5 DECISION TREES AND RANDOM FOREST

Nonparametric estimation is a statistical method that estimates the functional form of data fits when parametric modeling is absent and where there is no guidance from theory. Examples of nonparametric estimation are neural networks with layers of nonlinear activation functions on weighted features that attempt to fit the training data set, and kernel estimation method akin to fitting histograms on frequency counts to get a distributional form.

Decision Tree (DT) is a method to predict the value of a target variable (whether predicting a class/category or a continuous value) by a non-parametric way without employing some parametric functional form relating target to features such as in a logistic regression approach. A DT contains decision rules that are conditions to test each sample point on the training data set. For example, to partition red strawberries from red rambutans (an Australian friend of mine years ago called the latter 'hairy strawberries'), one could use the decision rule that if the fruit has hairs, then it is rambutan; otherwise, it is strawberry. Another example could be trying to partition between crocodiles and alligators. A decision rule could be that if the creature has rounded snout, then it is an alligator; otherwise, it is a crocodile. A difficult example is identifying a mugger from a police lineup where the other people in the lineup are innocent. One could use a decision rule based on observed features of the lineup suspects, such as whether the suspect has a proven alibi at the time of the crime. This may reduce the number of suspects, but there could still be many left, so more decision rules must be added sequentially to filter out the innocent ones.

The DT approach is typically supervised learning with a training data set and a test data set. In this chapter, we look at the usefulness of DT and the related Random Forest (RF) method of predicting binary classification. It is called the Categorical Variable DT when predicting binary or categorical groups. When used for predicting continuous value outputs, the DT may be called a Continuous Variable DT. Categorical Variable DT (and related RF) is generally found to be more accurate relative to use of Continuous Variable DT as the latter involves many more permutations of the target variable. However, due to the non-parametric nature, DT methods can produce unstable outcomes when data features have high variances as the decision rules on the partitioning can change sharply from one data point to another.

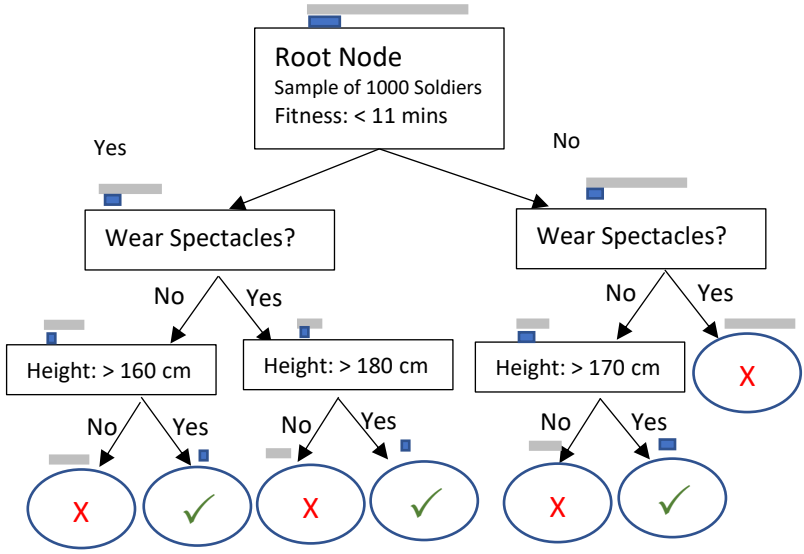
5.1 Decision Tree Method

Suppose we use DT to predict if a case belongs to the positive or the negative class, being binary partitions of all cases, i.e., any case could be put into the positive or the negative class, and that the classes are reasonably balanced. Balance refers to the number of cases in the positive class being of similar magnitude to the number of cases in the negative class. Let each case, besides its positive or negative label, also carry a set of attributes or features that can help in predicting its type/label.

We illustrate as follows how a DT can help to predict (1) if a soldier who passed BMT is a marksman (defined as one who shoots on target at least 84% of the times or 21 hits out of 25 shots at training targets), with only preliminary training, given the soldier's fitness (time in minutes to complete a 2.4 km run), height (in cm) and whether the soldier needs to wear spectacles or not; and (2) if a person would get a certain lung disease by age 60 given the person's fitness measured by number of hours of exercise a week, whether a regular smoker (at least one cigarette a day) or not, and whether there is a hereditary factor, i.e., if the person's parent (one or both) has had such a disease.

A trained DT based on a training sample of past 1000 soldiers for (1) could be as follows. The first root node is also a decision node on using fitness to split the total sample of 1000 soldiers. Value of a node refers to the number of positive cases (training sample marksmen) and number of negative cases (non-marksmen). For the illustration, this set of values is indicated by the relative length of the dark shade versus lighter grey bars above the decision node. The decision node is a node whereby a decision is made, e.g., if a sample point (a soldier in the training data set) has fitness run below 11 minutes (or else equal and above).

A feature is chosen (in some optimal way to be explained later) and a threshold of the feature is chosen. The latter refers to some values of the feature as the splitting point to divide the data associated with the feature into two subsets. The splitting or branching is based on 'Yes' or 'No' to the outcome of the decision rule. If the feature is categorical, then the splitting is based on 'Yes' or 'No' to the categorical rule. The decision node is associated with a decision rule. Data is split into subsets and moves along the branches to the next (lower) levels of decision nodes until the leaf or terminal nodes are reached. At each leaf (end) node, the model makes a prediction.



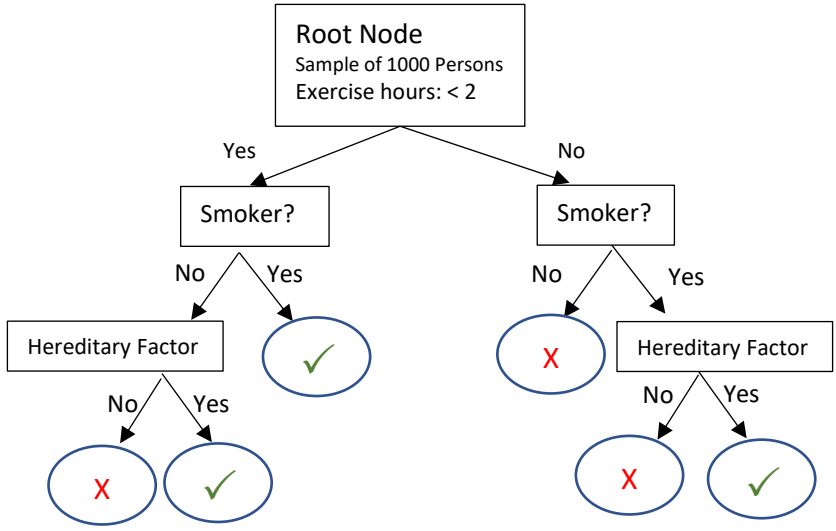
The rectangular boxes represent the decision nodes where splitting of the sample occurs along the branches. The ovals at the end of the tree (upside down) represents the leaf nodes or terminal nodes. These carry the prediction or final decision whether prediction (tick) is a marksman or (cross) not a marksman. In this example, the decision tree ends in a leaf with no further extension or decision box assuming no other features can be of help. Also, at the leaf, all remaining sub-samples are either all marksman or not marksmen. Hence there is no more sub-dividing. Recall that the bars above the boxes or ovals represent the number of marksmen (dark shade) and non-marksmen (lighter shade) from the historical sample used as training set. With each decision and split, the numbers of marksmen and non-marksmen are split into the next set of nodes.

The prediction rules according to this DT are: if fitness is < 11 mins, no spectacles, and height > 160 cm, or if fitness is < 11 mins, spectacles, and height > 180 cm, or if fitness is ≥ 11 mins, no spectacles, and height > 170 cm, then marksman; otherwise not in the other feature conditions. It is seen that with the given training set, one could continue to search for new features (associated with the targets) such that a chain of rules would lead to perfect discrimination – separating the targets into the different categories.

This could be over-fitting and may not produce a good accuracy on the test set. Some regularizations/constraints on the hyperparameters may be introduced such as limiting the depth (levels) of the DT instead of the default maximum (8 to 32 in the sklearn app), limiting to no decision node based on a feature if the improvement in the differentiation is minimal (we will discuss this in a while), limiting the total number of nodes, limiting to a minimum number of sample (points) on any node and not to split further, and so on.

Another example is a DT based on a training sample of past 1000 medical insurance buyers over a long period. Assume no moral hazard problem and that each insurance buyer starts off under the insurance scheme with no sign of contracting the disease.

A tick at the leaf denotes prediction of getting the disease; otherwise, it is a cross. The prediction rules according to this DT are: if exercise hours is < 2, if smoker, or, if exercise hours is < 2, if non-smoker but with hereditary factor, then disease is predicted. Or else, if exercise hours is ≥ 2, smoker and with hereditary factor, then disease is predicted. Otherwise, disease is not predicted.



Next, we explain how the DT chooses which feature to create the split, how is the split condition chosen, and when does the splitting stop.

At the root node, the sample (size N) provides a measure of Gini impurity that is defined as $1 - \sum_{k=1}^{m} p_k^2$ where m is the number of different categories in the sample and p_k is the empirical probability of being in category k. Thus, in the marksmen case, if the original sample has 200 marksmen and 800 non-marksmen, then if the positive case 1 (marksmen) has probability p_1 = 200/1000 = 0.2, and the negative case 2 (non-marksmen) has probability p_2 = 0.8. The Gini impurity would be $1 - (0.2^2 + 0.8^2) = 0.32$ in this binary classification. Note that if $p_1 = p_2 = 0.5$, then we have a maximum Gini impurity. But $p_1 = p_2 = 0.5$ is also the case for maximum entropy (uncertainty in distributional outcome) in the entropy measure of $-\sum_{k=1}^{m} p_k \ln(p_k)$. Hence lower Gini impurity is related to lower entropy or better resolution of the classification rules. Entropy is an alternative loss function to be minimized here. Hence the idea is to develop rules leading to the lowest Gini impurity in the next nodes. It is also possible to use a log loss or deviance function for binary classification where (log) loss is

$$loss = -\sum_{i=1}^{N} \left(Y_i \, p + (1 - Y_i)\,(1 - p) \right)$$

where N is the sample size, and p is the empirical probability of $Y_i = 1$ for any i.

The minimization of loss function as Gini impurity is done by (1) working on each feature considering splitting at different points – for a continuous variable feature, it means discretizing it and considering splits at each discrete points, e.g., splits at < 9 mins, < 9.1 mins, < 9.2 mins, ….., < 11 mins, < 11.1 mins, ….., <14 mins etc., and then (2) computing the Gini impurities in the new next layer binary nodes under each split, and (3) finding that split that produces the lowest weighted Gini impurities lower than that in the previous node (hence the largest gain in information – largest drop in entropy in the new split). For discrete valued features – the potential split point could be the mid-points of adjacent discrete values of the feature.

For example, the following could be a result of the split.

(Values) [Gini Impurity]	Yes	No	Weighted Gini Impurity

If < 10.9 min	(130,320) [0.41086]	(100,450) [0.29752]	(450/1000) × 0.41086 + (550/1000) × 0.29752 = 0.34853
If < 11 min	(120,300) [1 – (120/420)^2 – (300/420)^2 = 0.40816]	(80,500) [1 – (80/580)^2 – (500/580)^2 = 0.23781]	(420/1000) × 0.40816 + (580/1000) × 0.23781 = 0.30936
If < 11.1 min	(110,340) [0.369383]	(90, 460) [0.273719]	(450/1000) × 0.369383 + (550/1000) × 0.273719 = 0.31677

Suppose the split at Fitness < 11.0 min gives the lowest weighted Gini impurity or biggest drop in impurity from the previous node of 0.32. Suppose also that considering splits in all the other features do not produce any bigger drop in impurity. Then the first decision node is whether Fitness < 11.0 min. Note that the Gini impurity could be considered as the loss function to be minimized in this DT.

Building the DT then continues to the next level (down) when the next best information gain feature would be used to create the next split, and so on. Once the DT is trained using the training set, it is then used to predict the outcomes/categorizations in the test data set. All the test data X-variables (features) are used to decide on a positive or a negative case based on the trained decision rules. In case a leaf (final node) ends without fully discriminating against the two (or more) categories due to regularization, the empirical probability in the sample in that leaf can be used to make the prediction. The splitting of the sample into smaller subsets along each branch is called recursive distribution of the sample.

We briefly explain the Continuous Variable DT as follows. Suppose in the same marksmen problem above, we want to predict the actual shooting score – how many shots on target out of a total of 25 shots in the shooting range test. This is a discrete variable – but any continuous variable in the finite initial sample is considered a discrete value (no doubt with decimal places in the numbers).

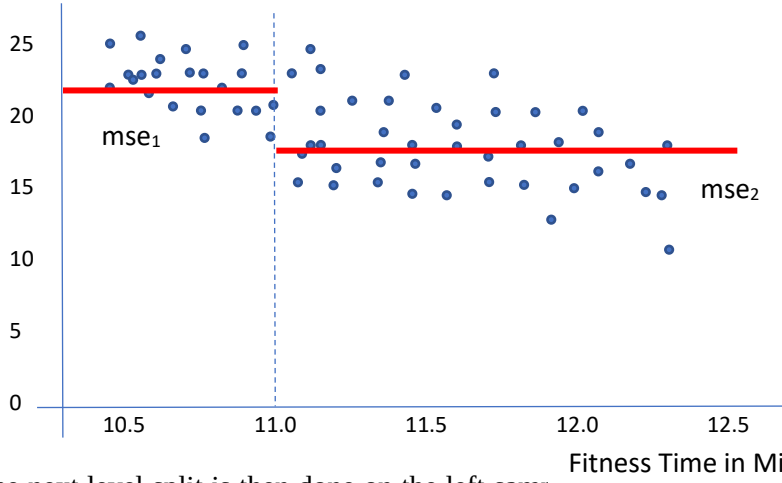
Continuous Variable DT works slightly differently whereby the loss criterion is squared error (default case) -- using DecisionTreeRegressor instead

of DecisionTreeClassifier (for classification) in sklearn. Continuous Variable DT is sometimes called a regression tree model. Suppose we plot the actual shoot score of the 1000 cases/subjects/samples against one of the features – Fitness.

The diagram serves only to illustrate as there would be 1000 points. Suppose at Fitness < 11.0 min, we obtain two partitions. On the left we have mean square error $mse_1 = \frac{\sum_{j=1}^{L}(Y_j - \bar{Y}^L)^2}{L}$ where L is the number of sample points with feature Fitness < 11.0 min and on the right $mse_2 = \frac{\sum_{j=1}^{R}(Y_j - \bar{Y}^R)^2}{R}$ where R is the number of sample points with feature Fitness ≥ 11.0 min. Total L + R = 1000 from the total number of sample before the split (earlier node). $\bar{Y}^L$ and $\bar{Y}^R$ are respectively the left sample and right sample means. Weighted variance or loss function in this case is L/(L+R) × mse_1 + R/(L+R) × mse_2. This is the same as

$$[\sum_{j=1}^{L}(Y_j - \bar{Y}^L)^2 + \sum_{j=1}^{R}(Y_j - \bar{Y}^R)^2] /(L+R).$$

The decision rule to split, i.e., at Fitness < 11 min, is chosen such that this loss function is the minimum across other possible splits and across all other feature splits and is also a variance reduction from the previous higher-level node.



The next level split is then done on the left sample, and next level split is done separately on the right sample using the next feature, and so on. Finally,

when all levels are done and the leaf node is on groups with the same shooting score, the DT can then be used to perform prediction of a future soldier's shooting score based on the new soldier's features. The predicted number would be on one of the leaf nodes. If the leaf ends with a small set of different shooting scores – then the mean can be taken as the predicted score given the decision rules along that DT.

The potential split points in the features are typically obtained as the mid-points of two adjacent discrete values of observations of the feature under consideration.

Note that for complex data, it is sometimes possible for the algorithm to use the same feature (used earlier for splitting closer to the root node) to split again at some points along a sub-tree. The way the splitting is done is based on the "greedy" algorithm approach that looks for the best option at hand without looking down the road if other not immediately best steps could instead be taken.

5.2 Worked Example DT – Data

This data set corporate_rating2.csv is obtained on public website from Kaggle – and there are acknowledgements of financialmodelingprep and opendata soft as the sources of the data. It is also in Kaggle, https://www.kaggle.com/datasets/agewerc/corporate-credit-rating.

The dataset contains 2029 credit ratings issued by major agencies from 2010 to 2016 on large US companies. For each credit rating, each company shows 25 accounting features. The other 6 columns in the data set consist of rating, name of firm, symbol of firm in exchange trading, the rating agency, date of rating, and the sector of the firm. I added a new column called 'Class" that contains "1" if the credit rating indicates investment grade, i.e., "AAA", "AA", "A", or "BBB". Any rating below "BBB" – considered as speculate grade – has a class value of "0".

The idea in this exercise using Chapter5-1.ipynb is to use the company financial/accounting variables/ratios to try to predict which binary class of investment grade or speculative grade the firm belongs to, using the published class as target for supervised training. A particular company may have several data points in the sample set as it might have obtained credit ratings, hence a classification as "1" or "0" at different calendar dates/times.

When adequately trained, the predictive model can be used to help new firms getting listed as well as investors/creditors of such firms to use the firm's internal accounting ratios (reported as accurately as possible in a format close to the published ones) to help predict which category the firm should belong to. Obviously if identified as investment grade, the firm would find it easier to float new shares (IPOs) and issue debts at a cheaper interest cost.

In codeline [3], the features information is shown.

```
In [3]: # Display the structure
        df_ratios.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2029 entries, 0 to 2028
Data columns (total 32 columns):
 #   Column                                Non-Null Count  Dtype
---  ------                                --------------  -----
 0   Rating                                2029 non-null   object
 1   Class                                 2029 non-null   int64
 2   Name                                  2029 non-null   object
 3   Symbol                                2029 non-null   object
 4   Rating Agency Name                    2029 non-null   object
 5   Date                                  2029 non-null   object
 6   Sector                                2029 non-null   object
 7   currentRatio                          2029 non-null   float64
 8   quickRatio                            2029 non-null   float64
 9   cashRatio                             2029 non-null   float64
 10  daysOfSalesOutstanding                2029 non-null   float64
 11  netProfitMargin                       2029 non-null   float64
 12  pretaxProfitMargin                    2029 non-null   float64
 13  grossProfitMargin                     2029 non-null   float64
 14  operatingProfitMargin                 2029 non-null   float64
 15  returnOnAssets                        2029 non-null   float64
 16  returnOnCapitalEmployed               2029 non-null   float64
 17  returnOnEquity                        2029 non-null   float64
 18  assetTurnover                         2029 non-null   float64
 19  fixedAssetTurnover                    2029 non-null   float64
 20  debtEquityRatio                       2029 non-null   float64
 21  debtRatio                             2029 non-null   float64
 22  effectiveTaxRate                      2029 non-null   float64
 23  freeCashFlowOperatingCashFlowRatio    2029 non-null   float64
 24  freeCashFlowPerShare                  2029 non-null   float64
 25  cashPerShare                          2029 non-null   float64
 26  companyEquityMultiplier               2029 non-null   float64
 27  ebitPerRevenue                        2029 non-null   float64
 28  enterpriseValueMultiple               2029 non-null   float64
 29  operatingCashFlowPerShare             2029 non-null   float64
 30  operatingCashFlowSalesRatio           2029 non-null   float64
 31  payablesTurnover                      2029 non-null   float64
```

Standard scaling is done in code line [6] on the features, excepting the Class target variable.

```
In [6]: # Normalization
        from sklearn.preprocessing import StandardScaler
        scaler = StandardScaler().fit(df_ratios_1)
        df_ratios_1 = scaler.transform(df_ratios_1)
        df_ratios_1=pd.DataFrame(df_ratios_1,columns=['currentRatio','quickRatio','cashRatio','daysOfSalesOutstanding',
                                                      'netProfitMargin', 'pretaxProfitMargin', 'grossProfitMargin',
                                                      'operatingProfitMargin','returnOnAssets','returnOnCapital','returnOnEquity',
                                                      'assetTurnover','fixedAssetTurnover','debtEquityRatio','debtRatio',
                                                      'effectiveTaxRate','freeCashFlowOperatingCashFlowRatio','freeCashFlowPerShare',
                                                      'cashPerShare','companyEquityMultiplier','ebitPerRevenue',
                                                      'enterpriseValueMultiple','operatingCashFlowPerShare',
                                                      'operatingCashFlowSalesRatio','payablesTurnover'])
```

This scaled data is then combined/concatenated with the Class variable in [11]. Note that we do not scale the Class variable here.

```
In [11]: TT=pd.concat([df_ratios['Class'],df_ratios_new],axis=1)
         TT.shape
         TT.tail()
```

Out[11]:

	Class	currentRatio	quickRatio	cashRatio	daysOfSalesOutstanding	netProfitMargin	pretaxProfitMargin	grossProfitMargin	operatingProfitMargin
2024	1	0.186827	0.135444	1.783407	-0.074822	-0.036575	-0.038419	0.955929	-0.045400
2025	0	-0.012870	-0.041268	-0.066739	-0.068183	-0.073966	-0.077771	-0.569938	-0.074482
2026	0	-0.060074	-0.054997	-0.120995	-0.041872	-0.047159	-0.048010	-0.216267	-0.035814
2027	0	-0.059442	-0.057857	-0.099558	-0.045460	-0.031518	-0.033485	-0.158141	-0.026830
2028	0	-0.055507	-0.049416	-0.129463	-0.040959	-0.084191	-0.083265	-0.192994	-0.033853

5 rows × 26 columns

Note: The displayed output in line [11] is not shown fully.

The total sample is then randomly split into 75% training set (1521 sample points) and 25% test set (508 sample points).

```
In [12]: Train, Test = train_test_split(TT,
                                       test_size=0.25,
                                       random_state=0)

         X_train, y_train = Train.iloc[:,1:26], Train.iloc[:,0] ### choose the 70 features excluding NPM
         X_test, y_test = Test.iloc[:,1:26], Test.iloc[:,0]
```

```
In [13]: X_train.shape, y_train.shape, X_test.shape, y_test.shape
```

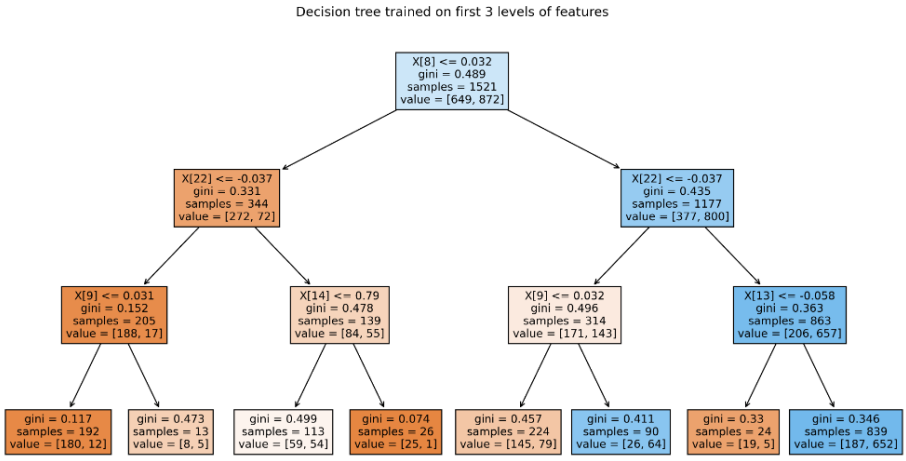
Out[13]: ((1521, 25), (1521,), (508, 25), (508,))

The sklearn DecisionClassifier algorithm is then performed on the training data set. See codeline [14]. In this preliminary DT classification, only a depth of 3 levels is applied. The root node and decision nodes and the branching are shown.

```
In [14]: from matplotlib import pyplot as plt
         from sklearn.tree import DecisionTreeClassifier
         from sklearn.tree import plot_tree

         dt1 = DecisionTreeClassifier(max_depth=3,criterion='gini').fit(X_train, y_train)

         fig, axes = plt.subplots(nrows = 1,ncols = 1,figsize = (15,8), dpi=300)
         plot_tree(dt1, filled=True)
         plt.title("Decision tree trained on first 3 levels of features")
         plt.show()
```



The text version of the tree is shown as follows.

```
In [15]: from sklearn.tree import export_text
         text_representation = export_text(dt1)
         print(text_representation)
```

```
|--- feature_8 <= 0.03
|   |--- feature_22 <= -0.04
|   |   |--- feature_9 <= 0.03
|   |   |   |--- class: 0
|   |   |--- feature_9 >  0.03
|   |   |   |--- class: 0
|   |--- feature_22 >  -0.04
|   |   |--- feature_14 <= 0.79
|   |   |   |--- class: 0
|   |   |--- feature_14 >  0.79
|   |   |   |--- class: 0
|--- feature_8 >  0.03
|   |--- feature_22 <= -0.04
|   |   |--- feature_9 <= 0.03
|   |   |   |--- class: 0
|   |   |--- feature_9 >  0.03
|   |   |   |--- class: 1
|   |--- feature_22 >  -0.04
|   |   |--- feature_13 <= -0.06
|   |   |   |--- class: 0
|   |   |--- feature_13 >  -0.06
|   |   |   |--- class: 1
```

It is seen that the Gini impurity (loss function) value at the root node is 0.489 with a total sample of 1521 in the training sample. The first decision rule at the root node is found in the feature8 – Operating Profit Margin – a key accounting variable of a firm. The rule based on the scaled number is that if its value < 0.032, then go to the left branch, if not go to the right branch.

The Gini impurity calculated based on the left branch now is 0.331 with a sample value [272,72], i.e., 272 cases of Class 0 from the root node of 649 and 72 cases of class 1 from the root node of 872. The Gini impurity calculated based on the right branch now is 0.435 with a sample value [377,800], i.e., 377 cases of Class 0 from the root node of 649 and 800 cases of class 1 from the root node of 872. The mean (weighted average) Gini impurity in the first depth level after the root node is (344/1521) × 0.331 + (1177/1521) × 0.435 = 0.4115, representing a Gini impurity decrease of 0.489 – 0.4115 = 0.0775. This feature X[8] at the split of 0.032, amongst all features and their split, has the largest

decrease in Gini impurity. This is thus selected as the first node (root node) decision rule.

It is seen that the splitting seems to push more class 1 to the right sub-trees. Intuitively lower operating profit margin indicates a weaker firm and hence to the left there are now more firms in "0" class than in "1" class.

On the left branch, the next decision node is based on feature22 "Enterprise Value Multiple" – which is EV/EBITA, the total value of a company (EV) (market cap plus debts and cash) relative to its earnings before interest, taxes, depreciation, and amortization (EBITDA). Lower multiple may indicate a weaker firm (this may conversely indicate a better value for acquisition). If this multiple is <= -0.037 (recall this is standard-scaled), then the branch goes to the next level on the left with sample value [188,17] – a high proportion of class 0 versus class 1. There is a further Gini impurity reduction of 0.331 – 0.152 = 0.179. The average/mean Gini impurity decrease, or Gini importance is the (relative sample size) weighted average of the Gini impurity decreases for both sides of the branching. Basically, as discussed, the decision node decides on the feature that yields the largest weighted average Gini importance.

If we use only this low depth of 3, then the testing results may not be accurate. For example, our decision rules would be (see the left most leaf on the third level) predicting "0" (since 180 versus 12 is obviously a higher odds for "0") if X[9] <=0.031, X[22]<=-0.037, and X[8]<=0.032, and so on for the other rules.

In code line [17], we develop the full DT without constraining the depth – letting the tree nodes expand till all leaves are pure. All features are also considered in each level of splitting with the greedy algorithm seeking the one with largest weighted average Gini impurity decrease. The DT text codes are shown in [18] and the structure of the DT is shown graphically in [20] with 22 levels of depth.

Code lines [17] – [26] show an accuracy of 70.275%, precision of 76% and recall of 72% for the case of class 1, and AUC of 70.02%.

5.3 Random Forest

Random Forest

Random Forest (RF) or Random decision trees is an ensemble (collection or group) method for classification as well as regression prediction. In a RF,

many, e.g., 500, DTs are run independently (or in parallel). Each of the 500 DTs is done on a sample bootstrap aggregation ("bagging"), i.e., if the training sample is 1521, after one DT is done as seen earlier, the next DT is constructed by resampling on the 1521 sample with replacement, i.e., some features cases/firms may appear more than once in a resampled sample of 1521.

Each of the trained 500 DTs gives its possibly different set of decision rules due to the resampling within the training data set and random selection of a smaller set of features. The X_train, and correspondingly y_train, changes for each re-sampling. Hence there are 500 different DTs and thus 500 different sets of decision rules. These are applied to the fixed 508 firms in y_train and y_test.

In the testing, each of the 500 trained DTs will, using its own trained decision rules, yield its prediction vector on the test sample of the same 508 firms. Each of these prediction vector from each of the 500 DTs are possibly different.

Suppose the trained 500 DTs produce 500 vectors of the 508 × 1 predictions on the 508 test set firms' target labels of "0"s or "1"s. Suppose their sum is shown as follows.

Sum of predicted test cases:

$$\begin{bmatrix}1\\0\\0\\1\\0\\1\\0\\\vdots\\1\end{bmatrix}+\begin{bmatrix}1\\1\\0\\1\\0\\0\\0\\\vdots\\1\end{bmatrix}+\begin{bmatrix}0\\0\\0\\1\\0\\1\\1\\\vdots\\0\end{bmatrix}+\cdots\cdots+\begin{bmatrix}1\\0\\1\\0\\0\\1\\0\\\vdots\\1\end{bmatrix}=\begin{bmatrix}312\\145\\98\\436\\24\\210\\56\\\vdots\\396\end{bmatrix}_{508\times1}$$

Dividing the 508 × 1 vector by 500 to show the averaged "vote" or probability of being "1"s – we obtain:

$$P=\begin{bmatrix}0.624\\0.29\\0.196\\0.872\\0.048\\0.42\\0.112\\\vdots\\0.792\end{bmatrix}$$

This means that the trained 500 DTs predicted, on averaging, based on the test set, that the first firm in the test set has predicted probability of "1" at probability 0.624. The second firm in the test set has predicted probability of "1" at probability of 0.29, and so on. Using probability > 0.5 as prediction of "1", if not "0", the prediction of the first firm is "1" and that of the second firm is "0" and so on.

Using the actual test y-value or actual firm's "1", "0" status, the confusion matrix of the RF can thus be produced, and the various prediction performance metrics can be computed. With a different threshold, e.g., 0.65, using probability > 0.65 as prediction of "1", if not "0", the prediction of the first firm is "0" and that of the second firm is "0" and so on. With a threshold, e.g., 0.25, using probability > 0.25 as prediction of "1", if not "0", the prediction of the first firm is "1" and that of the second firm is "1" and so on. Hence with different thresholds, and thus different corresponding confusion matrix, the ROC and its AUC can be computed.

In a RF Regression context, each predicted vector (508 × 1 vector of values) of the test set sample, based on each of the 500 trained DTs, is noted and together they are averaged to find the RF predicted value (508 × 1 vector). Just like an orchestra, an ensemble produces good accuracy even if individual DTs may falter or are weak performers. The averaging can produce better results as it reduces individual DT variance in the prediction outcomes.

We continue with the credit rating accounting data set used earlier in Section 5.2. First, we apply the Random Forest method. The sample is similarly split into 25% test size and 75% training size. The following shows code lines from Chapter5-2.ipynb file.

```
In [12]: Train, Test = train_test_split(TT,
                                        test_size=0.25,
                                        random_state=0)

         X_train, y_train = Train.iloc[:,1:26], Train.iloc[:,0] ### choose the 25 features
         X_test, y_test = Test.iloc[:,1:26], Test.iloc[:,0]

In [13]: X_train.shape, y_train.shape, X_test.shape, y_test.shape

Out[13]: ((1521, 25), (1521,), (508, 25), (508,))
```

Sklearn RandomForestClassifier is used with specification of 500 DTs (n_estimators = 500). RF method, besides bagging, also possibly randomizes by choosing a subset of the total set of 25 features for each DT. We set max_features = 'sqrt' which means that each DT randomly uses $\sqrt{25} = 5$ features of the 25 total in constructing the branching in each of the 500 DTs. Therefore, it is important to have many DTs or n_estimators. See code line[14].

```
In [14]: # Random Forest
         from sklearn.ensemble import RandomForestClassifier

         RF_model = RandomForestClassifier(n_estimators=500, random_state=1, max_features="sqrt", max_depth=34)
         ### default n_estimators/trees = 100; default criterion = 'gini'
         ### max_features default case is "sqrt"; unlike DecisionTree Classifier where default case = "none"
         ### If max_depth=None, then nodes are expanded until all leaves are pure or until all leaves contain less than
         ###   min_samples_split samples -- usu 2
         ### bootsrapping - bagging, default=true
         ### oob_scorebool, default=False -- Whether to use out-of-bag samples to estimate the generalization score.
         ###    Only available if bootstrap=True.
         RF_model.fit(X_train,y_train)

         y_pred_RF = RF_model.predict(X_test)
         Accuracy_RF = metrics.accuracy_score(y_test, y_pred_RF)
         print("RF Accuracy:",Accuracy_RF)

         RF Accuracy: 0.7952755905511811
```

In this investment grade/speculative grade prediction problem, the accuracy of the RF here is 79.53%. The Confusion matrix and the prediction performance classification report are shown as follows.

```
In [15]: from sklearn.metrics import confusion_matrix
         confusion_matrix = confusion_matrix(y_test, y_pred_RF)
         print(confusion_matrix)

         [[153  62]
          [ 42 251]]
```

```
In [16]: from sklearn.metrics import classification_report
         print(classification_report(y_test, y_pred_RF))
```

	precision	recall	f1-score	support
0	0.78	0.71	0.75	215
1	0.80	0.86	0.83	293
accuracy			0.80	508
macro avg	0.79	0.78	0.79	508
weighted avg	0.79	0.80	0.79	508

```
In [17]: ###       Assuming your target is (0,1), then the classifier would output a probability matrix of dimension (N,2).
         ###       The first index refers to the probability that the data belong to class 0, and the second refers to the probability
         ###           hat the data belong to class 1. Each row sums to 1.
         ### "RF_model.predict_proba(X_test)[:,1]" will print a single array of 508 numbers - the second column or the predicted prob
         ###    of test cases being 1.
         ### "RF_model.predict_proba(X_test)" will print array([[0.242, 0.758], [0.134, 0.866], ..... for 508 pairs
```

```
In [18]: import sklearn.metrics as metrics
         # calculate the fpr and tpr for all thresholds of the classification
         preds_RF = RF_model.predict_proba(X_test)[:,1]

         fpr, tpr, thresholds = metrics.roc_curve(y_test, preds_RF)
             ### matches y_test of 1's and 0's versus pred prob of 1's for each of the 508 test cases
             ###    sklearn.metrics.roc_curve(y_true, y_score,...) requires y_true as 0,1 input and y_score as prob inputs
             ###    this metrics.roc_curve returns fpr, tpr, thresholds (Decreasing thresholds used to compute fpr and tpr)
         roc_auc_RF = metrics.auc(fpr, tpr)
             ### sklearn.metrics.auc(fpr,tpr) returns AUC using trapezoidal rule
         roc_auc_RF
Out[18]: 0.8868640368283197
```

Compare for case y=1, precision of 80% and recall of 86% in Random Forest versus 76% and 72% for a single DT. The predicted probability of class 0 for each of the 508 test cases is given by code line [19].

```
In [19]: print(preds_RF)
```

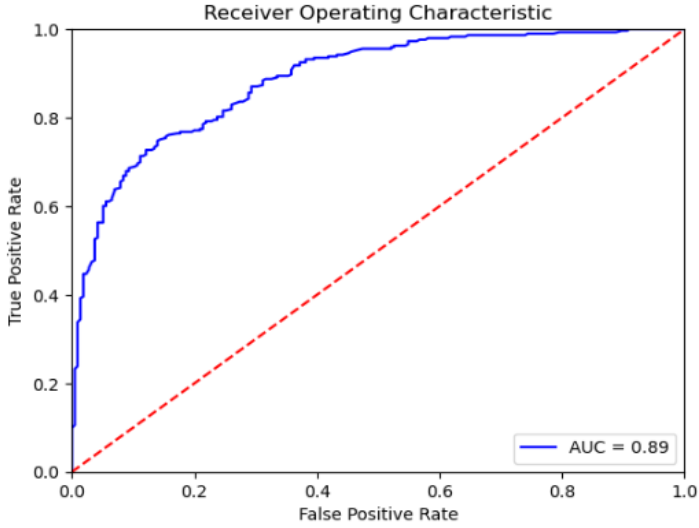
```
[0.758 0.866 0.49  0.508 0.98  0.888 0.836 0.722 0.628 0.82  0.6   0.816
 0.712 0.798 0.78  0.508 0.25  0.878 0.348 0.138 0.63  0.954 0.368 0.96
 0.522 0.634 0.752 0.76  0.204 0.23  0.794 0.36  0.354 0.734 0.636 0.156
 0.862 0.14  0.652 0.872 0.892 0.868 0.286 0.21  0.894 0.162 0.944 0.736
 0.674 0.58  0.836 0.938 0.794 0.636 0.226 0.68  0.534 0.718 0.882 0.552
 0.866 0.088 0.73  0.432 0.156 0.598 0.398 0.268 0.936 0.922 0.464 0.33
 0.834 0.968 0.096 0.386 0.796 0.452 0.846 0.872 0.966 0.046 0.94  0.774
 0.664 0.78  0.948 0.688 0.32  0.59  0.772 0.148 0.594 0.05  0.502 0.09
 0.328 0.238 0.936 0.822 0.77  0.286 0.872 0.55  0.78  0.738 0.294 0.848
```

```
 0.946 0.836 0.832 0.884 0.758 0.75  0.236 0.64  0.916 0.632 0.694 0.936
 0.686 0.522 0.334 0.154 0.834 0.888 0.248 0.114 0.574 0.64  0.732 0.446
 0.768 0.498 0.756 0.306 0.982 0.58  0.644 0.722 0.568 0.722 0.984 0.968
 0.68  0.784 0.158 0.744 0.808 0.974 0.202 0.908 0.354 0.376 0.27  0.302
 0.318 0.086 0.472 0.226 0.076 0.466 0.336 0.25  0.706 0.822 0.458 0.184
 0.826 0.348 0.954 0.786 0.474 0.082 0.112 0.916 0.918 0.544 0.856 0.912
 0.674 0.73  0.838 0.752 0.506 0.612 0.484 0.75  0.906 0.328 0.776 0.636
 0.672 0.87  0.3   0.522 0.11  0.708 0.764 0.466 0.058 0.778 0.528 0.782
 0.532 0.938 0.9   0.22  0.134 0.46  0.5   0.144 0.798 0.734 0.19  0.752
 0.502 0.63  0.958 0.934 0.672 0.994 0.956 0.552 0.752 0.268 0.3   0.868
 0.782 0.048 0.566 0.622 0.69  0.586 0.458 0.718 0.568 0.554 0.466 0.96
 0.722 0.734 0.164 0.044 0.934 0.818 0.836 0.128 0.828 0.34  0.218 0.312
 0.276 0.836 0.908 0.08  0.068 0.188 0.246 0.068 0.34  0.738 0.844 0.798
 0.874 0.74  0.558 0.696 0.654 0.262 0.536 0.86  0.164 0.632 0.644 0.288
 0.11  0.814 0.39  0.46  0.496 0.694 0.668 0.448 0.45  0.522 0.826 0.152
 0.28  0.816 0.798 0.136 0.456 0.778 0.056 0.172 0.682 0.456 0.882 0.546
 0.904 0.198 0.658 0.176 0.706 0.248 0.838 0.874 0.274 0.93  0.152 0.678
 0.966 0.91  0.358 0.04  0.578 0.556 0.48  0.326 0.93  0.254 0.358 0.792
 0.536 0.378 0.8   0.454 0.864 0.716 0.364 0.808 0.966 0.226 0.964 0.698
 0.772 0.894 0.406 0.402 0.07  0.638 0.378 0.654 0.172 0.358 0.912 0.102
 0.154 0.452 0.94  0.072 0.858 0.152 0.832 0.1   0.132 0.888 0.476 0.51

0.978 0.486 0.99  0.898 0.822 0.212 0.738 0.958 0.766 0.6   0.948 0.882
0.512 0.726 0.452 0.722 0.69  0.602 0.842 0.162 0.11  0.386 0.066 0.536
0.862 0.218 0.062 0.344 0.834 0.858 0.474 0.832 0.184 0.842 0.272 0.44
0.156 0.974 0.788 0.87  0.676 0.924 0.172 0.928 0.452 0.134 0.608 0.294
0.966 0.612 0.718 0.51  0.64  0.052 0.63  0.57  0.442 0.234 0.502 0.794
0.63  0.838 0.828 0.486 0.872 0.89  0.374 0.792 0.488 0.292 0.348 0.712
0.96  0.37  0.344 0.454 0.582 0.754 0.714 0.856 0.258 0.69  0.248 0.27
0.684 0.794 0.202 0.442 0.522 0.704 0.836 0.778 0.342 0.782 0.7   0.912
0.774 0.454 0.596 0.904 0.562 0.816 0.72  0.824 0.988 0.214 0.82  0.344
0.468 0.494 0.214 0.586 0.14  0.684 0.918 0.494 0.94  0.83  0.276 0.708
0.922 0.876 0.698 0.118 0.638 0.314 0.782 0.688 0.698 0.24  0.978 0.238
0.44  0.722 0.222 0.264 0.772 0.852 0.12  0.874 0.45  0.366 0.928 0.486
0.794 0.834 0.254 0.14 ]
```

The probabilities allow computation of the Receiver Operating Characteristic curve when the threshold for predicting the class is varied from 0 to 1. See code line [20].

```
In [20]: import matplotlib.pyplot as plt
         plt.title('Receiver Operating Characteristic')
         plt.plot(fpr, tpr, 'b', label = 'AUC = %0.2f' % roc_auc_RF)
         plt.legend(loc = 'lower right')
         plt.plot([0, 1], [0, 1],'r--')
         plt.xlim([0, 1])
         plt.ylim([0, 1])
         plt.ylabel('True Positive Rate')
         plt.xlabel('False Positive Rate')
         plt.show()
```



With RF, the AUC is 88.69%. This compares well with 70.02% AUC for a single DT.

With RF, we can also analyse the "Feature Importance". For each of the 500 DTs, the features that are involved in the splitting and hence Gini importance (mean decrease in Gini impurity) from the decision node down to the next two branched nodes are noted.

Over all the 500 DTs, the total Gini importance contributed by each of the 25 features are tabulated. This feature total is then divided by the total across the totals of the 25 features.

Note that for RF Regression, the contribution of each feature is the reduction of the loss function of variance or mean squared error. The higher the % of each feature in this contribution to overall Gini importance or overall Gini impurity decrease, the more important is this feature in the RF algorithm. Examining such features is called Feature Importance – this is shown as follows in code lines [21]-[22].

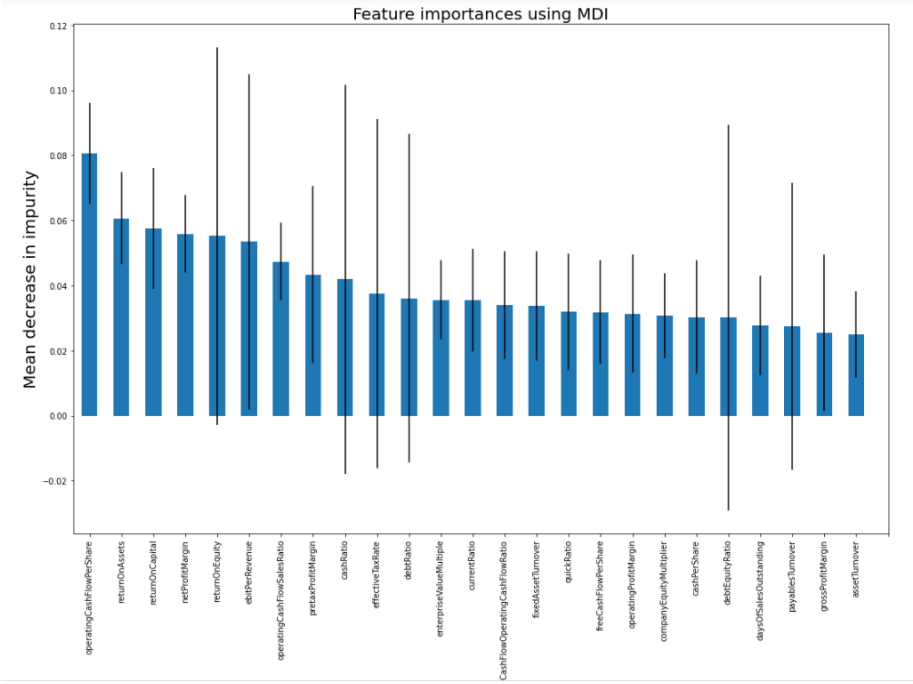
```
In [21]: import pandas as pd

forest_importances = pd.Series(importances, index=X_train.columns)
forest_importances.sort_values(ascending=False,inplace=True)

fig, ax = plt.subplots(figsize=(18,15))
forest_importances.plot.bar(yerr=std, ax=ax)
### thin line indicates 1 std err from the mean either way -- doesn't mean mean decrease is neg
ax.set_title("Feature importances using MDI",fontsize=20)  ### MDI is mean decrease in impurity
ax.set_ylabel("Mean decrease in impurity",fontsize=20)
ax.set(xticks=([ 0, 1, 2, 3, 4, 5, 6, 7, 8, 9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,
 24,25]))
### define own x-ticks to avoid clutter, entry must be a list
fig.tight_layout()

### Note: Negative feature importance value means that feature makes the loss go up.
### either model is underfitting with not enough iteration and not enough splitting use of feature or feature should be removed
```

The output shows that in order of importance (the mean decrease in impurity in this app is expressed as a %), the features are Operating Cashflow per share, Return on Assets, Return on Capital, Net Profit Margin, Return on Equity, EBIT per Revenue, and so on. These are key accounting variables/ratios of the firm that largely determine the credit quality and hence if the firm is of investment grade or of speculative grade.



When RF is applied, it reduces the variance of outcomes in a single DT. RF also randomizes selection of features. RF thus handles high dimensionality

well as it can reduce use of all features; it also handles missing values since any missing value amounts to having not selected that feature for that row. However, a RF prediction outcome is also harder to explain or interpret since there is no fixed set of features and weights as it is a collection of different DTs.

There is another popular ensemble technique to create an ensemble or collection of predictors. The idea is also to construct many DTs – but the difference is that the target variable for each subsequent DT is different. The idea with each subsequent DT is to solve for the net error left from the past DTs and to build stronger and more accurate DT starting from weaker ones. The latter technique is called gradient boosting and is covered in the next chapter.

References

Aurelien Geron, (2019), "Hands-on Machine Learning with Scikit-Learn, Keras & TensorFlow", 2nd ed., O'Reilly Publisher.
Marcos Lopez De Prado (2018), "Advances in Financial Machine Learning", Wiley.
https://medium.com/the-artificial-impostor/feature-importance-measures-for-tree-models-part-i-47f187c1a2c3