SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

RAMAPURAM CAMPUS

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

ACADEMIC YEAR 2021-22(EVEN SEMESTER)

18CSE479T - Statistical Machine Learning

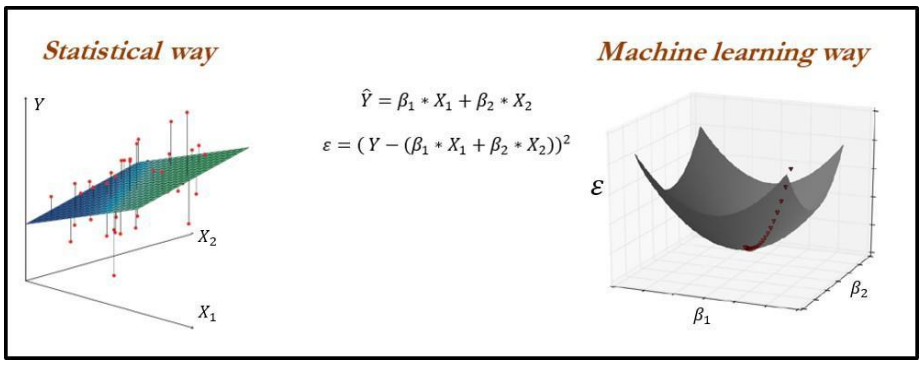
Unit - II

Comparison between regression and machine learning models-Compensating factors in machine learning models- Assumptions of linear regression Steps applied in linear regression modeling- Example of simple linear regression from first principles- Machine learning models - ridge and lasso regression-Example of ridge regression machine learning, Example of lasso regression machine learning model- Logistic Regression Versus Random Forest-Maximum likelihood estimation- Terminology involved in logistic regression- Applying steps in logistic regression modeling - Random forest-Example of random forest using German credit data -Grid search on random forest - Variable importance plot- Comparison of logistic regression with random forest.

I PARALLELISM OF STATISTICS AND MACHINE LEARNING

Comparison between regression and machine learning models

- Linear regression and machine learning models both try to solve the same problem in different ways.
- In the following simple example of a two-variable equation fitting the best possible plane, regression models try to fit the best possible hyperplane by minimizing the errors between the hyperplane and actual observations. However, in machine learning, the same problem has been converted into an optimization problem in which errors are modeled in squared form to minimize errors by altering the weights



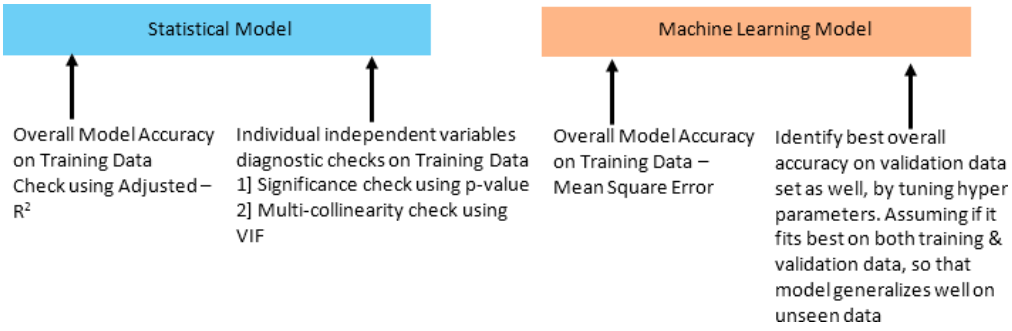
- In statistical modeling, samples are drawn from the population and the model will be fitted on sampled data. However, in machine learning, even small numbers such as 30 observations would be good enough to update the weights at the end of each iteration; in a few cases, such as online learning, the model will be updated with even one observation
- Statistical models are parametric in nature, which means a model will have parameters on which diagnostics are performed to check the validity of the model. Whereas machine learning models are non-parametric, do not have any parameters, or curve assumptions; these models learn by themselves based on provided data and come up with complex and intricate functions rather than predefined function fitting.
- Multi-collinearity checks are required to be performed in statistical modeling. Whereas, in machine learning space, weights automatically get adjusted to compensate the multicollinearity problem

Note : Multicollinearity occurs when two or more independent variables are highly correlated with one another in a regression model. This means that an independent variable can be predicted from another independent variable in a regression model

II Compensating Factors In Machine Learning Model

- Compensating factors in machine learning models to equate statistical diagnostics is explained with the example of a beam being supported by two supports. If one of the supports doesn't exist, the beam will eventually fall down by moving out of balance.
- A similar analogy is applied for comparing statistical modeling and machine learning methodologies here.
- The two-point validation is performed on the statistical modeling methodology on training data using overall model accuracy and individual parameters significance test. Due to the fact that either linear or logistic regression has less variance by shape of the model itself, hence there would be very little chance of it working worse on unseen data. Hence, during deployment, these models do not incur too many deviated results.

- However, in the machine learning space, models have a high degree of flexibility which can change from simple to highly complex.
- On top, statistical diagnostics on individual variables are not performed in machine learning. Hence, it is important to ensure the robustness to avoid overfitting of the models, which will ensure its usability during the implementation phase to ensure correct usage on unseen data.
- In machine learning, data will be split into three parts (train data - 50 percent, validation data - 25 percent, testing data - 25 percent) rather than two parts in statistical methodology.
- Machine learning models should be developed on training data, and its hyperparameters should be tuned based on validation data to ensure the two-point validation equivalence; this way, the robustness of models is ensured without diagnostics performed at an individual variable level



III ASSUMPTIONS OF LINEAR REGRESSION

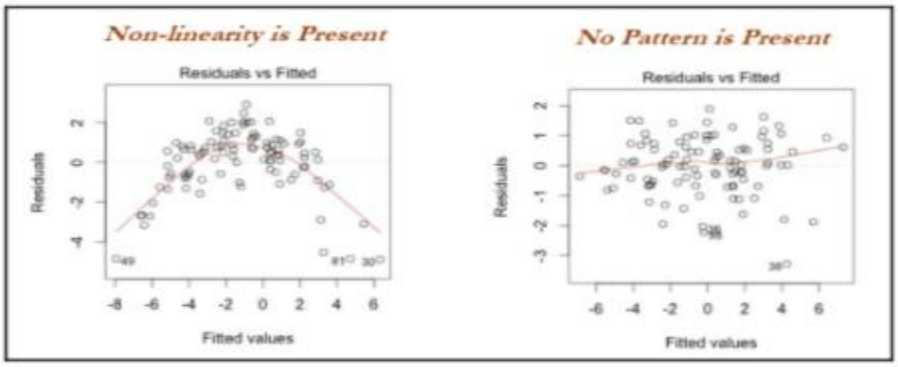
Linear regression has the following assumptions, failing which the linear regression model does not hold true:

- The dependent variable should be a linear combination of independent variables
- No autocorrelation in error terms
- Errors should have zero mean and be normally distributed
- No or little multi-collinearity
- Error terms should be homoscedastic

1.The dependent variable should be a linear combination of independent variables:

Y should be a linear combination of *X* variables. Please note, in the following equation, *X2* has raised to the power of *2*, the equation is still holding the assumption of a linear combination of variables

$$Y = A_0 + (\beta_1 * x_1) + (\beta_2 * x_2^2)$$



In the preceding sample graph, initially, linear regression was applied and the errors seem to have a pattern rather than being pure white noise; in this case, it is simply showing the presence of non-linearity. After increasing the power of the polynomial value, now the errors simply look like white noise.

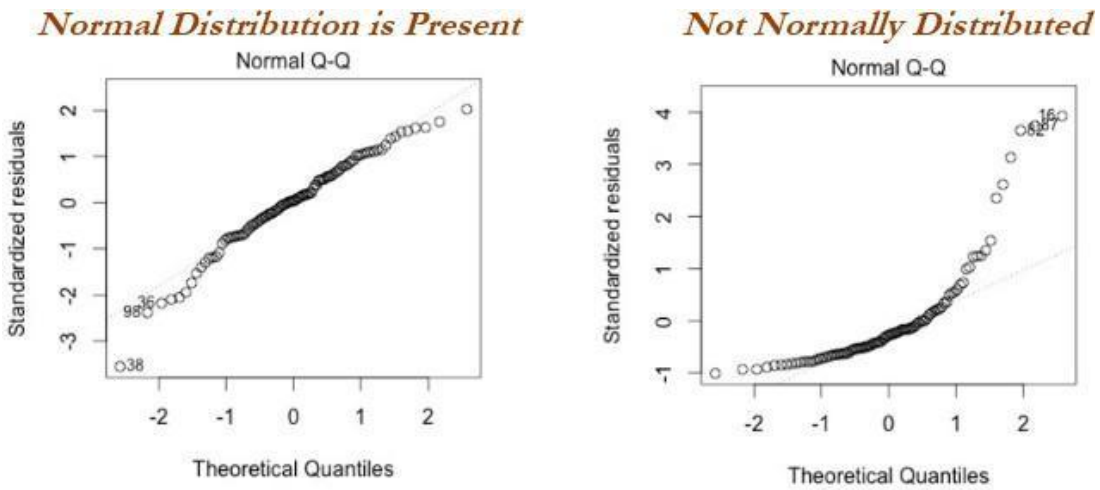
2.No autocorrelation in error terms

Presence of correlation in error terms penalized model accuracy

To diagnose: Look for the Durbin-Watson test. Durbin-Watson's *d* tests the null hypothesis that the residuals are not linearly auto correlated. While *d* can lie between *0* and *4*, if $d \approx 2$ indicates no autocorrelation, $0<d<2$ implies positive autocorrelation, and $2<d<4$ indicates negative autocorrelation

3.Error should have zero mean and be normally distributed

Errors should have zero mean for the model to create an unbiased estimate. Plotting the errors will show the distribution of errors. Whereas, if error terms are not normally distributed, it implies confidence intervals will become too wide or narrow, which leads to difficulty in estimating coefficients based on minimization of least squares:



To diagnose: Look into Q-Q plot and also tests such as Kolmogorov- Smirnov tests will be helpful. By looking into the above Q-Q plot, it is evident that the first chart shows errors are normally distributed, as the residuals do not seem to be deviating much compared with the diagonal-like line, whereas in the right-hand chart, it is clearly showing that errors are not normally distributed; in these scenarios, we need to reevaluate the variables by taking log transformations and so on to make residuals look as they do on the left-hand chart.

4.No or little multi-collinearity:

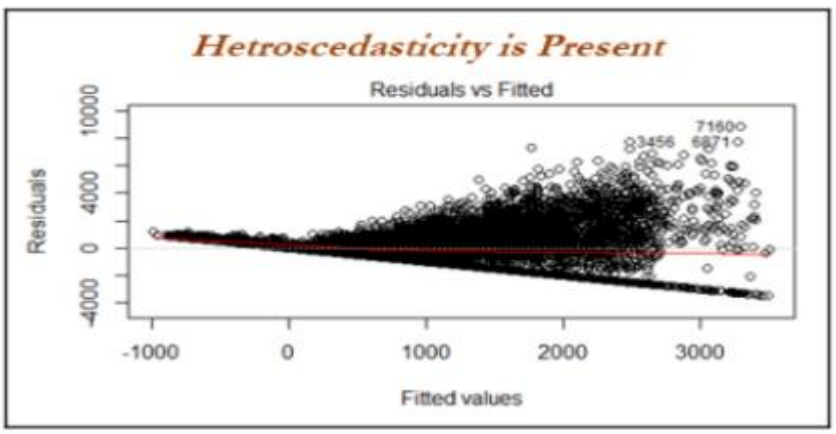
Multi-collinearity is the case in which independent variables are correlated with each other and this situation creates unstable models by inflating the magnitude of coefficients/estimates. It also becomes difficult to determine which variable is contributing to predict the response variable. *VIF* is calculated for each independent variable by calculating the R-squared value with respect to all the other independent variables and tries to eliminate which variable has the highest *VIF* value one by one:

$$VIF = \frac{1}{1 - R^2}$$

To diagnose: Look into scatter plots, run correlation coefficient on all the variables of data. Calculate the **variance inflation factor** (**VIF**). If *VIF <= 4* suggests no multi-collinearity, in banking scenarios, people use *VIF <= 2* also.

5.Errors should be homoscedastic

Errors should have constant variance with respect to the independent variable, which leads to impractically wide or narrow confidence intervals for estimates, which degrades the model's performance. One reason for not holding homoscedasticity is due to the presence of outliers in the data, which drags the model fit toward them with higher weights



To diagnose: Look into the residual versus dependent variables plot; if any pattern of cone or divergence does exist, it indicates the errors do not have constant variance, which impacts its predictions.

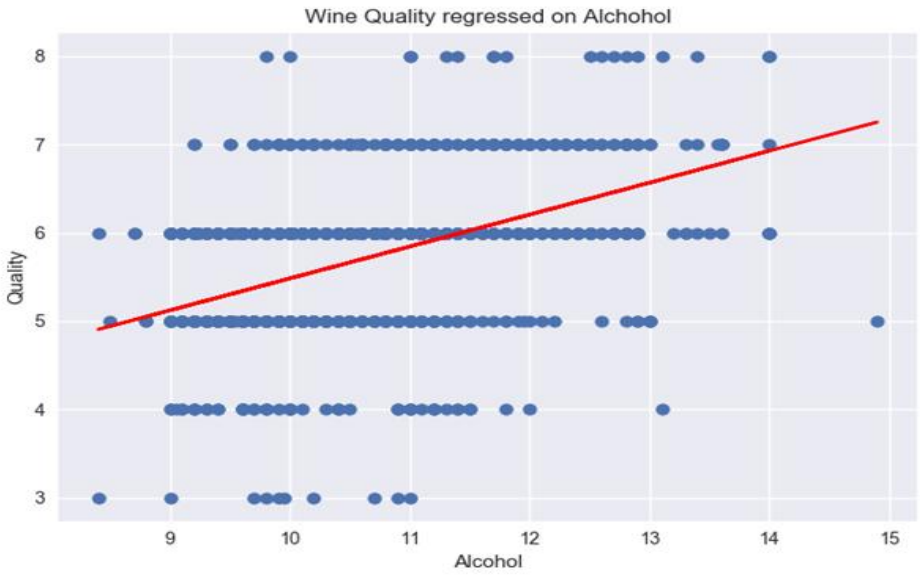
IV STEPS APPLIED IN LINEAR REGRESSION MODELING

The following steps are applied in linear regression modeling in industry:

1. Missing value and outlier treatment
2. Correlation check of independent variables
3. Train and test random classification
4. Fit the model on train data
5. Evaluate model on test data

V EXAMPLE OF SIMPLE LINEAR REGRESSION FROM FIRST PRINCIPLES

- UCI machine learning repository at https://archive.ics.uci.edu/ml/datasets/Wine+Quality.
- Simple linear regression is a straightforward approach for predicting the dependent/response variable *Y* given the independent/predictor variable *X*. It assumes a linear relationship between *X* and *Y*
- $\beta 0$ and $\beta 1$ are two unknown constants which are intercept and slope parameters respectively. Once we determine the constants, we can utilize them for the prediction of the dependent variable
- Residuals are the differences between the *i*th observed response value and the *i*th response value that is predicted from the model. Residual sum of squares is shown. The least squares approach chooses estimates by minimizing errors



$$Y \approx \beta_0 + \beta_1 X\,; \quad \hat{y} = \beta_0 + \beta_1 X$$

$$\hat{y}_i = \widehat{\beta_0} + \widehat{\beta_1} x_i \Rightarrow e_i = y_i - \hat{y}_i$$

$$RSS\ (Residual\ sum\ of\ squares) = e_1^2 + e_2^2 + \cdots + e_n^2$$

$$RSS = (y_1 - \widehat{\beta_0} - \widehat{\beta_1} x_1)^2 + (y_2 - \widehat{\beta_0} - \widehat{\beta_1} x_2)^2 + \cdots + (y_n - \widehat{\beta_0} - \widehat{\beta_1} x_n)^2$$

$$\widehat{\beta_1} = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{n}(x_i - \bar{x})^2} = \frac{cov(x,y)}{var(x)}; \widehat{\beta_0} = \bar{y} - \widehat{\beta_1}\,\bar{x}$$

$$\bar{y} \equiv \frac{1}{n}\sum_{i=1}^{n} y_i \;;\; \bar{x} \equiv \frac{1}{n}\sum_{i=1}^{n} x_i$$

- In order to prove statistically that linear regression is significant, we have to perform hypothesis testing.
- Let's assume we start with the null hypothesis that there is no significant relationship between *X* and *Y*
- Since, if $\beta 1 = 0$, then the model shows no association between both variables ($Y = \beta 0 + \varepsilon$), these are the null hypothesis assumptions;
- In order to prove this assumption right or wrong, we need to determine $\beta 1$ is sufficiently far from *0 so* that we can be confident that $\beta 1$ is nonzero and have a significant relationship between both variables

H_0: *There is no relationship between X and Y*

H_a: *There is relationship between X and Y*

$$H_0: \beta_1 = 0 \; ; \quad H_a: \beta_1 \neq 0$$

- It depends on the distribution of *β1*, which is its mean and standard error (similar to standard deviation).
- In some cases, if the standard error is small, even relatively small values may provide strong evidence that *β1 ≠ 0*, hence there is a relationship between *X* and *Y*.
- In contrast, if *SE(β1)* is large, then *β1* must be large in absolute value in order for us to reject the null hypothesis.

$$t = \frac{\widehat{\beta_1} - 0}{SE(\widehat{\beta_1})}$$

- We usually perform the t test to check how many standard deviations *β1* is away from the value *0*:

$$\bar{y} = \frac{1}{n}\sum_{i=1}^{n} y_i \; ; \; SS_{tot} = \sum_i (y_i - \bar{y})^2 \; ; \; SS_{reg} = \sum_i (f_i - \bar{y})^2 \; ; \; SS_{res} = \sum_i (y_i - f_i)^2 = \sum_i e_i^2$$

$$R^2 \equiv 1 - \frac{SS_{res}}{SS_{tot}}$$

- With this *t* value, we calculate the probability of observing any value equal to |*t*| or larger, assuming *β1 = 0*; this probability is also known as the p-value.
- If *p-value < 0.05*, it signifies that *β1* is significantly far from *0*, hence we can reject the null hypothesis and agree that there exists a strong relationship if *p-value > 0.05*, we accept the null hypothesis and conclude that there is no significant relationship between both variables
- To predict the dependent value and check for the R-squared value;
- if the value is *>= 0.7*, it means the model is good enough to deploy on unseen data
- if it is not such a good value (*<0.6*), we can conclude that this model is not good enough to deploy

VI MACHINE LEARNING MODELS – INTRODUCTION TO RIDGE AND LASSO REGRESSION

- In linear regression, only the **residual sum of squares** (**RSS**) is minimized
- In ridge and lasso regression, a penalty is applied (also known as **shrinkage penalty**) on coefficient values to regularize the coefficients with the tuning parameter λ
- When $\lambda=0$, the penalty has no impact, ridge/lasso produces the same result as linear regression, whereas λ -> ∞ will bring coefficients to zero.

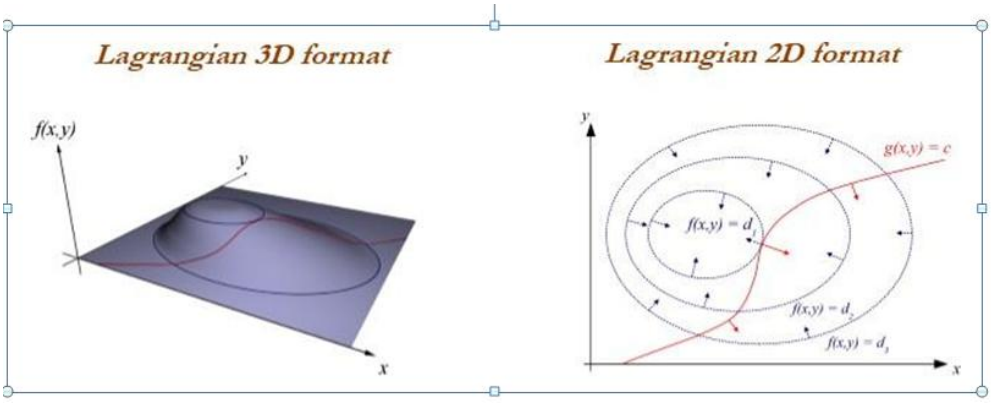
$$RSS\ (\beta) = \sum_{i=1}^{n} (y_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_{ij})^2$$

Lagrangian multipliers

- **Defintion of Lagrangian** : a function that describes the state of a dynamic system in terms of position coordinates and their time derivatives
- The Lagrange multiplier, λ, measures the increase in the objective function (f(x, y) that is obtained through a marginal relaxation in the constraint (an increase in k). For this reason, the Lagrange multiplier is often termed a shadow price.
- The method of Lagrange multipliers in Machine learning is a simple and elegant method of finding the local minima or local maxima of a function subject to equality or inequality constraints. Lagrange multipliers are also called undetermined multipliers.
- The objective is RSS subjected to cost constraint (*s*) of budget.
- For every value of λ, there is an *s* such that will provide the equivalent equations, as shown for the overall objective function with a penalty factor

$$Objective\ function\ to\ minimize\ in\ Ridge\ Regression = RSS(\beta) + \lambda \sum_{j=1}^{p} \beta_j^2$$

$$Objective\ function\ to\ minimize\ in\ Lasso\ Regression = RSS(\beta) + \lambda \sum_{j=1}^{p} |\beta_j|$$



- For any fixed value of λ, ridge regression only fits a single model and the model-fitting procedure can be performed very quickly
- One disadvantage of ridge regression
    - Given a situation where the number of predictors is significantly large, using ridge may provide accuracy, but it includes all the variables, which is not desired in a compact representation of the model;
- But in lasso, it will set the weights of unnecessary variables to zero

VII RIDGE REGRESSION

What is Ridge Regression?

- Ridge regression is a model tuning method that is used to analyse any data that suffers from multicollinearity. This method performs L2 regularization. When the issue of multicollinearity occurs, least-squares are unbiased, and variances are large, this results in predicted values being far away from the actual values.
- The cost function for ridge regression:

  Min(||Y – X(theta)||^2 + λ||theta||^2)

  Lambda is the penalty term. λ given here is denoted by an alpha parameter in the ridge function. So, by changing the values of alpha, we are controlling the penalty term. The higher the values of alpha, the bigger is the penalty and therefore the magnitude of coefficients is reduced.
- It shrinks the parameters. Therefore, it is used to prevent multicollinearity
- It reduces the model complexity by coefficient shrinkage

Lasso Meaning

- The word "LASSO" stands for Least Absolute Shrinkage and Selection Operator. It is a statistical formula for the regularisation of data models and feature selection.

Regularization

- Regularization is an important concept that is used to avoid overfitting of the data, especially when the trained and test data are much varying.
- Regularization is implemented by adding a "penalty" term to the best fit derived from the trained data, to achieve a lesser variance with the tested data and also restricts the influence of predictor variables over the output variable by compressing their coefficients.

Mathematical equation of Lasso Regression

Residual Sum of Squares + λ * (Sum of the absolute value of the magnitude of coefficients)

$$\sum_{i=1}^{n}(y_i - \sum_{j} x_{ij}\beta_j)^2 + \lambda \sum_{j=1}^{p} |\beta_j|$$

Where,

- λ denotes the amount of shrinkage.
- λ = 0 implies all features are considered and it is equivalent to the linear regression where only the residual sum of squares is considered to build a predictive model
- λ = ∞ implies no feature is considered i.e, as λ closes to infinity it eliminates more and more features
- The bias increases with increase in λ
- variance increases with decrease in λ

The key difference is in how they assign penalty to the coefficients:

1. Ridge Regression:
    - Performs L2 regularization, i.e. adds penalty equivalent to square of the magnitude of coefficients
    - Minimization objective = LS Obj + α * (sum of square of coefficients)
2. Lasso Regression:

- Performs L1 regularization, i.e. adds penalty equivalent to absolute value of the magnitude of coefficients
- Minimization objective = LS Obj + α * (sum of absolute value of coefficients)

- Ridge regression is a machine learning model in which we do not perform any statistical diagnostics on the independent variables and just utilize the model to fit on test data and check the accuracy of the fit. Here, we have used the scikit-learn package

```
>>> from sklearn.linear_model import Ridge

>>> wine_quality = pd.read_csv("winequality-red.csv",sep=';')
>>> wine_quality.rename(columns=lambda x: x.replace(" ", "_"),
inplace=True)


>>> all_colnms = ['fixed_acidity', 'volatile_acidity', 'citric_acid',
'residual_sugar', 'chlorides', 'free_sulfur_dioxide',
'total_sulfur_dioxide', 'density', 'pH', 'sulphates', 'alcohol']

>>> pdx = wine_quality[all_colnms]
>>> pdy = wine_quality["quality"]

>>> x_train,x_test,y_train,y_test = train_test_split(pdx,pdy,train_size =
0.7,random_state=42)
```

A simple version of a grid search from scratch is described as follows, in which various values of alphas are to be tested in a grid search to test the model's fitness:

```
>>> alphas = [1e-4,1e-3,1e-2,0.1,0.5,1.0,5.0,10.0]
```

Initial values of R-squared are set to 0 in order to keep track of its updated value and to print whenever the new value is greater than the existing value:

```
>>> initrsq = 0

>>> print ("\nRidge Regression: Best Parameters\n")
>>> for alph in alphas:
...       ridge_reg = Ridge(alpha=alph)
...       ridge_reg.fit(x_train,y_train)    0
...       tr_rsqrd = ridge_reg.score(x_train,y_train)
...       ts_rsqrd = ridge_reg.score(x_test,y_test)
```

The following code always keeps track of the test R-squared value and prints if the new value is greater than the existing best value:

```
>>>      if ts_rsqrd > initrsq:
...           print ("Lambda: ",alph,"Train R-Squared
value:",round(tr_rsqrd,5),"Test R-squared value:",round(ts_rsqrd,5))
...           initrsq = ts_rsqrd
```

This is shown in the following screenshot:

```
Ridge Regression: Best Parameters

Lambda:  0.0001 Train R-Squared value: 0.3612 Test R-squared value: 0.35135
```

Lasso regression is a close cousin of ridge regression, in which absolute values of coefficients are minimized rather than the square of values

The following implementation is similar to ridge regression apart from penalty application on mod/absolute value of coefficients:

```
>>> from sklearn.linear_model  import Lasso

>>> alphas = [1e-4,1e-3,1e-2,0.1,0.5,1.0,5.0,10.0]
>>> initrsq = 0
>>> print ("\nLasso Regression: Best Parameters\n")

>>> for alph in alphas:
...       lasso_reg = Lasso(alpha=alph)
...       lasso_reg.fit(x_train,y_train)
...       tr_rsqrd = lasso_reg.score(x_train,y_train)
...       ts_rsqrd = lasso_reg.score(x_test,y_test)

...       if ts_rsqrd > initrsq:
...           print ("Lambda: ",alph,"Train R-Squared
value:",round(tr_rsqrd,5),"Test  R-squared value:",round(ts_rsqrd,5))
...           initrsq = ts_rsqrd
```

```
Lasso Regression: Best Parameters

Lambda:  0.0001 Train R-Squared value: 0.36101 Test R-squared value: 0.35057
```

```
>>> ridge_reg = Ridge(alpha=0.001)
>>> ridge_reg.fit(x_train,y_train)
>>> print ("\nRidge Regression coefficient values of Alpha = 0.001\n")
>>> for i in range(11):
...     print (all_colnms[i],": ",ridge_reg.coef_[i])

>>> lasso_reg = Lasso(alpha=0.001)
>>> lasso_reg.fit(x_train,y_train)
>>> print ("\nLasso Regression coefficient values of Alpha = 0.001\n")
>>> for i in range(11):
...      print (all_colnms[i],": ",lasso_reg.coef_[i])
```

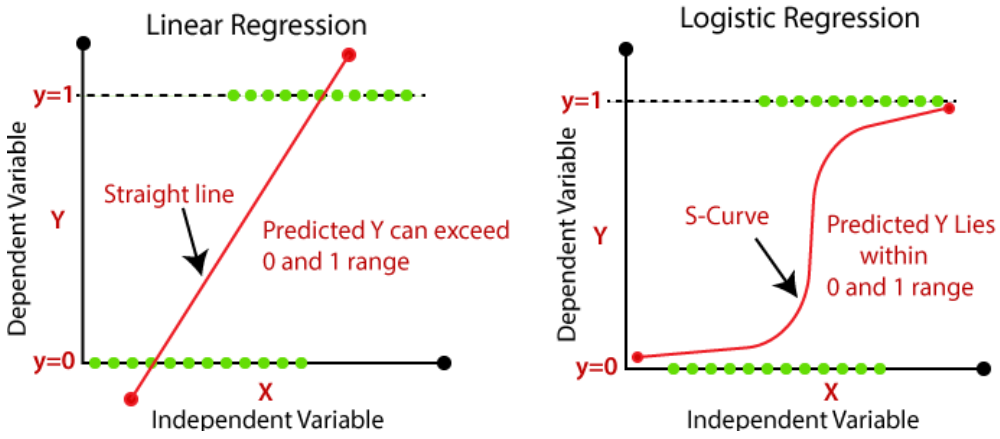
The following results show the coefficient values of both methods; the coefficient of density has been set to 0 in lasso regression, whereas the density value is -5.5672 in ridge regression; also, none of the coefficients in ridge regression are zero value

```
Ridge Regression coeficcient values of Alpha = 0.001

fixed_acidity :  0.015506587508
volatile_acidity :  -1.10509823549
citric_acid :  -0.248798655324
residual_sugar :  0.00401889539284
chlorides :  -1.68438396209
free_sulfur_dioxide :  0.00463690171096
total_sulfur_dioxide :  -0.00328376790411
density :  -5.5672717468
pH :  -0.362480017204
sulphates :  0.800919122803
alcohol :  0.299918244295
```

```
Lasso Regression coeficcient values of Alpha = 0.001

fixed_acidity :  0.0141495463691
volatile_acidity :  -1.09062360905
citric_acid :  -0.185295150047
residual_sugar :  -0.000136610246787
chlorides :  -1.05877579704
free_sulfur_dioxide :  0.00483164817515
total_sulfur_dioxide :  -0.00326722885596
density :  -0.0
pH :  -0.256901925871
sulphates :  0.694487540316
alcohol :  0.307756149124
```

VIII LOGISTIC REGRESSION

- Logistic regression is a process of modeling the probability of a discrete outcome given an input variable.
- It is used in statistical software to understand the relationship between the dependent variable and one or more independent variables by estimating probabilities using a logistic regression equation. This type of analysis can help you predict the likelihood of an event happening or a choice being made



Maximum Likelihood

- Maximum likelihood estimation is a method of estimating the parameters of a model given observations, by finding the parameter values that maximize the likelihood of making the observations, this means finding parameters that maximize the probability *p* of event *1* and *(1-p)* of non-event *0*

  probability (event + non-event) = 1

Example: Sample *(0, 1, 0, 0, 1, 0)* is drawn from binomial distribution. What is the maximum likelihood estimate of μ?

Solution: Given the fact that for binomial distribution $P(X=1) = \mu$ and $P(X=0) = 1- \mu$ where μ is the parameter:

$$\begin{aligned} Likelihood\ of\ \mu = L(\mu) \\ &= P(x=0) * P(x=1) * P(x=0) * P(x=0) * P(x=1) * P(x=0) \\ &= (1-\mu) * \mu * (1-\mu) * (1-\mu) * \mu * (1-\mu) \\ &L(\mu) = (1-\mu)^4 \mu^2 \end{aligned}$$

Here, *log* is applied to both sides of the equation for mathematical convenience; also, maximizing likelihood is the same as the maximizing log of likelihood:

$$\log(L(\mu)) = \log((1-\mu)^4 \mu^2)$$
$$\log(L(\mu)) = 4 * \log(1-\mu) + 2 * \log(\mu)$$

Determining the maximum value of μ by equating derivative to zero:

$$\frac{\partial}{\partial \mu} \log(L(\mu)) = 0$$
$$\Rightarrow 4 * \frac{1}{1-\mu} * (-1) + 2 * \frac{1}{\mu} = 0$$
$$-4 * \mu + 2 * (1-\mu) = 0$$
$$\mu = \frac{1}{3}$$

However, we need to do double differentiation to determine the saddle point obtained from equating derivative to zero is maximum or minimum. If the μ value is maximum; double differentiation of *log(L(μ))* should be a negative value:

$$\frac{\partial^2}{\partial \mu^2} \log(L(\mu)) = -4 * \frac{1}{(1-\mu)^2} - \frac{2}{\mu^2}$$

Even without substitution of μ value in double differentiation, we can determine that it is a negative value, as denominator values are squared and it has a negative sign against both terms. Nonetheless, we are substituting and the value is:

$$\frac{\partial^2}{\partial \mu^2}\log(L(\mu)) = -4 * \frac{1}{\left(1-\frac{1}{3}\right)^2} - \frac{2}{\left(\frac{1}{3}\right)^2} = -9 - 18 = -27$$

Hence it has been proven that at value $\mu = 1/3$, it is maximizing the likelihood. If we substitute the value in the log likelihood function, we will obtain:

$$L(\mu) = \left(1-\frac{1}{3}\right)^4 \frac{1}{3}^2 = 0.021948$$
$$\ln(L(\mu)) = \ln(0.021948) = -3.819$$
$$-2\ln(L(\mu)) = -2 * -3.819 = 7.63$$

The reason behind calculating *-2*ln(L)* is to replicate the metric calculated in proper logistic regression. In fact:

*AIC = -2*ln(L) + 2*k*

So, Logistic regression tries to find the parameters by maximizing the likelihood with respect to individual parameters.

IX TERMINOLOGY INVOLVED IN LOGISTIC REGRESSION

1.Information value (IV)

- This is very useful in the preliminary filtering of variables prior to including them in the model.
- IV is mainly used by industry for eliminating major variables in the first step prior to fitting the model, as the number of variables present in the final model would be about 10.
- Hence, initial processing is needed to reduce variables from 400+ in number or so.

$$Information\ Value = \ln\left(\frac{\%good}{\%bad}\right) * (\%good - \%bad)$$

$$Weight\ Of\ Evidence = \ln\left(\frac{\%good}{\%bad}\right)$$

Information Value	Predictive Power
<0.02	Useless for prediction
0.02 to 0.1	Weak predictor
0.1 to 0.3	Medium predictor
0.3 to 0.5	Strong predictor
> 0.5	Suspicious or too good predictor

Example: In the following table, continuous variable (price) has been broken down into deciles (10 bins) based on price range and the counted number of events and non-events in that bin, and the information value has been calculated for all the segments and added together. We got the total value as *0.0356*, meaning it is a weak predictor to classify events.

Range	Bin Number	Events	Non Events	%of Events [E] (Events/Total Events)	% of Non-Events [NE] (Non Events/Total Non Events)	[E] -[NE]	WOE = ln(E/NE)	IV = WOE * (E-NE)
0-50	1	40	394	5%	5%	0.003	0.062	0.0002
51-100	2	68	900	9%	12%	-0.024	-0.234	0.0056
101-150	3	78	984	10%	13%	-0.021	-0.186	0.0040
151-200	4	102	1194	14%	15%	-0.016	-0.111	0.0018
201-250	5	108	1218	15%	16%	-0.011	-0.074	0.0008
251-300	6	110	1164	15%	15%	-0.001	-0.010	0.0000
301-350	7	82	772	11%	10%	0.011	0.107	0.0012
351-400	8	46	330	6%	4%	0.019	0.379	0.0074
401-450	9	42	368	6%	5%	0.009	0.179	0.0017
>451	10	68	470	9%	6%	0.031	0.416	0.0129
	Total	**744**	**7794**				**I.V.Total**	**0.0356**

2.Akaike information criteria (AIC):

- The Akaike information criterion (AIC) is a mathematical method for evaluating how well a model fits the data it was generated from. In statistics, AIC is used to compare different possible models and determine which one is the best fit for the data.
- This measures the relative quality of a statistical model for a given set of data.
- During a comparison between two models, the model with less AIC is preferred over higher value

$$AIC = -2*ln(L) + 2*k$$

L = Maximum value of Likelihood (log transformation applied for mathematical convenience)

k = Number of variables in the model

3. **Receiver operating characteristic (ROC) curve**:

- This is a graphical plot that illustrates the performance of a binary classifier as its discriminant threshold is varied.
- The curve is created by plotting **true positive rate** (**TPR**) against **false positive rate** (**FPR**) at various threshold values.
- A threshold is a real value between *0* and *1*, used to convert the predicted probability of output into class.
- Ideally, the threshold should be set in a way that trade-offs value between both categories and produces higher overall accuracy
- *Optimum threshold = Threshold where maximum (sensitivity + specificity) is possible*

4. ***Confuion Matrix***

	Predicted: YES	Predicted: NO
Actual: YES	TP (True Positive)	FN (False Negative)
Actual: NO	FP (False Positive)	TN (True Negative)

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN}$$

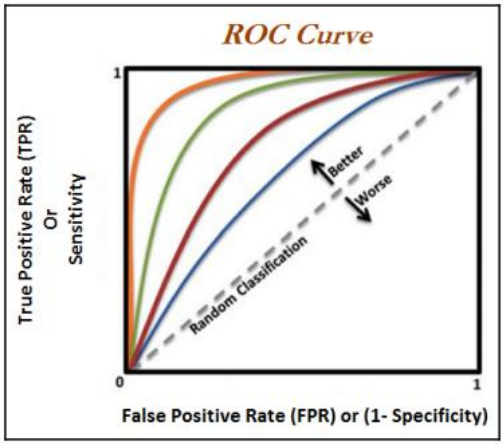
$$Precision = \frac{TP}{TP + FP}$$

$$Recall\ (Sensitivity\ or\ TPR) = \frac{TP}{TP + FN}$$

$$f1\ score = \frac{2 * Precision * Recall}{Precision + Recall}$$

$$FPR\ (1 - Specificity) = \frac{FP}{FP + TN}$$

$$Specificity\ (TNR) = \frac{TN}{FP + TN}$$



5. **Rank ordering**:
    - After sorting observations in descending order by predicted probabilities, deciles are created (10 equal bins with 10 percent of total observations in each bin).
    - By adding up the number of events in each decile, we will get aggregated events for each decile and this number should be in decreasing order, else it will be in serious violation of logistic regression methodology
6. **Concordance/c-statistic**:
    - The C-statistic (sometimes called the "concordance" statistic or C-index) is a measure of goodness of fit for binary outcomes in a logistic regression model.
    - This is a measure of quality of fit for a binary outcome in a logistic regression model.
    - It is a proportion of pairs in which the predicted event probability is higher for the actual event than non-event
    - In the following table, both actual and predicted values are shown with a sample of seven rows. Actual is the true category, either default or not; whereas predicted is predicted probabilities from the logistic regression model. Calculate the concordance value

Actual	Predicted
1	0.92
0	0.34
0	0.12
1	0.4
1	0.64
0	0.82
1	0.84

Actual	Predicted
1	0.92
1	0.4
1	0.64
1	0.84

Actual	Predicted
0	0.34
0	0.12
0	0.82

- For calculating concordance, we need to split the table into two (each table with actual values as *1* and *0*) and apply the Cartesian product of each row from both tables to form pairs
- The complete Cartesian product has been calculated and has classified the pair as a concordant pair whenever the predicted probability for *1* category is higher than the predicted probability for *0* category

Actual	Predicted	Actual	Predicted	Concordant pair	Discordant pair
1	0.92	0	0.34	✓	
1	0.92	0	0.12	✓	
1	0.92	0	0.82	✓	
1	0.4	0	0.34	✓	
1	0.4	0	0.12	✓	
1	0.4	0	0.82		✓
1	0.64	0	0.34	✓	
1	0.64	0	0.12	✓	
1	0.64	0	0.82		✓
1	0.84	0	0.34	✓	
1	0.84	0	0.12	✓	
1	0.84	0	0.82	✓	

$$\%\ concordant\ pairs = \frac{Number\ of\ Concordant\ pairs}{Total\ pairs} = \frac{10}{12} = 83.3\%$$

$$\%\ discordant\ pairs = \frac{Number\ of\ Discordant\ pairs}{Total\ pairs} = \frac{2}{12} = 16.67\%$$

$$\%\ tied\ pairs = \frac{Number\ of\ Tied\ pairs}{Total\ pairs} = \frac{0}{12} = 0\%$$

$$c\ statistic\ or\ c\ index = 0.5 + \frac{(\%Concordant\ pairs - \%Discordant\ pairs)}{2}$$
$$= 0.5 + \frac{(0.833 - 0.1667)}{2} = 0.83315 = 83.315\ \%$$

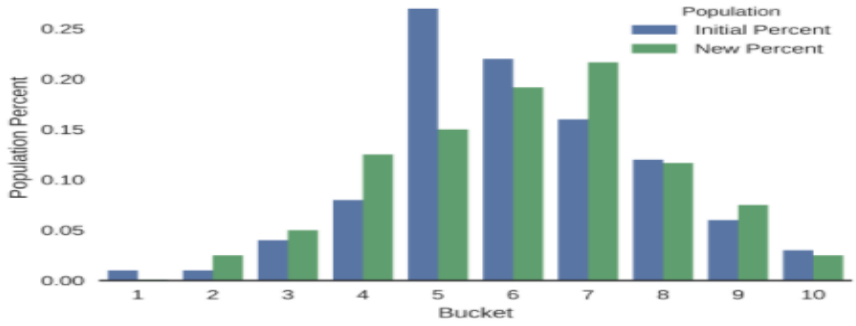
$$Somers'D = (\%\ Concordant\ pairs - \%\ Discordant\ pairs) = (0.833 - 0.1667)$$
$$= 0.663$$

- **C-statistic**: This is 0.83315 percent or 83.315 percent, and any value greater than 0.7 percent or 70 percent is considered a good model to use for practical purposes

7. **Divergence**: The distance between the average score of default accounts and the average score of non-default accounts. The greater the distance, the more effective the scoring system is at segregating good and bad observations.
8. **K-S statistic**: This is the maximum distance between two population distributions. It helps with discriminating default accounts from non-default accounts
9. **Population stability index (PSI)**:

- This is the metric used to check that drift in the current population on which the credit scoring model will be used is the same as the population with respective to development time:
- *PSI <= 0.1*: This states no change in characteristics of the current population with respect to the development population
- *0.1 < PSI <= 0.25*: This signifies some change has taken place and warns for attention, but can still be used
- *PSI >0.25*: This indicates a large shift in the score distribution of the current population compared with development time
- To calculate the PSI we first divide the initial population range into 10 buckets (an arbitrary number I chose), and count the number of values in each of those buckets for the initial and new populations, and then divide those by the total values in each

population to get the percents in each bucket. As expected, plotting the percents ends up looking like a discretized version of the original chart:



From here, we perform the actual PSI calculation for each bucket, and them sum them all up to get the overall PSI values for the distributions.

PSI Formula:

$$PSI = \sum \left((Actual\% - Expected\%) \times ln(\frac{Actual\%}{Expected\%}) \right)$$

Breakpoint Value	Bucket	Initial Count	New Count	Initial Percent	New Percent	PSI
-2.330642	1	1	0	0.01	0.001000	0.020723
-1.801596	2	1	3	0.01	0.025000	0.013744
-1.272550	3	4	6	0.04	0.050000	0.002231
-0.743504	4	8	15	0.08	0.125000	0.020083
-0.214458	5	27	18	0.27	0.150000	0.070534
0.314588	6	22	23	0.22	0.191667	0.003906
0.843633	7	16	26	0.16	0.216667	0.017181
1.372679	8	12	14	0.12	0.116667	0.000094
1.901725	9	6	9	0.06	0.075000	0.003347
2.430771	10	3	3	0.03	0.025000	0.000912

Interpretation:

We get a final PSI value of **0.153**, which indicates that there's a chance our population is shifting, and we may want to monitor it going forwards. Of course, this is just one way of calculating PSI by using equal size binning of 10 buckets. If we keep the 10 buckets but change our binning strategy to quantile bins, we end up with a different percent distribution and an overall lower estimate of **0.129**.

X APPLYING STEPS IN LOGISTIC REGRESSION MODELING

The following steps are applied in linear regression modeling in industry:

1. Exclusion criteria and good-bad definition finalization
2. Initial data preparation and univariate analysis
3. Derived/dummy variable creation
4. Fine classing and coarse classing

5. Fitting the logistic model on the training data
6. Evaluating the model on test data

XI RANDOM FOREST

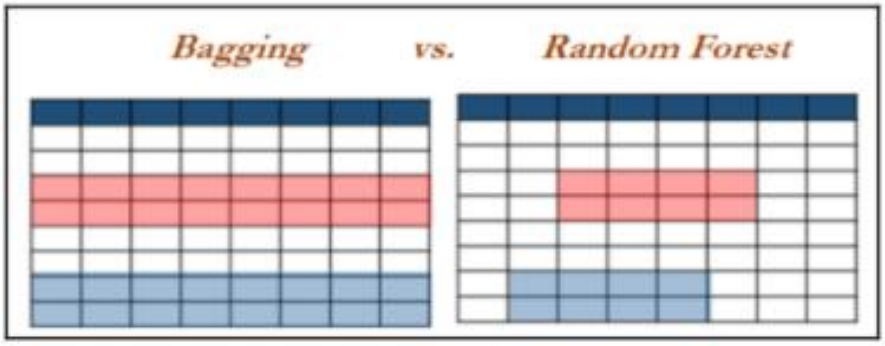
- Random Forest is a popular machine learning algorithm that belongs to the supervised learning technique.
- It can be used for both Classification and Regression problems in ML.
- It is based on the concept of ensemble learning, which is a process of combining multiple classifiers to solve a complex problem and to improve the performance of the model
- Random Forest is a classifier that contains a number of decision trees on various subsets of the given dataset and takes the average to improve the predictive accuracy of that dataset.
- Instead of relying on one decision tree, the random forest takes the prediction from each tree and based on the majority votes of predictions, and it predicts the final output.
- The greater number of trees in the forest leads to higher accuracy and prevents the problem of overfitting.
- RF focuses on sampling both observations and variables of training data to develop independent decision trees and take majority voting for classification and averaging for regression problems respectively
- In contrast, bagging samples only observations at random and selects all columns that have the deficiency of representing significant variables at root for all decision trees.
- This way makes trees that are dependent on each other, for which accuracy will be penalized.
- The following are a few rules of thumb when selecting sub-samples from observations using random forest

About 2/3 of observations in training data for each individual tree

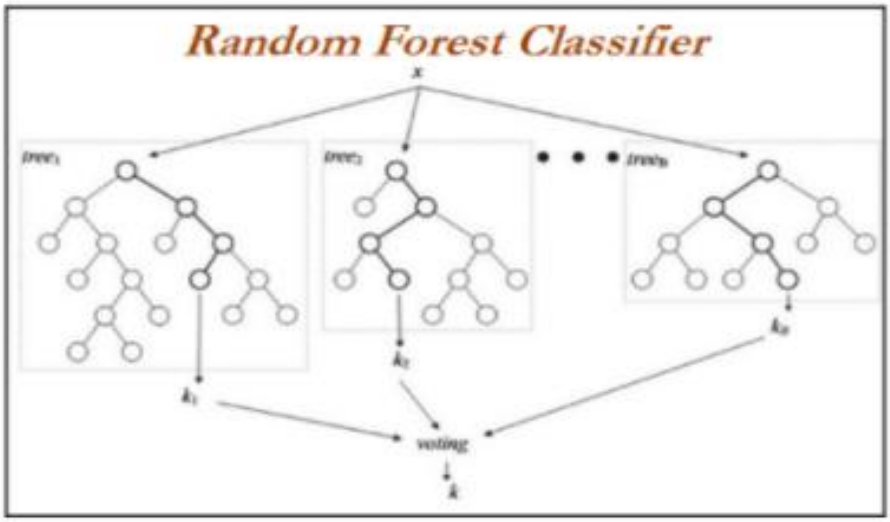
Select columns sqrt(p) -> For classification problem if p is total columns in training data

p/3 -> for regression problem if p is number of columns

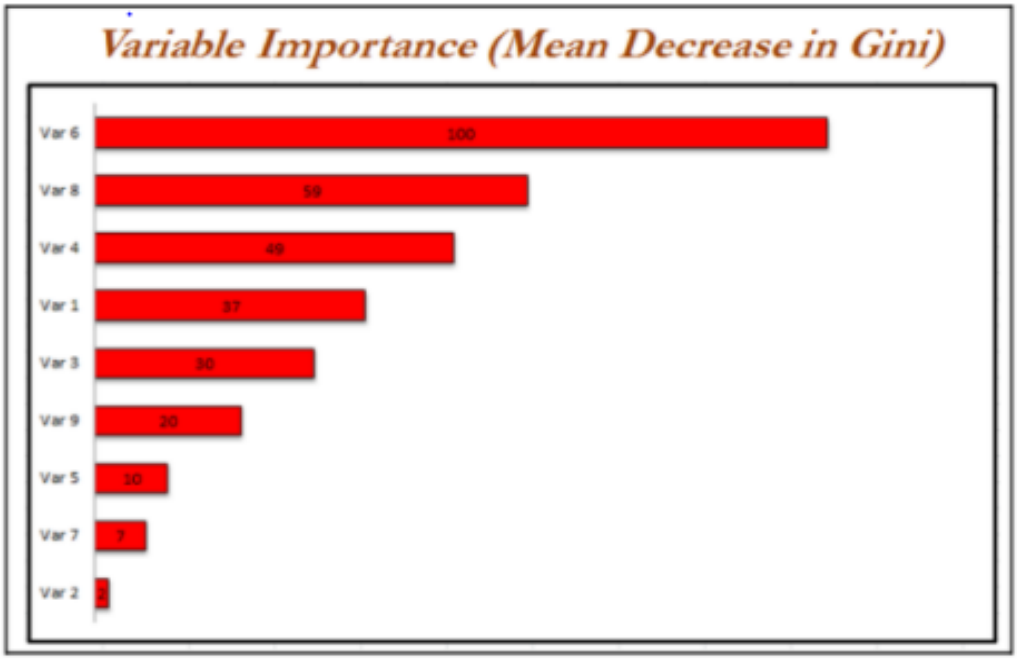
In the following diagram, two samples were shown with blue and pink colors, where, in the bagging scenario, a few observations and all columns are selected. Whereas, in random forest, a few observations and columns are selected to create uncorrelated individual trees.



In the following diagram, a sample idea shows how RF classifier works. Each tree has grown separately, and the depth of each tree varies as per the selected sample, but in the end, voting is performed to determine the final class.



Due to the ensemble of decision trees, RF suffered interpretability and could not determine the significance of each variable; only variable importance could be provided instead. In the following graph, a sample of variable performance has been provided, consisting of a mean decrease in *Gini*:



XII EXAMPLE OF RANDOM FOREST USING GERMAN CREDIT DATA

- The same German credit data is being utilized to illustrate the random forest model
- A very significant difference anyone can observe compared with logistic regression is that effort applied on data preprocessing drastically decreases.
- In RF, we have not removed variables one by one from analysis based on significance and VIF values, as significance tests are not applicable for ML models. However five-fold cross validation has been performed on training data to ensure the model's robustness
- In RF we have not removed the extra dummy variable from the analysis, as the latter automatically takes care of multi-collinearity.
- Random forest requires much less human effort and intervention to train the model.

Random forest applied on German credit data:

```
>>> import pandas as pd
>>> from sklearn.ensemble import RandomForestClassifier

>>> credit_data = pd.read_csv("credit_data.csv")
>>> credit_data['class'] = credit_data['class']-1
```

The creation of dummy variables step is similar to the logistic regression preprocessing step:

```
>>> dummy_stseca =
pd.get_dummies(credit_data['Status_of_existing_checking_account'],
prefix='status_exs_accnt')
>>> dummy_ch = pd.get_dummies(credit_data['Credit_history'],
prefix='cred_hist')
>>> dummy_purpose = pd.get_dummies(credit_data['Purpose'],
prefix='purpose')
>>> dummy_savacc = pd.get_dummies(credit_data['Savings_Account'],
prefix='sav_acc')
>>> dummy_presc = pd.get_dummies(credit_data['Present_Employment_since'],
prefix='pre_emp_snc')
>>> dummy_perssx = pd.get_dummies(credit_data['Personal_status_and_sex'],
prefix='per_stat_sx')
>>> dummy_othdts = pd.get_dummies(credit_data['Other_debtors'],
prefix='oth_debtors')
>>> dummy_property = pd.get_dummies(credit_data['Property'],
prefix='property')
>>> dummy_othinstpln =
pd.get_dummies(credit_data['Other_installment_plans'],
prefix='oth_inst_pln')
>>> dummy_housing = pd.get_dummies(credit_data['Housing'],
prefix='housing')
>>> dummy_job = pd.get_dummies(credit_data['Job'], prefix='job')
>>> dummy_telephn = pd.get_dummies(credit_data['Telephone'],
prefix='telephn')
>>> dummy_forgnwrkr = pd.get_dummies(credit_data['Foreign_worker'],
prefix='forgn_wrkr')
>>> continuous_columns = ['Duration_in_month', 'Credit_amount',
'Installment_rate_in_percentage_of_disposable_income',
'Present_residence_since','Age_in_years','Number_of_existing_credits_at_thi
s_bank',
'Number_of_People_being_liable_to_provide_maintenance_for']
>>> credit_continuous = credit_data[continuous_columns]
```

In the following example, data has been split 70-30. The reason is due to the fact that we would be performing five-fold cross-validation in grid search during training, which produces a similar effect of splitting the data into 50-25-25 of train, validation, and test datasets respectively.

```
>>> x_train,x_test,y_train,y_test = train_test_split( credit_data_new.drop(
['class'],axis=1),credit_data_new['class'],train_size =
0.7,random_state=42)
```

The random forest ML model is applied with assumed hyperparameter values, as follows:

- Number of trees is 1000
- Criterion of slitting is gini
- Maximum depth each decision tree can grow is 100
- Minimum observations required at each not to be eligible for splitting is 3
- Minimum number of observations in tree node should be 2

However, optimum parameter values needs to be tuned using grid search:

```
>>> rf_fit = RandomForestClassifier( n_estimators=1000, criterion="gini",
max_depth=100, min_samples_split=3,min_samples_leaf=2)
>>> rf_fit.fit(x_train,y_train)

>>> print ("\nRandom Forest -Train Confusion Matrix\n\n",
pd.crosstab(y_train, rf_fit.predict( x_train),rownames =
["Actuall"],colnames = ["Predicted"]))
>>> print ("\n Random Forest - Train accuracy",round(accuracy_score(
y_train, rf_fit.predict(x_train)),3))

>>> print ("\nRandom Forest - Test Confusion
Matrix\n\n",pd.crosstab(y_test, rf_fit.predict(x_test),rownames =
["Actuall"],colnames = ["Predicted"]))
>>> print ("\nRandom Forest - Test accuracy",round(accuracy_score(y_test,
rf_fit.predict(x_test)),3))
```

The test accuracy produced from random forest is 0.855

```
Random Forest - Train Confusion Matrix

Predicted     0     1
Actuall
0           501     0
1            18   185

Random Forest - Train accuracy 0.974


Random Forest - Test Confusion Matrix

Predicted     0    1
Actuall
0           210    1
1            43   49

Random Forest - Test accuracy 0.855
```

XIII GRID SEARCH ON RANDOM FOREST

Grid search has been performed by changing various hyperparameters with the following settings. However, readers are encouraged to try other parameters to explore further in this space.

Number of trees is (1000,2000,3000)

Maximum depth is (100,200,300)

Minimum samples per split are (2,3)

Minimum samples in leaf node are (1,2)

Import `Pipeline` as follows:

```
>>> from sklearn.pipeline import Pipeline
>>> from sklearn.model_selection import train_test_split,GridSearchCV
```

The `Pipeline` function creates the combinations which will be applied one by one sequentially to determine the best possible combination:

```
>>> pipeline = Pipeline([
('clf',RandomForestClassifier(criterion='gini'))])
>>> parameters = {
  ...     'clf__n_estimators':(1000,2000,3000),
  ...     'clf__max_depth':(100,200,300),
  ...     'clf__min_samples_split':(2,3),
  ...     'clf__min_samples_leaf':(1,2) }
```

In the following, grid search utilizes cross-validation of five to ensure robustness in the model, which is the ML way of creating two-point validation of the model:

```
>>> grid_search = GridSearchCV(pipeline,parameters,n_jobs=-1, cv=5,
verbose=1, ... scoring='accuracy')
>>> grid_search.fit(x_train,y_train)

>>> print ('Best Training score: %0.3f' % grid_search.best_score_)
>>> print ('Best parameters set:')
>>> best_parameters = grid_search.best_estimator_.get_params()
>>> for param_name in sorted(parameters.keys()):
  ...     print ('\t%s: %r' % (param_name, best_parameters[param_name]))


>>> predictions = grid_search.predict(x_test)

>>> print ("Testing accuracy:",round(accuracy_score(y_test,
predictions),4))
>>> print ("\nComplete report of Testing
data\n",classification_report(y_test, ... predictions))

>>> print ("\n\nRandom Forest Grid Search- Test Confusion Matrix\n\n",
pd.crosstab(y_test, predictions,rownames = ["Actuall"],colnames =
["Predicted"]))
```

```
Fitting 5 folds for each of 36 candidates, totalling 180 fits
[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed:   47.7s
[Parallel(n_jobs=-1)]: Done 180 out of 180 | elapsed:  4.0min finished
Best Training score: 0.820
Best parameters set:
        clf__max_depth: 300
        clf__min_samples_leaf: 1
        clf__min_samples_split: 3
        clf__n_estimators: 1000
Testing accuracy: 0.8911

Complete report of Testing data
             precision    recall  f1-score   support

          0       0.87      1.00      0.93       211
          1       0.98      0.65      0.78        92

avg / total       0.90      0.89      0.88       303
```

```
Random Forest Grid Search- Test Confusion Matrix

Predicted    0   1
Actuall
0          210   1
1           32  60
```

XIV VARIABLE IMPORTANCE PLOT

- Variable importance plot provides a list of the most significant variables in descending order by a mean decrease in Gini.
- The top variables contribute more to the model than the bottom ones and also have high predictive power in classifying default and non-default customers
- Grid search does not have variable importance functionality in Python scikit- learn, hence we are using the best parameters from grid search and plotting the variable importance graph with simple random forest scikit-learn function.
- Whereas, in R programming, we have that provision, hence R code would be compact here.

```
>>> import matplotlib.pyplot as plt
>>> rf_fit = RandomForestClassifier(n_estimators=1000, criterion="gini",
max_depth=300, min_samples_split=3,min_samples_leaf=1)
>>> rf_fit.fit(x_train,y_train)
>>> importances = rf_fit.feature_importances_
>>> std = np.std([tree.feature_importances_ for tree in
rf_fit.estimators_], axis=0)
>>> indices = np.argsort(importances)[::-1]

>>> colnames = list(x_train.columns)
# Print the feature ranking
>>> print("\nFeature ranking:\n")
>>> for f in range(x_train.shape[1]):
...     print ("Feature", indices[f], ",", colnames[indices[f]],
round(importances [indices[f]],4))

>>> plt.figure()
>>> plt.bar(range(x_train.shape[1]), importances[indices], color="r", yerr=
std[indices], align="center")
>>> plt.xticks(range(x_train.shape[1]), indices)
>>> plt.xlim([-1, x_train.shape[1]])
>>> plt.show()
```

XV COMPARISON OF LOGISTIC REGRESSION WITH RANDOM FOREST

- In the following table, both models explanatory variables have been put in descending order based on the importance of them towards the model contribution.
- In the logistic regression model, it is the p-value (minimum is a better predictor), and for random forest it is the mean decrease in Gini (maximum is a better predictor).
- Many of the variables are very much matching in importance like,
    - status_exs_accnt_A14
    - credit_hist_A34
    - installment_rate_in_percentage_of_disposable_income
    - property_A_24
    - Credit_amount
    - Duration_in_month

Logistic Regression - Summary		
Variable	Co-efficient	p-value
const	-2.1782	0
status_exs_accnt_A14	-1.5561	0
cred_hist_A34	-0.9717	0
per_stat_sx_A93	-0.7401	0
Installment_rate_in_percentage_of_disposable_income	0.3783	0
purpose_A41	-1.3009	0.002
property_A124	0.7233	0.007
status_exs_accnt_A13	-1.1011	0.009
oth_inst_pln_A142	1.1535	0.009
purpose_A43	-0.5359	0.017
Credit_amount	0.0001	0.026
Number_of_existing_credits_at_this_bank	0.3876	0.041
cred_hist_A31	0.9028	0.042
Duration_in_month	0.0185	0.057

Variable Importance - Random Forest		
Feature Number	Variable Name	Mean Decrese in Gini
55	Credit_amount	0.1075
58	Age_in_years	0.0826
54	Duration_in_month	0.0729
3	status_exs_accnt_A14	0.0446
56	Installment_rate_in_percentage_of_disposable_income	0.0355
57	Present_residence_since	0.0315
0	status_exs_accnt_A11	0.0307
8	cred_hist_A34	0.0231
1	status_exs_accnt_A12	0.0191
59	Number_of_existing_credits_at_this_bank	0.0191
19	sav_acc_A61	0.0172
9	purpose_A40	0.017
39	property_A124	0.0157
30	per_stat_sx_A92	0.0155
31	per_stat_sx_A93	0.0155
38	property_A123	0.0155
42	oth_inst_pln_A143	0.0152
44	housing_A152	0.0149
36	property_A121	0.0149
25	pre_emp_snc_A72	0.0147
6	cred_hist_A32	0.0147
48	job_A173	0.014
5	cred_hist_A31	0.0139
26	pre_emp_snc_A73	0.0137
13	purpose_A43	0.0131
49	job_A174	0.0131
51	telephn_A192	0.0128
50	telephn_A191	0.0127
12	purpose_A42	0.0127
27	pre_emp_snc_A74	0.0127
28	pre_emp_snc_A75	0.0126
4	cred_hist_A30	0.0123
23	sav_acc_A65	0.0122
40	oth_inst_pln_A141	0.0115