02-Sep-2023 video notes

**Covariance**

It determines the strength of relationship between two variables

Its value goes from –infinity to +infinity

If value of one variable doesn’t change it means there is no covariance between two variables (like if one variable remains 5,5,5,5,5… and other may be 5,10, 15 ,45 etc)

If we change the scale (tens, hundreds, 1000s) of variable the strength of relationship doesn’t change but the value of covariance changes. So its bit confusing

If the value of any variable doesn’t change it means, there is no covariance

Positive value means good relationship Negative means bad relationship 0 means no relationship

**Correlation**

It’s almost same as covariance however data in this case is standardized and doesn’t affect by change in scale. Its value changes from -1 to +1

Machine Learning

What is machine learning?

Machine learning is a branch of [artificial intelligence (AI)](https://www.ibm.com/topics/artificial-intelligence) and computer science which focuses on the use of data and algorithms to imitate the way that humans learn, gradually improving its accuracy.

**Four types**

**Supervised** (when you have both features and targets available)

**Regression** problem (when the label data is of continuous nature age, height, sale, loss BMI, most of the time labeled data is float nature etc)

Simple regression (one input + one output)

Multivariable regression (multiple inputs + one output)

Classification problem (when the label data is of categorical nature good/bad, fail/pass

etc)

In both aforesaid subcategories of Supervised Learning features can be continuous as well as categorical. It can further be divided into binary classification (2 possible outcomes or multiclassification (more than 2 outcomes)

**Unsupervised** (only different features data is available but no labelled data)

Clustering (grouping features on the basis of similarity). Categorizing people and placing them in different clusters for decision making YouTube is an example, customer segmentation on the basis of their similar features in purchase (bank, Amazon)

Semi Supervised (Supervised + unsupervised) (Not used now a days because now we have strong algorithms available

**Reinforcement Learning**

In ML we deal with the situation or problem of predicting the needed output (outcome, target, class, labels, outputs) on the basis of certain inputs (features, Independent variables, predictor). ML is the process of teaching computer

**Linear Regression**

It Is the most fundamental algorithm mainly used for solving the regression problems.

Based on the concept of line with slope and intercept so its called linear

Note: Mostly they ask for lifecycle of data science

to be continued….

**Regression or Regression Analysis**

It is the process of predicting the values of dependent variables/labels (marks, height, weight) on the basis of input variables or features

Difference between correlations and regression

Essentially, you must know when to use correlation vs regression. Use **correlation** to summarize the strength and degree of the relationship between two or more numeric variables. Use **regression** when you’re looking to predict, optimize, or explain a number response between the variables (how x influences y).

Correlation quantifies the strength of the linear relationship between a pair of variables, whereas regression expresses the relationship in the form of an equation.

Regression is used for time series modelling and to find causal effect relationship between variables and forecasting (labels)

It is also used to see the extent of the impact of multiple independent variables on dependent variables

It can also measure these effects even when independent variables are on different scales (some features are in tens, some are in hundreds and other are in thousands)

Why we do scaling

Feature scaling is the process of normalizing the range of features in a dataset. Real-world datasets often contain features that are varying in degrees of magnitude, range, and units. Therefore, in order for machine learning models to interpret these features on the same scale, we need to perform feature scaling.

Types of scaling (Standardization, Normalization)

**Linear Regression (least square algorithm)**

Simple Linear Regression (only one feature (Discrete/continuous is used to predict the label/target (Continuous only)) using one feature only.

Multi-linear Regression (More than one features are used to predict the label (continuous only) (99% of the time we deal with multilinear regression problems)

How a linear regression predicts means behind the scene what happening

It is basically trying to find the best fit line (y=mx+C+e). Y is prediction, e is the error .For multilinear this equation will be extended further like y=C+m1x1+m2x2+m3x3…………..

Y is response or target variable

X is the feature

m is the coefficient (slope) (how y is changing w.r.t x)

C is the intercept (point where line cuts at y-axis)

Both m and C are model coefficients

Will sale of the car is impacted by not investing in radio advertisement or by investing in tv ads etc. these kinds of answers

To predict salary someone gives his phone number.. A nonsense because there is no relationship.

Import matplotlib.pyplot as plt

Import seaborn as sns

Import pandas as pd

Import numpy as np

Import random

%inline matplotlib #used to save the graphs

data=pd. read\_csv(‘file.csv’)

data.head()

data.shape() finding the total numbers of rows and columns

data.info() finding types of variables (int, float string), if something is string remove it

data.describe() finding the details of data (total count, mean, min, max std, Q1..Q3)

data.isna().sum() finding the total null values

#visualize the relationship between features and target using scatterplot

fig, axs=plt.subplots(1,3) #fig variable is not essential but used to save picture

data.plot(kind=’scatter’, x=’TV’, y=’sales’, ax=axs[0], figsize=(12,6))

#figsize parameter is not essential

data.plot(kind=’scatter’, x=’radio’, y=’sales’, ax=axs[1])

data.plot(kind=’scatter’, x=’newspaper’, y=’sales’, ax=axs[2])

fig.savefig(‘testdata.jpg’)

Here each data scientist is different in visualization if the data, which is okay. How much relationship is what we need to find (correlation coefficient), not good bad or average relationship.

If you need to find the relationship/non-relationship between any variable and label you need proofs, visualization is not enough. That’s what is done in actual projects.

Now visually it seems that tv and radio have strong correlation with sales than newspaper

But it is to be confirmed

Some general Questions are

What’s the relationship between ads and sales?

How prominent is that relationship?

Which ad types contribute to sales

How each ad contributes to sales?

Can sales be predicted on the basis of advertisement?

When we fit a line in Linear Regression it is known as best fit line, prediction and regression line

Best slope and intercept values for a line depends upon the distribution of data

A best fit line is a tradeoff between bias and variance ( a line that fit every data point of training dataset has highly bias). A line when test point are plotted shows lots of value for RSS is high variance line.

Y=mx+C is the prediction formula for linear regression

Residual (R) or error: the difference between true observed values and prediction lines (predicted values) R=y-(mx+c) (residual for one point only)

No one can predict 100% if there is 100% prediction there could be luck or data is corrupted or line is over-fitted

How do you know it is the best-fit line?

A best-fit line is one with minimum residual or min error. Residual is the distance between actual label and predicted label.

Saying on the basis of one difference about the performance of algorithm is easy but when there are thousands of differences we need to have a number to judge the conclusion that’s where we need sum of residual and then sum of the RSS (Sum of square if residuals). If you add all the residual without squaring them, it may give zero. So that is why squaring is required. So this way a model is judged based on results of RSS. It should be as minimum as possible.

**Gradient Descent**

Internally it tries to improve the accuracy of the model on the basis of results (error). An iterative process improve (reduce error) itself on the basis of available data and respective trend.

It is used to calculate the best values for m and C for which residual is minimum or near global minima (zero error).

Data fluctuates all the time for example for a student 2 years of experience, we are giving him 2 lacs but for other student with same years of experience we are giving 4lacs. Therefore, it is possible to predict with 100% accuracy or with no error. So model results in no further reduction of error and at that point gives the optimum value for slope and intercept of the model. learning done, ready for future prediction

Keep scikitlearn or official document of any library open when working on the ML projects

Following order remain same until last model of ML

# Create X=feature and Y=Label

X=data[[‘TV’]]

Y=data.sales

#follow the usual sklearn pattern import, instantiate, fit (model training)

From sklearn.linear\_model import LinearRegression

Lm=LinearRegression()

Lm.fit(X,Y)

Print(lm.intercept\_)

Print(lm.coef\_)

We can predict manually through equation or through model

To predict manually we need values of slope and intercept of the model, input(feature) can be any value you want to get prediction upon.

Desmos website to visualize m and C values

Y(predict)=0.04 x 50 + 7 (Manually)

#Make a dataframe

X\_new=pd.DataFrame({‘TV’,[[50]]})

Lm.predict([[50]]) (through model) #for one sample only

We know our model was best fit but how much best fit, so to answer it we need R2 value

**Model Confidence (R2 statistics – one of the Metrics for regression)**

The model after fit and predict functions predicts a certain value[[9]]. Now we want to know and evaluate and we want to get confidence that how good is our model in making predictions. R2 or R2 statistics gives the measure of fit of the model.

R2=TSS (variation of mean w.r.t data points)-RSS (variation of model line around the points)/TSS (variation of mean w.r.t data points)

R2=1- RSS/TSS

We compared the performance of model variation with mean variation. Because mean line will always give max variation so it’s a good reference point for calculating the predicting performance of model. Less variation as compared to mean line means good model

The most generic way to calculate the fit of linear model is by computing the R-Squared value. It explains the proportion of Variance i.e the proportion of variance in the observed data which model explains, or reduction in error over the null model( it predicts only mean of the observed responses, thus it only has intercept and no slope)

Its value lies between 0-1. The value closer to 1 is better as it shows more variance is explained by the model

Here the question we solve is that: does the model we used predict the label better than the mean( a null model with no slope) than question is how much better? Its always gives results between 0-1 because variation around the line will never be greater than variation around the mean.

R2=0.75 or 75% means our model is 75% accurate in predicting or in other words there is 75% less variation around the model than the mean. It also means that 75% of variation in data is explained by the selected feature. Or 25% error or 25% variation is not explained by the model, it could be explained by other features.

R2 is same as R(correlation) so if someone says R=0.9 for any two variables then it means their relationship explains 81% of variations.

R2 is better than R because it is easy to interpret

When R=0.5 R2=0.25 means 25% of original variation is explained

When R=0.7 R2=0.49 means 50% of original variation is explained

So, R2 is the percentage of variation explained by the relationship b/w two variables

When anyone gives you, R just square it for easy interpretation

from sklearn.metrics import r2\_score

# pass all Tv Records and predict sales

Predicted\_sales=lm.predict(X) #for all samples

X.head()

Predicted\_sales

r2\_score (y\_true=y, y\_pred=Predicted\_sales) # mentioning y\_true is not mandatory

Adjusted R2 statistics (use mostly for multi variable problems)

Because as we increase the input features the value of R2 will automatically increases. It doesn’t essentially mean that the new added features are strongly correlated with label. It s just because of mathematics. Even our model is not improving with the addition of new features but its value is increasing. So this could be misleading. To rectify this issue we use adjusted R2. It penalize the excessive use of those features that do not correlate

When predictor in the equation of R2 adjusted become zero R2 adjusted becomes R2

Finally any R2 value appears good is really good or not depends upon domain

R2=1- (1-R2)(N-1)/N-P-1

N=number of samples

P=features or predictors

Using R2 for multiple linear regression is not advised but people do use

It’s a tool for comparing the performance of different models

Multiple linear Regression

Bo+B1x1+B2x2+B3x3……

B1 , B2 and B3 are different because each feature is impacting the label differently

X= data[[‘TV’,’radio’,’Newspaper’]]

Y=data.sales

lm=LinearRegression()

lm.fit(X,Y)

print(‘intercept’, lm.intercept\_)

print(‘coef TV’, lm.coef\_[0])

print(‘coef radio’, lm.coef\_[1])

print(‘coef newspaper’, lm.coef\_[2])

As discussed previously in scatter graphs that newspaper is not correlated (1st proof) but we didin’t make the final decision now coef\_ for newspaper= -0.00145 (2nd proof) it means it is affecting the sales in negative manner or your sale will not make any difference by giving ads to newspapers. Now to get the confidence we will check adjusted r2-score including Newspaper and without including Newspaper, if there is no difference in adjusted r2-score value it means TV and Radio are best features for predicting sales correctly (3rd proof).

X= data[[‘TV’,’radio’,’Newspaper’]]

Y=data.sales

lm.fit(X,Y)

predicted\_sales=lm.predict(X)

r2\_score( y, predicted\_sales)

X= data[[‘TV’,’radio’,]]

Y=data.sales

lm.fit(X,Y)

predicted\_sales=lm.predict(X)

r2\_score( y, predicted\_sales)

This until now is very simple used case, in practice there are lots of other things to be included

R2 for multilinear regression is not a good approach better to use adjusted R2 (penalizes model complexity to control overfitting).But it again under penalizes the complexity that why better approach for feature selection is cross-validation. Its more reliable to choose which of the models will best generalizes as it is better estimate of out-of-sample error

In cross-validation the algorithm the proportion of data for training and testing is shuffled and after all combination cross validation gives the result for an algorithm with highest result. Then cross validation of other algorithms are performed and at the end of cross validation of different algorithms. Algorithms with best cross validation results is selected for further implementation.

IN ML estimating parameter is called training the algorithm

Evaluating the method is called testing the algorithm

**09-September 2023 video**

Project-1 (Linear Regression)

**Prediction of admission chance on the basis of students various scores**

**import pandas as pd**

**import numpy as np**

**import matplotlib.pyplot as plt**

**from sklearn.linear\_model import LinearRegression**

**from sklearn.preprocessing import StandardScaler**

**from sklearn.model\_selection import train\_test\_split**

**import pickle # to save and the final model**

**import seaborn as sns**

**import warnings**

**warnings.filterwarnings(‘ignore’)**

**data=pd.read\_csv(‘Admission\_prediction.csv’)**

**data.head( ) # Gives first five rows of the dataset**

**data.shape # gives total number of rows and colmns**

**#serial number in a dataset is useless and a nominal data (no order importance) just name, drop it during cleaning**

**DATA Preprocessing steps**

1. **Data Cleaning**
2. **EDA Analysis (Exploratory data Analysis) how data is trending**
3. **Finding and treating Outliers( if Any) whether data is skewed right or left**
4. **Feature selection for model building (dropping unwanted columns)**
5. **Data standardization**
6. **Train Test split**

**data=data.drop(columns=[‘Serial No.’]) #To remove unnecessary column**

**data.head( )**

**data.describe( ) #describe statistics of dataset or health check of dataset**

**data.info( )**

**data.isnull( ).sum( ) # give null values in each column**

**data.isnull( ).sum( ).sum( ) # give null values in whole DataFrame**

**# Based on the available data, perform the documentation and note down important findings or Observations. Keep in mind it is just observation not conclusion we will come back after having more proofs to take final decisions. For which we will plot graphs to visualize and If see any problems in graph we will come back and fix it.**

**Observe like how many Null values are present in each feature columns and how it would be dropped or filled. Second observation is mean and standard deviation simultaneously, it is done only on continuous (unlimited and float) data columns. Check whether there is a reasonable relation between mean and deviation. For example if you have mean cricket score is 360 and standard deviation is+-80, it doesn’t make much sense because this is too big window in comparison of standard deviation +-10. The lesser the standard deviation as compared to mean, the more the quality of data (confirm using displot function that whether it forms bell shape curve or not because we always look for normally distributed data). Also observer difference between min, 25%, 50%, 75% and max, again it is performed only on continuous data. it also should be reasonably small not too big gaps, if too big means some issues, note down this observation also, it could be due to outliers. For this purpose one important step is to apply mean( ) function to continuous columns and mode function for categorical columns data to fill the Null values. Because if we apply mean function to the categorical data it will return float value but categorical values are mostly discrete. The entire analysis needs to be done on features we never touch labels. After this removal of null values apply describe ( ) function to perform the step of observation again because may be values of some columns might be changed after removal of Null values. Even after cleaning you may find outliers then drop them also before scaling and training any model.**

**Note:**

**How to differentiate between continuous and discrete data.**

**Discrete data is dotted line whereas continuous data is line with infinite uncountable number of points. Where discrete values can be counted. For example time can be divided into many infinitely small units so it can’t be counted whereas apples and stars both can be counted**

**# Filling the null values**

**data[‘University Rating’]=data[‘University Rating’].fillna(data[‘University Rating’].mode( )[0])**

**data[‘TOEFL Score’]= data[‘TOEFL Score’].fillna(data[‘TOEFL Score’].mean( ))**

**data[‘GRE Score’]=data[‘GRE Score’].fillna(data[‘GRE Score’].mean( ))**

**# Let’s see how the data is distributed for every column**

**plt.figure(figsize=(20,15), facecolor=’red’)**

**plotnumber=1**

**for column in data:**

**if plotnumber<=8:**

**ax=plt.subplot(2,4,plotnumber)**

**sns.distplot(data[column])**

**plt.xlabel(column, fontsize=20)**

**plotnumber+=1**

**plt.tight\_layout( ) or plt.show( )**

**# Linear Regression assumes that continuous data (features only) is normally distributed**

**# Correlated with label or not**

**# check outliers using boxpltot if there is some skewness in continuous column data**

**Divide dataset into features and Labels**

**Y=data[‘Chance of Admit’]**

**X=data.drop(columns= [‘Chance of Admit’])**

**Relationship analysis for feature selection (Visualize Relationship)**

**plt.figure(figsize=(15,10), facecolor=’yellow’)**

**plotnumber=1**

**for column in data:**

**if plotnumber<=8:**

**ax=plt.subplot(2,4,plotnumber)**

**plt.scatter(X[column],Y)**

**plt.xlabel(column, fontsize=20)**

**plt.ylabel(‘Chance of Admit’, fontsize=10)**

**plotnumber+=1**

**plt.tight\_layout( )**

**#Even if you think by observing that there is 1% relationship between any feature and label consider it don’t drop. It’s not essential that we always see a long straight-line trend. An upward trend is also a good observation for having a relationship. Here two different data scientists may differ in opinion**

**Data scaling Z= (X-mean)/standard deviation #it is Z-statistics**

**Dividing by std ensures large numbers divided by correspondingly large std deviation ad small numbers are divided by correspondingly smaller deviation hence keeping or scaling in equitable ranges. Another problem is that computer may take long time in calculating features with large values (not must). Also we want to make data unit less or unit independent.No information is lost just we are reducing propotion of magnitude of each feature equally**

**For help $ help(LinearRegression)**

**Scaler=StandardScaler( )**

**X\_scaled=Scaler.fit\_transform(X)**

**Train test split**

**Training is giving almost 75% questions (features) and answers (label)**

**Test is only testing by giving 25% questions (features) and hiding remaining 25% actual responses( actual labels) and test how algorithm predicts response (labels).**

**Sequence is always**

|  |  |
| --- | --- |
| **1(train) X-train** | **3(train) Y-train** |
| **2(test) X-test** | **4(test) Y-test** |

**# split data into train and test parts. Model will be built on training data and tested on test data.**

**x\_train,x\_test,y\_train,y\_test=train\_test\_split(X\_scaled, Y, test\_size=0.25, random\_state=348)**

**#Selecting test\_size value is variable mostly vary b/w 20-30%, on large dataset it doesn’t matter, by default it is 25% also random\_state can be any number it’s the concept of seed(any number) to generate psudorandom numbers**

**y\_train.head()**

**Model instantiating and training**

**regression=LinearRegression( )**

**regression.fit(x\_train, y\_train)**

**data.tail(2) #give data of last two student kept in 25% testing portion**

**Predict the chance of admission by checking the model for only one new person coming to know his chance of admission and giving the different scores he has secured. As we already have used the object(named scaler) for StandardScaler we can use it here as well for transformation, without ftting of new data for prediction.**

**Print(‘Chance of Admisison: ‘ regression.predict(scaler.transform([[314, 103, 2 ,2 ,3, 2.81, 0]])**

**#You can save the model to local file system and use it in future for prediction**

**pickle.dumb(regression, open(‘reg\_model’, ‘wb’) #it makes a kind of exe file for user means he would only run the model but couldn’t see anything**

**#loading the model**

**loaded\_model= pickle.load(open(‘reg\_model’, ‘rb’)) #this is written by user to run the model**

**a=loaded\_model.predict(scaler.tranform([[314, 103, 2 ,2 ,3, 2.81, 0]]) #scaler.tranform is better but not essential**

**10 September 2023 video**

**Adjusted R-square to check how well the model learnt on data**

**#Let’s check how well model fits on training data**

**regression.score(x\_train,y\_train) #means this part of data is given to model to check how well the model understood the trend,less percentage means there could be variation in data or other features are still required to increase the percentage of**

**r squared value**

**#Let’s check how well model fits on test data**

**Regression.score(x\_test,y\_test)**

**Test side data Rsquare value is less because it has less data to be learnt on.**

**Let’s plot and visualize**

**x\_test[1]**

**y\_pred=regression.predict(x\_test)**

**y\_pred**

**plt.scatter(y\_test,y\_y\_pred) #y\_test is actual chance of admisison**

**plt.xlabel(“Actual Chance of Admission)**

**plt.ylabel(“predicted chance of admission”)**

**plt.title(“Actual vs model predicition”)**

**plt.show( )**

**#when you get a plot there must be some outliers because our model can’t be 100% accurate**

**#Now after this stage we need to find how much error is generated by the model**

**#When we talk about r score we talk about highest score and accuracy but when we talk about lower score it means error (overall error in x-test). Client is more focused on the error. Also r2 shows how well is our model whereas error shows how bad is our model:**

**In all regression models we have three new metrics to check error:**

**MAE (Mean Average Error): Represent Average Error means it considers all the points for calculating the error(related as well as Outliers so overall average will affect).**

**Like avg of 2+4+3/3=3 but for same number with one outlier (also called Noise) 99+3+3/3=35**

**So as initial case points have no big outlier it gives MAE of 3 whereas later give 35 which is confusing to evaluate algorithm performance by MAE**

**MAE=(Predicted value – Actual Value)/ average number of samples**

**MSE (mean squared Error)**

**As MAE considers all the points and cause confusion so MSE penalizes by neglecting the outliers or noises to get a clear picture of error. So giving MAE as well as MSE to client helps him to evaluate the noise in the data. It’s hard to interpret than MAE because it is squared value and not in base units**

**RMSE(root mean squared error): Most popular metric to evaluate model error because its penalizes the noise and bring MSE in base unit for easy interpretation**

**#Finally when we built a model we need to present R2-score adjusted R2-score**

**MAE, MSE, RMSE metrics for all regression models. For classification there are other performance metrics.**

**# How to calculate MAE,MSE,RMSE**

**from sklearn.metrics import mean\_squared\_error, mean\_absolute\_error**

**y\_pred=regression.predict(x\_test)**

**mean\_absolute\_error(y\_test,y\_pred)**

**mean\_squared\_error(y\_test,y\_pred)**

**root\_mean\_squared\_error=np.sqrt(mean\_squared\_error(y\_test,y\_pred))**

**Now after seeing all the metrics it is up to client whether he may use this model or not. It is possible that model may fail it is because of data integrity problem not your or client fault. In real projects it is very very difficult to achieve 75% accuracy where data doesn’t vary a lot. Mostly accuracy lie b/w 50-60%. So if a model achieve 90% accuracy client will doubt and to avoid this you need to check overfitting factor of your model.**

**MODEL SUCCESSFULLY BUILT**

**Now let’s check whether our model overfiiting our data using Regualarization**

**Overfitting**

**Lets say we have 100 questions and we break 75 for training and 25 for testing our model. Let’s say our test set have 20 questions similar to what appearing in training set and 5 are something new. Obviously model can easily get 20/25\*100=80% accuracy. This is overfitting, means the training data is also reflected in test data. On the other hand if you give new set of data that is not appearing in training set. Your model may achieve 60,65%.**

**TO avoid overfitting we use Regularization techiques:**

**LASSO (a.k.a L1 Form)( Least Absolute Shrinkage and Selection operator)**

**LASSO internally checks the correlation of all features with label and completely disregard all features with less correlation. It selects relevant features that’s why its is a selector operator.**

**Ridge (L2-form)**

**Ridge doesn’t disregard irrelevant features completely rather it minimizes the importance (maybe 0.0001) of said features.**

**Elasticnet (mix of two) not very famous**

**Why use regularization**

**It helps in reducing the variance of model without the increasing the bias substantially. Bias is when model is fitted tight to every training point. But have high RSS value w.r.t to test points.**

**In gradient descent we use iteration on the basis of data to reduce error in every iteration. But we didn’t know how fast or how slow the model was learning. For example learning too fast is good but one may omit certain information or learning very slowly also reduces the processing efficiency. So we have to regularize the learning rate or learning speed for getting optimum learning rate.**

**For example our model is learning at 10step/time learning rate. So when it reaches global minima it is possible that the model should take 2step at a time to reach near global minima. But at learning rate of 10 steps it goes on the other side, so it can never come near to global minima or to reducing error. So in linear regression we can’t tell the model how slow or fast model should learn. But in LASSO and Ridge we can tell how slow you read to reach global minima.**

**from sklearn.linear\_model import Ridge, Lasso, RidgeCV, LassoCV**

**#LassoCV returns best alphas(learning rate) after max iteration**

**#CV means cross validation**

**In short CV means, Checking the performance between various test data sets. Means ist time train and test on some data and check the performance. Then again train bust take test with different question data**

**#Normalize means min-max scaling, each feature is transformed to a range between 0 and 1 . This is done by subtracting minimum value of feature from each feature and dividing by range (max-min)**

**lasscv=LassoCV(alphas=None, max\_iter=10) #max\_iter means we are asking lassocv to read our data ten times.learning rate will always between zero to one. So what lassocv does that it start learning at different different learning rates say 0.0001, 0.0002……1.0. And the rate at which gets the best R2 score it gives that point as best learning rate to be used further in lasso**

**lasscv.fit(x\_train, y\_train)**

**alpha=lasscv.alpha\_**

**lasso\_reg=Lasso(alpha)**

**lasso\_reg.fit(x\_train, y\_train)**

**lasso\_reg.score(x\_test,y\_test)**

**#when you run the model you find that linear regression model and this model is giving almost same result which shows that features taken are all relevant and affecting labels. Here point to be noted is that we are not using lasso for proving the overfitting factor of Linear Regression model, it is an entirely different model that can be separately use inspite of LinearRegression. As Lasso and Ridge has advantage of having optimum learning rate that can avoid the overfitting phenomenon, it takes lot of time to process or learning data as compared to linear regression . Because for every learning rate it takes different different time. NLP models may take weeks or months to learn so cost is associated because you can’t do it on local platform so you use AWS google cloud.**

**Implementation of RidgeCV**

**ridgecv=RidgeCV(alphas=np.arange(0.001,0.1,0.01)**

**#we are giving alpha values because we want the model to consume less time in training. Same could be done in LassoCV as well**

**ridgecv.fit (x\_train, y\_train)**

**alpha1=ridgecv.alpha\_**

**ridge\_reg=Ridge(alpha1)**

**ridge\_reg.fit(x\_train,y\_train)**

**ridge\_reg.score(x\_test,y\_test)**

**16 Sep 2023 videos**

Logistic Regression

It is a supervised learning algorithm used only for classification purpose. Particularly for binary classification only. For multiclass we have other algorithms

In logistic regression we also used linear line formula y=mx+C , as probabilities are continuous numbers like 0.3, 0.5, 0.6, but the values come as a outcome are transformed by sigmoid function into 0 or 1 as per the threshold value of the said function.Numbers above threshold are marked as 1 and below threshold are marked as 0. Resulting in binary classification of the class 0 means No 1 means Yes. Simply

Note: it could be used for regression only in the condition when the data points are on the straight line. So we can’t use linear regression for classification because it produces continuous number but for classification we need categorical data.

Why we use sigmoid function

It’s a simple method to introduce non-linearity in the model

It’s range is bounded between 0 and 1. So it is useful to be utilized for logistic regression

It’s derivative is easily to calculate than other functions that is helpful in gradient descent calculation

**Classification**

In classification we deal with qualitative or categorical data

There are two types of classification binary (only two classes or only two labels, suffering from cancer YES or NO on the basis of given features) classification and multiclass classification (more than 2 outcomes or classes types of flowers to be chosen from three different flowers on the basis of its various features).

**Evaluation if the classification model** After result of classification problem we measure how accurate our model is, we do it by various metrics. In regression the difference between the actual and predicted values is accuracy. But the metrics we use in linear regression can’t be used in the evaluation of classification models.

In interviews they focus most of the time on the metrics of classification models.

In classification problems all the metrics are derived from confusion matrix

|  |  |
| --- | --- |
| TP | FP |
| FN | TN |

**Actual**

**Positive (1) Negative (0)**

**positive**

**predicted**

**Negative**

**TP: A value that was predicted as positive and is positive in actual**

**FP: A value predicted as positive but is negative in actual**

**FN: A value predicted as negative but in actual is positive**

**TN: A value predicted as negative and is Negative in actual**

**Typ1 Error: FP are called type-1 error**

**Type2 error: FN are called type-2 error**

**In covid cases type-2 error is more dangerous because more people that are affected by covid are walking around with assumption that they aren’t affected and causing harm to others**

**In business type-1 error is more dangerous because expert prediction about something to be true though it’s not in actual may make people to invest in the businesses and leading to loss. In business type-2 error may prevent you from making profit but doesn’t lead you to lose. Means it is predicted that a business is not worth investing though it was and you didn’t invest.**

**In conclusion which error to be reduced more, depends upon the condition we facing.**

**Accuracy TP+TN/TP+FP+FN+TN**

**It’s used to see overall performance of model. It measures how many predictions were correct w.r.t all predictions. It’s not a very good indicator of model if you have imbalanced data. Like if you have more data regarding on feature. For that purpose, we need other measures or metrics to evaluate model performance.**

**Sensitivity or Recall TP/TP+FN**

**It measures how many positive result were predicted correctly by model of all actual positive results. It is a good measure in the cases when we are trying to reduce false Negatives. Examples in case of health we want to correctly increase the TP by reducing the FN**

**Precision**

**It measures out of all positive predictions how many were actually positive**

**TP/TP+FP**

**In business we want to increase the TP while reducing the FP.**

**Furthermore, Example of spam email programming is good to understand that where recall or sensitivity is important and where precision is important**

**There are cases where not the money involved not health involved, then accuracy could be a good choice. like whether I am going to get my medical seat or not. It is the harmonic mean of precision and recall**

**F1-Score**

**When both money and health are involved and we need to reduce both FNs and FPs**

**2\*(recall\*precision)/recall+precision**

**Specificity or true negative rate:**

**It represents that how specific the model is while predicting true negatives, Mathematically:**

**Specificity=TN/FP+TN**

**It can quantify that how many negatives were predicted by the model as compared to total number of negatives or non-favorable outcomes.** **This metric is of interest if you are concerned about the accuracy of your negative rate and there is a high cost to a positive outcome — so you don't want to blow this whistle if you don't have to. Example how much covid negatives in actual are predicted as Negatives**

**False positive rate**

**How many were negative were predicted as positives (1-Specificity)**

**ROC (Receiver operator characteristics)**

**It helps in setting right threshold value for sigmoid function**

**Also, It helps in selecting best model for classification among various classification model for any particular dataset. The models’ ROC is built and plotted (It plots true negative vs true positive and the model having highest AUC (Area Under the curve)in ROC AUC Curve is finally selected for classification.**

**ROC AUC are only meant for classification models evaluation**

**Implementation of classification model (diabetics\_patient.csv)**

**import pandas as pd**

**import numpy as np**

**import matplotlib.pyplot as plt**

**from sklearn.linear\_model import LogisticRegression**

**from sklearn.preprocessing import StandardScaler**

**from sklearn.model\_selection import train\_test\_split**

**from statsmodels.stats.outliers\_influence import variance\_inflation\_factor**

**import pickle # to save and the final model**

**from sklearn.metrics import accuracy\_score, confusion\_matrix, roc\_curve, roc\_auc\_score,**

**import seaborn as sns**

**import warnings**

**warnings.filterwarnings(‘ignore’)**

**data=pd.read\_csv(‘diabetes.csv’)**

**data.head( )**

**data.shape**

**data.describe( )**

**data.info( )**

**data.isnull( ).sum( )**

**Think relastically about the data**

**#if in any feature data if std is high and mean is less it doesn’t make sense at all, BP zero, skin thickness zero not possible. Google for realistic min max values of features like blood pressure, sugar etc if you din’t have domain knowledge**

**#all steps switch back and forth**

**# replacing zero values by the mean of the column, if you have huge amount of data like in millions you can delete zeros rather than replacing them with mean**

**data[‘BMI’]= data[‘BMI’].replace(0, data[‘BMI’].mean( ))**

**data[‘BloodPressure’]= data[‘BloodPressure’].replace(0, data[‘BloodPressure’].mean( ))**

**data[‘Glucose’]= data[‘Glucose’].replace(0, data[‘Glucose’].mean( ))**

**data[‘Insulin’]= data[‘Insulin’].replace(0, data[‘Insulin’].mean( ))**

**data[‘SkinThickness’]= data[‘SkinThickness’].replace(0, data[‘SkinThickness’].mean( ))**

**data.describe( )**

**In the diabetic dataset std deviation is very high as compared to mean, so it must be treated. So we need to identify the skewness of the data**

**# let’s see how data is distributed for every column, as almost all except pregnancies have continuous data.**

**plt.figure(figsize=(20,25), facecolor=’yellow’)**

**plotnumber=1**

**for column in data:**

**if plotnumber<=9:**

**ax=plt.subplot(3,3,plotnumber)**

**sns.distplot(data[column])**

**plt.xlabel(column, fontsize=20)**

**plotnumber+=1**

**plt.tight\_layout( ) or plt.show( )**

**#Also while visualizing data you need to look for reality (whether something is possible or not in real world). So pregnancies showing 20 which is extremely rarenowadays. Glucose has no skewness. Maybe some skewness in BP graph. There is a definite right skewedness in skinthickness. In insulin and BMI as well. In diabetes pedigree and age also. So it means threre exist outliers that could disturb the learning pattern of algorithm. We never check outcome/label at all.**

**#Now we will keep only features**

**df\_features=data.drop(‘Outcome’, axis=1)**

**# Visualize the outliers using boxplot**

**plt.figure(figsize=(20,25))**

**graph=1**

**for column in df\_features:**

**if graph<=9:**

**plt.subplot(3,3,graph)**

**ax=sns.boxplot(data=df\_features[column]) #in boxplot no x and y only dataset**

**plt.xlabel(column, fontsize=15)**

**graph+=1**

**plt.tight\_layout( )**

**# If left skewed outliers could be seen below the line(min level). If outliers above (max level) the line seen it means right skewness exists.**

**# So what should be done with outliers, should they be kept or delete. It is not always the case that these outliers should be deleted. It depends upon the nature of problem you are solving. So for example we need to think that BP range given like 600, 800 or a range from 600 to 800, is it realistic, if it is not then delete it. Now where to keep outliers, say for example bank balance of poor to upper middle class could be from 1000 to 10 lacs, so if the accounts of super wealthy people also exist in data it will appear as a outlier, so we can’t delete it because the super wealthy cases are real. So in the upcoming classes we will learn how to keep these outliers while not losing the data, in short the entire data will be convert in different form.**

**Outlier detection formula**

**# Ist quartile**

**q1=data.quantile(0.25)**

**q3=data.quantile(0.75)**

**iqr=q3 – q1**

**higher\_side\_outliers=q3 + (1.5 \* iqr )**

**lower\_side\_outliers=q1 - (1.5 \* iqr )**

**q1**

**#Note down the outliers, then work on removing those outliers, usually we don’t have outliers in descrete data (pregnancies), but considering the reality (20 pregnancies not possible) we want to take out the outlier from this as well.**

**# Validating an outlier**

**Preg\_high= (q3. pregnencies + (1.5 \* iqr. pregnancies))**

**Preg\_high**

**# check index which have higher values**

**np\_index=np.where(data[‘pregnencies’]>preg\_high) #it gives the index of columns having value greater than preg\_high**

**np\_index**

**# Drop the index which we found in the above cell**

**data=data.drop(data.index[np\_index])**

**data. shape**

**data. reset\_index( ) #to set the values index in series again, whenever you delete samples for any reason apply this function to set the table properly**

**data.shape**

**#Now same steps for Blood pressure column**

**bp\_high= (q3. BloodPressure + (1.5 \* iqr. BloodPressure))**

**print(bp\_high)**

**index=np.where(data[‘BloodPressure’]>bp\_high)**

**data=data.drop(data.index[index])**

**print(data.shape)**

**data. reset\_index( )**

**# Apply same code on all features to remove the outlier just change the feature name. Apply Q1 formula alongwith Q3 also on features having outliers on the lower sides as well. Very careful to note some people while copy paste make the mistake.**

**Sometime clients give the range that how much data could easily be deleted in case of cleaning. Otherwise there are millions of rows so some deletion may not causes problems.**

**bp\_low=(q1.BloodPressure – (1.5 \* iqr.BloodPressure)**

**index=np.where(data[‘BloodPressure’]<bp\_low)**

**data=data.drop(data.index[index])**

**print(data.shape)**

**print(data.reset\_index( ))**

**#Now after above step recheck the data distribution to see whether the skewness has been removed or reduced, so perfect bell shape curves. Except in pregnancy because it is startingfrom zero**

**#Don’t try to remove outliers multiple times, only one time is enough. You find outliers after first removal because there is data variance every time as you remove outlier**

**plt.figure(figsize=(20,25), facecolor=’yellow’)**

**plotnumber=1**

**for column in data:**

**if plotnumber<=9:**

**ax=plt.subplot(3,3,plotnumber)**

**sns.distplot(data[column])**

**plt.xlabel(column, fontsize=20)**

**potnumber+=1**

**plt.tight\_layout( ) or plt.show( )**

**#Now finding relationship and keeping the features having relationship with label.**

**#seprate features and label**

**X=data.drop(columns=[‘Outcome’])**

**Y=data[‘Outcome’]**

**# Let’s see how features are related to class, we use stripplot because our data is categorical**

**plt.figure(figsize=(15,20))**

**plotnumber=1**

**for column in data:**

**if plotnumber<=9:**

**ax=plt.subplot ( 3,3,plotnumber)**

**# In stripplot we have x and y axes. On x-axis we have label and on y axis we have features**

**sns.stripplot(x=Y, y=X[column], hue=y)**

**plotnumber+=1**

**plt.show( )**

**#The graph shows outcomes (healthy (0) or diabetic (1)),for details see video from (1:44) time, It seems all features are related, if you think some feature is not relevant drop it, but if some feature has even 1% relationship with label it means it is impacting**

**# Find multicollinearity problem to see whether one feature is dependent upon other feature or not**

**scalar=StandardScalar( )**

**X\_scaled=scalar.fit\_transform(X)**

**X\_scaled.shape[1] #shape shows (rows,columns) so shape[1] shows column only**

**Now find multicillinearity using Variance inflation factor (VIF).**

**# Now consider we have age, experience and salary column. Salary can be predicted using age columns as well as experience column. And also experience increases with age so that means age and experience are collinear( have relationship). Therefore, if I can predict the salary through any of the aforementioned column separately then what’s the point in keeping both age and experience columns we can drop one of them (think about millions of record, if two feature are collinear you are just adding millions of data uselessly). Another issue is that if you keep feeding more and more features to your model it may get overfit (because it tries to learn everything in the features)**

**# We need to create a new DataFrame to keep a column for vif and another column for features**

**vif=pd.DataFrame( )**

**vif[“vif”]=[variance\_inflation\_factor(X\_scaled,i) for i in range(X\_scaled.shape[1])]**

**#if code is inside the list it is called list comprehension, discussed in python section**

**vif[“features”]=X.columns**

**#Lets check the values**

**Vif**

**Note:range(X-scaled.shape[1])= range between 0 to 8 column**

**I=1=pregnancy column, first passes to variance\_inflation\_factor(X\_scaled,i=1,2,3) and i is compared with all X\_scaled columns one by one means 1 is compared with 1 to 8 then 2 will compared with 1 to 8 and so on. It uses vif= 1/1-R2 formula while comparing and come up with vif score.**

**Finally we have all the score against the features that we will compare with project specific score. Normally vif>5 means collinearity exists and <5 means not exist. But it is not standard it varies as per project which they decide on the basis of their data.**

**Who set it, may be client, project lead etc**

**# Now splitting of dataset**

**x\_train,x\_test,y\_train,y\_test=train\_test\_split(X\_scaled,y,test\_size=0.25, random\_state=255)**

**pre-processing steps completed**

**# Model Building**

**log\_reg=LogisticRegression( )**

**log\_reg.fit(x\_train,y\_train)**

**#let’s see how our model perform on test data**

**y\_pred=log\_reg.predict(x\_test)**

**y-pred**

**24 Sep 2023 video**

**# now we will look how the model is coming up with 0 and 1 for different columns**

**log\_reg.predict\_proba(x\_test)**

**#it will just show how algorithm assigned 0s and 1s , it shows two columns first for 0 and second for 1, the row in which probability in 0 column is high as compared to 1 column it assigns 0 and then vice versa. And also if probability of any column is high the value of probability shows how confidently the algorithm shows the prediction in terms of percentage. So even the values are 0 and 1 but 1 doesn’t mean 100% likewise 0 doesn’t means no confidence. It depends upon the values we get from proba function.**

**# accuracy of the model**

**accuracy=accuracy\_score(y\_test, y\_pred)**

**accuracy**

**#confusiuon matrix**

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

**confusion\_matrix**

**#instead of recalling all recall, precision, F1 indiviually we can call altogether**

**from sklearn.metrics import classification\_report**

**print(classification\_report(y\_test,y\_pred))**

**#So model predicts precision of healthy person separately not overall precision, same for recall and F1-Score, Support is basically size of x\_test samples.**

**In bottom section it gives overall values for each metrics as well. Weighted\_avg will be discussed later**

**#Now roc\_curve it tells how much model will fluctuate on this dataset, if your data don’t have too much variance means not fluctuating in different features (rather following a pattern) means your model will not fluctuate during making predictions**

**It will create three data**

**fpr,tpr,thresholds=roc\_curve(y\_test,y\_pred)**

**#[0] threshold means no instance predicted (it should be read from 0-max), so this thresholds value model will decide on the basis of fluctuation and variance in the data and how much model is understanding the dataset. It may be 0,1,2 or 3. Mean for different fluctuation levels model has to assign different thresholds level to segregate data between various thresholds. So every model think differently.threshold values are read from right to left. Tpr,fpr are read normally.similar tpr,fpr values show model is not fluctuating a lot. We will work on ROC again when we build 4 to 5 models. So this logistic regression ROC only**

**print(‘Thresholds: ‘, thresholds)**

**print(“True positive Rate”, tpr)**

**Print(“False Positive rate: “, fpr)**

**plt.plot(fpr, tpr, color=’orange’, label=’ROC’)**

**plt.plot([0,1], [0,1], color=’darkblue’, linestyle=’—‘)**

**plt.xlabel(‘False Positive Rate’)**

**plt.ylabel(‘True Positive Rate’)**

**plt.title(‘Receiver operating characteristics (ROC) Curve’)**

**plt.legend( )**

**plt.show( )**

**#How much area ROC is covering is AUC (Area Under the curve)**

**auc\_score=roc\_auc\_score(y\_test,y\_pred)**

**print(auc\_score)**

**#Significance of ROC\_AUC in real life**

**# In real life we build 4 to 5 models on a dataset so we select the model having best AUC score**

**Advantages**

**Simple and easy to implement**

**Output is more informative than other models**

**It expresses**

**Disadvantages**

**Not effective for data that is not linearly possible**

**Not as powerful as other classification**

**Multiclass classification are much easier with other classification models**

**It can predict only categorical data**

**KNN (K-nearest neighbor)**

**It is a supervised learning algorithm used for both regression and classification but mostly it is used for classification. Given a dataset with different classes, it tries to predict the correct class of the test data by calculating the distance between the test data and all the training points. It then selects the K-points already assigned to algorithm that lies to the nearest of test point (for classification). Then it calculates the probability of test points belonging to the classes of the K training points and the class with highest probability is selected. In case of regression the predicted value is the mean of K selected training points.**

**The test point whose class is to be decided the algorithm calculates its distance from all the training points of different classes. So if K=3 (neighbors) is given to algorithm it will selects the nearest three points belonging to any of the classes. Say if there are two classes (red and green). If it selects two green points due to shortest distance then probability =2/3 and for 1 green closest point P=1/3. Therefore, due to red higher probability the test point will be assigned to red class**

**It is possible that if k=5 then the probability of green class may become high. So optimum value of K is also crucial. Usually odd value of k is chosen. So best k value will be discussed later. Most famous distance calculation is eculidean. Also manhattans is used.**

**It is also known as lazy learner because it doesn’t create model on the basis of training data it waits for test data. Once test data is provided then only it starts generalizing the training data to predict or classify test data. BAYSEIAN, SVM, LOGISTIC REGRESSION ARE EAGER LEARNERS.**

**Advantages**

**Mathematics behind algo is simple**

**For both classification and regression**

**No need to create model or do hyperparameter training**

**Knn doesn’t make any assumption for the distribution of the given data**

**Not much time in training phase**

**Cons**

**Finding the optimum value OF K**

**IT takes a lot of time to compute the distance between each test sample and training sample**

**Since model is not saved beforehand (lazy learner), so every time for a new test value it follows the same steps**

**Since we need to store whole training set for every test set it takes a lot of space**

**Not suitable for high dimensional data**

**Expensive in testing phase**

**Python Implementation is based on cancer dataset**

**import pandas as pd**

**import numpy as np**

**import matplotlib.pyplot as plt**

**from sklearn.preprocessing import StandardScaler**

**from sklearn.neighbors import KNeighborsClassifier**

**from sklearn.model\_selection import train\_test\_split**

**import pickle # to save and the final model**

**from sklearn.metrics import accuracy\_score, confusion\_matrix, classification\_report**

**import seaborn as sns**

**import warnings**

**warnings.filterwarnings(‘ignore’)**

**#import plotly**

**df=pd.read\_csv(‘breast cancer.csv’, index\_col=0)**

**df.head()**

**print(“shape is: “, shape)**

**print(dfinfo())**

**df.describe**

**df.isnull( ).sum( )**

**In dataset diagnostic is label (M=malignant, B=benign)**

**Many terminologies are new to you the company provides you KT(knowledge Transfer) session.**

**Whenever working on classification model check data is balanced or imbalanced**

**Check in the label column only**

**For example, If B=90 and M=10 samples then the model trains mostly on B samples and the chances are most of the time it predicts B. so this is called imbalanced dataset. Again if 70/30 then deciding the data condition is based on the domain. You need to consult senior advice. So whenever you face imbalance data consult superior. Solution to solve imbalanced dataset will be discussed later**

**df.diagnosis.value\_counts( )**

**you can also plot frequency charts to see data balance**

**sns.countplot(x=’diagnosis’, data=df)**

**plt.show( )**

**df.shape ( )**

**We have seen in the section where we performed multicollinearity where we drew the grpahs, but there are other ways through which directly selects the best features (features that impact the label most).**

**Using SelectKBest feature selection methods, K means quantiy of best features needed K=10 means 10 best features**

**SelectKBest uses f\_classif function to find best features where f\_classif uses ANOVA test. Internally it takes all features and generate f\_score, the more f\_score means a feature is highest contributor, so a fscore is assigned to all features, it keeps all the feature in highest to lowest. Now its up to data scientist how many features to select for this purpose there are techniques**

**from sklearn.feature\_selection import SelectKbest, f\_classif**

**# Replace diagnosis (label column) by binary codes as ML models never understand text**

**df[‘diagnosis’]= df[‘diagnosis’].replace ({‘M’:1 , ‘B’:0})**

**X=df.drop (‘diagnosis’, axis=1)**

**Y=df.diagnosis**

**best\_features=SelectKBest(score\_func=f-classif, k=17) #score\_func=f\_classif selected**

**#K=17 is selected because before that K=13,14 15,16 were checked and k=17 was found best. Furthermore, K=18,19,20 doesn’t cause considerable changes to f\_score. So it’s a hit and trial method**

**#f\_classif is type of score to priorities features behind the scene it uses ANOVA**

**fit= best\_features. fit (X, Y)**

**df\_scores=pd. DataFrame (fit. scores\_) #to find the scores of features**

**df\_columns=pd.DataFrame(X.columns) #we don’t know the scores belong to which column so we extract columns by above command and index table generated as a result shows which index number column has highest f\_score**

**30 Sep 2023**

**#concatenate dataframes means final table should have features, f\_scores and names of columns**

**feature\_scores= pd. concat([df\_columns, df\_scores], axis=1)**

**#Names of columns in the table**

**feature\_scores.column=[‘Feature\_Name’, ‘Score]**

**print(feature\_scores.nlargest(17,’Score’) #Print top 17 columns nsmallest will give ascending order**

**Now these 17 columns will be used to built models**

**MODEL BUILDING**

**new\_X=df[[‘ ‘, ‘ ‘]] #write all seventeen features in the brackets**

**new\_X.describe( ) #here you need to check data of features that are selected and not selected there would be huge difference between accurate values of means, mode, etc b/w two groups also before going to standardscaler step check outliers and remove them**

**scalar=StandardScaler( )**

**X\_scalar= scalar.fit\_transform (new\_X)**

**In KNN standard scaler is not mandatory because mostly it gave same accuracy**

**Building model to test unexposed data**

**X\_train,X\_test,Y\_train,Y\_test= train\_test\_split (X\_scalar,Y, test\_size=0.25,random\_state=355)**

**#Now let’s take another way to get acuuracy report. Write one function and call as many times as you want to check accuracy\_score of different models**

**def metric\_score(clf, X\_train,X\_test,Y\_train,Y\_test, train=True):**

**#clf is model name it can be xyz and it will be replaced by KNN model in our case, train=true means I want my control to go inside to get training set**

**if train:**

**y\_pred=clf.predict(X\_train)**

**print(“\n==============train Results==============”)**

**print(f ”Acurracy Score: {accuracy\_score(y\_train,y\_pred) \* 100:.2f}%”)**

**elif train=False:**

**pred=clf.predict(X\_test)**

**print(“\n==============test Results==============\n”)**

**print(f ”Acurracy Score: {accuracy\_score(y\_test,y\_pred) \* 100:.2f}%”)**

**print(“\n\n test classification report\n”,classification\_report(Y\_test,Y\_predict,digit=2)**

**Above function is general and will be used in all the models. In this model we want to see training results so that we may get the idea that how good our model understand the training data because if training results are good we can expect good test results as well but not essentially. But if too much difference in results we have to think for alternate steps. If they are close means model is perfect (visit 44:44 minutes to learn again)**

**#Inintialize KneighborClassifer**

**Knn=KNeigborsClassifier( )**

**#Model training**

**Knn.fit(X\_train,Y\_train)**

**#Call the function and pass dataset to check train and test scores**

**metric\_score(knn, X\_train,X\_test,Y\_train,Y\_test, train=true)**

**metric\_score(knn, X\_train,X\_test,Y\_train,Y\_test, train=false)**

**If training score is 99 and test score is 95 that fine. But if test score is 75 or 80 it means we may increase this score but not essentially So to increase test score we use hyper\_parameter tuning. Here KNN is kind of complete but not full-fledge as we try to increase test\_score. Now we perform cross validation to check whether our model is overfitting. Its may be ovrfitting if we are getting more than 80%**

**There are certain techniques of cross-validation**

**Hold-out method:we have previously did by train\_test\_split , the hold out method.As we are holding the test data and only giving the train data it is called hold-out method. It is computationally less costly but it can have a high variance because it depends heavily on which data points end up in training set and which ends up in test set.the evaluation will be different everytime the division changes. It trains only one time and we have to trust it. Example is in this method you are asking only one person to predict**

**K-Fold cross\_validation (K=5 means five iterations for dataset)**

**In this method we divide our dataset in which such a way that in each iteration a row becomes a test set and other becomes training set. It is like if we have 5 rows in each iteration we have one prediction and in all iterations we have 5 predictions and we take average of those predictions so that if one prediction is very high other balance it by average. It makes copies of dataset as per k values and always select different divisions for training and testing. So if in one copy/iteration model is overfitting it get balanced by other iterations. So why we can’t use it every time because it is of good quality but require more cost and time. If hold-out=4 hours time then k=5 means 20 hours**

**Here selection of cross validation is based on factors like cost and time**

**LOOCV (Leave one out cross validation)**

**It is the most demanding technique in terms of cost and time. Because it always keeps one sample for testing and use other for training means thousand iterations for thousand samples. At the end it also takes average**

**It is a special type of k-fold method where k becomes n.**

**K-fold method for demo purpose only you don’t need to practice it**

**from sklearn.model\_selection import Kfold, cross\_val\_score**

**k\_f=Kfold(n\_splits=3)**

**k\_f**

**#Split the array into train and test, we are doing it in for loop because it makes split=3 copies of dataset**

**for train, test in k-f.split ([12,23,35,46,51,63,75,86,96,108]):**

**print(‘train :’ train, ‘test:’ test)**

**Cross validation score to check if the model is overfitting**

**Cross\_val\_score (knn, X\_scalar, y, cv=5)**

**#As above command give scores but we need one score so we take mean below**

**Cross\_val\_score (knn, X\_scalar, y, cv=5).mean( )**

**So one time training gives 98% and k\_fold gives 97%**

**BIAS AND VARIANCE NOT EXPLAINED**

**Bias represents the error introduced by model’s assumptions (like giving more important to weight feature having values 65,67 than height feature having values 6.7,5.7, model may give more importance to weight because it assumes so on the basis of large digit we can’t control it, as model is not focusing on height while making patterns it may leads to underfitting) and simplification about the data. A model with high bias tends to be too simplistic making it unable to capture true underlying patterns in the data that leads to under-fitting. As a result, model perform poorly on both training as well as testing data because it oversimplifies the problem.**

**Variance on the other hand refers to model’s sensitivity to fluctuations or noise in the training data (it try to memorize outliers). A model with high variance is too complex and flexible essentially memorizing the noise in the training data rather than generalizing well to new unseen data.**

**The goal of ML is to strike a balance between Bias and variance so that a model generalizes well to unseen data. This target is achieved through model selection, hyper-parameter tuning and regularization to prevent overfitting.**

**Hyper parameter tuning**

**See model was learning on its own but in hyperparameter tuning we are giving parmeters to model that how it should learn**

**There are two techniques GridSearchCV and RandomSearch CV**

**We use GridSearchCV to choose parameters for higher accuracy, its like preparing dish in which you periodically taste salt and spices for quality**

**Let’s use GridSearchCV. See initially in KNN, k=5 was value by default we weren’t controlling that value but if we may change that value or select an optimum value accuracy might change. Also through GridSearchCV method we will know which KNN method is better either kd\_tree or brute force method. How GridSearchCV knows the best K value and which KNN method is good obviously we tell ,As we will put both methods in param\_grid. Leaf size is related to kdtree , we are randomly give options regarding leave size and asking GridSearchCV about optimum leaf size. You can give any number of options you want more options more training time**

**from sklearn.model\_selection import GridSearchCV**

**param\_grid= { ‘algorithm’: [‘kd\_tree’, ‘brute’],**

**‘leaf\_size’ : [3,5,6,7,8],**

**‘n\_neighbors’:[3,5,7,9,11,13]}**

**#So how GridSearch works**

**It takes Kd\_tree and 3 and 3 for leaf and neighbors as first combination.Kd\_tree 3 for leaf and 5 for neighbors and then with brute and other combinations. All permutations and combinations. So for all combination it going to train.so eventually it picks combination of parameter that gives highest training score**

**#tell below which model you use because above params are for KNN**

**gridsearch=GridSearchCV (estimator=knn, param\_grid=param\_grid)**

**gridsearch.fit(X\_train,Y\_train)**

**# it gives best score of the combination we are going to use**

**gridsearch.best\_score\_**

**#it gives us best algorithm best leaf size and best neighbor size**

**gridsearch.best\_estimator\_**

**#Now with above optimum combination initialize knnclassifier**

**Knn=KNeighborsClassifier(algorithm=’kd\_tree’,leaf\_size=3, n\_neighbors=3)**

**knn.fit(X\_train,Y\_train)**

**#Call the function and pass dataset to check train and test scores**

**metric\_score(knn, X\_train,X\_test,Y\_train,Y\_test, train=true)**

**metric\_score(knn, X\_train,X\_test,Y\_train,Y\_test, train=false)**

**# if you want to check the confusion matrix you can check**

**#Always use cost and time definition regarding utilization of algorithms it makes good impression**

**# confusion matrix**

**y\_pred=knn.predict(x\_test)**

**cfm=confusion\_matrix(y\_test,y\_predict)**

**cfm**

**Ist October 2023 Test**

**7th Octoboer 2023**

**ENCODING AND IMPUTING TECHNIQUES (Generic techniques applied to all datasets)**

**Previously we use fillna or replace methods to fill missing values but there are advanced techniques like imputation. Question can be different methods for treating the nulls. Also ML algorithm doesn’t understand text so we need to use encoding techniques to replace those text with numbers.**

**1.LabelEncoder**

**2.OneHotEncoder**

**3.BinaryEncoder**

**4.get\_dummies**

**#Make a dataset**

**Import warnings #if you don’t want to see warnings**

**Warnings.filterwarnings(‘ignore’)**

**import pandas as pd**

**df=pd.DataFrame({‘salary’:[1000,20000,45000,60000,45700],**

**‘city’: [‘hyderabad’,’karachi’,’Lahore’,’peshawar’,’sukkurr,’Multan’]**

**‘gender’:[‘Male’,’female’,’female’,’female’,’male,’male’], ‘Exp’:{1,2,3,4,5,None]})**

**from sklearn.preprocessing import LabelEncoder**

**lab\_enc=LabelEncoder( )**

**df2=lab\_enc.fit\_transform(df[‘city’])**

**# it converts all categories into numbers and arrange them in alphabetical order means 0 for city starting with A 1 for city starting with B etc**

**As it will generate only one column we need to convert it in series**

**pd.series(df2)**

**#Re-assign to df**

**df[‘city’]=df2**

**df**

**#practice on gender**

**Now we want to encode as well as impute in one shot. There are various imputers like SimpleImputer, KNN Imputer, iterativeImputer**

**from sklearn.preprocessing import OneHotEncoder**

**from sklearn.impute import SimpleImputer**

**from sklearn.compose import make\_column\_transformer #helps doing encoding and imputation simultaneously**

**ohe=OneHotEncoder( )**

**si=SimpleImputer( )**

**import pandas as pd**

**df=pd.DataFrame({‘salary’:[1000,20000,45000,60000,45700],**

**‘city’: [‘hyderabad’,’karachi’,’Lahore’,’peshawar’,’sukkurr,’Multan’]**

**‘gender’:[‘Male’,’female’,’female’,’female’,’male,’male’], ‘Exp’:{1,2,3,4,5,None]})**

**ct=make\_column\_transformer((ohe,[‘city’,’Gender’]),(si,[‘Exp’]),remainder=’passthrough’)) # to keep all other columns as it is**

**Depending on the situation any encoding and imputing techniques combination can be used**

**encoded=pd.DataFrame(ct.fit\_transform(df))**

**Now in the dataset columns will increase from original dataset. Because it going to arrange catagories in rows in such a way that if three categories then 1 0 0 in next row 0 1 0 and in third row 0 0 1. Now fourth and fifth columns will be Female and Male. Simple imputer internally uses mean to remove nan in Experience data column. If the column to be impute is category go to import SimpleImputer inside brackets write stratedy=mode)**

**Difference between fillna and imputer is that if I have three columns with NaN , I have to use fillna three times but with imputer I can place all three columns. But if you have different types of columns (one continuous other categorical) you need to make seprate objects of imputers si,si1,si2 with different strategy values**

**#To replace feature numbers with names again**

**encoded=pd.DataFrame(ct.fit\_transform(df), columns=[‘citynames’,’genders’…..])**

**Now in above example we had few categorical attributes but what if we have thousands of categories in dataset, its not possible for us to write down all names in place of numbers. No in that case we left it as it is , just rename label**

**get\_dummies**

**it is not a seprate technique that needs to be imported it is part of pandas**

**it is used to reduce the number of columns we get when we encode categories using OneHotEncoder.**

**df1=pd.get\_dummies(df[[‘city’,’gender’]])**

**df1=pd.get\_dummies(df[[‘city’,’gender’]]), drop\_first=true)**

**drop\_first is by default false, but when we make it true it drops first column in each category. Also get\_dummies is an encoding techniques that keeps the name of features rather than numbers. When you think we are not going to lose much info then use get\_dummies. When you want ML to treat all categories equally then use OneHotEncoder or get\_dummies. So use Label encoder when you want to give importance to any category more than other, also use labelEncoder when you don’t want to create many columns**

**Ordinal Encoder**

**from sklearn.preprocessing import OrdinalEncoder**

**import pandas as pd**

**Empl=pd.DataFrame({‘position’:[‘SE’, ‘Manager’, ‘TeamLead’,’SSE’],**

**‘project’:[‘A’,’B’,’C’,’D’], ‘salary’:[25,000,85000,71000,48000]})**

**ord\_enc=OrdinalEncoder(categories=[[‘SE’,’SSE’,’TeamLead’,’Manager’],[‘A’,’B’,’C’,’D’]])**

**encoded\_df=ord\_enc.fit\_tranform(Empl[[‘position’,’project’]])**

**use this model if you want to tell the model about seniority of position so that highest position get highest weightage. In category we place from low position to high**

**stop at 1:05 hours still remaining**

**08-Oct-2023 video**

**Decision Tree**

**It is only model in ML where we can see what’s happening behind the scene.**

**Supervised learning model, used for both regression and classification**

**Mostly for classification it is better, it is a greedy approach and conditional algorithm. For buying car we consider budget, monthly emi, fuel, marriage budgets. Now which feature is most important is called root node. So it has yes or no condition then in case of yes or no another sub node until its form a inverted tree. In case of No tree branch disconnected. Sub-node are leaf-nodes. If from second to last decision leaf there are three stages it means tree depth=3, in this root node and its decisions stage is not included.**

**Tree pruning**

**Let’s say there are thousands of features/nodes, now depth is too big and its take too much time in training and making decision when it go through all the nodes. Prune means to cut, so to avoid all the nodes to reduce time. Let’s say total tree depth=400 but you prune 300 levels and want to take decision at 100th level to reduce time. How pruning will discuss later. It is reduced to decrease complexity and variance. We have before and after building a model called pre-pruning and post-pruning respectively.**

**Now how to decide which factor/node/feature is most important and to be placed at root node. Every factor is important so it is difficult and it will also not vary person to person so there are techniques to follow because for 1000s of feature we must need some techniques or criteria. So we have two criteria**

1. **Entropy**
2. **Gini-index**

**It helps in finding 1st important 2nd important 3rd important**

**Look for recorded video in LMS and check its math**

**Entropy**

**Lets say we have four features and one label (regression/classification) so entropy looks how much information f1 has for label, how much f2 about label then f3 and f4 and so on. Its like how much 4 friends info about football. Some may have 45%, some 30 and so on. So the feature having highest info about label will become the root node. Information percentage or contribution is called information gain. Then second highest contributor to label then third and so on.**

**Gini impurity or Gini indexing. It is opposite to entropy because it looks for how less a feature knows or contribute about the label. Less impurity (lack of information) most information. So lowest impurity to highest impurity.**

**Which one to use , default criteria of model is Gini because it has less calculation its math is simple. But if you ask is there any criteria to choose entropy or Gini. Yes, if your most features are categorical then entropy. If your features are more continuous then Gini is better. It is not a standard it’s a good guess by data scientist. Or use both to see which performs better in terms of accuracy. But there would be not big difference in results most of the time.**

**Different Algo for Decision tree**

**ID3 is first version (iterative Dichotomiser) we don’t use because it’s only useful for classification.. It only accepts categorical features**

**So they introduce C4.5 it was bit improved as it accepts both continuous and discrete features but give only classification results**

**So best and currently used is CART (classification and Regression technique/Algorithm/model).**

**Advantages of Decision Tree**

**It can be used for both regression and classification**

**Its rules are easy to grasp**

**It can be understood just by visualization**

**Scaling and normalization not needed ( as it is going to divided is yes or No then what’s the point in scaling the data)**

**Disadvantages**

**A small change in data can destabilize the model because of greedy approach (for root node =10 then all below tree works as per 10 value but of root node=10.5 then below entire leafs may change**

**Probability of overfitting is very high (so we have other technique to resolve it)**

**14 October**

**Heatmap in 8 October**

**As compared to other classification algorithm it takes more time to get trained**

**Python implementation of decision tree (Now its implemented on multiclass classification means more than 2 classes)**

**import pandas as pd**

**from sklearn.tree import DecisionTreeClassifier/Regressor**

**from sklearn.model\_selection import train\_test\_split,GridSearchCV**

**from sklearn.preprocessing import StandardScaler**

**from sklearn.metrics import accuracy\_score, confusion\_matrix**

**roc\_curve,roc\_auc\_score,classification\_report**

**import matplotlib.pyplot as plt**

**import seaborn as sns**

**import warnings**

**warnings.filterwarnings(‘ignore’)**

**data=pd.read\_csv(‘wine.csv’)**

**data.head( )**

**data.sample(n=10) (top 10 samples or rows are shown)**

**data.shape**

**data.isna( ).sum( )**

**#In wine data quality has label 5,6,7 classes as per quality**

**from sklearn.preprocessing import OrdinalEncoder**

**order\_encoder = OrdinalEncoder(categories=[[‘low’,’Medium’,’High’]])**

**df1=ord\_encoder.fit\_transform(data[[‘Alcohol\_content’]])**

**df1**

**# Override alcohol contents column with coder**

**data[‘Alcohol\_content’]=df1**

**data.head( )**

**# plotting Heatmap (Correlation matrix) it’s a technique to find relationship strength between features and labels and among labels (multi-collinearity)**

**df\_corr=data.corr ( ).abs( ) #abs means all positive values, this will get the co\_efficient of one variable vs all other variables (Relationship)**

**plt.figure (figsize=(14,8))**

**sns.heatmap(df\_corr, annot=true, annot\_kws={‘size’=10}) #annot and annot\_size(coeff font size) are optional annot=true will plot correlation\_coefficients in the Heatmap**

**plt.show( )**

**#it will show colors and correlation co-efficients, greater the number greater the relation 0.12 means 12% and likewise.**

**Check the colors also brighter means high and dark leads to low we see for brightest for multicollinearity, you can check left or right from 1 diagonal. Now from observation it could be seen that alcohol and alcohol\_content are multicollinear they have high correlation coefficient of around 0.85. But we have to sure by making scatter plot between alcohol and alcohol\_content.**

**plt.scatter(data.alchohol,data.alcohol\_content)**

**plt.xlabel(‘Alcohol’)**

**plt.ylabel(“Alcohol\_content’)**

**plt.title(Alcohol vs Alcohol\_content)**

**plt.show( )**

**In graph one can see that if Alcohol is 7 to 9.5 the Alcohol\_content is 0 but when it oncreasing the the alcohol\_content also increasing means there is a positive relationship b/w both.**

**Now it means we have to remove one feature from these two. So we will see which feature coefficient is higher (contributing more) when compared with label we keep that feature and drop the other**

**# Note: here remember that VIF technique can also be used. Don’t think it was valid only for regression.**

**Now dataset is ready for model building so drop label and multicollinear feature**

**X=data.drop(columns=[‘Alcohol\_content’,’quality’]**

**Y=data[‘quality’]**

**X\_train,X\_test,Y\_train,Y-test = train\_test\_split(X,Y, test\_size=0.25, random\_state=41)**

**Functions to repeatedly check accuracy\_score of different models**

**def\_metric\_score(clf,X\_train,X\_test,Y\_train,Y\_test, train=true)**

**if train:**

**Y\_pred=clf.predict(x\_train)**

**print(“\n==============train Results==============”)**

**print(f ”Acurracy Score: {accuracy\_score(y\_train,y\_pred) \* 100:.2f}%”)**

**elif train=False:**

**pred=clf.predict(X\_test)**

**print(“\n==============test Results==============\n”)**

**print(f ”Acurracy Score: {accuracy\_score(y\_test,y\_pred) \* 100:.2f}%”)**

**print(“\n\n test classification report\n”,classification\_report(Y\_test,Y\_predict,digit=2)**

**# Model initialization**

**clf=DecisionTreeClassifier( )**

**clf.fit(X\_train,Y\_train)**

**#call the function**

**metric\_score(clf, X\_train,X\_test,Y\_train,Y\_test, train=true) #for training score**

**metric\_score(clf, X\_train,X\_test,Y\_train,Y\_test, train=false) #for testing score**

**Support is X\_test data**

**Confusion matrix for multiclassfication**

**Y\_pred=clf.predict(X\_test)**

**Confusion\_matrix(Y\_test,Y\_pred)**

**#Converting into data\_frame to understand it better**

**Pd.DataFrame(confusion\_matrix(Y\_test,Y\_pred), columns=[‘Q3’, ‘Q4’,‘Q5’,‘Q6’,‘Q7’,‘Q8’], index=[‘Q3’, ‘Q4’,‘Q5’,‘Q6’,‘Q7’,‘Q8’])**

**Let’s see how tree looks like it has nothing to do with algorithm accuracy. It’s just for visualization from time: 1:51:10 till end ,sometime due to low configuration it is difficult to visualize, so don’t waste too much time. Its also available on datatrained github link**

**Now in above algorithms, difference between training and testing accuracy is too high therefore we will apply hyperparameter tuning techniques.**

**For decision tree models we are improving four parameters by passing different values**

**Grid\_param = {**

**‘criterion’: [‘gini’,’Entropy’],**

**‘max\_depth’: range(10,15), #the max depth of the tree. Actually it was 70 but I don’t want to go all the way to 70 despite I want to check between 10 to 15 level that which 5 are best.**

**‘min\_samples\_leaf’: range(10,15), #the min number of samples required at leaf node**

**‘min\_samples\_split’: range(10,15), ##the min number of samples required to split an internal node means if node is less than or greater than certain number of samples than don’t split. Furthermore it will split when the label is dependent upon two features otherwise not.**

**‘max\_leaf\_nodes’:range (5,15) #means how many leaf nodes at the end to determine label, if None then unlimited number of leaf node**

**}**

**grid\_search=GridSearchCV(estimator=clf,**

**Param\_grid=grid\_param,**

**cv=5,**

**n\_jobs=-1) #use all the cores of the system for performance improvement**

**grid\_search.fit(X\_train,Y\_train)**

**best\_parameters=grid\_search.best\_params\_**

**print(best\_parameters)**

**#Initiate Decision tree with new parameters and start training**

**Play with numbers and keep changing the values to check how more score could be achieved and you get idea of dealing this process 10-15 times doesn’t matter**

**clf=DecisionTreeClassifier(criterion=’entropy’, min\_samples\_split=4,min\_samples\_leaf=2,max\_depth=8, max\_leaf\_nodes=20)**

**clf.fit(X\_train,Y\_train)**

**# call the function and pass dataset to check train and test scores**

**metric\_score(clf,X\_train,X-test,Y-train,Y-test,train=true) #for training score**

**metric\_score(clf,X\_train,X-test,Y-train,Y-test,train=false) #for testing score**

**#Now we will make decision tree again after tuning**

**#prediction for confusion matrix again after tuning**

**Y\_pred=clf.predict(X\_test)**

**#Converting into dataframe to understand this in better way**

**pd.DataFrame(confusion\_matrix(y\_test,y\_pred),columns=[Q3,Q4,Q5,Q6,Q7,Q8], index=[ Q3,Q4,Q5,Q6,Q7,Q8])**

**#Visualise tree (not mandatory) available at 41:06 and github repository**

**See when required , change tuning parameters as accordingly to improve training and testing accuracies**

**Ensemble (collection) Approach (Bagging , Boosting): to solve complex problems**

**Bagging (bootstrap aggregation)**

Means using multiple models of same kind (means either 10-KNN models or 10-Logistic Regression etc) and making decision (regression/classification) for supervised learning models.

For example, you buy a car and ask a single model. It may say yes/No. But asking 10 similar models can give more reliability and if majority of the models out of 10 say Yes then you can confidently buy a car. Using gridsearchCV we can find out what the best number of models(estimators-10,20,30) to use for decision making for a particular dataset.

From1:01 an example is defined

In bagging if bootstap=yes means with replacement means you are using 5 models of same kind (either all KNN, all Decision tree etc) they give same result. So, they give different results when the data provided to them will be different. So what happened first model randomly select different features (how many features is preset by programmer but which feature can’t be determined in advance). Second model will pick three features but as bootstrap=yes it can pick the same features previously picked by model-1 and so on by model 3,4,5….All models are internally independent. But if bootstrap=false no model will pick same feature selected previously by another model. It is called w/o replacement and pasting.

Finally it goes with the majority decision made by models in case of classification. In case of regression it takes the average of all decision made by all models of same kind.

By doing this we can avoid biasdness as majority is better than single opinion.

Question can be then why we are not using bagging always, answer is time and cost factor .

**Out of Bag Evaluation:** while bagging operation we don’t seprate data into train/test split. So we don’t perform cross validation or LOOCV. As bagging operation automatically separate some features for testing. This portion of data is not given to any model. What happen that sample provided from features to model are not 100% but portion of a particular sample. Remaining is set aside for testing.

**In bagging there are many models baggingclassifier, RandomForest, stacking, booting there are many models but they are not used much but RandomForest is mostly used now.**

**15 October 2023**

**from sklearn.ensemble import BaggingClassifier**

**from sklearn.neighbors import c**

**import warnings**

**warning.filterwarning(‘ignore’)**

**#for individual KNN**

**from sklearn.dataset import load\_breast\_cancer**

**dataset= load\_breast\_cancer**

**X=dataset.data**

**Y=dataset.target**

**from sklearn.model\_selection import train\_test\_split**

**X-train,X\_test,Y-train,Y-test= train\_test\_split(X,Y,random\_state=4)**

**knn= KNeighborsClassifier(n\_neighbors=5)**

**knn.fit(X\_train,Y\_train)**

**knn.score(X\_test,Y\_test)**

**Now let’s use bagging with true over KNN classifier to see our score improves or not:for KNN x 9 models**

**bag\_knn=BaggingClassifier(KNeighborsClassifier(n\_neighbors=5),n\_estimators=9,max\_samples=0.7,bootstrap=True,random\_state=3,oob\_score=True)**

**#oob\_score is means out of bag evaluation score of data**

**bag\_knn.fit(X\_train,Y\_train)**

**bag\_knn.score(X\_test,Y\_test)**

**scores improves with bagging, great**

**Now lets use pasting with bagging**

**pasting\_knn=BaggingClassifier(KNeighborsClassifier(n\_neighbors=5),n\_estimators=9,max\_samples=0.7,bootstrap=false,random\_state=3)**

**pasting\_knn.fit (X\_train, Y\_train)**

**pasting\_knn.score(X\_test,Y\_test)**

**#Bagging classifier is only to get understanding but for project RandomForest is used. The main idea behind bagging is to decrease over-fitting it may or may not increase accuracy as compared to single model implementation**

**# when to set bootstrap=true, when you don’t have too many features but when you have 100 or 1000s of features bootstrap=false is better**

**Random Forest**

**It’s the main algorithm in the bagging. Useful for classification but also be used for Regression. It works same as bagging but little bit different as it collects different features but not all the samples from any particular feature. Means in RandomForest each model picks all the features but not all the samples. One model may pick top rows of all features, second may pick middle rows of all features, third model may pick bottom rows of all features or randomly each model select different samples from all features.**

**By default Random Forest uses Decision Tree as a base model. Because it has low bias high variance. As we studied decision trees tend to overfit data. So one solution is to use bagging with decision tree as a base model. But an issue is that in bagging there is not sufficient independence among dataset, means they are correlated. So advantage of random forest over bagging is that it brings changes to the bagging algorithm to decrease correlation. The idea is to generate more randomness while generating trees.**

**import pandas as pd**

**import numpy as np**

**import matplotlib.pyplot as plt**

**import seaborn as sns**

**from sklearn.preprocessing import StandardScaler**

**from sklearn.model\_selection import train\_test\_split, GridSearchCV**

**from sklearn.ensemble import RandomForestClassifier/Regressor**

**from sklearn.metrics import accuracy\_score, confusion\_matrix, classification\_report**

**import warnings**

**warning.filterwarning(‘ignore’)**

**%matplotlib.inline**

**df=pd.read\_csv(‘cardio\_train.csv’, sep=’;’)**

**df.head( )**

**print(df.shape)**

**df.drop(‘id’,axis=1,inplace=True)**

**df. describe()**

**#Make report on outliers, null values, etc**

**df.drop\_duplicates(inplace=True)**

**df.shapes**

**# Make sure to remove id column before removing duplicates because ids are always unique so duplicate dropping will not work. Also form data.describe you feel some data is not correct so use those features only in the program below**

**Plt.figure(figsize=(20,15))**

**Plotnumber=1**

**for column in df[[‘age’,’height’,’ap\_hi’,’ap\_low’,’weight’]]:**

**if plotnumber<=6:**

**ax=plt.subplot(3,2,plotnumber)**

**sns.distplot(df[column])**

**plt.xlabel(column,fontsize=20)**

**plotnumber+=1**

**plt.tight\_layout ( )**

**#Now from results it can be seen that there are outliers so one way is to use boxplot and outlier detection method. It took lot of time because we did it for each feature in which we decided to remove data from different features using IQR to remove get rid of Outliers. Another option is using or converting normal distribution into Z-distribution or standard Normal distribution. It is nothing but standard scaler that makes sure mean=0 and standard deviation=1. Mean is zero because some data is in positive and some is in negative. Std=1 covers 68%(between 1 and -1) of data, std=2(95% of data), std=3(99.97%). Because in datascientest community it is assumed that most of outliers lie beyond std=3, 99.97%(0.3% are outiers) so they omit this much of data to remove outliers in Z-distribution. Mostly it works. So if you think my data has lots of outlier than take std=2 rather than 3**

**from scipy.stats import zscore**

**z\_score=zscore( df[[‘age’,’height’,’ap\_hi’,’ap\_low’,’weight’]])**

**#Absolute Z-score means all numbers 0.1,0.2, -0.1,-0.2…. between 1 and -1**

**abs\_z\_score=np.abs(z\_score)**

**filtering\_entry=(abs\_z\_score<3).all(axis=1) #all data between three standard deviations**

**df=df[filtering\_entry] #need 3 standard deviation data from main dataframe, if you think its taking more outliers you can choose 2, 2.7, 2.8 std as well also 3.1 is possible**

**df. describe()**

**df.head( )**

**#for huge number of features Z-distribution is better than boxplot**

**#visualize how many outliers are removed**

**Plt.figure(figsize=(20,15))**

**Plotnumber=1**

**for column in df[[‘age’,’height’,’ap\_hi’,’ap\_low’,’weight’]]:**

**if plotnumber<=6:**

**ax=plt.subplot(3,2,plotnumber)**

**sns.distplot(df[column])**

**plt.xlabel(column,fontsize=20)**

**#if you are not satisfied go back and change abs\_z\_score values**

**#for selecting features we have stripplot, selectkbest, VIF, heatmap. Here this process is omitted intentionally. Do it yourself**

**Another way of outlier removal**

**columns\_to\_sqrt = ['DailyRate','DistanceFromHome','EmployeeNumber','HourlyRate','MonthlyIncome','MonthlyRate','PercentSalaryHike','TotalWorkingYears','YearsAtCompany','YearsInCurrentRole','YearsWithCurrManager']**

**for column in columns\_to\_sqrt:**

**data[column] = np.sqrt(data[column])**

**print(data)**

**Stop at 01:01:28**

**Project 2 (Regression) Discussion (Fliprobo)**

# In dataframe table data.column is same as data[‘column’]

# To handle date column

data[‘date]=pd.to\_\_datetime(data.date, format=’%Y-%m-%d’)

df.info()

df.set\_index(‘date’, inplace=True) # Now date is set as index column instead of original index (making date index will remove date from input features, date is selected because its unique, ID , Passport No. can also be set as index or any unique column)

#checking unique values in each column

df.nunique( ).to\_frame (“No. of Unique Values”)

#checking value counts of each column means which values (samples) are how many time

for i in df.columns:

print(df[i].value\_counts( ))

print(“\n”)

**# Checking duplicate values in DataFrame**

**df.duplicated( ).sum( )**

**# Q2 or 50% is second Quartile and median value, If Q2 is greater than the mean it means data is left skewed and vice versa (right skewed).**

**# if there is huge difference between max and Q3(75%) then it means it has outliers**

**# To find exact either formula.skew method to find exact skewness**

**# IQR method, z-Score**

**# Cross check various method**

**sns.lmplot (x=’open’,y=’close’, data=df, palette=’colorblind’)**

**# to see relation between features and features and features and label simply correlation, scatter plot can be also used, outlier can also be detected, lm also shows best fit line**

**# we can use regplot, scatterplot as well as lmplot, difference is in scatterplot bestfit line is not present, in regplot bestfit line is present,**

**plt.figure(figsize=(20,15), color=’red’)**

**p=1**

**for i in df:**

**if P<=17:**

**plt.subplot(5,4,p)**

**sns.regplot(x=’close’,y=i,data=df,color=’r’)**

**plt.xlabel(‘close’)**

**plt.ylabel(i)**

**p=+1**

**plt.show( )**

**# Also pairplot for find relationship between different pairs**

**sns.pairplot(data=df, palette=’Dark2”)**

**#high positive=+1**

**#low negative**

**-0.3 to +0.3 considered as low correlation**

**-0.5 to +0.5 medium correlation**

**-0.7 to 0.7 and above strong correlation**

**# sns.lineplot(data=df.iloc[:,:-1]**

**# visit seaborn website to see all type of plots**

**# check the outliers**

**plt.figure(figsize=(20,25))**

**p1=**

**for i in df:**

**if p<=13:**

**plt.subplot(5,4,p)**

**sns.boxplot(df[i], palette=”set2\_r”)**

**plt.xlabel(i)**

**p+=1**

**plt.show( )**

**# Use domain knowledge, if not then discuss with expert, so whether outliers are accepted or not it depends on domain knowledge, but if I don’t have domain knowledge I will try to reduce the outliers**

**#use Z Score or IQR method (samplevalue-mean/standard deviation), we don’t remove outlier from categorical data and target data**

**from scipy.stats import zscore**

**out\_feature=df[[‘open’,’High’,’low’]] #DataFrame format**

**z=np.abs(zscore(out\_features))**

**#99.7 data is in range of -3 to +3**

**# so beyond this value threshold=3 not -3 because we have converted it in absolute values above**

**np.where(z>3)**

**# first array shows row number and second array shows column number**

**z.iloc[723,3] # to confirm that value at row 723, and column 3 is greater than 3**

**# Now removing the data above zscore and creating new DataFrame so that original df may preserve and could be used for IQR method**

**df1=df[(Z<3).all(axis=1)] # keeping all data where z value is le**

**df1.shape**

**# in pandas axis=1 is column wise operation but in numpy column=1 is row wise**

**# we don’t detect outliers from target and categorical but we remove all rows from them.**

**# Calculate data loss percentage**

**print(“old DataFrame :”, df.shape[0])**

**print(“new DataFrame :”, df1.shape[0])**

**print(“data loss percentage…”,(df.shape[0]-df1.shape[0]/df.shape[0])\*100**

**# consistent rows are mandatory because its needed for train\_test\_split**

**# Now we can apply IQR method on original df**

**Q1=out\_feature.quantile(0.25)**

**Q3=out\_feature. quantile(0.75)**

**IQR=Q3 – Q1**

**#min value #max value**

**# if any value less than the min value or any value greater than the max value we will keep it otherwise drop it so below is opposite that is why we are using ~**

**Any means if any datapoint is matching**

**df2=df[~((df <(Q1- 1.5\*IQR)) | (df > (Q3+1.5\*IQR))).any(axis=1)]**

**df2.shape**

**# Again calculate dataloss percentage between df and df2**

**# if data loss of z score is less than select z score over IQR or vice versa**

**# so make df=df1**

**#In case of IQR do df=df2**

**# Now check how data is distributed, it two bells are forming its okay because it can be bimodal type data its not skewness**

**# Then check skewness**

**df.skew**

**# it can give negative value for left skew and positive value for right skew so 0.12 and -0.16 are acceptable skewness where 1 and beyond are high skewness**

**# Removing skewness using cuberoot method of remaining features**

**df[‘column’]=np.cbrt[df[‘column’]**

**df. Skew( )**

**# or apply distplot on columns**

**sns.distplot(df[’volume’], color =’m’, kde\_kws= {“shade”:True}, hist=False)**

**# checking correlation**

**df.corr( )**

**# or by heatmap**

**# we can either use vif or PCA to handle multicollinearity but first seprate target and independent variables**

**X=df1.drop(“close”, axis=1)**

**Y=df1[“close”]**

**from sklearn.preprocessing import StandardScaler**

**scaler=StandardScaler( )**

**x=Pd.DataFrame(scaler.fit\_transform(x), columns=x.columns)**

**x**

**# Checking Variance Inflation Factor (VIF)**

**from statmodels.stats.outliers\_influence import variance\_inflation\_factor**

**vif=pd.DataFrame( )**

**vif[“VIF values”]= [variance\_inflation\_factor (x.values, i) for i in range (len(x.columns))]**

**vif[‘features’]=x.columns**

**vif**

**# if threshold is 10 then any value below 10 isn’t removed but determining threshold depends on data and removing column should be done on the basis that how much its correlated with target and how much percentage of data could be lost upon removing that column**

**# use PCA method or regularization (ridge or LASSO if you are not dropping the column**

**# Modelling**

**Finding best random state**

**from sklearn.model\_selection import train\_test\_split**

**from sklearn.metrics import mean\_absolute\_error**

**from sklearn.metrics import mean\_squared\_error**

**from sklearn.metrics import r2\_score**

**from sklearn.ensemble import GradientBoostingRegressor**

**from sklearn.linear\_model import LinearRegression**

**Creating train test split**

**# finding best state**

**maxAccu=0**

**maxRS=0**

**for i in range (1,200): #More range more time**

**x\_train,x\_test,y\_train,y\_test=train\_test\_split (x,y,test\_size=0.3, random\_state=i)**

**lr=LinearRegression**

**lr.fit(x\_train,y\_train)**

**pred=lr.predict(x\_test)**

**acc=r2\_score(y\_test,pred)**

**if acc>maxAccu:**

**maxAccu=acc**

**maxRS=i**

**print(“Maximum r2 score is :”, maxAccu,”on\_Random\_state”,maxRS)**

**x\_train,x\_test,y\_train,y\_test=train\_test\_split (x,y,test\_size=0.3, random\_state=maxRS)**

**from sklearn.svm import SVR**

**from sklearn.ensemble import ExtraTreeRegressor**

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

**# default cross val score for classifivcation is accuracy and in regression it is r2\_score**

**# check metrics on sklearn website**

**score= cross\_val\_score(LR,x,y,cv=5,scoring=’r2’)**

**cv=5 means my data is shuffled in 5 equal parts and each part is randomly selected**

**print(score)**

**print(score.mean()) # because score are five values so to make it one**

**print(“Difference between R2 score and CV score is..” (r2\_score(y\_test,pred\_LR) -score.mean())\*100)**

**# difference means the the model previous to CV was giving whatever is the difference more value of r2\_score due to overfitting**

**# Similar way do it for all models cv=5 or little more is ok lesser is not good**

**# CV score can be negative, it happens because each model is different**

**Now model having least difference between r2\_score and cv score will be selected and gridsearchcv will be performed on the same**

**from sklearn.model\_selection import GridSearchCV**

**# applying hyperparameter tuning doesn’t always increase performance, if algorithm is already giving best performance then how it is possible. Further it depends on hyper parameter tuning values**

**param={‘alpha’:[1.0,0.05’,0.4,2],’fit\_intercept’:[True,False],**

**‘solver’:[‘auto’,’svd’,’cholesky’,’lsqr’,’sag’,’saga’,’lbfgs’],**

**‘positive’:[False,True],**

**‘random\_state’:[1,4,10,20]**

**}**

**gscv=GridSearchCV(Ridge( ), param, cv=5)**

**gscv.fit(x\_train,y\_train)**

**gscv.best\_params\_**

**Model=Ridge(write parameters of Best params)**

**Model.fit(x\_train,y\_train)**

**Pred=Model.predict(x\_test)**

**Print(“R2\_Score is: “, r2\_score(y\_test,pred)**

**Print(“Mean Absolute error is: “, mean\_absolute\_error(y\_test,pred)**

**Print(“mean squared erro is: “, mean\_squared\_error(y\_test,pred)**

**Print(“Root mean square is: “, np.sqrt(mean\_squared\_error(y\_test,pred)))**

**# Once it is done we will save the model**

**import joblib**

**# either joblib or pickle can be used to save the model**

**# joblib.dump(Model, “TeslaStock.pk1)**

**import pickle**

**filename=’TeslaStock.pk1’**

**pickle.dump(Model, open(filename,’wb’)) #saved model**

**# open save model and convert file into binary using rb**

**import pickle**

**## open save model and read file using wb**

**loaded\_model=pickle.load(open(’TeslaStock.pk1’,’rb’))**

**result=loaded\_model.score(x\_test,y\_test)**

**print(result\*100)**

**conclusion=pd.DataFrame([loaded\_model.predict(x\_test)[:],y\_test[:]], index=[‘Predicted’,’Original’])**

**conclusion**

**# if training and test data are in two different files**

**Step1: Perform EDA, preprocessing, feature engineering etc**

**Step2: split dataset into features and target variable and then train and test dataset using train test split**

**Find best model, do hyper parameter tuning and save the model**

**For train dataset file you need both step1 and step2**

**For test datasetfile**

**You need to do all the same activities(even same methods) that you have done in step 1 with train data only**

**Now load the saved model (model which we saved in step2 for train dataset) and use test datasetfile to predict the target variable**

**Project 1 Discussion (CLASSIFICATION)**

**Most important thing for ML projects?**

**Problem statement is most important because everything is based on understanding it correctly and it will help in collecting and analysis of data**

**import pandas as pd**

**import numpy as np**

**import seaborn as sns**

**import matplotlib.pyplot as plt**

**import warnings**

**warnings.filterwarnings(‘ignore’)**

**# Importing Dataset when it is raw, click on raw option of github link and copy paste that particular link into pd.read argument, Second approach is file downloading and uploading the downloaded file into jupyter. Third option is to give location address of file present in your system**

**df= pd.read\_csv(‘webaddress’)**

**df**

**# How to check whether the target is binary or multiclass classification**

**Check multiple samples to confirm**

**df.head(15)**

**df.tail(30)**

**# Exploratory Data Analysis (two types: non-graphical and graphical analysis)**

**df.shape**

**df.columns**

**# checking in list form**

**df.columns.tolist**

**df.dtypes**

**# Sometime float datatype is in object datatype because the number 23.4 are written as’23.4’ that is string.**

**# NaN (Null value) doesn’t change datatype of columns because it means no datatype**

**df.isnull( ).sum( )**

**df.isnull( ).sum( ) .sum( )**

**df.info( )**

**# null values can also be visualized using heatmap**

**sns.heatmap(df.isnull( ))**

**# check reason of datatypes of different columns. it will show all unique values in any column, it may be string. It gives array of unique values**

**# for loop can also be applied for multiple values**

**df[‘TotalCharges’].unique( )**

**# to check number of Unique values**

**df[‘TotalCharges’].nunique( )**

**# Checking the value counts(or number of unique values) of each column, it will also show whether the problem is binary classification or of multiclass classification**

**#dtype given in answers is not the datatype of column it is the datatype of frequency which is mostly in integer64**

**# There can be white spaces or missing values in the column these are different from Nan values. So it can also be a reason that float datatype can be shown as object datatype alongwith some values are string**

**for i in df.columns:**

**print(df[i].value\_counts( ))**

**print(‘\n’)**

**# checking blank values/whitespaces within column using unique function**

**df.loc[df[‘TotalCharges’]==” “]**

**# filling missing values with any value and again check value counts, length of column will be reduced by replacing whitespaces with null because null value doesn’t has any datatype**

**df[‘TotalCharges’]= df[‘TotalCharges’].replace(“ “, np.nan)**

**df[‘TotalCharges’].value\_counts( )**

**df.isnull( ).sum( )**

**# converting object datatype into float datatype by typecasting**

**df[‘TotalCharges’]= df[‘TotalCharges’].astype(float)**

**df[‘TotalCharges’].dtype**

**df.info ( )**

**# checking mean of the columns**

**print(np.mean(df[“Totalcharges”].mean( ))**

**# replacing nan value using mean method using fillna, it is applied one by one on each column, for multiple columns imputer can also be used, but it will be applied on float and object columns separately with strategies mentioned as strategy=’mean’ for float and strategy=’most frequent’ for mode used for objects**

**df[“Totalcharges”]= (df[“Totalcharges”].fillna (df[“Totalcharges”].mean( ))**

**df.isnull( ).sum( )**

**sns.heatmap(df.isnull( ), cmap=”cool\_r”)**

**# Now to further check whether white spaces have been replaced or not by mean**

**# row number 488 and all the columns, 488 because there was a white space before at that location**

**# applying mean w/o converting whitespace into nan will cause problem**

**# if 50% or more values in any column are null drop that column because it will cause duplication of values problem. Or use iterative approach**

**df.iloc[488,:]**

**# Seprating Numerical and categorical columns**

**# Checking for categorical columns**

**categorical\_col=[ ]**

**for i in df.dtypes.index:**

**if df.dtypes[i] == “object”:**

**categorical\_col.append(i)**

**print(“categorical columns : “, categorical\_col)**

**# Checking for numerical columns**

**numerical\_col=[ ]**

**for i in df.dtypes.index:**

**if df.dtypes[i] != “object”:**

**numerical\_col.append(i)**

**print(“numerical columns : “, numerical\_col)**

**# checking number of unique values in each column in dataframe form**

**df.nunique( ).to\_frame(“No of unique values”)**

**# CustomerID has unique ID that can be dropped as it won’t help in analysis part because of lack of any pattern**

**df.drop(“customerID”, axis=1)**

**df.head( )**

**# checking list of counts, and unique values of target**

**df[“target”].unique**

**df[“target”].value\_counts( )**

**Description of the numerical columns of dataset**

**df.describe( )**

**# relation between mean and median w.r.t skewness**

**# if mean=median no skewness**

**# if mean>median right skewness**

**# if mean<median left skewness**

**Analysis**

**# The counts are same means there are no missing values in the dataset**

**# Write down skewness status of all columns in view of median and mean comparison**

**# If there is huge difference between 75% and max it means outliers are present in data**

**# Also check std and IQR**

**# difference between mean and std should be less**

**Data Visualization (graphical Analysis)**

**# Univariatre Analysis**

**# difference between univariate (analysis of single column), bivariate (analysis between two columns) and multivariate (analysis of multiple columns) is**

**# Visualize the number of targets that are yes and No, so that class imbalance problem of target could be found**

**ax= sns.countplot(x=’target’, data=df)**

**print(df[‘target’].value\_counts ()**

**# So, write down whether there is class imbalance present or not 70:30 is termed as class imbalance**

**# class imbalance will be resolved by oversampling method later on**

**# In similar fashion check the imbalance of object datatypes columns of features, it will show histograms for two, three and more classes also**

**print(df[‘object\_feature’].value\_counts( ))**

**ax=sns.countplot(x=‘object\_feature’, data=df)**

**plt.show( )**

**# Now let’s check how data is distributed in remaining columns**

**plt.figure(figsize= (10,6), facecolor=’white’)**

**plotnumber=1**

**for col in numerical\_col:**

**if plotnumber<=4:**

**ax=plt.subplot(2,2,plotnumber)**

**sns. distplot(df[col], color=’m’)**

**plt.xlabel(col, fontsize=12)**

**plt.yticks(rotation=0, fontsize=10)**

**plotnumber+=1**

**plt.tight\_layout( )**

**# Difference between categorical, numerical and continuous data**

**# continuous data doesn’t mean float values, because if unique values are ten and repeated several times, it means the data can be grouped on the basis of unique values, so it can be categorized, and it can be string and number. So, it is numerically categorized.**

**# But if all the values are unique and total size of data of that particular column has difference very less and it is not possible to categorize. In chart it can be seen as well, as in categorical data has multiple separate histograms whereas in continuous data column histograms are connected. So, it can be multiclass classification.**

**Bivariate Analysis ( For checking the relationship between target value and features and features vs features). Scatter plot is used to know how columns are affecting each other and how data is distributed**

**plt.title (“ comparison between tenure and seniorcitizen”)**

**sns.stripplot(x= “seniorcitizen”, y=”tenure”, data=df)**

**plt.show( )**

**# Another graph for comparison**

**plt.title(“comparison between tenure and totalcharges”)**

**sns.scatterplot(x=”tenure”, y=”TotalCharges”, data=df, hue=’churn’, palette=’bright’)**

**plt.show( )**

**# palette is colorcoding for target**

**# hue is the column by which we are taking the reference**

**# In this graph one can find that how target values of “Yes” and “No” are responding w.r.t to x axis feature and y axis feature.**

**# Similarly comparison of two features in histogram style with keeping target in view**

**plt.title(“comparison between churn and gender”)**

**sns.barrplot(x=”gender”, y=”seniorcitizen”, data=df, hue=’churn’, palette=’winter’)**

**plt.show( )**

**sns.catplot(x=”targetcolumn”, col=’featurecolumn’, data=df, kind=”count”, palette=”spring\_r”)**

**# go to seaborn website to learn about more plot types through official documentation**

**Multivariate**

**sns.pairplot(df, hue= “churn”, palette=”Dark2”)**

**plt.show( )**

**# It gives distribution of pair**

**plt.figure(figsize=(10, 16), facecolor='red')**

**plotnumber = 1**

**for col in numerical\_col:**

**if plotnumber <= 13:**

**ax = plt.subplot(3, 5, plotnumber)**

**sns.boxplot(data1[col], palette='Blues')**

**plt.xlabel(col, fontsize=15)**

**plt.yticks(rotation=0, fontsize=10)**

**plotnumber += 1**

**plt.tight\_layout()**

**# It is applied only on numerical data, so for finding skewness you need mean or median which doesn’t exist in case of categorical data, for target also we don’t remove the skewness or outliers**

**# standard deviation is used to calculate how far the numbers (data) is distributed**

**# Variance means how far the data is away from the mean**

**# single line in boxplot means whole data lies on that line min,25,50,75,max**

**# IQR is difference between 1st and third quartile**

**Checking for skewness**

**df.skew( )**

**# 0.5 to -0.5 is normal skewness of real world data. So if skewness go out this range we try reduce the skewness**

**# if after applying various skewness methods the value of skewness doesn’t come within the normal range, then the existing skewness will be considered as normal skewness of that specific data**

**Removing skewness using cuberoot**

**df[‘feature’]=np.cbrt(df[‘feature’])**

**df.skew( )**

**# Checking graphically the skewness for single column**

**sns.distplot(df[‘feature’], color=’m’, kde\_kws={“shade”:True,} hist=False)**

**Plt.show( )**

**Encoding categorical columns**

**# Encoding categorical columns using OrdinalEncoder, it is applied on target as well**

**from sklearn.preprocessing import OrdinalEncoder**

**OE=OrdinalEncoder( )**

**for i in df.columns:**

**if df[i].dtypes==”object”:**

**df[i]=OE.fit\_transform(df[i].values.reshape (-1,1))**

**df**

**# fit and transform are two methods. when we aren’t aware of number of rows then (-1,1) do it automatically**

**df.info ( )**

**df.describe( )**

**# Checking the correlation between target and features and features and features, multicollinearity is when two independent variables are correlated with each other**

**cor=df.corr( )**

**cor**

**# corr range is -1 to 1 ,**

**# better is to use heatmap**

**plt.figure(figsize=(20, 15))**

**sns.heatmap(df.corr( ), linewidths=0.1, fmt=”.1g”, linecolor=’black’, annot=True, cmap=”Blues\_r”)**

**plt.yticks(rotation=0)**

**plt.show( )**

**# This heatmap shows relationship between features and target and features with features**

**And so on…**

**# Here 0 =no relation, -0.3 to +0.3 less correlation, greater than -0.5 to +0.5 moderate correlation, greater than -0.7 to +0.7 high correlation**

**# Now to find correlation of each variable with target only**

**cor[‘target’\_column].sort\_values(ascending=False)**

**Visualizing the relation between features and label using barplot**

**plt.figure(figsize=(22, 7))**

**df.corr()[‘targetcolumn’].sort\_values(ascending=False).drop([‘churn’]).plot(kind=’bar’,colors=”m”)**

**plt.xlabel(‘Features’, fontsize=15)**

**plt.ylabel(‘Target’, fontsize=15)**

**plt.title(“correlation between label and features using barplot, fontsize=20)**

**plt.show( )**

**# if multicollinearity exist then one can use vif or PCA to handle it**

**Seprating the columns**

**X=df.drop(“targetcolumn”, axis=1)**

**Y=df[“targetcolumn”]**

**# Scaling methods are applied only on independent variables, it is used to avoid biasdness or making units related to scale so that model treat all samples equally ( mean=0 std=1 for standard scaler, for min,max scaling the range is between 0 to 1**

**from sklearn.preprocessing import StandardScaler**

**scaler=StandardScaler( )**

**X=pd.DataFrame(scaler.fit\_transform(X), columns=X.columns)**

**X**

**# Now to treat multicollinearity issue we are applying variance inflation factor (VIF) in each scaled columns , it always applies on independent variables, so no target variable**

**from statsmodels.stats.outliers\_influence import variance\_inflation\_factor**

**# To make a dataframe**

**vif=pd.DataFrame( )**

**vif[“VIF Values”]= [variance\_inflation\_factor(X.values,i) for i in range(len(X.columns))]**

**vif[“Features”]=X.columns**

**vif**

**# Lowest vif value is 1 not less than 1, if any feature has vif=1 it means it is not correlated with any other independent variable. If any feature has vif=12 has more correlation with other independent values. Normally in most of the cases any feature having vif value more than 5 should be removed. But in real life data we don’t rush to lose the data. So we takes vif=10 as threshold to drop the feature instead of 5. So any feature vif>10 is dropped. If two or more than two independent features have vif>10 then we remove one with highest vif value**

**X.drop(‘feature\_to\_be\_dropped’, axis=1, inplace=True)**

**# Again checking vif values to confirm whether multicollinearity still exist or not**

**vif=pd.DataFrame( )**

**vif[“VIF Values”]= [variance\_inflation\_factor(X.values,i) for i in range(len(X.xolumns))]**

**vif[“Features”]=X.columns**

**vif**

**# It can be found that vif value of other features may be dropped because vif based on interrelationship between independent features. So one may affect other**

**Y.value\_counts( )**

**# if values in the target variable isn’t balanced we will apply resampling techniques like oversampling, undersampling, randomoversampling, SMOTE (synthetic oversampling method method**

**!pip install imblearn #install the library**

**from imblearn.over\_sampling import SMOTE**

**SM=SMOTE( )**

**X1,Y1=SM.fit\_resample(X,Y)**

**Y.value\_counts( )**

**# Steps we have done**

**Imported libraries**

**Non-graphical Analysis**

**graphical Analysis**

**Univariate, bivariate, multivariate Analysis**

**Encode the categorical columns**

**Checked outliers and remove if present**

**Check skewness and remove if it is present**

**Check multicollinearity and remove if it is present**

**Apply feature scaling methods on numerical features**

**Apply resampling methods to balance the classes (in classification problems)**

**Also apply feature selection method**

**Hypothesis testing, feature important method….and so on (optional)**

**Modelling**

**Finding the best random state (for randomly selecting the data), it is the state where accuracy of any model is highest**

**from sklearn.model\_selection import train\_test\_split**

**from sklearn.ensemble import RandomForestClassifier**

**from sklearn.metrics import accuracy\_score**

**maxAccu=0**

**maxRS=0**

**for i in range(1,200):**

**X\_train, X\_test, Y\_train, Y\_test=train\_test\_split(X1,Y1,test\_size=0.3, random\_state=i)**

**RFR= RandomForestClassifier( )**

**RFR.fit(X\_train,Y\_train)**

**pred=RFR.predict(X\_test)**

**acc=accuracy\_score(Y\_test,pred)**

**if acc>maxAcc:**

**maxAccu=acc**

**maxRS=i**

**print(‘Best Accuracy is: “, maxAccu, ‘at random\_state: ‘,maxRS)**

**X\_train, X\_test, Y\_train, Y\_test=train\_test\_split(X1,Y1,test\_size=0.3, random\_state=maxRS)**

**Classification Algorithms**

**from sklearn.ensemble import RandomForestClassifier, ExtraTreesClassifier**

**fromsklearn.linear\_model import LogisticRegression**

**from sklearn.svm import SVC**

**from sklearn.ensemble import GradientBoostingClassifier, AdaBoostClassifier, BaggingClassifier**

**from sklearn.metrics import classification\_report, confusion\_matrix, roc\_curve, accuracy\_score**

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

**RandomForestClassifier**

**RFC= RandomForestClassifier ( )**

**RFC.fit(X\_train,Y\_train)**

**predRFC=RFC.predict(X\_test)**

**print(accuracy\_score(Y\_test,predRFC))**

**print(confusion\_matrix(Y\_test, predRFC))**

**print(classification\_report(Y\_test,predRFC))**

**# Similarly do the model fitting and predicting for all models one by one**

**Cross Validation Score-to validate the true accuracy of the models, if you haven’t divided data in train,test,validate manner**

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

**# to check cv score for every aforementioned model one by one**

**# in classification algorithms, default scoring parameter is accuracy. For regression default scoring parameter is r2\_score**

**# high cv values means more processing time**

**score= cross\_val\_score(RFC,X1,Y1)**

**print(score)**

**print(score.mean( ))**

**print(“Difference between accuracy score and cross validation score is: “, accuracy\_score(Y\_test,predRFC) – score.mean( ))**

**# Model with least difference will be the best model for onward hyperparameter operation**

**Hyperparameter tuning**

**# Used for finding optimum combination of parameters for a model, RandomSearchCV is less accurate and less time consuming**

**from sklearn.model\_selection import GridSearchCV**

**parameters={‘criterion’: [‘gini’,’Entropy’],**

**‘random\_state’: [10,50,1000].**

**‘max\_depth’: [0,10,20],**

**‘n\_jobs’:[-2,-1,1],**

**‘n\_estimators’:[50,100,200,300]}**

**GCV=GridSearchCV(ExtraTreeClassifier( ), parameters, cv=5)**

**GCV.fit(X\_train,Y\_train)**

**GCV.best\_params\_**

**# check best params for any model on sklearn website**

**Final\_model=ExtraTreeClassifier(criterion= ‘entropy’, max\_depth=10, n\_estimators=100, n\_jobs= -2, random\_state=50)**

**Final\_model.fit(X\_train,Y\_train)**

**Pred=Final\_model.predict(X\_test)**

**acc= accuracy\_score(Y\_test,pred)**

**print(acc\*100)**

**plotting ROC and compare AUC for all models used, not valid anymore by sklearn website**

**# plotting for all models used**

**from sklearn import datasets**

**from sklearn import metrics**

**from sklearn import model\_selection**

**from sklearn.metrics import plot\_roc\_curve # this method is deprecated (not used ) from sklearn, visit the official website, you can find alternate methods**

**disp=plot\_roc\_curve(ET, X\_test, Y\_test) # ax =Axes with confusion matrix**

**plot\_roc\_curve(RFC, X\_test, Y\_test, ax=disp.ax\_)**

**plot\_roc\_curve(LR, X\_test, Y\_test, ax=disp.ax\_)**

**plot\_roc\_curve(SVC, X\_test, Y\_test, ax=disp.ax\_)**

**plot\_roc\_curve(ABC, X\_test, Y\_test, ax=disp.ax\_)**

**plot\_roc\_curve(GC, X\_test, Y\_test, ax=disp.ax\_)**

**plot\_roc\_curve(BC, X\_test, Y\_test, ax=disp.ax\_)**

**plt.legend(prop={‘size’:11}, loc=’lower right’)**

**plt.show( )**

**# Correct method for roc**

**# AUC is used to show whether the model is able to distinguish between classes or not**

**from sklearn import metrics**

**fpr, tpr, thresholds=metrics.roc\_curve(Y\_test,pred)**

**roc\_auc=metrics.auc(fpr,tpr)**

**display=metrics.RocCurveDisplay(fpr=fpr, tpr=tpr, roc\_auc=roc\_auc, estimator\_name=Final\_model)**

**display.plot( )**

**Saving the model using.pkl (pickle format)**

**import joblib**

**joblib.dump(Final\_model,”File\_name.pkl”)**

**Predicting using saved model**

**let’s load the model**

**joblib.load(“File\_name.pkl”)**

**prediction=model.predict(X\_test)**

**prediction**

**clear display of prediction**

**a=np.array(Y\_test)**

**df=pd.DataFrame( )**

**df [‘prediction’]=prediction**

**df[‘original’]=a**

**# Use your own methods like function for implementing models**

**# Machine learning is a hit and trail methods, check performance of model repeatedly by making changes**

**Project 2 Discussion (REGRESSION)**

**import pandas as pd**

**import numpy as np**

**import seaborn as sns**

**import matplotlib.pyplot as plt**

**import warnings**

**warnings.filterwarnings(‘ignore’)**

**df=pd.read\_csv(‘filename.csv’)**

**df**

**df.head( )**

**df.shape**

**print(df.columns)**

**print(“Min Date:”,df.Date.min( ))**

**print(“Max Date:”,df[‘Date’].max( ))**

**df.dtypes**

**# whether datatypes are correct or not**

**df.isnull( ).sum( )**

**# datetime is itself a datatype for dates**

**sns.heatmap(df.isnull( ))**

**# Converting the datatype of Date column form Object to datetime**

**df[‘Date’]=pd.to\_datetime(df.Date, format=’%Y-%m-%d’)**

**df.info( )**

**# Setting Date as DataFrame index**

**df.reset\_index(‘Date’, inplace=True)**

**df**

**# checking number of unique values in each column**

**df.nunique( ).to\_frame(‘No. of unique values’)**

**# checking the value counts of each column**

**for in df.columns:**

**print([df[i].value\_counts( ))**

**print(‘\n’)**

**# drop columns if they have same value**

**df.drop(‘feature’, axis=1, inplace=True)**

**df.head( )**

**# checking duplicate values in the dataframe, rows having similar values, if present drop them**

**print(“total duplicated values are: “, df.duplicated( ).sum( ))**

**# To get statistical Summary**

**df.describe( )**

**# describe method can also be used with heatmap**

**plt.figure(figsize=(22,10))**

**sns.heatmap(df.describe( ), annot=True, fmt=’0.2f’, linewidth=0.2,linecolor=’black’,cmap=’spectral’)**

**plt.xlabel(‘Figure’,font\_size=14)**

**plt.ylabel(“Feature\_Name’,font\_size=14)**

**plt.title(“Descriptive\_graph”, font\_size=16)**

**plt.show( )**

**# to check relation between feature and label with best fit line**

**sns.lmplot(x=’feature’,y=’target’, data=df, palette=’colorblind’)**

**# Do it for all features with labels**

**Plt.figure(figsize=(20,25))**

**Plotnumber=1**

**for columns in df:**

**if plotnumber<=17:**

**plt.subplot(5,4,plotnumber)**

**sns.regplot(x=’target’,y=columns, data=df, color=’r’)**

**plt.xlabel(‘target’)**

**plt.ylabel(columns)**

**plotnumber+=1**

**plt.show( )**

**# regplot is same as lmplot**

**# Scatter plot is same as reg but it doesn’t show best fit line**

**plt.figure(figsize=(20,25))**

**Plotnumber=1**

**for columns in df:**

**if plotnumber<=17:**

**plt.subplot(5,4,plotnumber)**

**sns.scatterplot(x=’target’,y=columns, data=df, color=’r’)**

**plt.xlabel(‘target’)**

**plt.ylabel(columns)**

**plotnumber+=1**

**plt.show( )**

**# pairplot is used to find relationship between feature and feature and feature and target**

**For Multivariate Analysis and data distribution**

**sns.pairplot(data=df, palette=”Dark2”)**

**#**

**plt.figure(figsize=(18,6))**

**plt.plot(df.Close, label=’closing\_price’)**

**plt.xlabel(“Time”)**

**plt.ylabel(“Stock Price’)**

**plt.title(‘tesla stock price”)**

**plt.show( )**

**#lineplot is used**

**plt.figure(figsize=(15,8)**

**sns.lineplot(data=df.iloc[:,:,-1]) # selecting all rows and columns except last column**

**plt.ylabel(‘stockvallue’)**

**plt.title(“general trend of all values”)**

**# let’s check the outlier using boxplot**

**plt.figure(figsize= (10, 16), facecolor=’white’)**

**plotnumber=1**

**for col in numerical.col:**

**if plotnumber<=4:**

**ax=plt.subplot(2,2,plotnumber)**

**sns.boxplot(df[col], palette=”set2\_r”)**

**plt.xlabel(col, fontsize=15)**

**plt.yticks(rotation=0, fontsize=10)**

**plotnumber+=1**

**plt.tight\_layout( )**

**#use Z Score or IQR method (samplevalue-mean/standard deviation), we don’t remove outlier from categorical data and target data**

**from scipy.stats import zscore**

**out\_features=df[[‘open’,’High’,’low’]] #DataFrame format**

**z=np.abs(zscore(out\_features))**

**#99.7 data is in range of -3 to +3**

**# so beyond this value threshold=3 not -3 because we have converted it in absolute values above**

**np.where(z>3)**

**# first array shows row number and second array shows column number**

**z.iloc[723,3] # to confirm that value at row 723, and column 3 is greater than 3**

**# Now removing the data above zscore and creating new DataFrame so that original df may preserve and could be used for IQR method**

**df1=df[(z<3).all(axis=1)] # keeping all data where z value is le**

**df1.shape**

**# in pandas axis=1 is column wise operation but in numpy column=1 is row wise**

**# we don’t detect outliers from target and categorical but we remove all rows from them.**

**# Calculate data loss percentage**

**print(“old DataFrame :”, df.shape[0])**

**print(“new DataFrame :”, df1.shape[0])**

**print('data loss percentage:', ((data1.shape[0] - df1.shape[0]) / data1.shape[0]) \* 100)# consistent rows are mandatory because its needed for train\_test\_split**

**# Now we can apply IQR method on original df**

**Q1=out\_feature.quantile(0.25)**

**Q3=out\_feature. quantile(0.75)**

**IQR=Q3 – Q1**

**# if any value less than the min value or any value greater than the max value we will keep it otherwise drop it so below is opposite that is why we are using ~**

**Any means if any datapoint is matching**

**df2=df[~((df <(Q1- 1.5\*IQR)) | (df > (Q3+1.5\*IQR))).any(axis=1)]**

**df2.shape**

**# Again calculate dataloss percentage between df and df2**

**print(“old DataFrame :”, df.shape[0])**

**print(“new DataFrame :”, df2.shape[0])**

**# if data loss of z score is less than select z score over IQR or vice versa**

**# so make df=df1**

**#In case of IQR do df=df2**

**# Now check how data is distributed, it two bells are forming its okay because it can be bimodal type data its not skewness**

**# Then check skewness**

**df.skew**

**# it can give negative value for left skew and positive value for right skew so 0.12 and -0.16 are acceptable skewness where 1 and beyond are high skewness**

**# Removing skewness using cuberoot method of remaining features**

**df[‘column’]=np.cbrt[df[‘column’]**

**df. skew( )**

**# or apply distplot on columns**

**sns.distplot(df[’volume’], color =’m’, kde\_kws= {“shade”:True}, hist=False)**

**# checking correlation**

**df.corr( )**

**# or by heatmap**

**sns.heatmap(df.describe( ), annot=True, fmt=’0.2f’, linewidth=0.2,linecolor=’black’,cmap=’spectral’)**

**plt.xlabel(‘Figure’,font\_size=14)**

**plt.ylabel(“Feature\_Name’,font\_size=14)**

**plt.title(“Descriptive\_graph”, font\_size=16)**

**plt.show( )**

**# we can either use vif or PCA to handle multicollinearity but first seprate target and independent variables**

**X=df1.drop(“close”, axis=1)**

**Y=df1[“close”]**

**from sklearn.preprocessing import StandardScaler**

**scaler=StandardScaler( )**

**x=Pd.DataFrame(scaler.fit\_transform(x), columns=x.columns)**

**x**

**# Checking Variance Inflation Factor (VIF)**

**from statmodels.stats.outliers\_influence import variance\_inflation\_factor**

**vif=pd.DataFrame( )**

**vif[“VIF values”]= [variance\_inflation\_factor (x.values, i) for i in range (len(x.columns))]**

**vif[‘features’]=x.columns**

**vif**

**# if threshold is 10 then any value below 10 isn’t removed but determining threshold depends on data and removing column should be done on the basis that how much its correlated with target and how much percentage of data could be lost upon removing that column**

**# use PCA method or regularization (ridge or LASSO if you are not dropping the column**

**# Modelling**

**Finding best random state**

**from sklearn.model\_selection import train\_test\_split**

**from sklearn.metrics import mean\_absolute\_error**

**from sklearn.metrics import mean\_squared\_error**

**from sklearn.metrics import r2\_score**

**from sklearn.ensemble import GradientBoostingRegressor**

**from sklearn.linear\_model import LinearRegression**

**Creating train test split**

**# finding best state**

**maxAccu=0**

**maxRS=0**

**for i in range (1,200): #More range more time**

**x\_train,x\_test,y\_train,y\_test=train\_test\_split (x,y,test\_size=0.3, random\_state=i)**

**lr=LinearRegression**

**lr.fit(x\_train,y\_train)**

**pred=lr.predict(x\_test)**

**acc=r2\_score(y\_test,pred)**

**if acc>maxAccu:**

**maxAccu=acc**

**maxRS=i**

**print(“Maximum r2 score is :”, maxAccu,”on\_Random\_state”,maxRS)**

**x\_train,x\_test,y\_train,y\_test=train\_test\_split (x,y,test\_size=0.3, random\_state=maxRS)**

**from sklearn.svm import SVR**

**from sklearn.ensemble import ExtraTreeRegressor**

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

**# Apply all models and calculate r2\_socre**

**# default cross val score for classification is accuracy and in regression it is r2\_score**

**# check metrics on sklearn website**

**score= cross\_val\_score(LR,x,y,cv=5,scoring=’r2’)**

**cv=5 means my data is shuffled in 5 equal parts and each part is randomly selected**

**print(score)**

**print(score.mean()) # because score are five values so to make it one**

**print(“Difference between R2 score and CV score is..” (r2\_score(y\_test,pred\_LR) -score.mean())\*100)**

**# difference means the the model previous to CV was giving whatever is the difference more value of r2\_score due to overfitting**

**# Similar way do it for all models cv=5 or little more is ok lesser is not good**

**# CV score can be negative, it happens because each model is different**

**Now model having least difference between r2\_score and cv score will be selected and gridsearchcv will be performed on the same**

**from sklearn.model\_selection import GridSearchCV**

**# applying hyperparameter tuning doesn’t always increase performance, if algorithm is already giving best performance then how it is possible. Further it depends on hyper parameter tuning values**

**param={‘alpha’:[1.0,0.05’,0.4,2],’fit\_intercept’:[True,False],**

**‘solver’:[‘auto’,’svd’,’cholesky’,’lsqr’,’sag’,’saga’,’lbfgs’],**

**‘positive’:[False,True],**

**‘random\_state’:[1,4,10,20]**

**}**

**gscv=GridSearchCV(Ridge( ), param, cv=5)**

**gscv.fit(x\_train,y\_train)**

**gscv.best\_params\_**

**Model=Ridge(write parameters of Best params)**

**Model.fit(x\_train,y\_train)**

**Pred=Model.predict(x\_test)**

**Print(“R2\_Score is: “, r2\_score(y\_test,pred)**

**Print(“Mean Absolute error is: “, mean\_absolute\_error(y\_test,pred)**

**Print(“mean squared error is: “, mean\_squared\_error(y\_test,pred)**

**Print(“Root mean square is: “, np.sqrt(mean\_squared\_error(y\_test,pred)))**

**# Once it is done we will save the model**

**import joblib**

**# either joblib or pickle can be used to save the model**

**# joblib.dump(Model, “TeslaStock.pk1”)**

**import pickle**

**filename=’TeslaStock.pk1’**

**pickle.dump(Model, open(filename,’wb’)) #saved model**

**# open save model and convert file into binary using rb**

**import pickle**

**## open save model and read file using wb**

**loaded\_model=pickle.load(open(’TeslaStock.pk1’,’rb’))**

**result=loaded\_model.score(x\_test,y\_test)**

**print(result\*100)**

**conclusion=pd.DataFrame([loaded\_model.predict(x\_test)[:],y\_test[:]], index=[‘Predicted’,’Original’])**

**conclusion**

**# if training and test data are in two different files**

**Step1: Perform EDA, preprocessing, feature engineering etc**

**Step2: split dataset into features and target variable and then train and test dataset using train test split**

**Find best model, do hyper parameter tuning and save the model**

**For train dataset file you need both step1 and step2**

**For test dataset file**

**You need to do all the same activities (even same methods) that you have done in step 1 with train data only**

**Now load the saved model (model which we saved in step2 for train dataset) and use test dataset file to predict the target variable**

**Web Scraping**

**Beauitiful Soup**

It’s a technique to extract large amount of data from websites

Can be structured in scv format for further Analysis

How does web scraping works?

Code sends a request to server hosting the specific website

Code downloads that page source code

It filters through the page looking for HTML elements we have specified and extracting instructed content

For example we need all titles in h2 tag: code would request the site’s content from its server and download it

Then it go through page’s html code and looks for h2 tag

Upon finding h2 it copy whatever inside h2 tag and save it in the specified format.

**A webpage is made generally by 4 types of files**

**HTML: contains main content of the webpage**

**CSS: styling the webpage**

**JS: brings interactivity to the webpage**

**Image file: JPG/PNG for showing images in the webpage**

**We are interested in extracting data so we use html file**

**BeauitifulSoup practice session**

**!pip install requests**

**# used to send request to webpage server to get the source/html code of the webpage**

**!pip install bs4**

**# used to parse (Analyze and breaking into small parts) the source code and to extract the required information from parsed structure**

**Importing required libraries**

**from bs4 import BeauitifulSoup**

**import requests**

**send get request to webpage server to get the source code**

**page=requests.get(‘weblink’)**

**Page Content**

**Soup=BeauitifulSoup(page.content)**