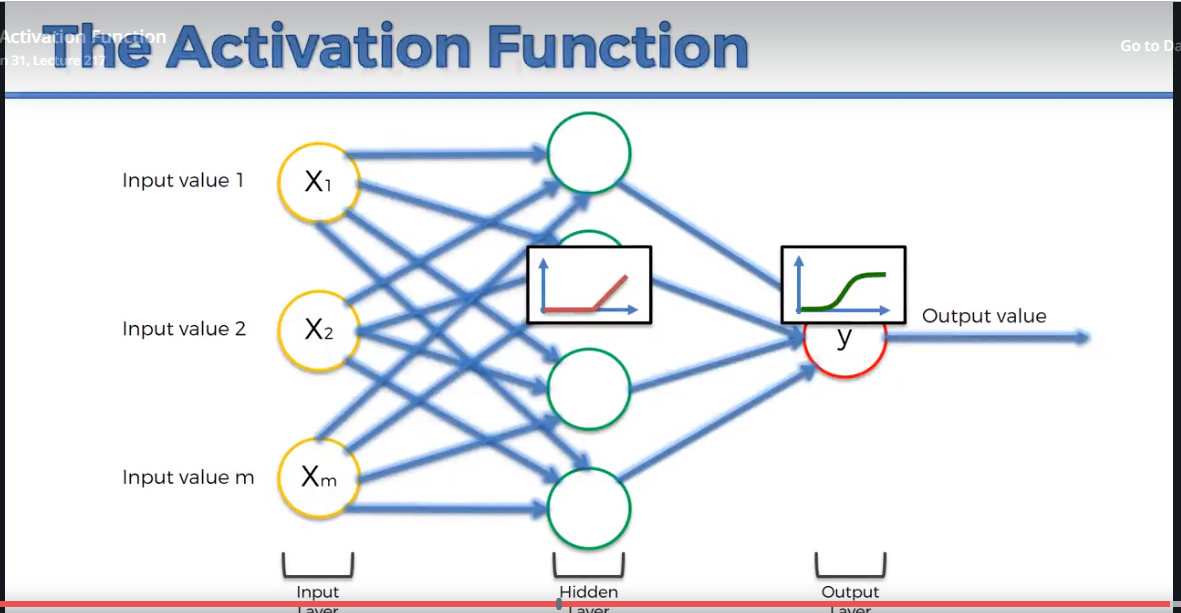
Omgamganapathayenamaha

Omgamganapathayenamaha

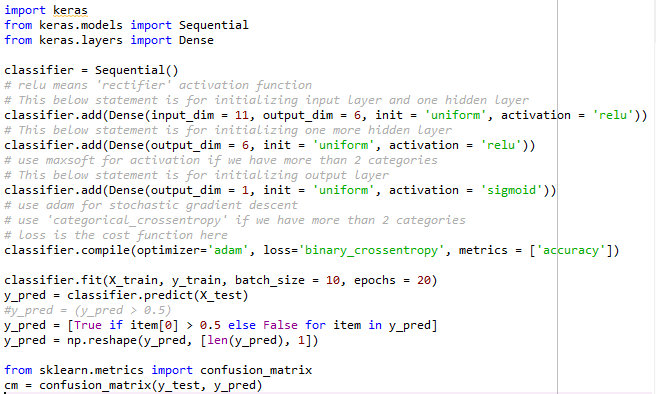
Omgamganapathayenamaha

Omgamganapathayenamaha

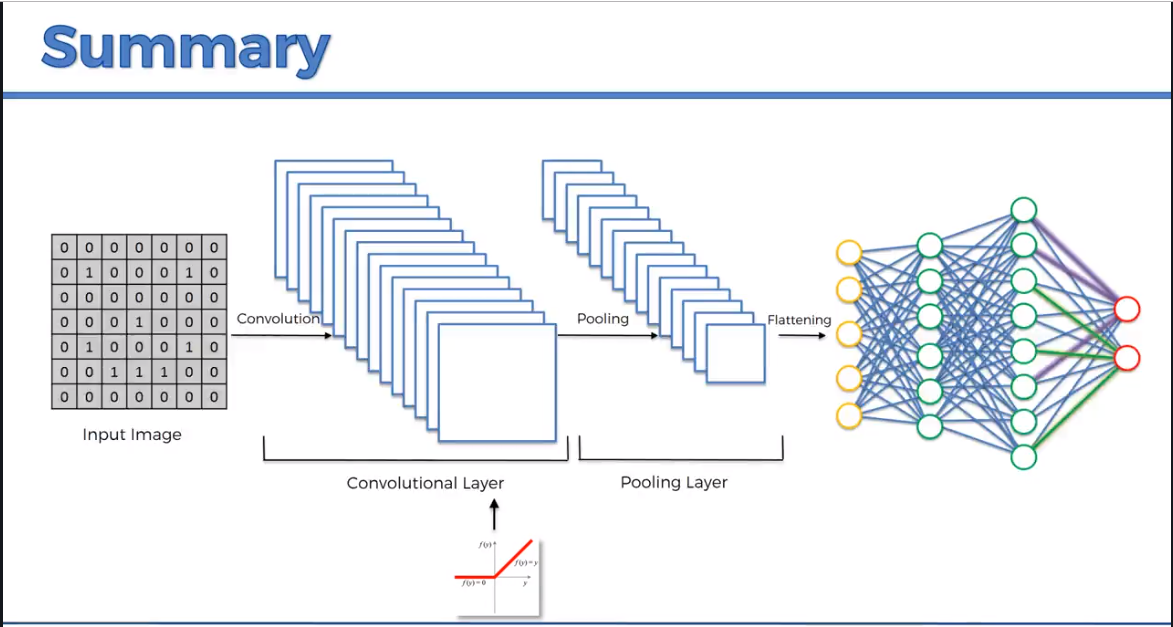
1. Deep Learning
   1. Artificial Neural Networks for Regression and Classification
   2. Convolution Neural Networks for Computer Vision
   3. Recurrent Neural Networks for Time Series Analysis
   4. Self Organizing Maps for Feature Extraction
   5. Deep Boltzmann Machines for Recommendation Systems
   6. Auto Encoders for Recommendation Systems
2. Neurons
   1. Dendrites
   2. Axon
   3. Synapse
3. Neural Network
   1. Neurons are connected to each other to form a Neural Network
   2. Each Neuron will have a
      1. Input Layer
      2. Hidden Layer (not a single layer but multiple Hidden layers might be present)
      3. Output Layer
4. Artificial Neural Networks
   1. Inputs – x1, x2, …, xm
   2. Weights – w1, w2, …, wm
   3. Activation Function
   4. Output(s)
5. Types of Activation Functions
   1. Threshold
      1. Theta = 0 if x <= 0 else Theta = 1
   2. Sigmoid
      1. Theta = 1 / 1 + e power -x
   3. Rectifier (Most Popular)
      1. Theta = max(x, 0)
   4. Hyperbolic Tangent
      1. Theta = 1 - e power -2x / 1 + e power -2x
6. We can apply Activation Functions like below:
   1. Apply Activation Function say Threshold (or) Sigmoid at Hidden Layer on Sum(wixi) and send the result to output layer
   2. Apply Rectifier at Hidden Layer and Sigmoid at Output Layer



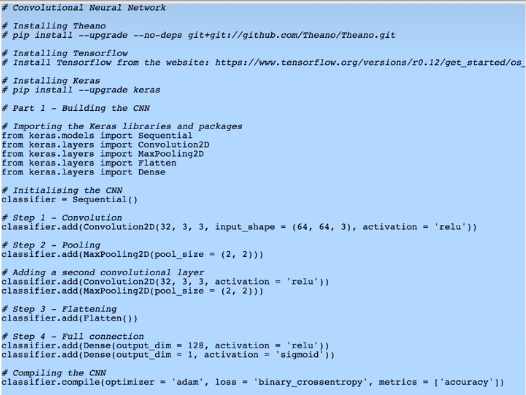
1. Weights are adjusted using ‘Back Propagation’ method
2. How Back Propagation works?
   1. Gradient Descent will start with some random weights
   2. Y-hats are calculated for all the input data
   3. Cost function is applied on all the y-hats i.e., C = ½ \* Sum(y-hat - yi) \*\* 2
   4. If C is more, Gradient Descent will check which way to move based on the slope, C and other factors and will re-calculate the weights
   5. Steps ‘b’ to ‘e’ are repeated until ‘C’ becomes very less
3. Gradient Descent is called ‘Batch Gradient Descent’ and has a problem on finding the ‘Local Minima’ instead of ‘Global Minima’
4. ‘Stochastic Gradient Descent’ works better and finds ‘global minima’ as below:
   1. Y-hats are calculated for each row of data
   2. ‘C’ is calculated and weights are updated after y-hat is calculated for every row of data
5. In between, there is a method called ‘Mini-batch gradient descent’ where it considers batches of input rows before updating the weights
6. One run through the input data is called an ‘epoch’. Run the algorithms through many epochs so that the weights get settled down
7. Theano Library
   1. Open Source Numerical Computational Library
   2. Runs on both CPU and GPU
   3. Developed my ML group of People at University of Montreal
8. TensorFlow
   1. Open Source Numerical Computational Library
   2. Runs on both CPU and GPU
   3. Developed my ML group at Google (Google Brain)
9. With Theano and TensorFlow, it takes lot of lines of code to build DeepLearning Models
10. Keras
    1. Wrapper around Theano and TensorFlow
    2. Amazing Library to build Deep Learning Models and Deep Neural Networks in less lines of code
    3. Installing keras
       1. conda install -c conda-forge keras (On the Anacaonda Prompt in Windows)
       2. conda install -c conda-forge keras (On the Terminal)
11. Libraries to use toward Artificial Neural Networks:
    1. import keras
    2. from keras.models import Sequential
    3. from keras.layers import Dense

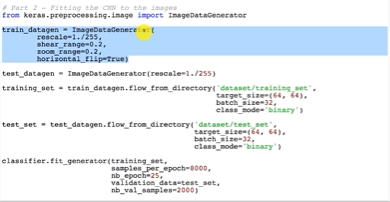


1. Artificial Neural Networks – Geoffrey Hinton
2. Convolutional Neural Networks - Yann Lecun
3. CNN deals with classifying/labeling images
4. Assume we have a 300x300 pixel B/W Image
   1. This image is converted to a 300x300 Matrix
   2. Each cell is a value between 0 to 255
5. Assume we have a 300x300 pixel Color Image
   1. This image is converted to 3 (300x300 Matrices) (Red Matrix, Blue Matrix, Green Matrix)
   2. Each cell is a value between 0 to 255
6. Identifying features from these Matrices is called Convolution
7. How to identify features?
   1. Use Matrices called ‘Feature Detectors’ or ‘Filters’ and select a stride
   2. Apply them on original image matrix
   3. We will get something called a ‘Feature Map’
8. On one single image, we will apply lot of ‘feature detectors’ and will get same number of ‘feature maps’ which we will store corresponding to that ‘image’
9. ReLU = Rectifier Linear Unit
10. Apply ‘Rectifier Activation Function’ on all the ‘Feature Maps’ generated
11. In ANN, generally, ‘Rectifier Activation Function’ is used on all the hidden layers and ‘Sigmoid Activation function’ on the output layer
12. Max Pooling
    1. On the feature maps, Consider 2x2 grids with a stride of 2
    2. In each 2x2 grid, consider the max value in that 2x2 grid
    3. Main reason for applying Pooling is “Spatial Invariance”
       1. Feature can be detected whether it is tilted or it is on the left corner, center or whether it is in some angle
    4. Doing Pooling this way also has few advantages:
       1. Features (max values) are retained … noise is eliminated
       2. Size is reduced to 75% of the original
       3. Eliminates over-fitting
13. Original Image -> Apply Feature Detectors -> Feature Maps -> Apply reLU -> Apply Max Pooling on each Feature Map -> Pooled Feature Maps -> Flattening
14. In Flattening, each ‘Pooled Feature Map’ is converted into a vector to pass it as an input to ANN
15. Point to note here is, while training, not only the weights but also the ‘feature detectors’ (filters) will also get changed



1. If we have more than one output in a Neural Network, generally, the outputs are independent and their values don’t sum up to 1. Softmax function when applied on these outputs will bring these output values (y values) in the range of 0 to 1 and also it will make sure the sum of all these y values is equal to 1
2. Assume the output values are y1, y2, y3. When softmax function is applied, these values will become eg., y1 will become (e power y1 / (e power y1 + e power y2 + e power y3))
3. Loss Functions / Error Functions
   1. Mean Squared Error
   2. Cross Entropy
4. Cross Entropy is only used in Classification and can better guide Gradient Descent than Mean Squared Error
5. Mean Squared Error is a good choice for Regression problems
6. Steps to code CNN
   1. Initialize a Sequential layer of Stacks
   2. Add a Convolution layer
   3. Add a Pooling layer
   4. Add a Flattening layer
   5. Do Image Pre-processing of both the training set and test set of images
   6. Add a Fully connected ANN
7. For good results, “add more than one convolution layers”
8. For good results, “your each hidden layer should contain around 128 neurons”
9. For good results, “input image size should be of 128 x 128 or 256 x 256”
10. We can initialize a Neural Network in 2 ways
    1. As a Sequence of Layers (using Sequential class from keras.models)
    2. As a Graph
11. Loss function ‘binary\_crossentropy’ is same as the “Logistic Regression” cost function as we need the Binary outcome ‘0’ for ‘1’ from CNN and also the activation function which is used in the output node is the sigmoid function
12. Feature Scaling is compulsory in Artificial Neural Networks
13. In CNN, Feature Scaling is done in the form of Image Augmentation
14. Image Augmentation eliminates overfitting. Overfitting is a situation where we get good results on training set and not so good results on test set
15. To get good results, we need lot of images in the training set
16. If we don’t have lot of images, then we can do image augmentation and it will apply transformations by rotating, flipping, shifting, shearing the images and generates the images/data





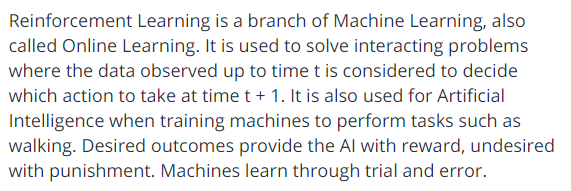
Omgamganapathayenamaha

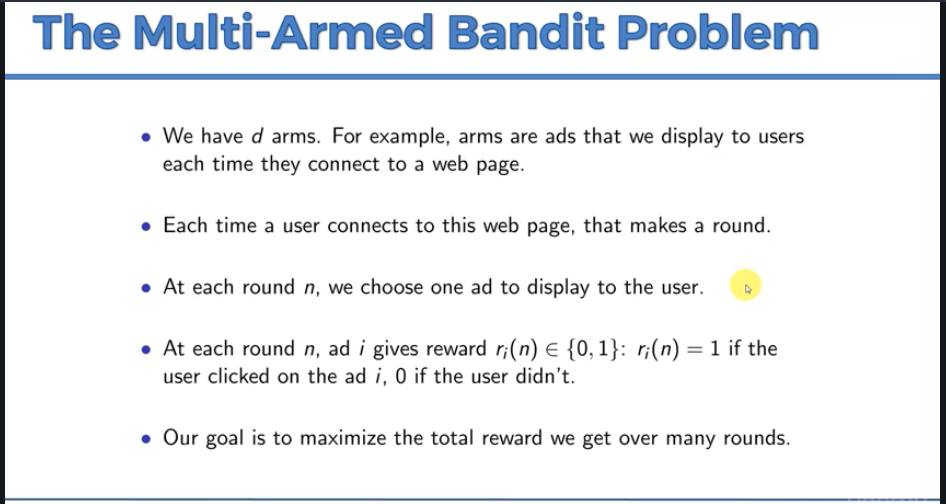
Omgamganapathayenamaha

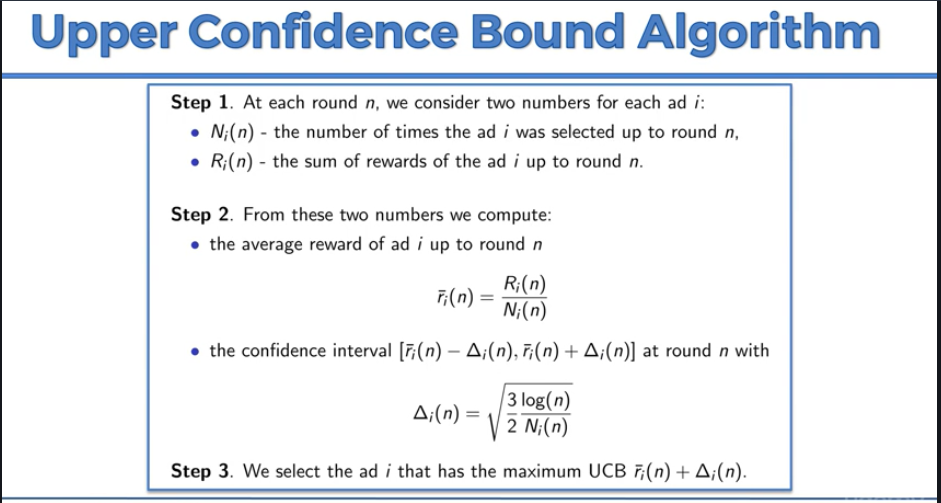
Omgamganapathayenamaha

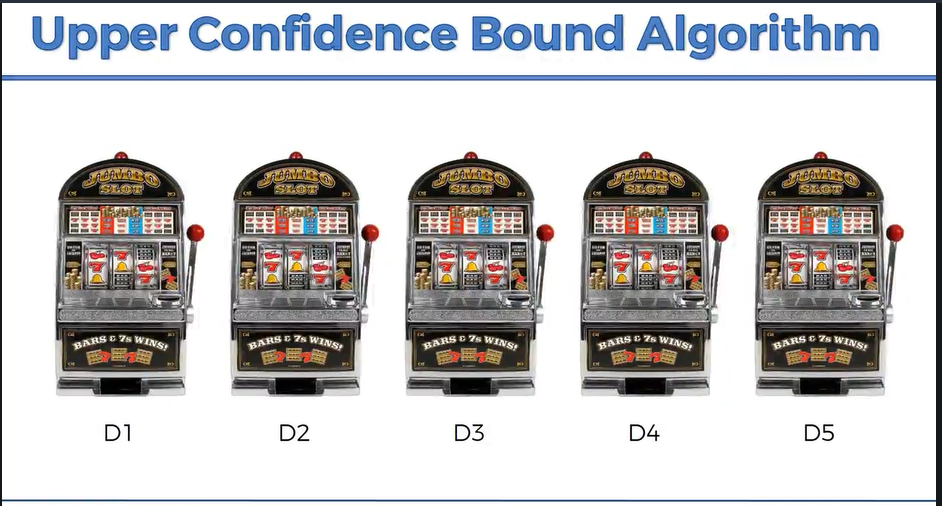
Omgamganapathayenamaha

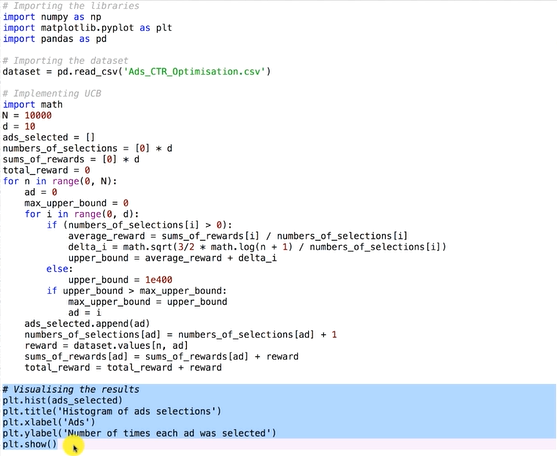
Reinforcement Learning











Omgamganapathayenamaha

Omgamganapathayenamaha

Omgamganapathayenamaha

Omgamganapathayenamaha

Dimensionality Reduction

1. Feature Selection - Backward Elimination, Forward Selection, Bidirectional Elimination, Score Comparison
2. Feature Extraction

Principal Component Analysis (PCA)

