67.1 - Explainable AI: LIME

Explainable AI is all about why a given model works in a specific way, for a given specific input. There are certain modes that are more complex (like the Deep Learning models), and are difficult to interpret, whereas the classical ML models are easy to interpret. Explainable AI is important in critical situations where we have to explain, given an input 'x_i' to the model, if we get the output 'y_i', we should be able to explain why we got the output as 'y_i'.

One way to provide explainability is using the feature importance. If our input is d-dimensional (ie., $x_i \in R^d$), and if it is passed through a machine learning model 'f', and gives the output 'y_i', then we can say which is feature is contributing most to the model, on the basis of its corresponding component in the weight vector.

In the case of Linear models, we can find out which feature is contributing more, on the basis of the components of the weight vector. In Tree-based models, we can find out the important features, on the basis of the Gini Impurity (or) Information gain. In Deep Learning models, there are many specialized techniques, and one of them is using Integrated Gradients using which we can get the feature importance.

When it comes to models like kernel SVM, we could not obtain the feature importance directly. The main goal here is to obtain model agnostic explainability, which means whatever ML model we might be using, the feature importance can be obtained. Two such libraries used for providing the feature importance irrespective of the ML model, are LIME and SHAP.

LIME stands for **Local Interpretable Model Agnostic Explanations**.

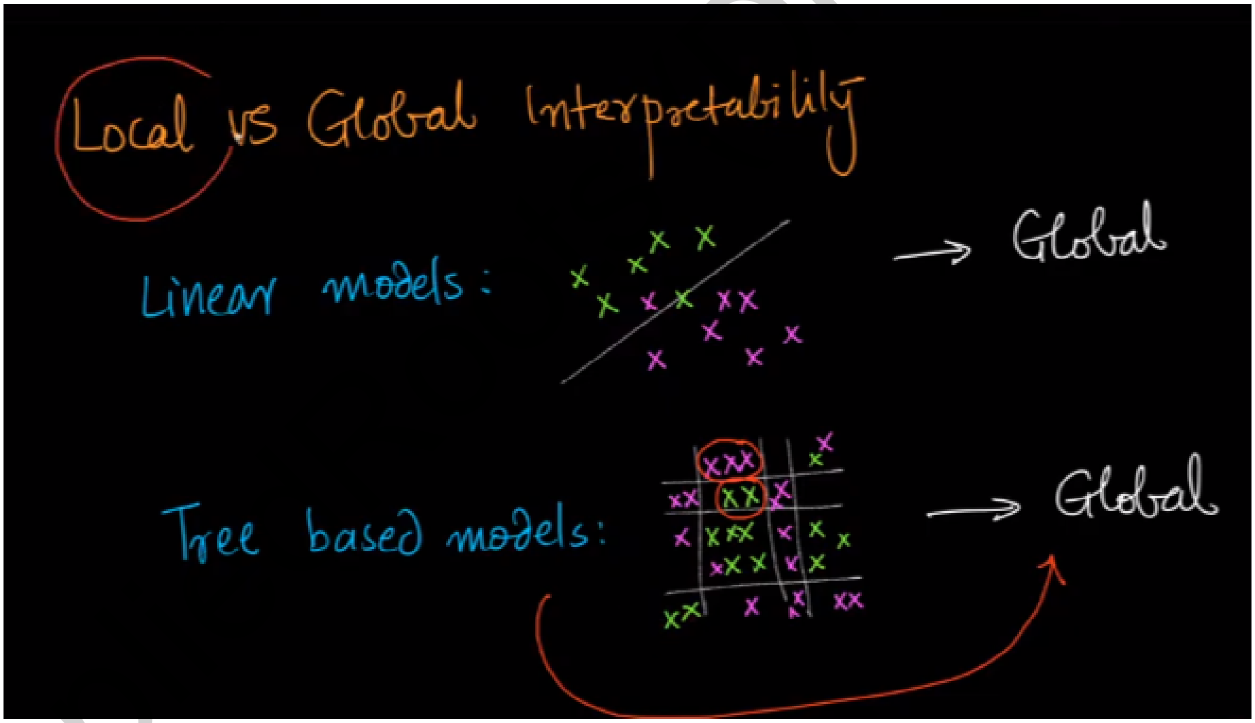
AppliedRoots (Draft Copy)

Local Interpretability vs Global Interpretability

Global Interpretability

The linear models have a global explanation which means they try to find a plane that separates the points belonging to different classes. The interpretability obtained from the linear models is global interpretability, which means the interpretability is obtained from the perspective of a global structure. Hence we call the linear models as the global interpretable models.

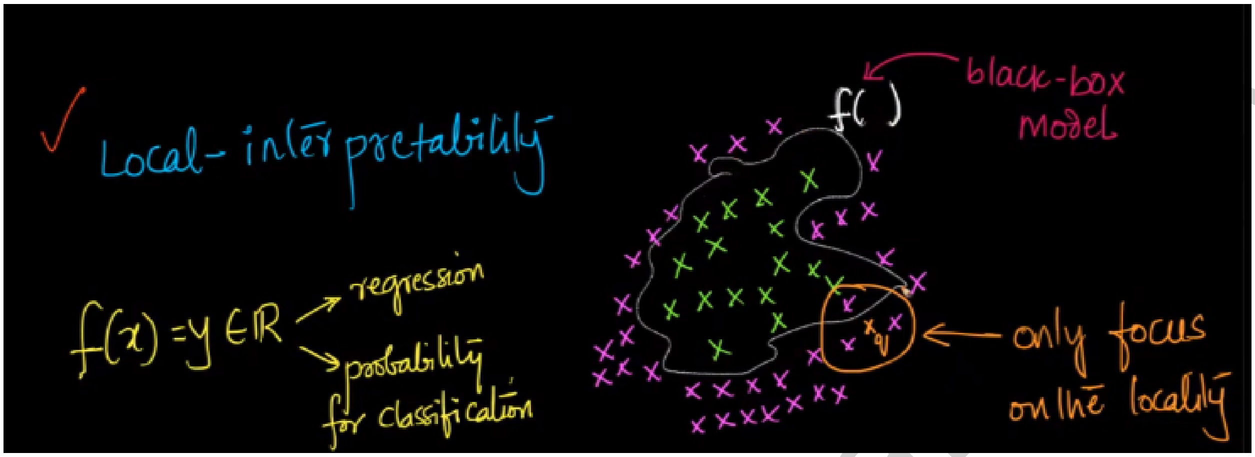
When we look at the tree-based models, they divide the entire space into different regions using the axes parallel planes. Even for the tree-based models, the global structure is taken into consideration, and are global interpretable models, as we aren't taking the neighborhood into consideration.



App Draft Copy)

Local Interpretability

Let us assume we have a black-box model ‘f’, and the decision surface of the model is as shown in the figure below



The decision surface looks complex, and this kind of decision surface is mostly seen in non-linear models like Deep Learning models, Kernel SVM, etc. In Local interpretability, if we pass a query point ‘x_q’ as an input to the model, it gives a predicted output as ‘y_q^’. Here we do not bother about the global structure, but we bother only about why we are getting the predicted output ‘$y^_q$’, for a given query point ‘x_q’, and we focus on the neighborhood of the point ‘x_q’. Our main focus is on how the model is behaving at the prediction task, in the locality/neighborhood of the point ‘x_q’.

So the difference between the Global Interpretability and the Local Interpretability is, in Global Interpretability, once we have the feature importance scores, it doesn’t matter where the query point ‘x_q’ is, the explanation is going to be the same because the feature importance values are computed globally. But in the case of Local Interpretability, we don’t care about what is happening all over, and we care only about what is happening in the locality/neighborhood of the query point ‘x_q’.

In order to make the math look simpler, we shall assume the black-box model ‘f’ gives a real-valued number as an output. The black-box model ‘f’ discussed so far could be a regression or a classification model.

$f(x) = y \in R$

If ‘f’ is a regression model → ‘y’ is the real-valued output

If ‘f’ is a classification model → ‘y’ is the probabilistic output (ie., $P(y_i=1|x_i)$)

AppliedRoots (Draft Copy)

Why is the model behaving in a specific way in the locality of ‘x_q’ in Local Interpretability?

The LIME models only look at the local interpretability and take the locality/neighborhood into consideration, but not the global structure.

LIME - Intuition

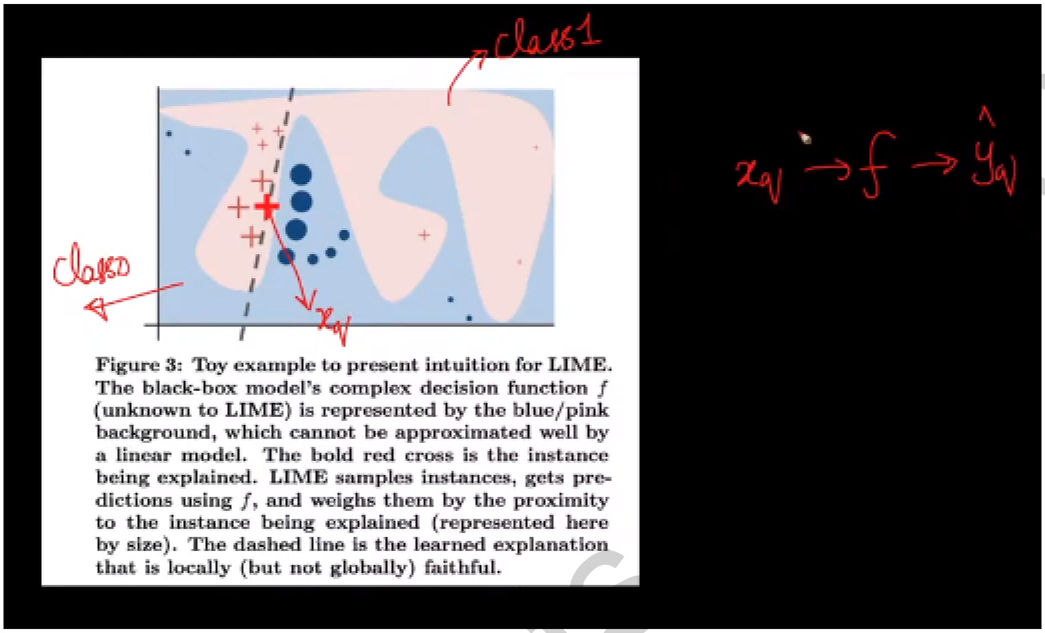


Figure 3: Toy example to present intuition for LIME. The black-box model's complex decision function f (unknown to LIME) is represented by the blue/pink background, which cannot be approximated well by a linear model. The bold red cross is the instance being explained. LIME samples instances, gets predictions using f, and weighs them by the proximity to the instance being explained (represented here by size). The dashed line is the learned explanation that is locally (but not globally) faithful.

Let us assume we have a decision surface as shown in the figure above. Let us assume, all the points present to the left side of this decision surface belong to the positive class, and all the points present to the right side of this decision surface belong to the negative class.

Let us assume we have a query point ‘x_q’, and let us denote the prediction by the model ‘f’ made on this point as ‘y_q^’. Now we have to explain why the model has given the output as ‘y_q^’.

The main intuition of LIME is sampling a few points in the neighborhood of the query point ‘x_q’, and weighing them, such that the points nearer to the query point ‘x_q’ are given the more weightage, and the points far away from the query point ‘x_q’ are given the less weightage. So far we have the model ’f’ trained on the entire training dataset. Now we have to fit a model ‘g’ only on these sampled points, such that the predictions made by the model ‘g’ on these sampled points, will approximate the predictions made by our initial model ‘f’. Here we call this model ‘g’ a **Surrogate model**.

AppliedRoots (Copy)

If ‘g’ is a linear model, then we can obtain the feature importance easily. As the prediction made by the surrogate model ‘g’ on the sampled points approximate the predictions made by the model ‘f’, the model ‘g’ can capture the facts on why the model ‘f’ is giving the output ‘y_q^’, for a given query point ‘x_q’.

Math and Algorithm

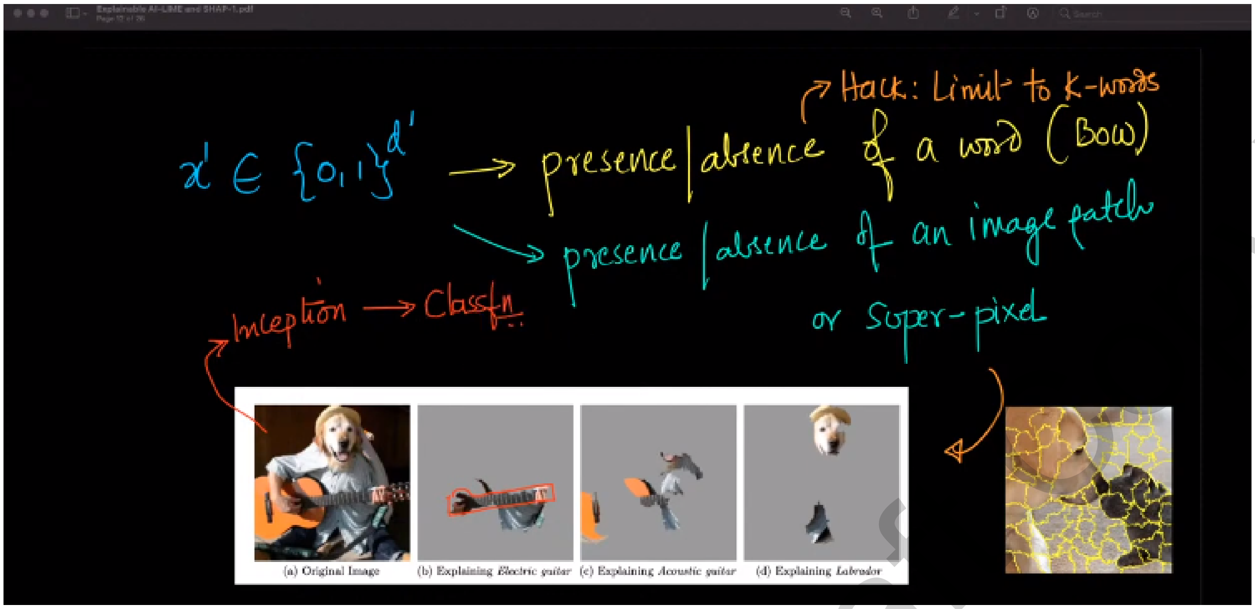
Let us assume we have the data in d-dimensional form (ie., $x \in R^d$), and ‘f’ is the black-box model. Whenever we pass our text data through a Word2Vec or any other model, we get a vector representation, but it is not interpretable.

So for every original vector $x_i \in R^d$, we have another corresponding vector $x_i^{l} \in \{0,1\}^d$. This x^l is a binary vector of d^l dimensions, and it is an interpretable representation. Let us see how to create interpretable representations.

Suppose we have a text with the words w_1, w_2, w_3, w_4, ….. And we pass this text through some Deep Learning model, let us assume we get a 512-dimensional BERT vector (which is non-interpretable). We shall assume this 512-dimensional vector is our data vector ‘x_i’$\in R^d$. Here our machine learning model ‘f’ has to be actually trained on ‘x_i’, but as ‘x_i’ is non-interpretable, we shall create an interpretable representation of ‘x_i’. In this case, that interpretable version of ‘x_i’ is a vectorization of the text data into Binary BOW, and let’s denote these binary vectors as ‘x_i^{l}’. Here Binary BOW vectorization catches the information about the presence of words in the given text. If a word ‘w_i’ is present in the given text, its component in the vector would be 1. Otherwise, it would be 0.

In the case of text data, the components of ‘x_i^{l}’ denote the presence/absence of the words. In the case of image data, the components of ‘x_i^{l}’ denote the presence/absence of image patch or superpixels in the image. An image can be divided into many superpixels (a superpixel is a region covered by a combination of one or more pixels with the same color tone)

AppliedRoots (Draft Copy)


Explainable AI-LIME and SHAP-1.pdf
Page 12 of 26

$x' \in \{0,1\}^{d'}$ → presence | absence of a word (BoW)
→ Hack: Limit to k-words
→ presence | absence of an image patch
or Super-pixel
Inception → Classfn

(a) Original Image
(b) Explaining Electric guitar
(c) Explaining Acoustic guitar
(d) Explaining Labrador

So here we are dividing the image into a group of superpixels, and the new vector representation ‘$x_i^{|}$’ consists of these superpixels as the components. When we apply the LIME, the components associated with those superpixels present in the image would be initialized to 1, and the remaining superpixels would be initialized to 0.

If our data is in tabular form(normally the dataset with many features), then

a) If we have categorical features, they can be converted into binary vector form using one-hot encoding.
b) If we have real-valued features, then we can convert them into binary vector form using the feature binning technique.

We have the surrogate model ‘g’ that approximates the model ‘f’ in the neighborhood of the query point ‘x_q’. Let ‘G’ be the class of easily interpretable models (like Logistic Regression, Decision Trees, SVM, etc), Then $g \in G$.

$\Omega(g)$ → Time Complexity of ‘g’ (It depends on the depth of the decision tree (or) the number of non-zero weights. If either of these two parameters increases, the time complexity becomes larger). We have to keep this time complexity as much low as possible.

AppliedRoots (Draft Copy)

The surrogate model ‘g’ works only on the interpretable representation of the points (ie., $g(x_i^{|})$), but not on the non-interpretable representation of the points (like $g(x_i)$)

Proximity Function

$\pi_x(z)$ → proximity measure between ‘x’ and ‘z’. It measures how close the two points ‘x’ and ‘z’ are.

Π_x → Locality of ‘x’

Local Fidelity and Interpretability

$L(f,g,\Pi_x)$ → In the locality/neighborhood defined by ‘Π_x’, whatever function ‘g’ we fit, it has to be very close to ‘f’. If ‘L’ is large, it means the surrogate model ‘g’ is not approximating the model ‘f’. We have to minimize this loss function ’L’ as much as possible.

So the objective here is to find the surrogate model ’g’ that belongs to the class of easily interpretable models, such that it approximates the actual model ‘f’ closely, minimizes the loss ‘L’ and also it(ie., ‘g’) should not be a complex model.

$\xi(x) = \min_{g \in G} L(f,g,\Pi_x) + \Omega(g)$ --- (1)

In the above function, both ‘f’ and ‘Π_x’ are constant. It is only ‘g’ that varies.

This is the optimization problem we have to solve. Here

G → Class of Interpretable models

L → Squared Loss (as $f(x) \in R$)

$\Pi_x(z) = \exp\{-D(x,z)^2/\sigma^2\}$ → Exponential Kernel

The loss function is

$L(f,g,\Pi_x) = \Sigma_{z,z' \in Z} \Pi_x(z)\ (f(z)-g(z^{|}))^2$

$\xi(x) = \min_{g \in G} L(f,g,\Pi_x) + \Omega(g)$

$\Omega(g)$ → K-Lasso → L_1 regularized Linear Regression with only top ‘K’ features

Along with minimizing the loss, we also have to minimize the time complexity(ie., the number of non zero weights)

AppliedRoots (Draft Copy)

Advantages

- It works on Images, Text, and Tabular data. In images, it works on the superpixels, in text, it works on BOW, and in tables, it works as one-hot encoding(on categorical features), and feature binning(on numerical features)
- It is highly interpretable if we are using Lasso (or) low depth trees for 'G'.
- The actual data 'x' always need not be the same as 'x^{l}'. Here 'x' may(sometimes) or may not be interpretable, whereas 'x^{l}' is interpretable.

Drawbacks

- It is difficult to define a neighborhood. There is no clear strategy to define a neighborhood, and also it may or may not be a good choice of choosing the neighborhood.
- As the neighborhood is defined on the basis of the exponential kernel, it is difficult to find out the kernel width. We have to try different values of kernel width and see which one gives sensible interpretations.
- It is difficult to choose/define the distance metric to compute D(x,z) in high-dimensional data.
- There might be a few real-world instances where the interpretable transformations of the data (ie., 'x^{l}') might not be straightforward.
  $\mathbf{x_i \in R^d} \rightarrow \mathbf{x^{l} \in \{0,1\}^{d'}}$ (After transformation)
- Instability due to the lack of robustness in the explanations. Experiments have proved
  a) On simulated data, the explanations on the close points have varied
  b) Repeating the sampling could give different explanations.
  c) Lack of robustness. This is the main disadvantage of LIME.

AppliedRoots (Draft Copy)

Alternative to LIME

- ➢ We have a technique called SHAP which works as an alternative to LIME.
- ➢ SHAP uses the shapely values from the game theory.
- ➢ It gives concrete mathematical guarantees of the machine learning systems
- ➢ SHAP is a time-consuming process.
- ➢ It is harder to interpret the explanations of SHAP. It might lead to misinterpretations.

Reference Links for Theory and Code on LIME

https://christophm.github.io/interpretable-ml-book/index.html
https://lime-ml.readthedocs.io/en/latest/
https://towardsdatascience.com/lime-how-to-interpret-machine-learning-models-with-python-94b0e7e4432e
https://towardsdatascience.com/interpreting-image-classification-model-with-lime-1e7064a2f2e5
https://towardsdatascience.com/what-makes-your-question-insincere-in-quora-26ee7658b010

Note

Please go through the reference links given above, and for any queries related to LIME, please feel free to post them in the comments section or mail us at mentors.diploma@appliedroots.com

AppliedRoots (Draft Copy)

67.2 - Explainable AI: SHAP

Game Theory - Intuition

When we apply the game theory in machine learning, our 'd' features (ie., f_1, f_2, …., f_d) in the dataset are the players. All these features work cooperatively to generate the model output. The shapely values measure how much each of the features has contributed to the output.

Intuition

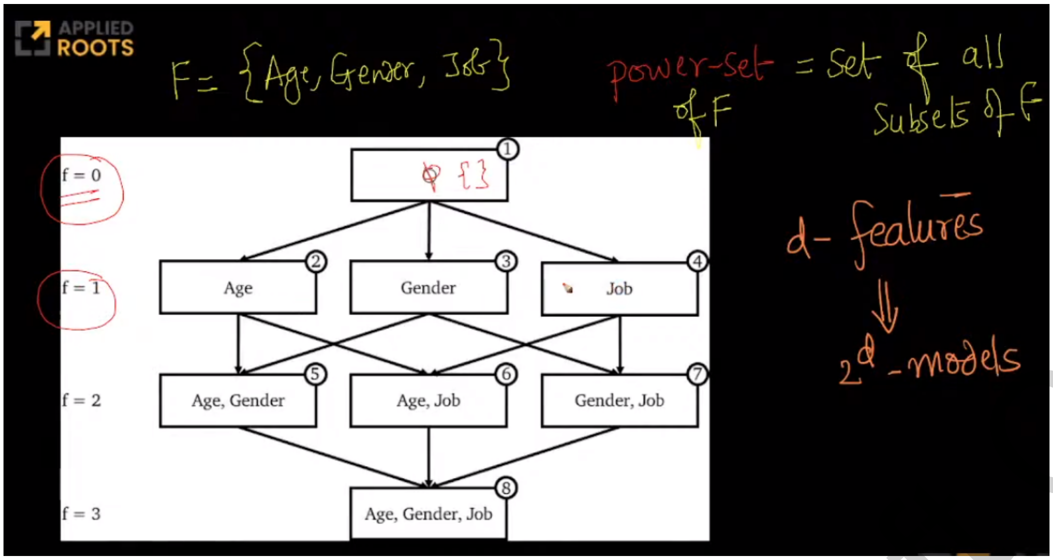
Let us take a regression problem as an example. We have to predict the salary of a person when the factors like age, gender, and job are given. So Age, Gender, and Job are the features, and the salary column is the output. The actual output for a data point 'x_q' is denoted as 'y_q', and the predicted output is denoted as 'y_q^'. Let us assume 'y_bar' is the average of all the **y_i's** in the train data (ie., 'y_bar' is the average salary of all the persons in the training data). Now we have to find out how much do these features (ie., f_1, f_2, …., f_d) contribute to the model prediction for a given query point 'x_q'.

Let us now create a power set of features. A power set of features is a set consisting of all possible subsets of features. For example, in our dataset, we have the features Age, Gender and Job. So the possible combinations are

a) Features taken 0 at a time. (ie., {} or ∅)
b) Features taken 1 at a time. (ie., {Age}, {Gender}, {Job})
c) Features taken 2 at a time. (ie., {Age, Gender}, {Age, Job}, {Gender, Job})
d) Features taken all the 3 at a time. (ie., {Age, Gender, Job})
So if we have 'd' features, then the total number of possible subsets = $\mathbf{2^d}$.
As we have d=3, the total number of possible subsets = $2^3 = 8$.

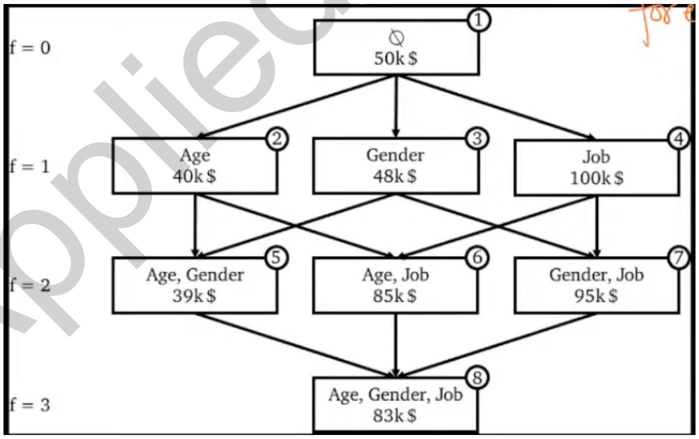
So we have to build 8 models using each of the above subset, and make predictions on the training data. The models and the features to be used are given below

AppliedRoots (Draft Copy)



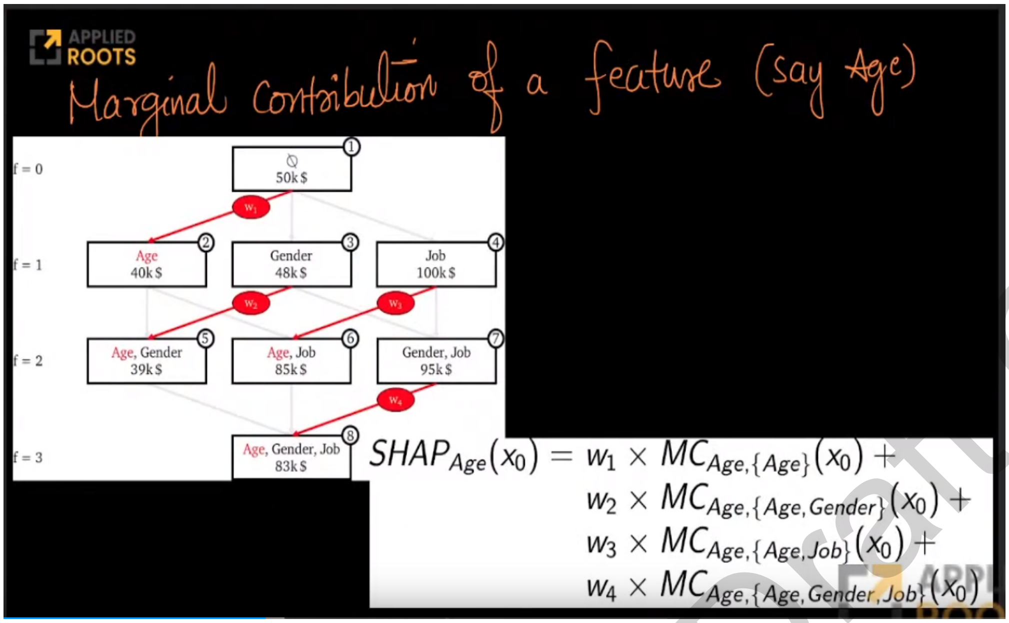
Model 1 → Features to be used: {}
Model 2 → Features to be used: {'Age'}
Model 3 → Features to be used: {'Gender'}
Model 4 → Features to be used: {'Job'}
Model 5 → Features to be used: {'Age', 'Gender'}
Model 6 → Features to be used: {'Age', 'Job'}
Model 7 → Features to be used: {'Gender', 'Job'}
Model 8 → Features to be used: {'Age', 'Gender', 'Job'}

In the figure given above, we are creating the edges only between the pairs of subsets that differ only by one feature. If a query point 'x_q' is given, it has to be passed through each of these 8 models, and the output values for this query point is obtained from the prediction made by all these 8 models. Let those predicted values be as shown in the figure below



AppliedRoots (Draft Copy)

We shall define the marginal contribution of a feature. Let us take the example of computing the marginal contribution of the 'Age' column.



Marginal contribution of a feature (say Age)

$$SHAP_{Age}(x_0) = w_1 \times MC_{Age,\{Age\}}(x_0) + w_2 \times MC_{Age,\{Age,Gender\}}(x_0) + w_3 \times MC_{Age,\{Age,Job\}}(x_0) + w_4 \times MC_{Age,\{Age,Gender,Job\}}(x_0)$$

Marginal contribution of 'Age' column with respect to an empty set for the point 'x_0' = $MC_{Age,\{Age\}}$ (x_0) =
= (Output of the model with only 'Age' column - Output of the model with no features)
= (Output of Model 2) - (Output of Model 1)
= 40K$ - 50K$
= -10K$

Marginal contribution of 'Age' column with respect to the 'Gender' column for the point 'x_0' = $MC_{Age,\{Age, Gender\}}$ (x_0) =
= (Output of the model with the features {'Age', 'Gender'} - Output of the model with the features {'Gender'})
= (Output of Model 5) - (Output of Model 3)
= 39K$-48K$
= -9K$

Marginal contribution of 'Age' column with respect to the 'Job' column for the point 'x_0' = $MC_{Age,\{Age, Job\}}$ (x_0) =
= (Output of the model with the features {'Age', 'Job'} - Output of the model with the features {'Job'})

AppliedRoots (Draft Copy)

= (Output of Model 6) - (Output of Model 4)
= 85K\$-100K\$
= -15K\$

Marginal contribution of ‘Age’ column with respect to the ‘Job’ and ‘Gender’ columns for the point ‘x_0’ = $MC_{Age,\{Age,\ Job,\ Gender\}}(x_0)$ =
= (Output of the model with the features {‘Age’, ‘Job’, ‘Gender’} - Output of the model with the features {‘Job’, ‘Gender’})
= (Output of Model 8) - (Output of Model 7)
= 83K\$-95K\$
= -12K\$

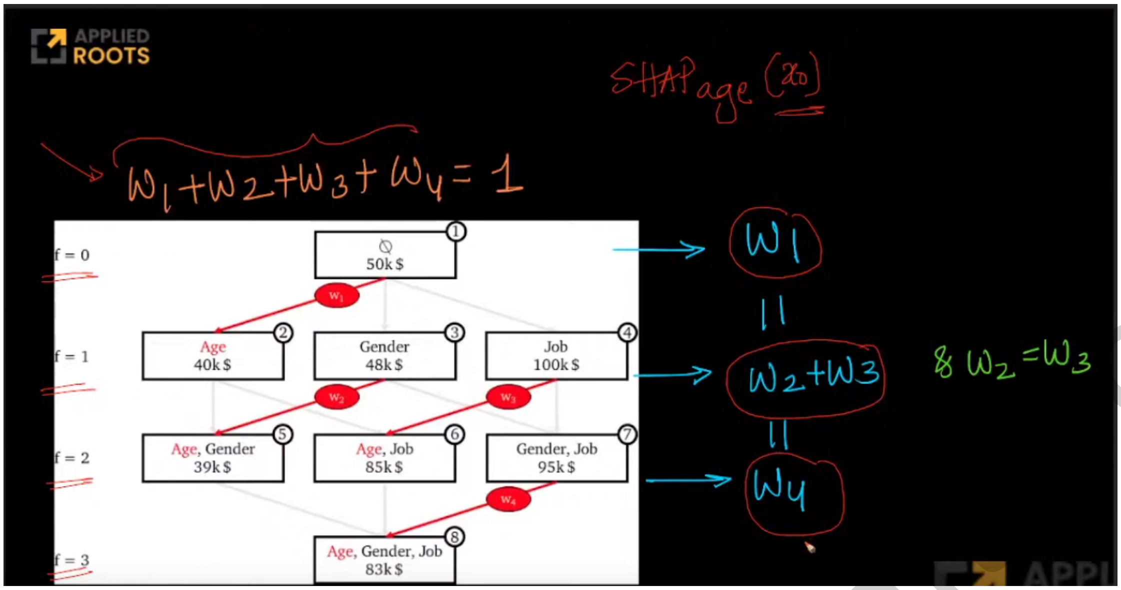
$$\textbf{SHAP}_{Age}(x_0) = w_1 * MC_{Age,\{Age\}}(x_0) + w_2 * MC_{Age,\{Age,\ Gender\}}(x_0) + w_3 * MC_{Age,\{Age,\ Job\}}(x_0) + w_4 * MC_{Age,\{Age,\ Job,\ Gender\}}(x_0)$$

So the shapely value of a feature is the weighted sum of all the marginal contributions of the feature. It is like a brute-force approach. There is a simple technique to compute these weights (ie., w_1, w_2, w_3 and w_4). Our main aim here is to compute the Shapely values of the ‘Age’ column with respect to the point ‘x_0’.

How to compute these weights?

1) One of the important properties these weights have to satisfy is the sum of all these weights should be equal to 1. In the above example,
   $\mathbf{w_1 + w_2 + w_3 + w_4 = 1}$
2) Another important property of shapely values is that the sum of weights at each level should be the same.

AppliedRoots (Draft Copy)



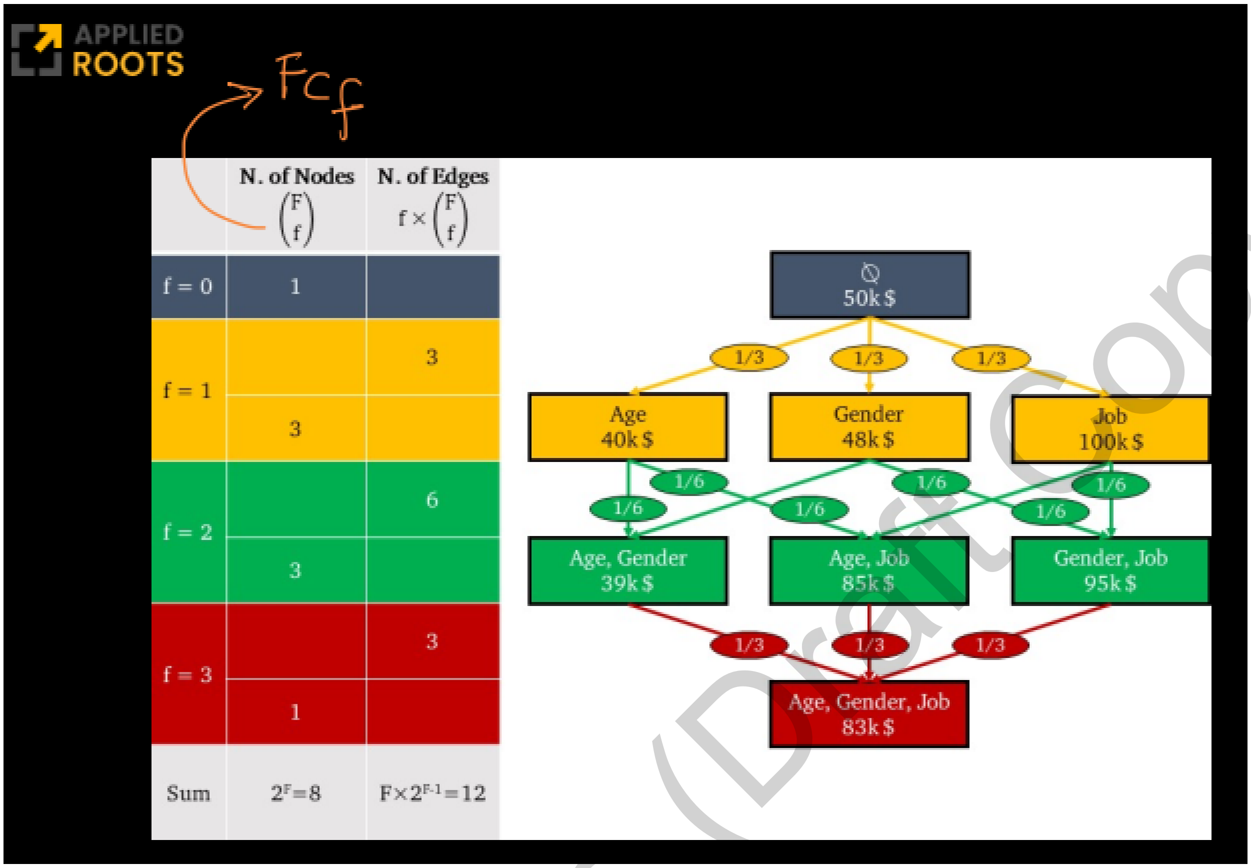
In the above picture, computing the shapely value of 'Age', we could see that

a) When we are going from no features to one feature (ie., from {} to {'Age'}, we have the weight 'w_1'.
b) When we are going from one feature to 2 features (ie., from {'Gender'} to {'Age', 'Gender'}, and from {'Job'} to {'Age', 'Job'}), we have the weights w_2 and w_3 respectively.
c) When we are going from 2 features to 3 features (ie., from {'Gender', 'Age'} to {'Age', 'Gender', 'Job'}), we have the weight w_4.

So the sum of the weights at each level should be the same.
(ie., $\mathbf{w_1 = (w_2+w_3) = w_4}$)

3) Also another important property is that if there are multiple weights in the same layer, each of these weights should be equal.
(For example, when we are moving from 1 feature to 2 features, we have the weights 'w_2' and 'w_3'. These two weights should be equal. (ie., $\mathbf{w_2 = w_3}$))
It is mandatory for the weights in the shapely values to satisfy these properties.
Number of nodes at each level = $\mathbf{Fc_f}$
Number of edges at each level = $\mathbf{f * Fc_f}$
Where F → Total Number of features in the dataset
f → Number of features in the subset at that level.

AppliedRoots (Draft copy)



	N. of Nodes $\binom{F}{f}$	N. of Edges $f \times \binom{F}{f}$
f = 0	1	
f = 1		3
	3	
f = 2		6
	3	
f = 3		3
	1	
Sum	$2^F=8$	$F \times 2^{F-1}=12$

From the example, of computing the shapely values of ‘Age’ feature, we have

F = 3

At level 0, we have no features at all. So f=0

At level 1, we consider one feature at a time. So f=1

At level 2, we consider two features at a time. So f=2

At level 3, we consider all the three features. So f=3.

So while computing the shapely values of the ‘Age’ column,

Total number of nodes $= Fc_{f=0} + Fc_{f=1} + Fc_{f=2} + Fc_{f=3}$

$= 3C_0 + 3C_1 + 3C_2 + 3C_3$

$= 1 + 3 + 3 + 1$

$= 8$

Total number of edges $= (f_{f=0}*Fc_{f=0}) + (f_{f=1}*Fc_{f=0}) + (f_{f=2}*Fc_{f=0}) + (f_{f=3}*Fc_{f=0})$

$= (0*3C_0) + (1*3C_1) + (2*3C_2) + (3*3C_3)$

$= 0 + 3 + 6 + 3$

$= 12$

AppliedRoots (Draft Copy)

Sum of total Number of Nodes at all the levels = $\mathbf{2^F}$ = 2^3 = 8.
Sum of total Number of Edges at all the levels = $\mathbf{F*2^{F-1}}$ = $3*2^{3-1}$ = 12.

Let us look at the formula for computing the shapely value once again.

$$\mathbf{SHAP_{Age}(x_0) = w_1 * MC_{Age,\{Age\}}(x_0) + w_2 * MC_{Age,\{Age,\ Gender\}}(x_0) + w_3 * MC_{Age,\{Age,\ Job\}}(x_0) + w_4 * MC_{Age,\{Age,\ Job,\ Gender\}}(x_0)}$$

Here

$w_1 = (f_{f=1}*Fc_{f=1})^{-1} = (1*3C_1)^{-1}$ = ⅓
$w_2 = (f_{f=2}*Fc_{f=2})^{-1} = (2*3C_2)^{-1}$ = (½)*(⅓) = ⅙
$w_3 = (f_{f=2}*Fc_{f=2})^{-1} = (2*3C_2)^{-1}$ = (½)*(⅓) = ⅙
$w_4 = (f_{f=3}*Fc_{f=3})^{-1} = (3*3C_3)^{-1}$ = (⅓)*1 = ⅓

When we substitute these weight values in the formula for $\mathbf{SHAP_{Age}(x_0)}$, we get

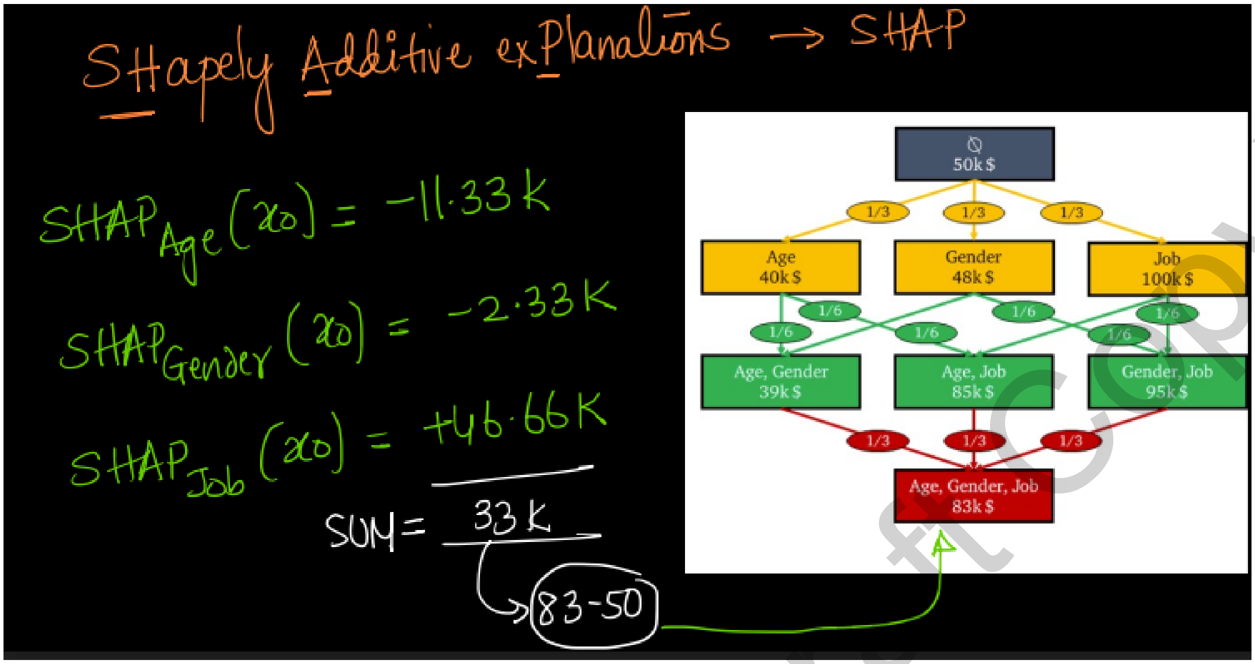
$SHAP_{Age}(x_0)$ = (⅓)*(-10K\$) + (⅙)*(-9k\$) + (⅙)*(-15K\$) + (⅓)*(-12K\$)
= -11.33K\$

So the generic formula is given as

$$\boxed{SHAP_{feature}(x) = \sum_{set:feature \in set} [|set| \times \binom{F}{|set|}]^{-1} [Predict_{set}(x) - Predict_{set \setminus feature}(x)]}$$

From the above equations, we have computed the value of $SHAP_{Age}(x_0)$.
Similarly we have to compute the values of $SHAP_{Gender}(x_0)$ and $SHAP_{Job}(X_0)$.
We'll get the values as
$SHAP_{Age}(x_0)$ = -11.33K\$
$SHAP_{Gender}(x_0)$ = -2.33K\$
$SHAP_{Job}(x_0)$ = 46.66K\$
The sum of all the Shapely values = (-11.33) + (-2.33) + 46.66 = 33K\$

AppliedRoots (Draft Copy)



So if we pass a data point ‘x_0’ through the model without any features, then the output would be 50K$. In case, if we pass the same data point ’x_0' through the model with all the 3 features (ie., ‘Age’, ‘Gender’ and ‘Job’), then it gives the output as 83K$.

SHAP stands for **Shapely Additive Explanations**. The shapely values of all the features are explaining us why this simple model (ie., the model built using null set as features) is differing from the final model (ie., the model built using all the features) in terms of the output, and this difference is split among the 3 features (ie., ‘Age’, ‘Gender’ and ‘Job’), and is explained clearly by these 3 features.

Sum of shapely values = y^$_{\{all\ features\}}$ - y^$_{\{no\ features\}}$

Now we shall look at setting up the explainability, which is the same as that of LIME. In LIME, if our given data was ‘x’, we have created an interpretable representation out of it, and it is denoted as $\mathbf{x^{l} \in \{0,1\}^{M}}$

Given any ‘x^{l}’, there exists a function ’h_x’ which gives ‘x’, when ‘x^{l}’ is passed through it. (ie., $\mathbf{x = h_x(x^{l})}$)

Let us denote the original machine learning model as ‘f’ and the surrogate model as ‘g’, then $\mathbf{g(x^{l}) \approx f(h_x(x^{l}))}$

Additive Feature Attribution methods have an explanation model that is a linear function of binary variables.

AppliedRoots (Draft Copy)

$\mathbf{g(z^{|}) = Ø_0 + \Sigma_{i=1}^{M} Ø_i Z_i^{|}}$

Where M → Number of dimensions in interpretable representation

This function is another form of representing

$\mathbf{y\text{^}_{\{all\ features\}} = y\text{^}_{\{no\ features\}}}$ + **Sum of shapely values**

Expectations in Probability

Let us now see how this concept of Shapely values can be understood from the perspective of probability.

$\mathbf{E(X) = \Sigma_x\, x.P(X{=}x) = \int_x x.P(x)\, dx}$

We can understand the expectation of 'X' as the expected or average value of 'X'.

$\mathbf{E(X|Y{=}y_i) = \Sigma_x\, x.P(X{=}x_i)}$

Note: Please go through the link given below to learn about Conditional Expectation
https://en.wikipedia.org/wiki/Conditional_expectation

The conditional expectation values are equivalent to the shapely values. So wherever we have to compute the shapely values, we have to compute the conditional expectations. The computation of the shapely values is very time consuming and is of exponential time complexity. In general,we compute these computational expectations using approximation algorithms with some accuracy.

Kernel SHAP (Combination of LIME + Shapely Values)

Kernel SHAP uses the same neighborhood function and also the surrogate function, same as LIME. The only difference between Kernel SHAP and LIME is, in Kernel SHAP instead of finding out the feature importance using a linear model, we compute the shapely values in determining 'y_0^' given 'x_0', by using the whole math we used in shapely values.

Kernel SHAP is **Model Agnostic**. LIME had a fundamental problem on how to define the neighborhood function. Kernel SHAP solves this problem using the function given below. In kernel SHAP,

$\mathbf{\pi_x(z^{|}) = (M-1)/((MC_{|z'|}) * |Z^{|}| * (M-|Z^{|}|))}$

AppliedRoots (Draft Copy)

Where M → Number of dimensions in x^l (x^l is an interpretable representation)
$|z^l|$ → Number of features in z^l (z^l is a binary vector which says how many features are present)

In the above function, 'M' is constant.
If $|z^l|$ is very large or very small, then **$\pi_x(z^l)$** is larger.
If $|z^l|$ is in between (ie., neither too large nor too small), then **$\pi_x(z^l)$** is smaller.

Note

Even with the usage of all approximation algorithms, Kernel SHAP still takes time, as compared to standard LIME. There was a research paper published on TREESHAP which is a model specific approximate shapely computation. TREESHAP is much faster than the regular Kernel SHAP. The major disadvantage of TREESHAP is its interpretation is not as good as that of Kernel SHAP.

Advantages of SHAP

- ➢ It has a solid theory. If we have the shapely values and if we use a kernel SHAP, and if we take the sufficient amount of time to compute everything, then we can explain everything with proofs.
- ➢ We can combine LIME and SHAP to have actual interpretable models. We can use SHAP on top of LIME, to create Kernel SHAP.
- ➢ TREE SHAP is faster than Kernel SHAP

Disadvantages of SHAP

- ➢ Kernel SHAP is the most generic form of Model Agnostic SHAP, but it is slow.
- ➢ TREE SHAP results are unintuitive. We can't trust in the results obtained from TREE SHAP.
- ➢ Another drawback with both Kernel SHAP and TREE SHAP is they both are approximation based models, and the feature dependence is ignored. All the approximations are performed assuming all the features are independent.

AppliedRoots (Draft Copy)

Reference Links

https://towardsdatascience.com/shap-explained-the-way-i-wish-someone-explained-it-to-me-ab81cc69ef30?gi=964927471e98
https://en.wikipedia.org/wiki/Conditional_expectation
https://arxiv.org/pdf/1705.07874.pdf
https://shap.readthedocs.io/en/latest/index.html
https://drive.google.com/file/d/1Oc3gsx00ieNmHhmNwlPqKok3P8999o-m/view

Note

As the code part was discussed in the video lecture, we aren't documenting it here. Please go through the reference links mentioned above and if you have any queries, please post them in the comments section (or) please feel free to drop us an email.

AppliedRoots (Draft Copy)