QF634 APPLIED QUANTITATIVE RESEARCH METHODS
LECTURE 5

Lecturer: Prof Emeritus Lim Kian Guan

Ensemble Prediction

Decision Trees, Random Forest

DECISION TREES

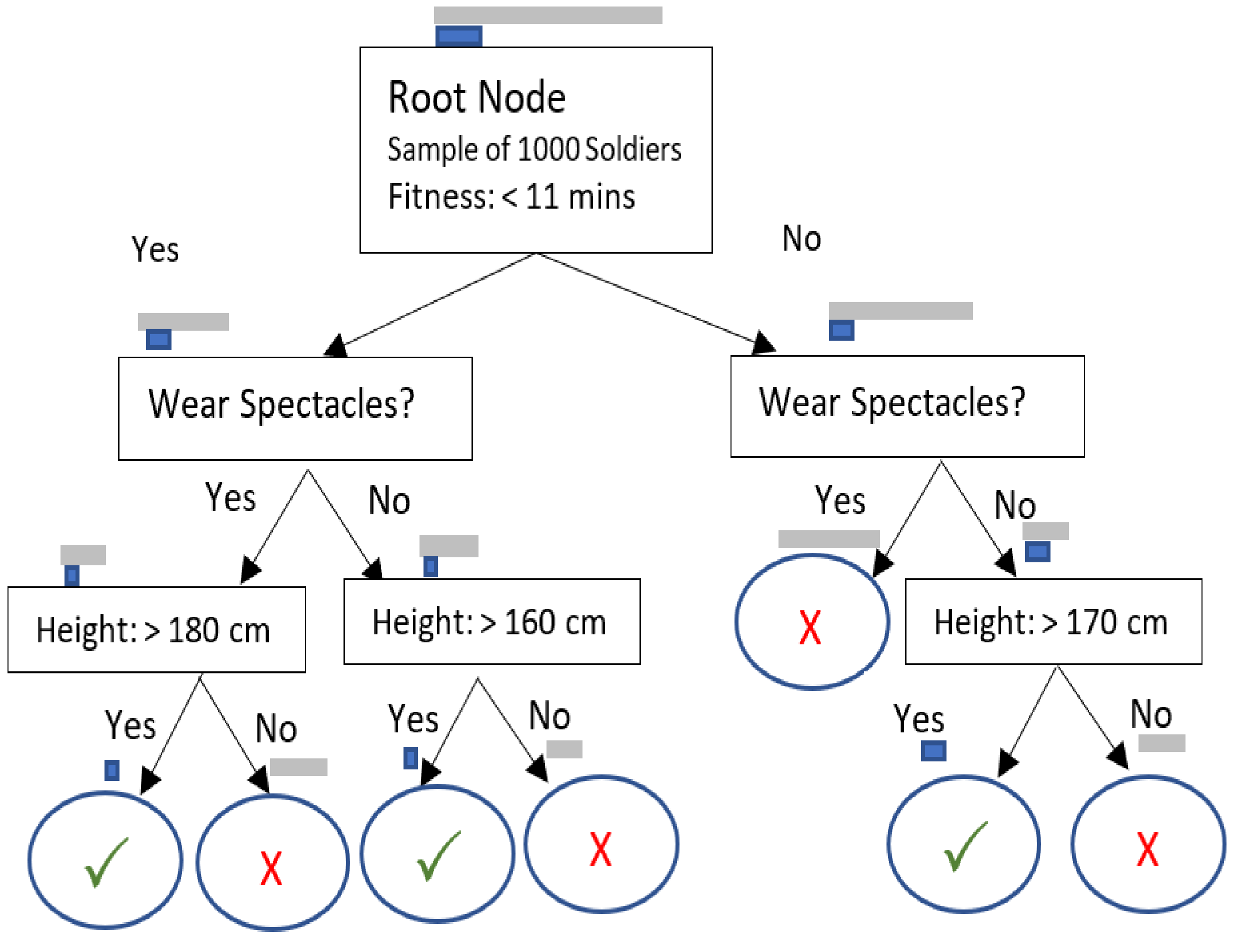
- Nonparametric estimation is a statistical method that estimates the functional form of data fits when parametric modeling is absent and where there is no guidance from theory. Examples of nonparametric estimation are neural networks with layers of nonlinear activation functions on weighted features 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 supervised learning 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.

DECISION TREES

- DT 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.
- 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.
- 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), 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.

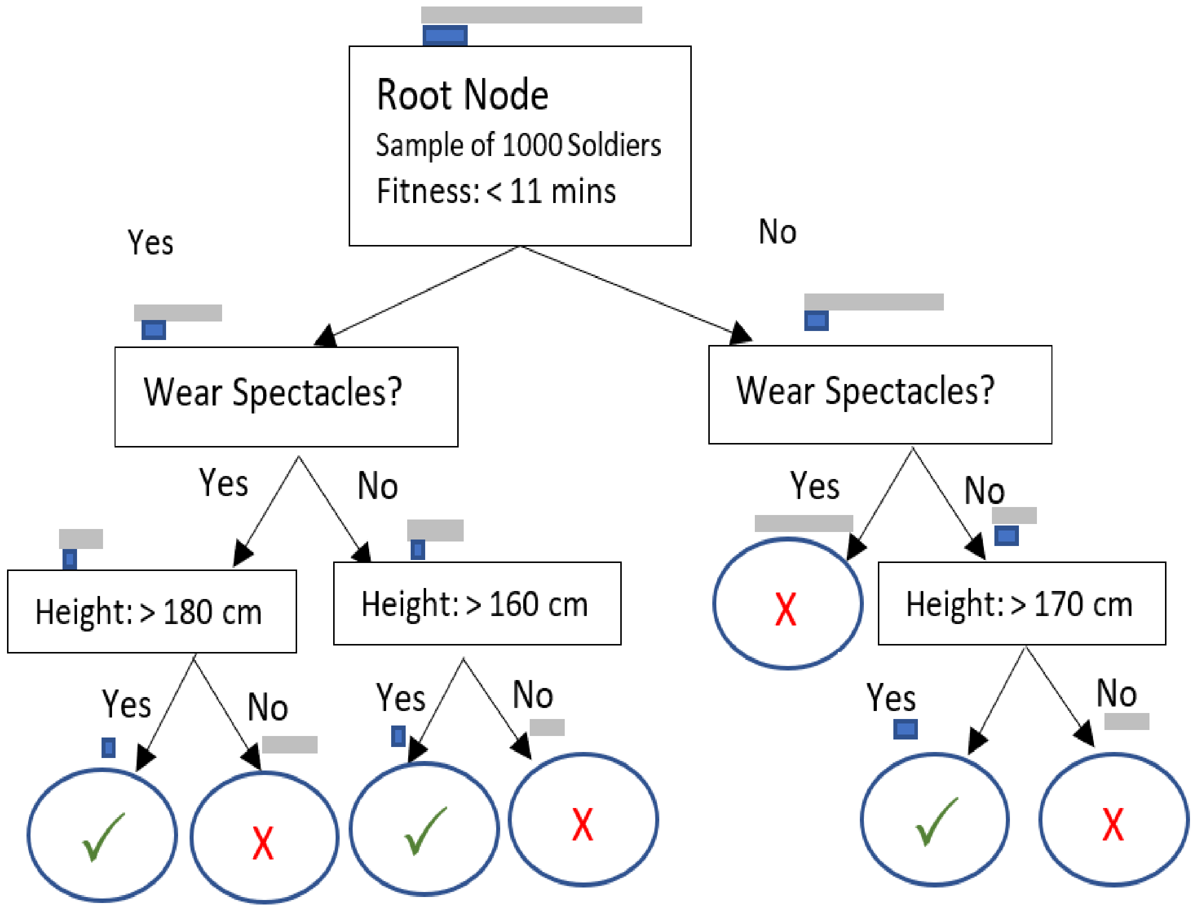
DECISION TREES

- 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).



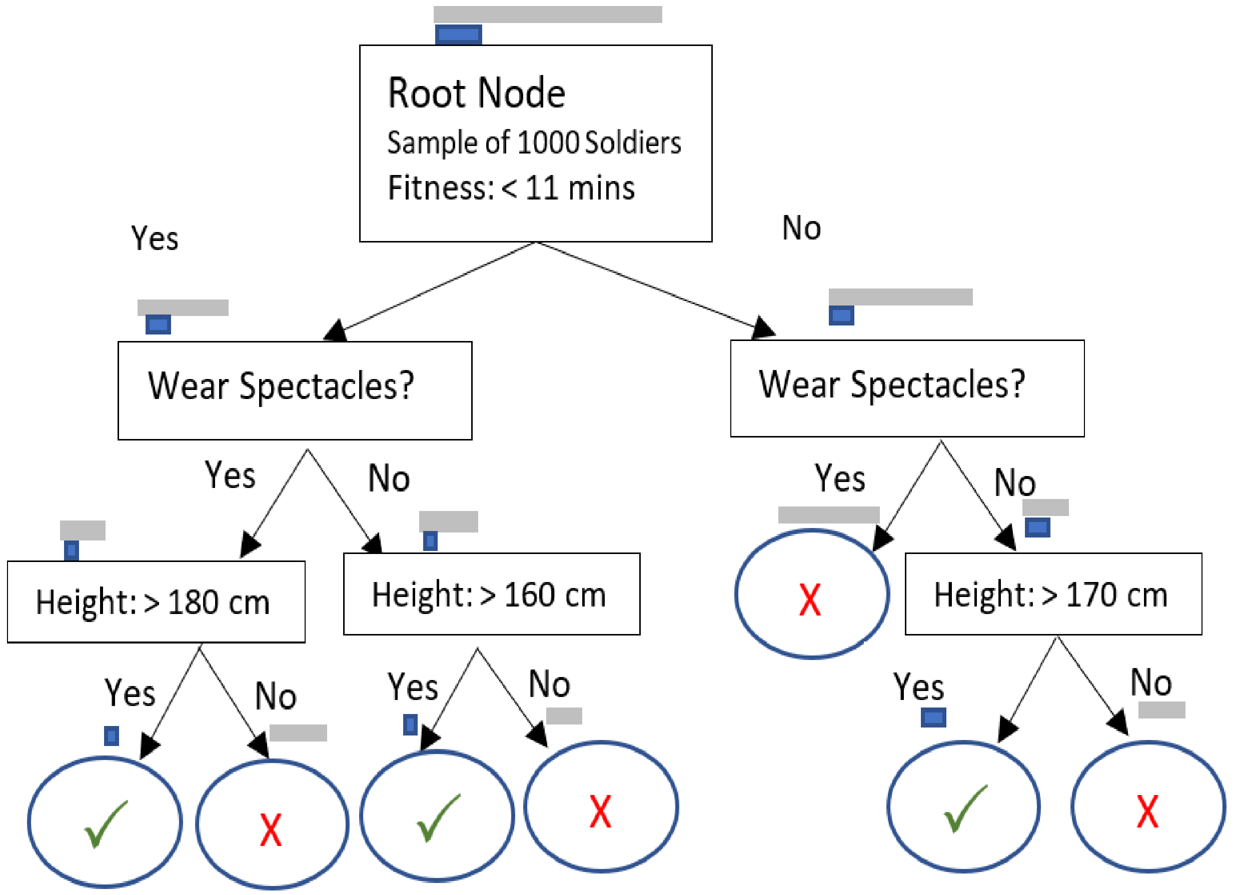
DECISION TREES

- 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 levels (lower) of decision nodes until the leaf or terminal nodes are reached.
- At each leaf (end) node, the model makes a prediction.



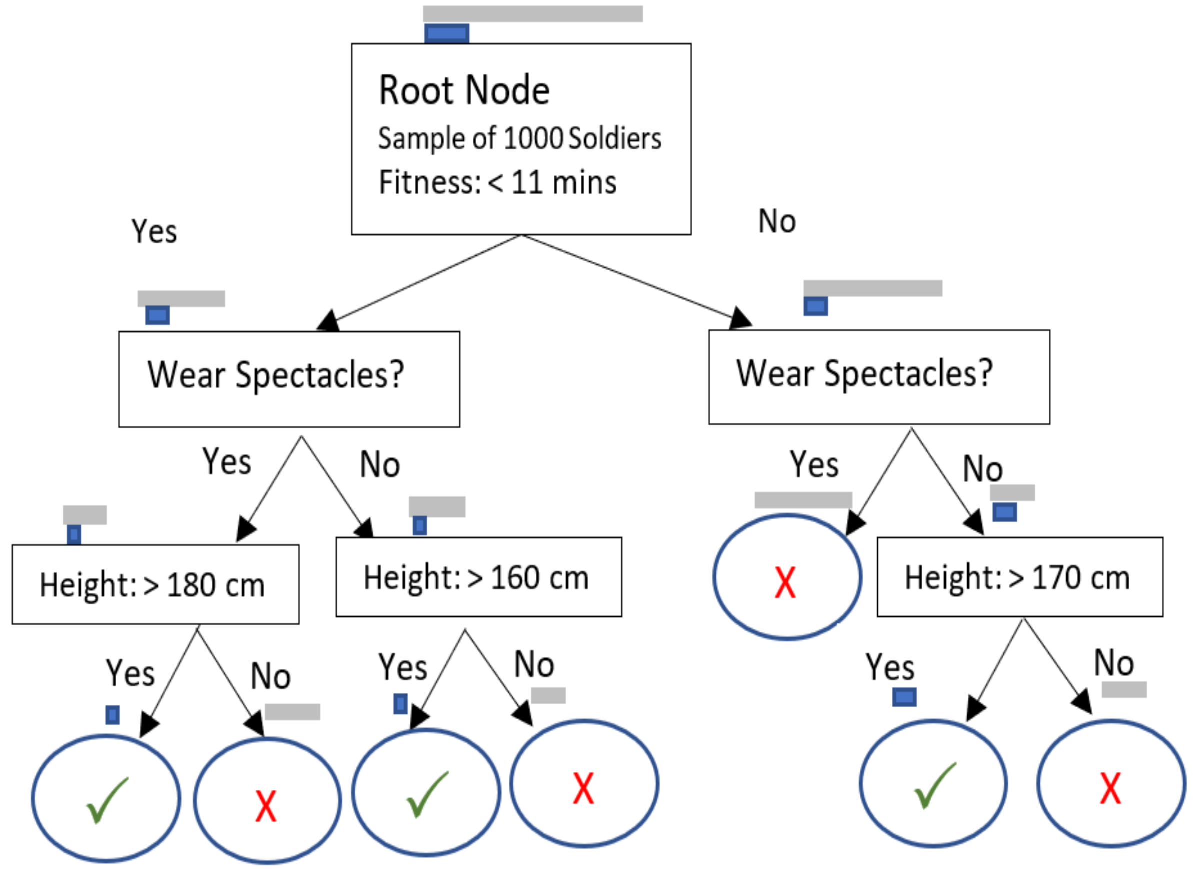
DECISION TREES

- 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.

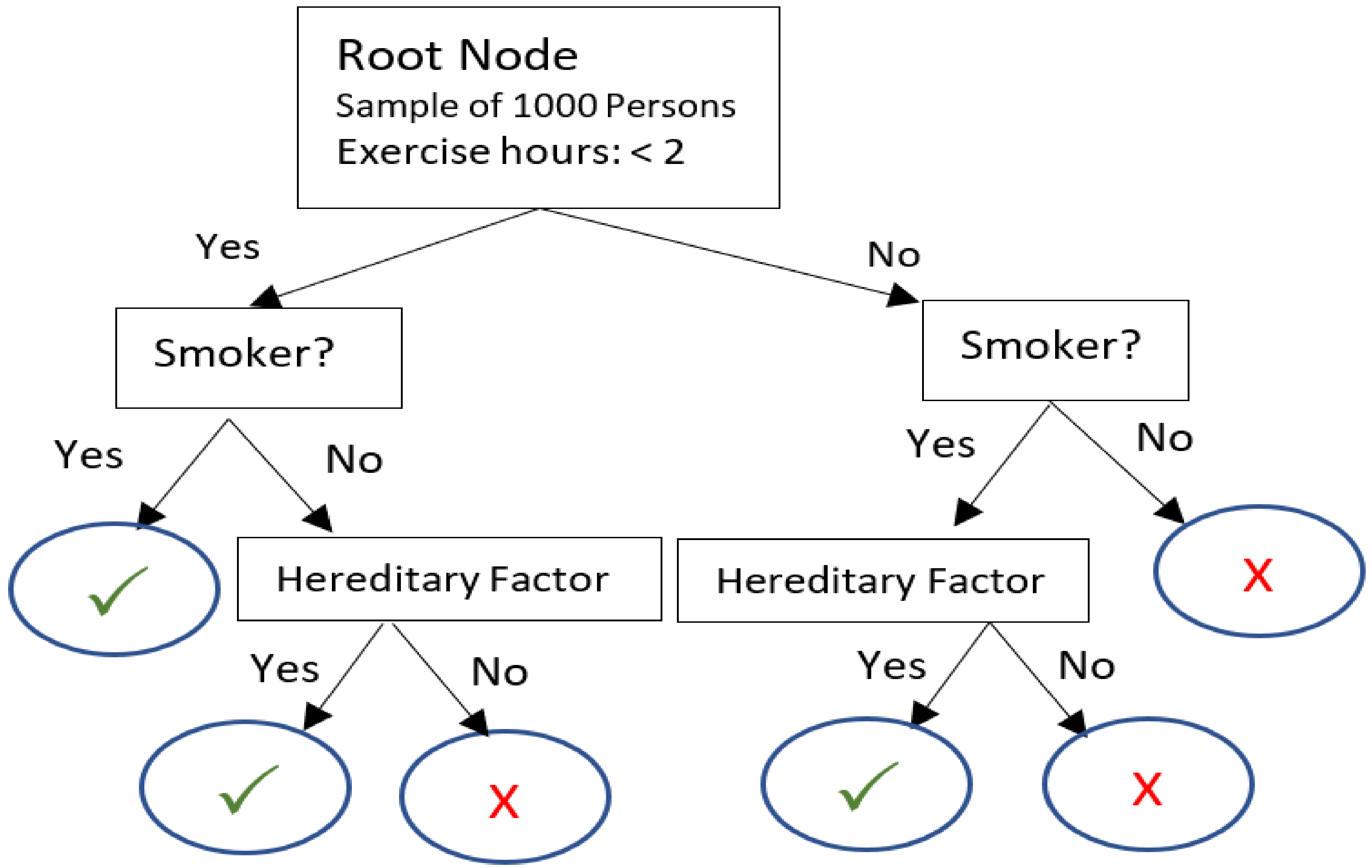


DECISION TREES

- 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 hyper-parameters 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, 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.



DECISION TREES – another example



DECISION TREES – How it Splits?

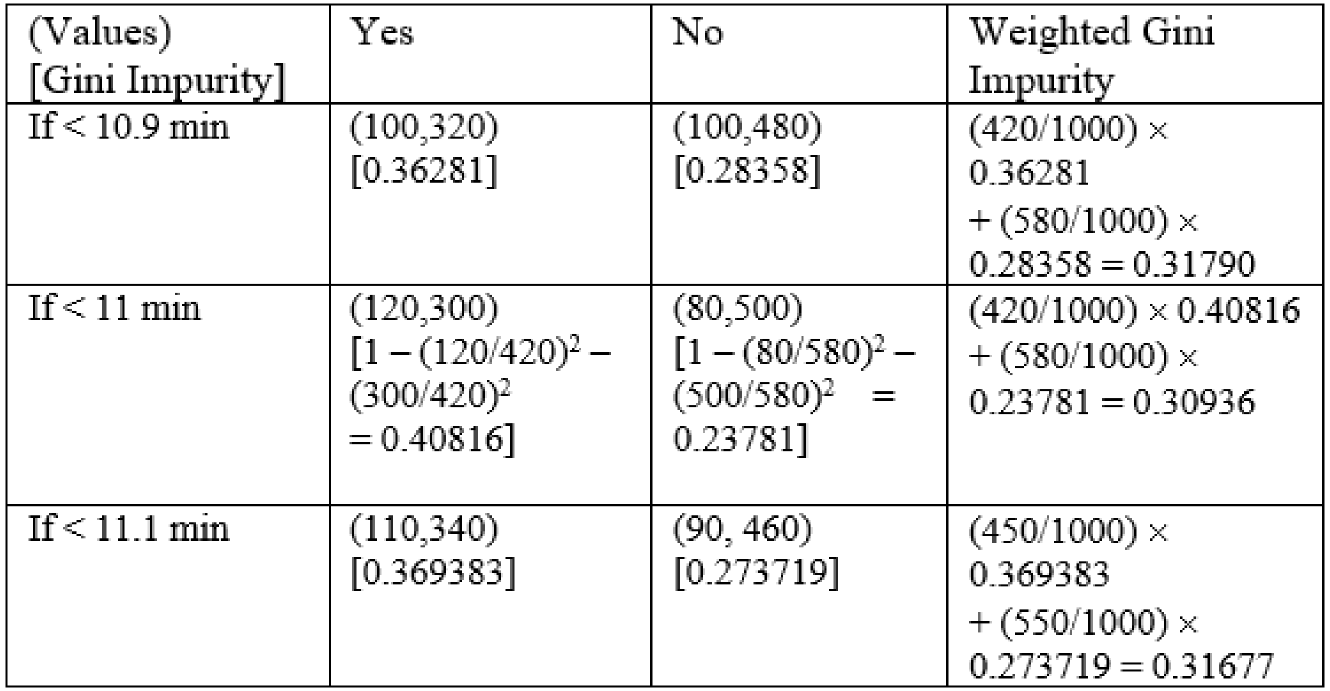
- At the root node, the sample (size N) provides a measure of Gini impurity that is defined as $1 - \sum_{k=1}^{m} p_k^2$ where m is the number of different categories in the sample and p_k is the empirical probability of being in category k.
- Thus, in the marksmen case, if the original sample has 200 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.

DECISION TREES – How it Splits?

- 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)
- The discrete points – potential split point could be the mid-points of adjacent discrete values of the feature.

DECISION TREES – How it Splits?

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



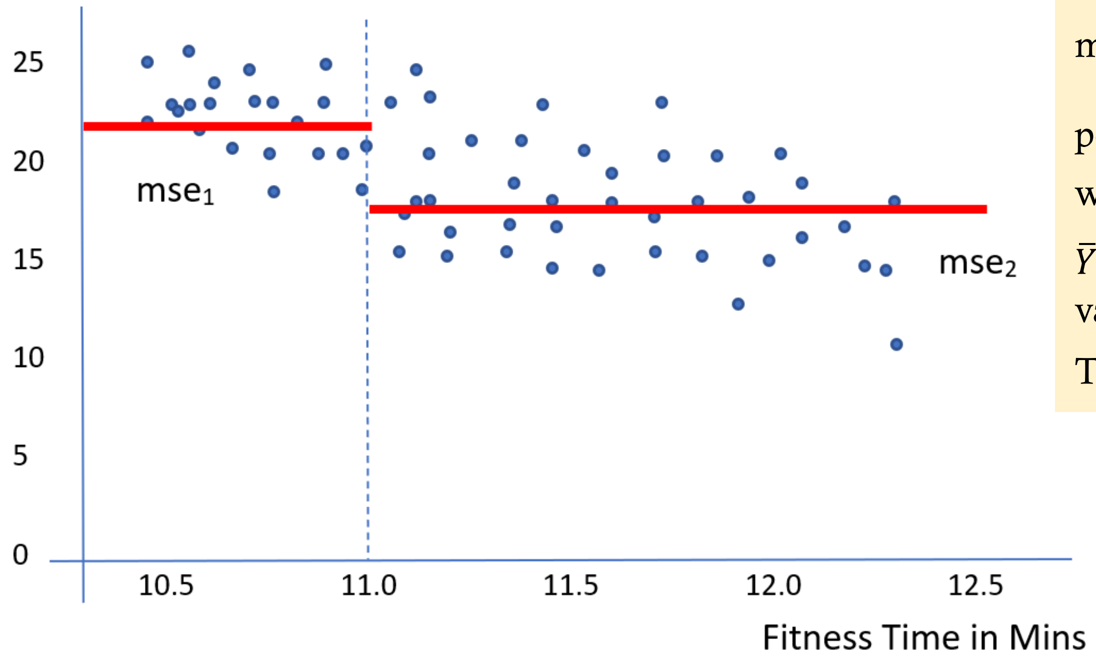
(Values) [Gini Impurity]	Yes	No	Weighted Gini Impurity
If < 10.9 min	(100,320) [0.36281]	(100,480) [0.28358]	$(420/1000) \times 0.36281 + (580/1000) \times 0.28358 = 0.31790$
If < 11 min	(120,300) $[1 - (120/420)^2 - (300/420)^2 = 0.40816]$	(80,500) $[1 - (80/580)^2 - (500/580)^2 = 0.23781]$	$(420/1000) \times 0.40816 + (580/1000) \times 0.23781 = 0.30936$
If < 11.1 min	(110,340) [0.369383]	(90, 460) [0.273719]	$(450/1000) \times 0.369383 + (550/1000) \times 0.273719 = 0.31677$

DECISION TREES – How it Splits?

- 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.
- 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. It is a “greedy” algorithm.
- 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 the testing, the test sample point labels/types are not used.

Continuous Variable DT

- 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. 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.



Suppose at Fitness < 11.0 min, we obtain two partitions. On the left we have mean square error $\text{mse}_1 = \frac{\sum_{j=1}^{L}(Y_j - \bar{Y}^L)^2}{L}$ where L is the number of sample points with feature Fitness < 11.0 min and on the right $\text{mse}_2 = \frac{\sum_{j=1}^{R}(Y_j - \bar{Y}^R)^2}{R}$ where R is the number of sample points with feature Fitness ≥ 11.0 min.

$\bar{Y}^L$ and $\bar{Y}^R$ are respectively the left sample and right sample means. Weighted variance or loss function in this case is $L/(L+R) \times \text{mse}_1 + R/(L+R) \times \text{mse}_2$. 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.

Continuous Variable DT

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.

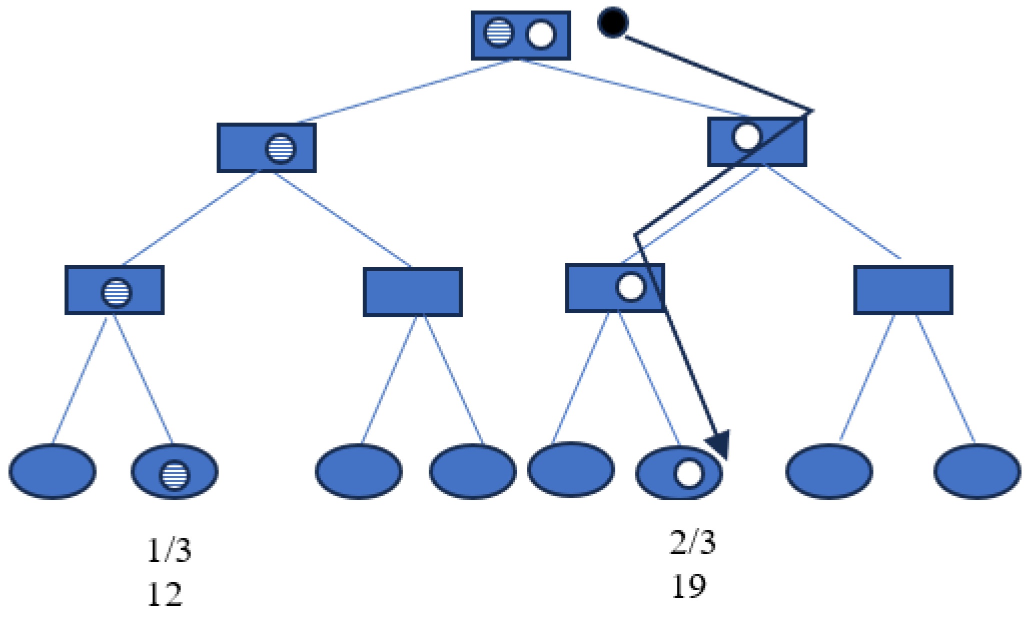
Finally, when all levels are done and the leaf node is on groups with the same shooting score, the DT can then be used to perform prediction of a future soldier’s shooting score based on the new soldier’s features.

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.

Just as in a categorical variable DT, once the DT is trained using the training set, it is then used to predict the outcomes in the test data set. All the test data X-variables (features) are used to decide on the predicted shooting score based on the trained decision rules. In the testing, the test sample point labels/types are not used for the prediction but are used afterward to measure the prediction performance.

Prediction

- The prediction in testing is illustrated as follows. The following is a trained decision tree showing the root node on top down to the leaf nodes. The DT and decision rules were arrived at during the training by making use of the knowledge of the training set labels.
- A trained sample point from the training data set may be the white circle. Given its features and the decision rules on the features, it ended up in leaf node indicating 2/3 (if it is a categorical variable DT, i.e., label is categorical). The trained DT indicates that the leaf (terminal node) with 2/3 meant there were 2/3 positive cases and 1/3 negative case there in the training set.
- Suppose the black circle is a test sample point or instance, and the decision rules in the fitted or trained tree led it to land on the leaf with 2/3 indicating it has a probability of 2/3 being a positive case. In this case of predicted probability, since the usual threshold is 0.5, predicted Prob (label Y=1) = 2/3 implies it is predicted as category Y=1.



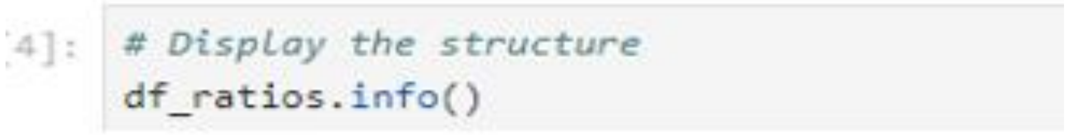
The Prediction Problem

Please upload Chapter5-1.ipynb and follow the computing steps in Jupyter Notebook

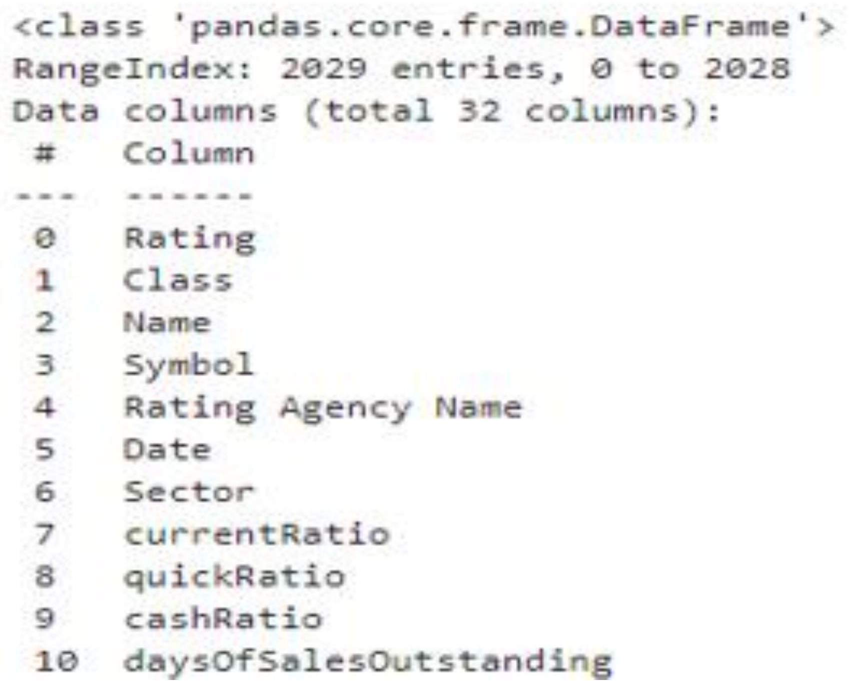
- This data set corporate_rating2.csv is obtained on public website from Kaggle. The dataset contains 2029 credit ratings issued by major agencies from 2010 to 2016 on large US companies. For each credit rating, each company shows 25 accounting features. The other 6 columns in the data set consist of 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 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.
- 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.

Please upload Chapter5-1.ipynb and follow the computing steps in Jupyter Notebook

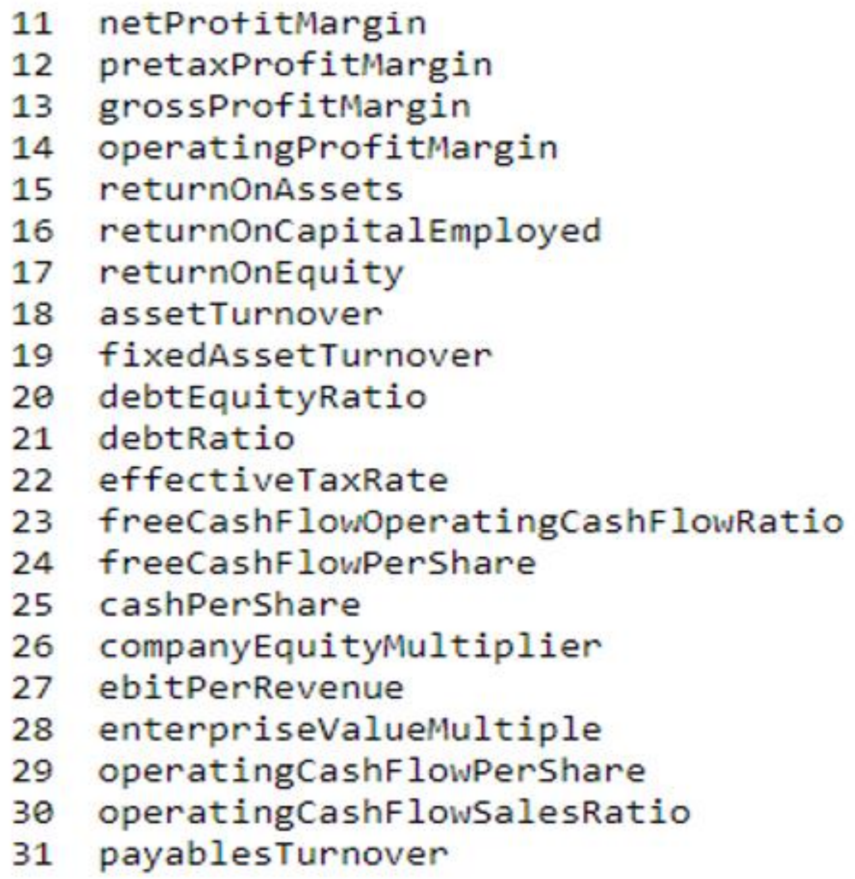
In codeline [4], the features information is shown.



```
4]:  # Display the structure
     df_ratios.info()
```



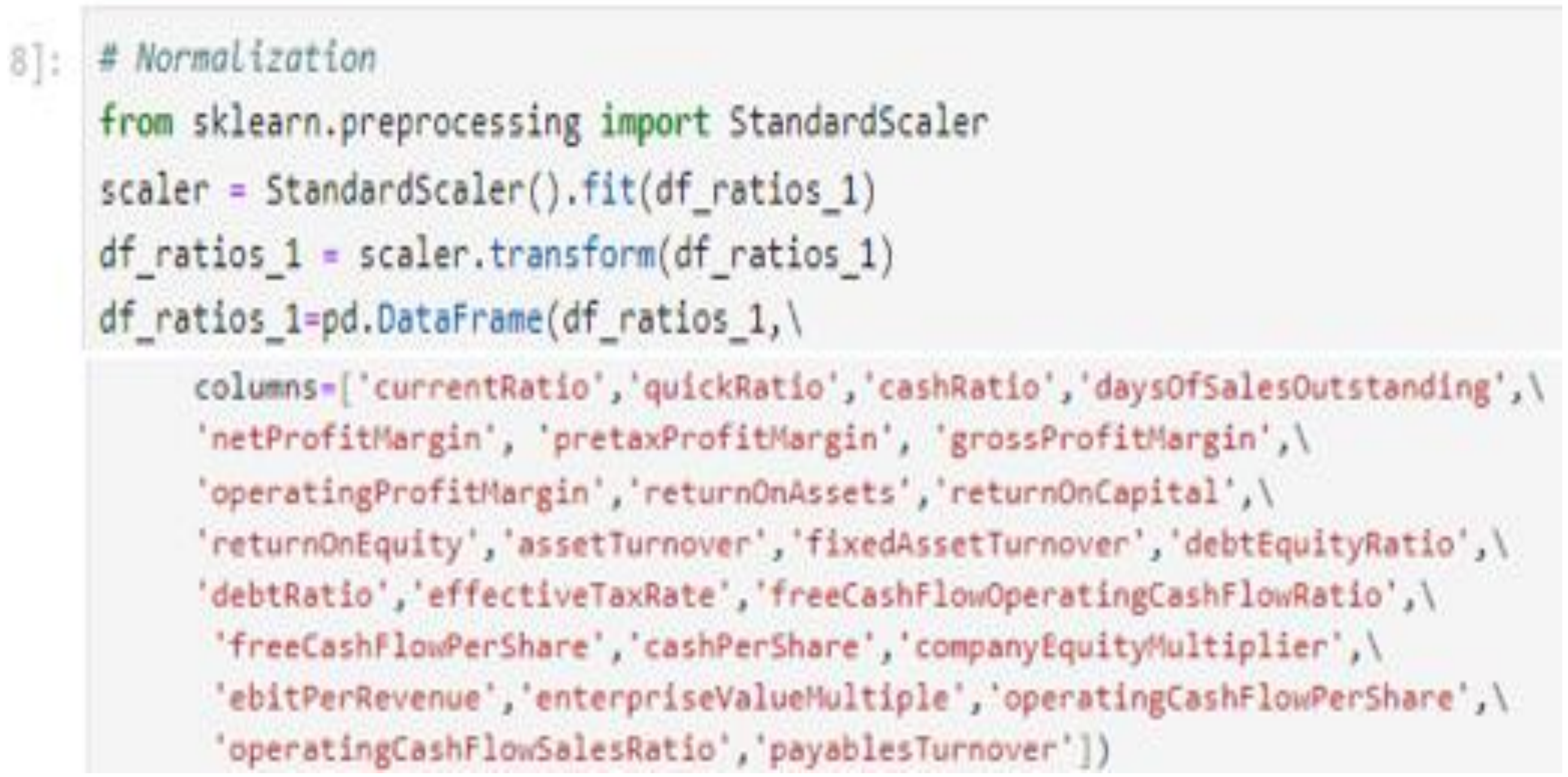
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2029 entries, 0 to 2028
Data columns (total 32 columns):
 #   Column
---  ------
 0   Rating
 1   Class
 2   Name
 3   Symbol
 4   Rating Agency Name
 5   Date
 6   Sector
 7   currentRatio
 8   quickRatio
 9   cashRatio
 10  daysOfSalesOutstanding
```



```
 11  netProfitMargin
 12  pretaxProfitMargin
 13  grossProfitMargin
 14  operatingProfitMargin
 15  returnOnAssets
 16  returnOnCapitalEmployed
 17  returnOnEquity
 18  assetTurnover
 19  fixedAssetTurnover
 20  debtEquityRatio
 21  debtRatio
 22  effectiveTaxRate
 23  freeCashFlowOperatingCashFlowRatio
 24  freeCashFlowPerShare
 25  cashPerShare
 26  companyEquityMultiplier
 27  ebitPerRevenue
 28  enterpriseValueMultiple
 29  operatingCashFlowPerShare
 30  operatingCashFlowSalesRatio
 31  payablesTurnover
```

Please upload Chapter5-1.ipynb and follow the computing steps in Jupyter Notebook

Standard scaling is done in code line [8] on the features, excepting the Class target variable. The scaling reduces the variances of the features.

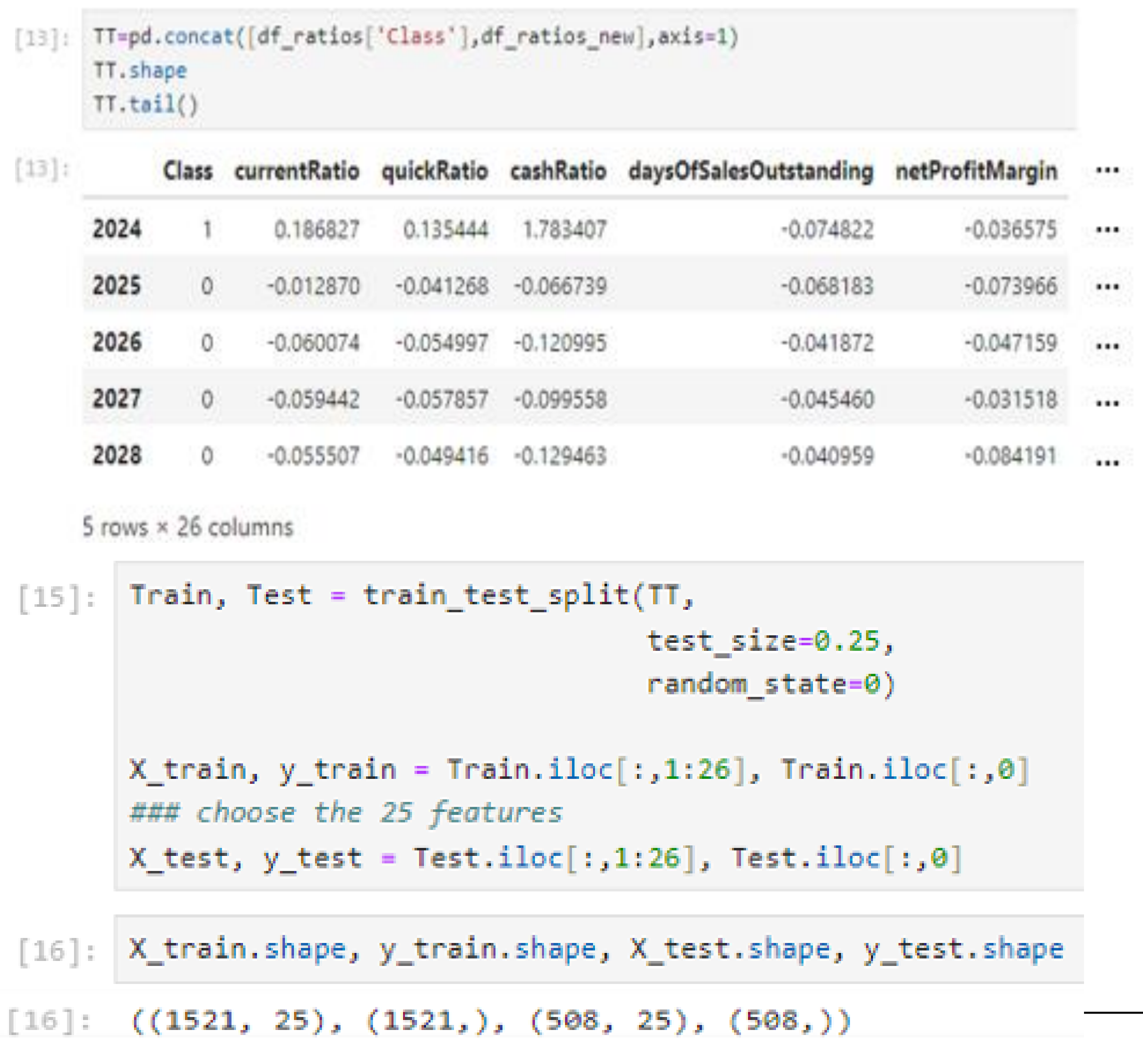


```
8]: # Normalization
    from sklearn.preprocessing import StandardScaler
    scaler = StandardScaler().fit(df_ratios_1)
    df_ratios_1 = scaler.transform(df_ratios_1)
    df_ratios_1=pd.DataFrame(df_ratios_1,\
          columns=['currentRatio','quickRatio','cashRatio','daysOfSalesOutstanding',\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

Please upload Chapter5-1.ipynb and follow the computing steps in Jupyter Notebook

This scaled data is then combined/concatenated with the Class variable in [13]. Note that we do not scale the Class variable here.

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



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

[13]:

	Class	currentRatio	quickRatio	cashRatio	daysOfSalesOutstanding	netProfitMargin	...
2024	1	0.186827	0.135444	1.783407	-0.074822	-0.036575	...
2025	0	-0.012870	-0.041268	-0.066739	-0.068183	-0.073966	...
2026	0	-0.060074	-0.054997	-0.120995	-0.041872	-0.047159	...
2027	0	-0.059442	-0.057857	-0.099558	-0.045460	-0.031518	...
2028	0	-0.055507	-0.049416	-0.129463	-0.040959	-0.084191	...

5 rows × 26 columns

```
[15]: 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]
```

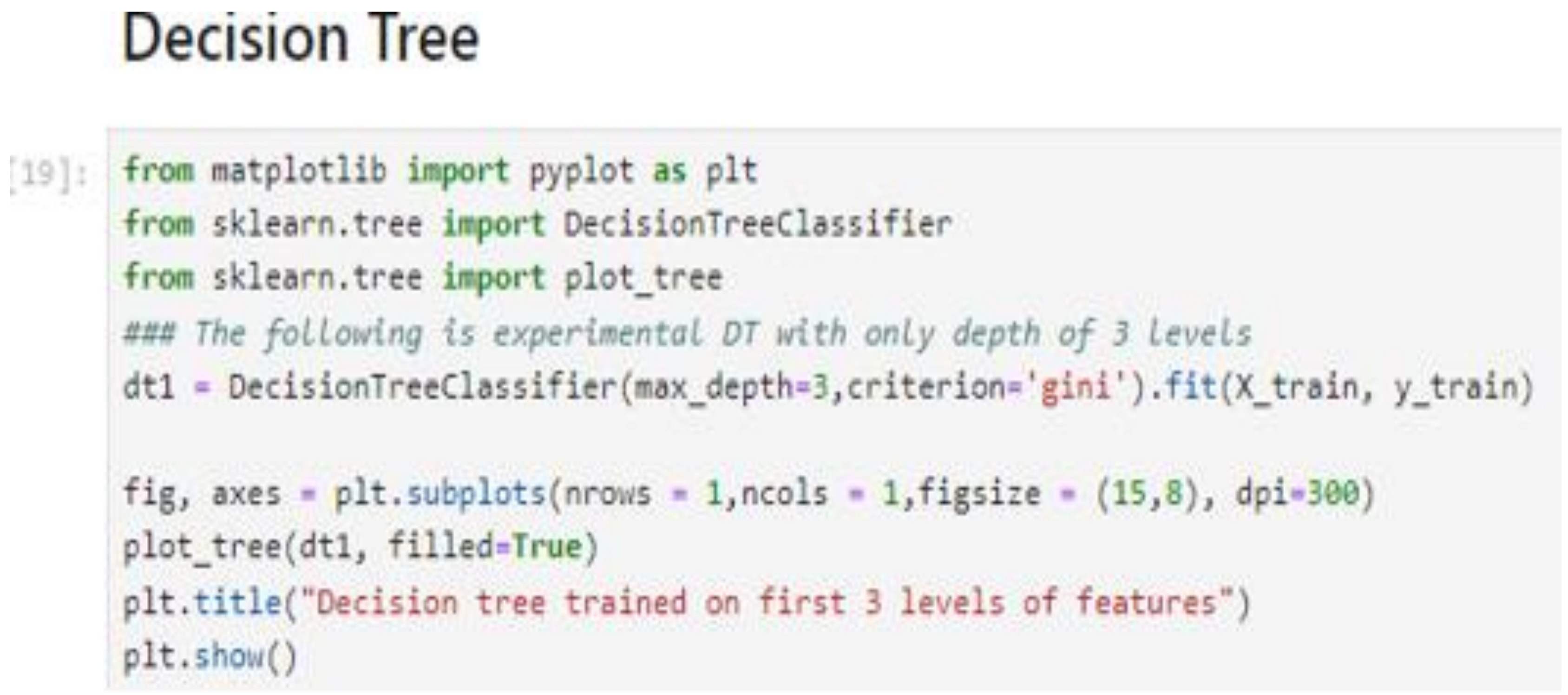
```
[16]: X_train.shape, y_train.shape, X_test.shape, y_test.shape
```

```
[16]: ((1521, 25), (1521,), (508, 25), (508,))
```

sklearn DecisionClassifier algorithm

Please upload Chapter5-1.ipynb and follow the computing steps in Jupyter Notebook

The sklearn DecisionClassifier algorithm is then performed on the training data set. See codeline [19]. In this preliminary DT classification, only a depth of 3 levels is applied. The root node and decision nodes and the branching are shown.

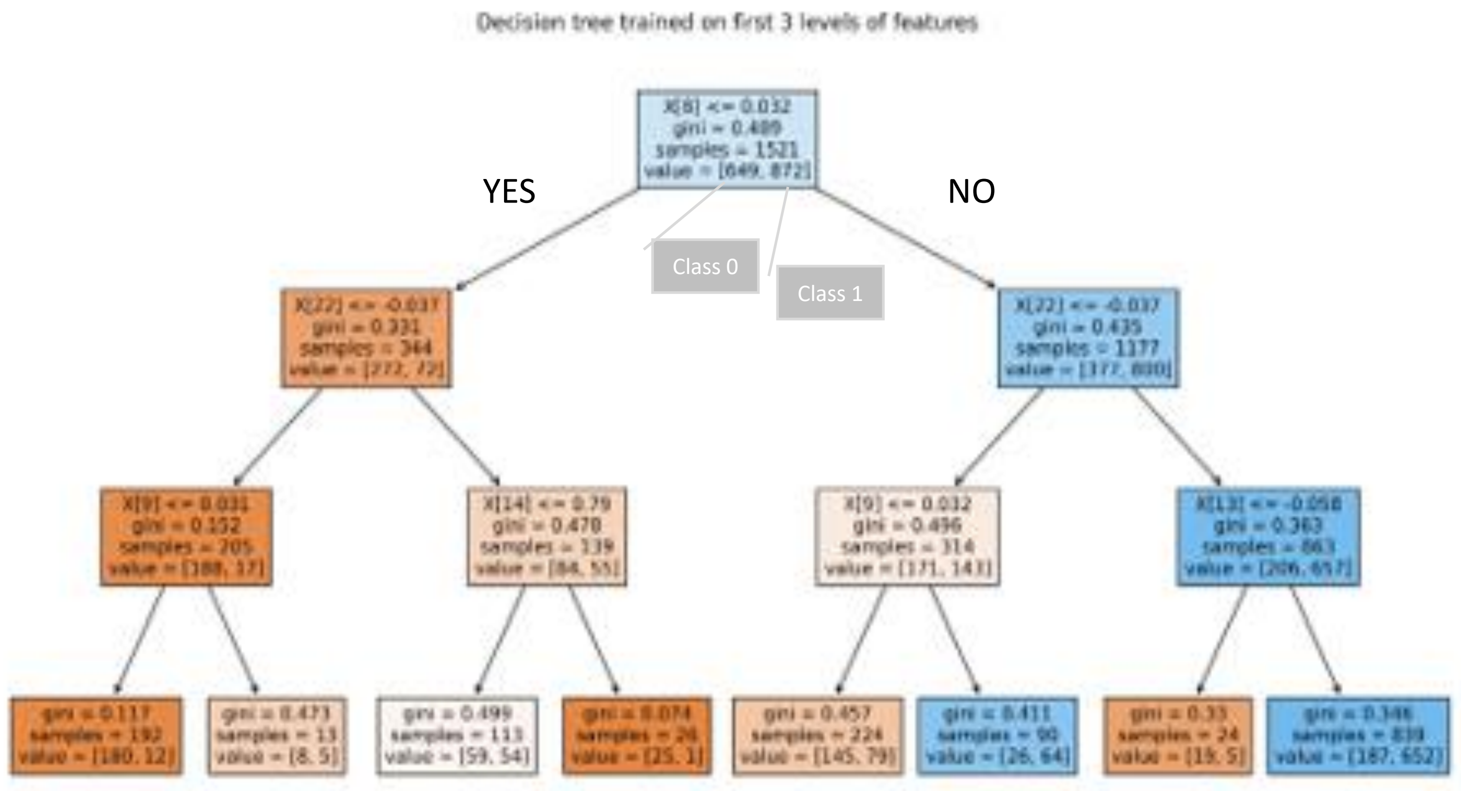


Decision Tree

```
[19]: from matplotlib import pyplot as plt
      from sklearn.tree import DecisionTreeClassifier
      from sklearn.tree import plot_tree
      ### The following is experimental DT with only depth of 3 levels
      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()
```

sklearn DecisionClassifier algorithm

Please upload Chapter5-1.ipynb and follow the computing steps in Jupyter Notebook



When DT depth levels are not constrained, Code lines [22] – [29] show an accuracy of 70.669%, precision of 75% and recall of 74% for the case of class 1. Precision and recall for class 0 are 65% and 67%. AUC for class 0 is 70.12% with 22 depth levels for a single DT.
Note: High recall in case 0 is important for this situation of being able to predict a low rating firm correctly.

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.
- When it comes to the testing, each of the 500 trained DTs will, using its own trained decision rules, yield its prediction vector on the test sample of 508 firms. Each of these prediction vector from each of the 500 DTs are possibly different. (The X_train changes for each re-sampling; so are the X_test. The 508 firms in y_train, y_test remain the same.)

Random Forest

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

Sum of predicted test cases:

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

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

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

On averaging, based on the test set, 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.

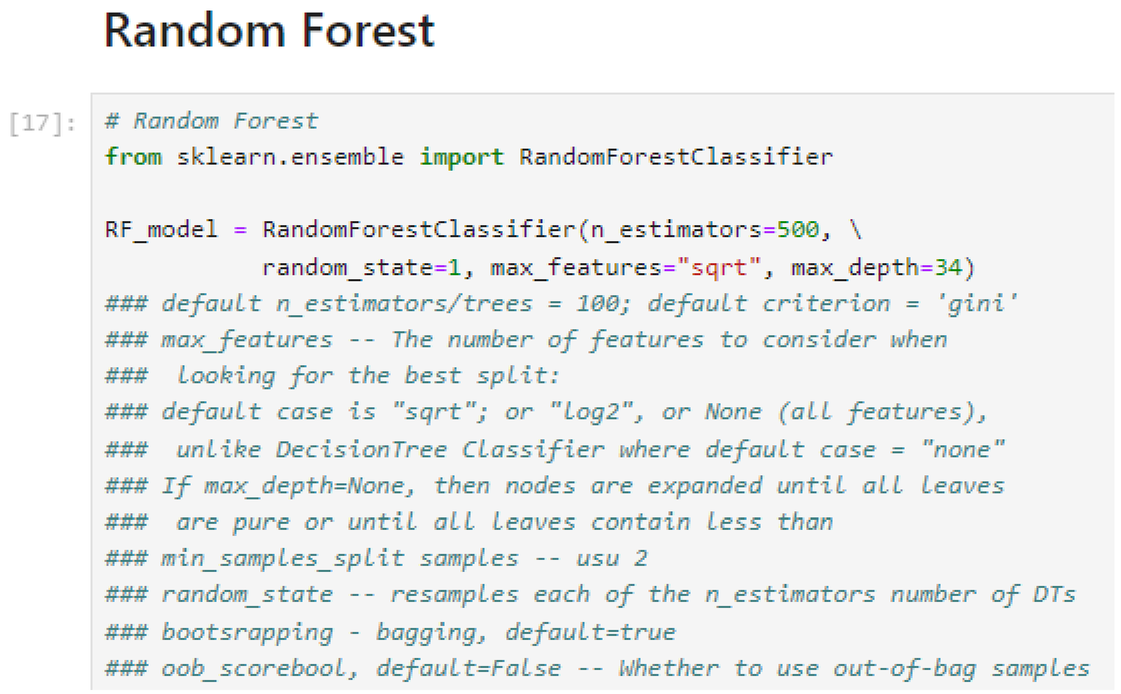
Random Forest

- 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.

RANDOMFORESTCLASSIFIER

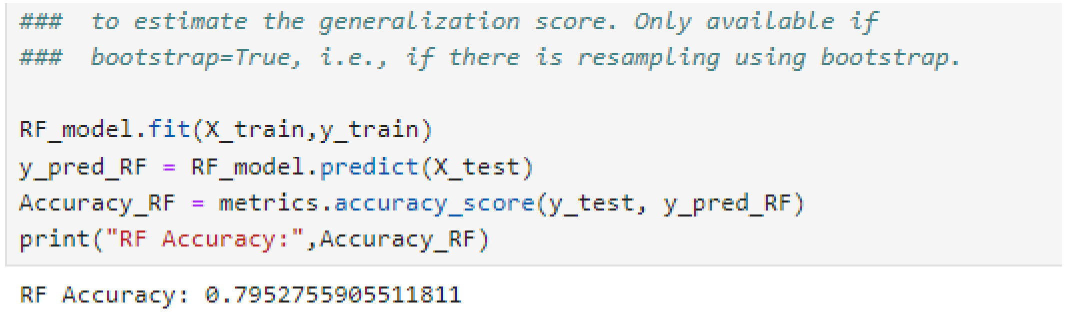
Please upload Chapter5-2.ipynb and follow the computing steps in Jupyter Notebook

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 √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.



Random Forest

```
[17]: # 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 -- The number of features to consider when
      ###  looking for the best split:
      ### default case is "sqrt"; or "log2", or None (all features),
      ###  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
      ### random_state -- resamples each of the n_estimators number of DTs
      ### bootsrapping - bagging, default=true
      ### oob_scorebool, default=False -- Whether to use out-of-bag samples
```



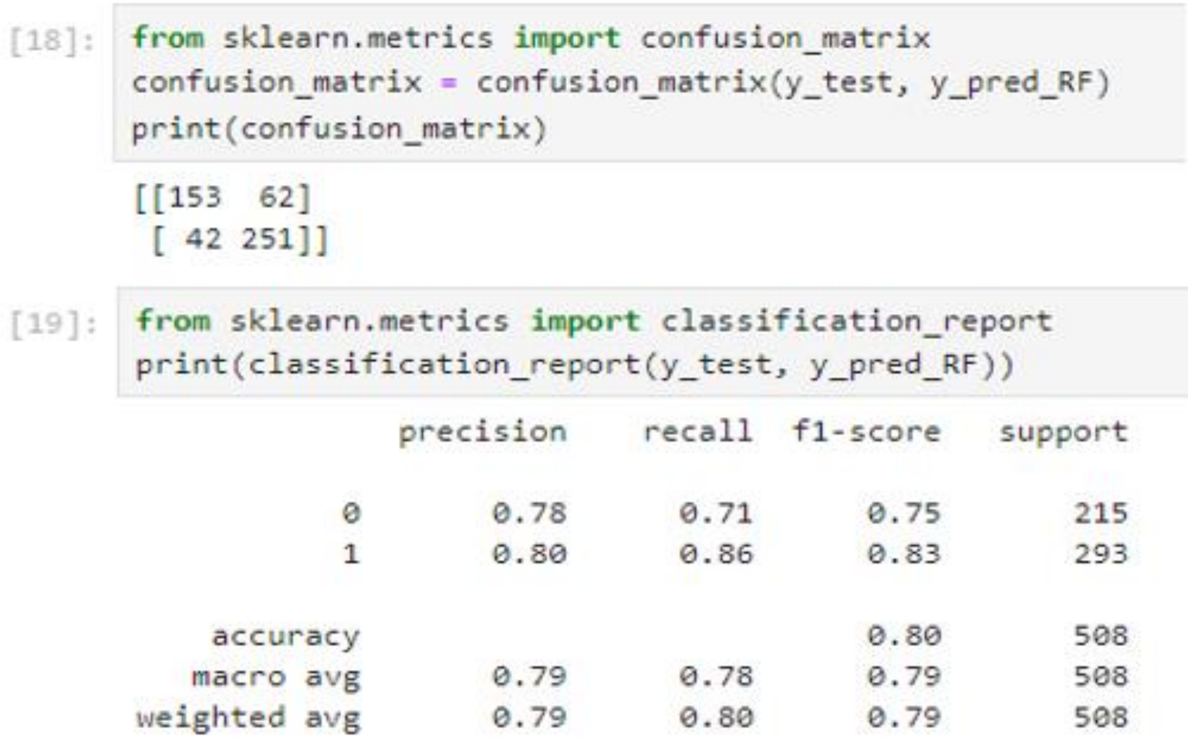
```
###  to estimate the generalization score. Only available if
###  bootstrap=True, i.e., if there is resampling using bootstrap.

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
```

RANDOMFORESTCLASSIFIER

Please upload Chapter5-2.ipynb and follow the computing steps in Jupyter Notebook

In this investment grade/speculative grade prediction problem, the accuracy of the RF here is 79.53%. The AUC is 88.69%. The Confusion matrix and the prediction performance classification report are shown as follows.



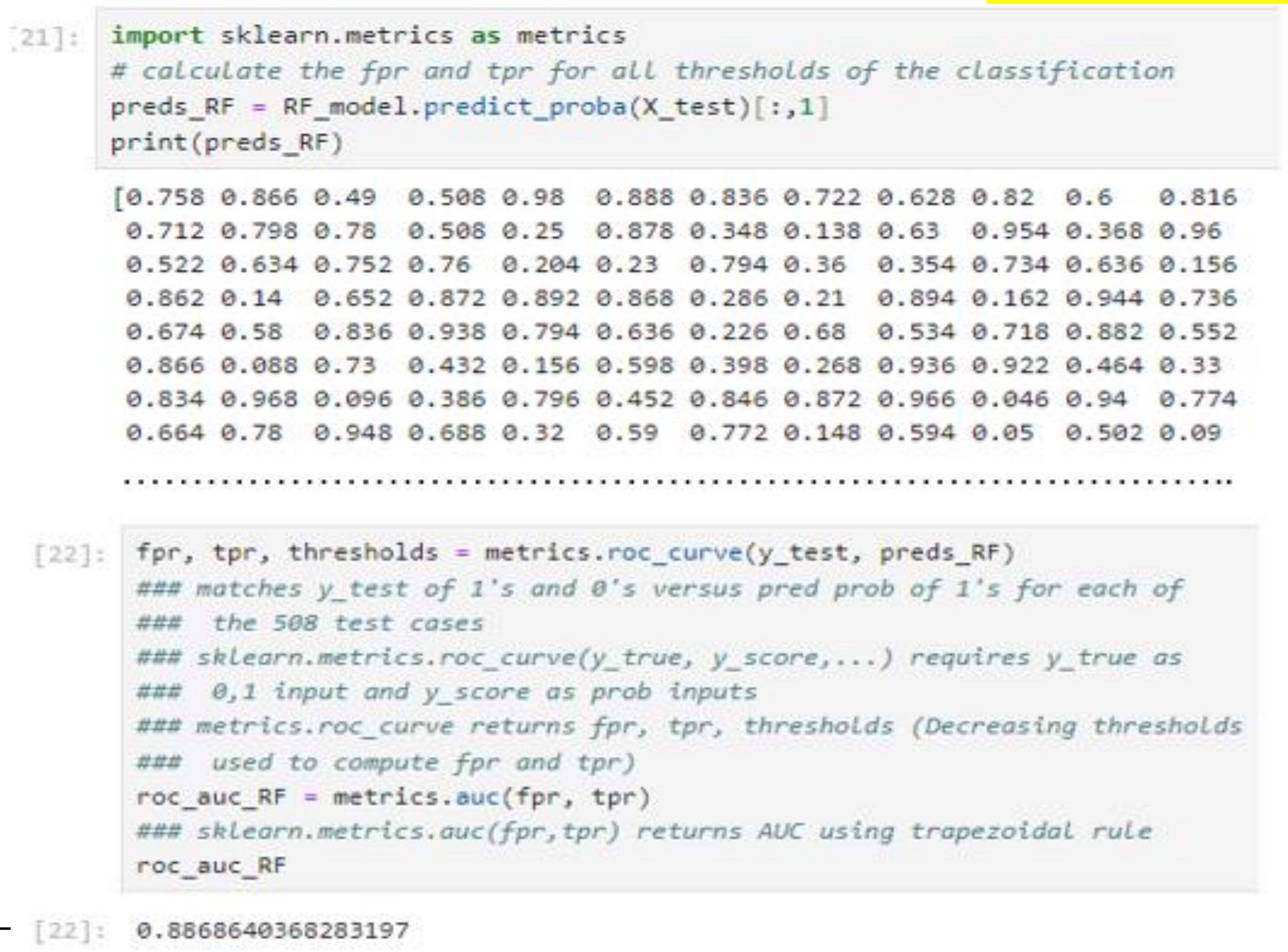
```
[18]: from sklearn.metrics import confusion_matrix
      confusion_matrix = confusion_matrix(y_test, y_pred_RF)
      print(confusion_matrix)

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

```
[19]: 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
```

Compare for case Y=1, precision of 80% and recall of 86% versus 75% and 74% for a single DT.

For class Y = 0, RF also has higher precision of 78% and higher recall of 71% compared to those of a single DT of 65% and 67% respectively.

Please upload Chapter5-2.ipynb and follow the computing steps in Jupyter Notebook



```
[21]: 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]
      print(preds_RF)
```

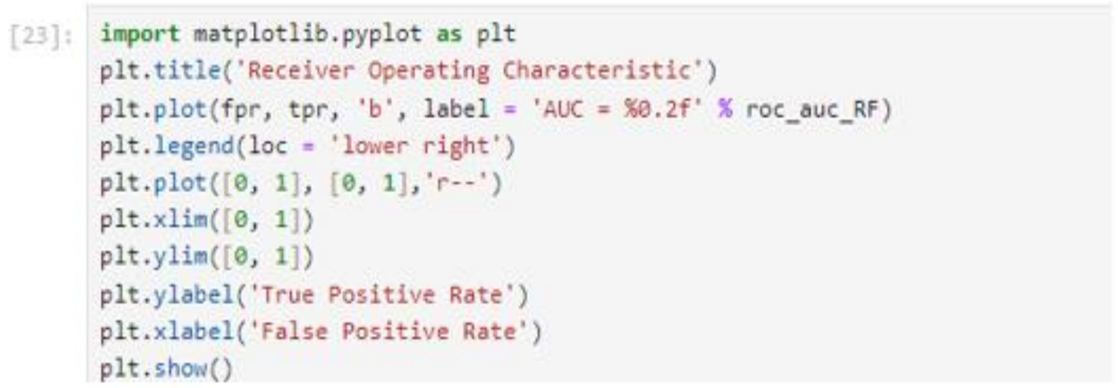
```
[0.758 0.866 0.49  0.508 0.98  0.888 0.836 0.722 0.628 0.82  0.6   0.816
 0.712 0.798 0.78  0.508 0.25  0.878 0.348 0.138 0.63  0.954 0.368 0.96
 0.522 0.634 0.752 0.76  0.204 0.23  0.794 0.36  0.354 0.734 0.636 0.156
 0.862 0.14  0.652 0.872 0.892 0.868 0.286 0.21  0.894 0.162 0.944 0.736
 0.674 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

 ........................................................................
```

```
[22]: 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
      ### 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

[22]: 0.8868640368283197
```

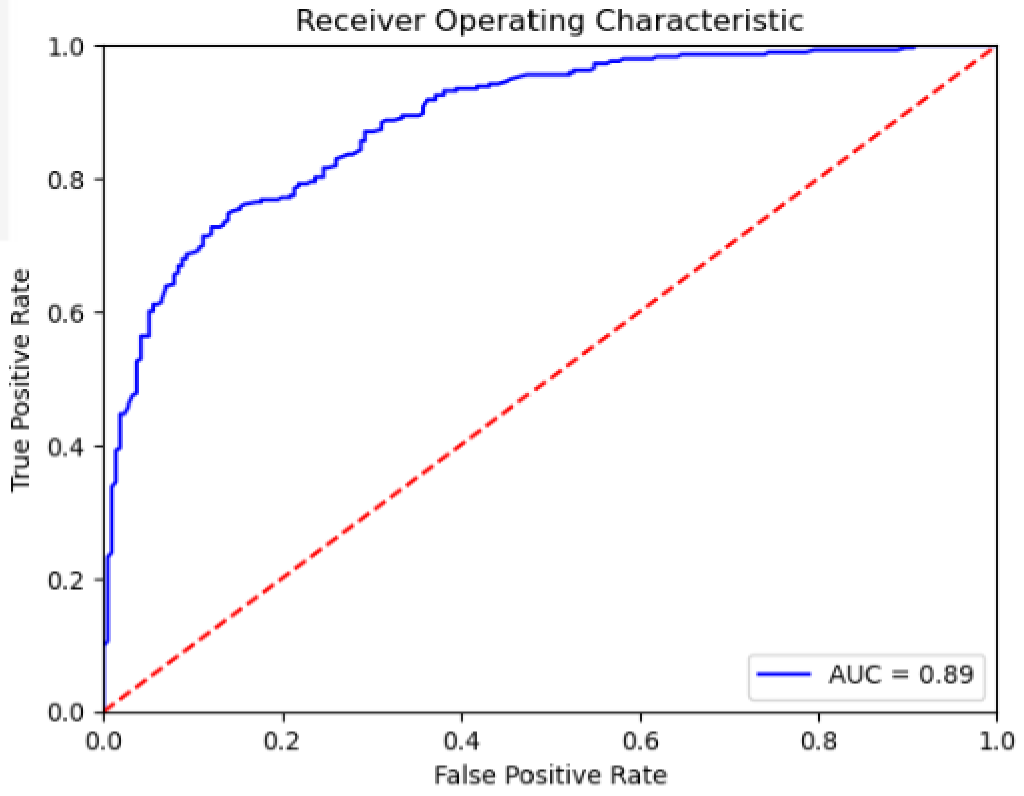
Compare with 70.12% for a single DT

Area under ROC

Please upload Chapter5-2.ipynb and follow the computing steps in Jupyter Notebook



```
[23]: 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()
```



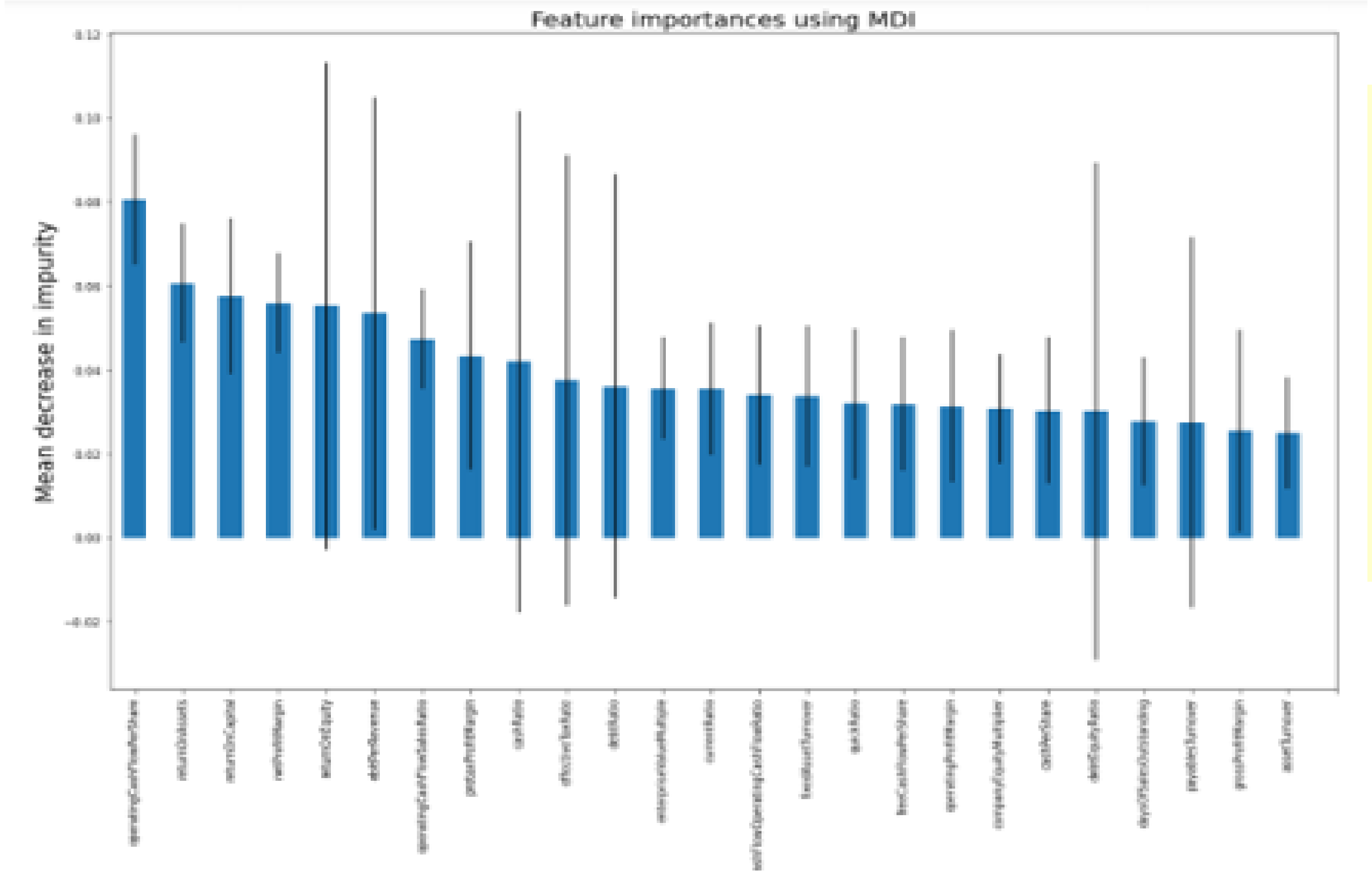
RANDOMFORESTCLASSIFIER

Please upload Chapter5-2.ipynb and follow the computing steps in Jupyter Notebook

- 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 sum total across the totals of the 25 features.
- This ratio is called the Feature Importance of the particular feature. The ratios of all features sum to 100%. The higher the % of each feature in this contribution to overall Gini importance or overall Gini weighted impurity decrease, the more important is this feature in the RF algorithm. The examining of feature importance is shown as follows in code line [25].

RF FEATURE IMPORTANCE

Please upload Chapter5-2.ipynb and follow the computing steps in Jupyter Notebook

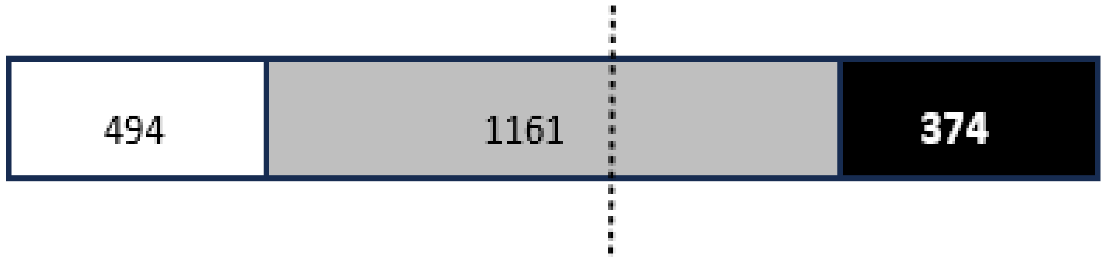


It can be seen that in order importance, the features are Operating Cashflow per share, Return on Assets, Return on Capital, Net Profit Margin, Return on Equity, EBIT per Revenue, and so on.

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.

MULTI-CLASS CLASSIFICATION

Sometimes binary classification may not be the best way to classify if the types have the following distributions where the numbers indicate the instances.

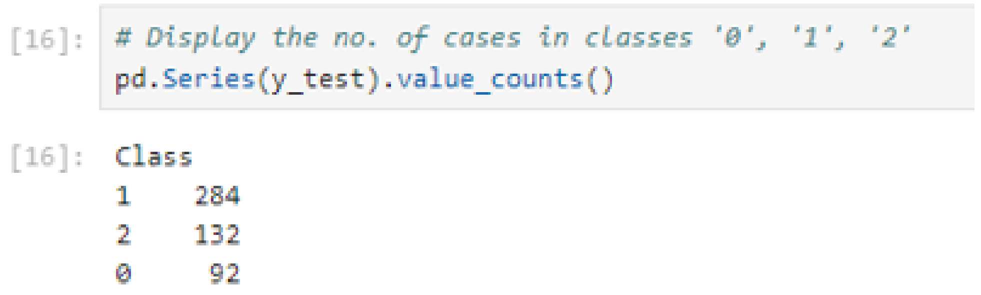


These classes are akin to the class 2 for white sector, class 1 for grey sector, and class 0 for black sector. Banks facing potential borrowing firms would try to accurately predict their unknown rating status, especially if they have bad ratings belonging to class 0 or the black sector. Banks would also like to predict if the potential client is a good firm of class 2 so that more favourable lending rates can be given to maintain longer client relationship. Hence it is more relevant to perform multi-class (3-class) prediction in this case.

MULTI-CLASS CLASSIFICATION

Please upload Chapter5-2M.ipynb and follow the computing steps in Jupyter Notebook

- We continue with the corporate ratings data set in Section 5.2 but now re-classify the firms into 3 classes, viz., Class 2 for AAA, AA, A ratings, class 1 for BBB, BB ratings, and class 0 for B, CCC, CC, C, and D ratings. The code file is Chapter5-2M.ipynb and the data set is corporate_rating3.csv.
- Standard scaling is done, as in the binary classification, to the 25 features in code line [7]. The data set is similarly divided into 75% training data and 25% test data (assuming validation phase has been done). There are 508 test cases and the distribution of these in classes 2,1,0 is shown as follows.

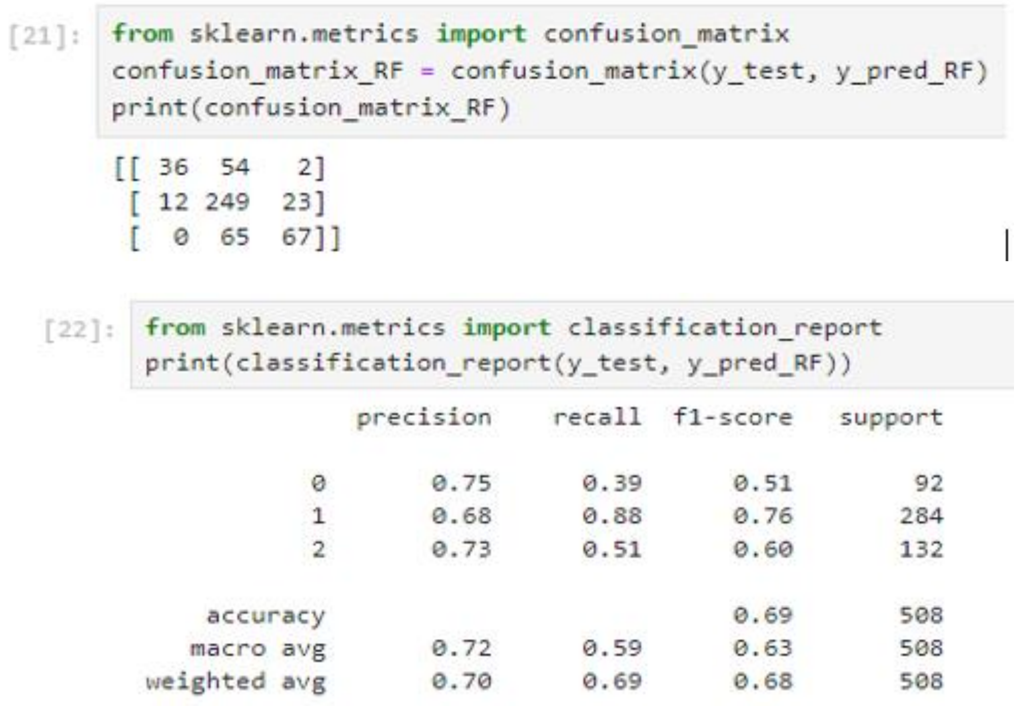


```
[16]: # Display the no. of cases in classes '0', '1', '2'
      pd.Series(y_test).value_counts()

[16]: Class
      1    284
      2    132
      0     92
```

The Random Forest method is performed on this multi-class prediction problem. Code line [18] shows that with ensemble of 1000 DTs, the accuracy is 69.29%.

MULTI-CLASS CLASSIFICATION

Code lines [21] and [22] show the computed confusion matrix and the classification report for this multi-class prediction.



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

      [[ 36  54   2]
       [ 12 249  23]
       [  0  65  67]]
```

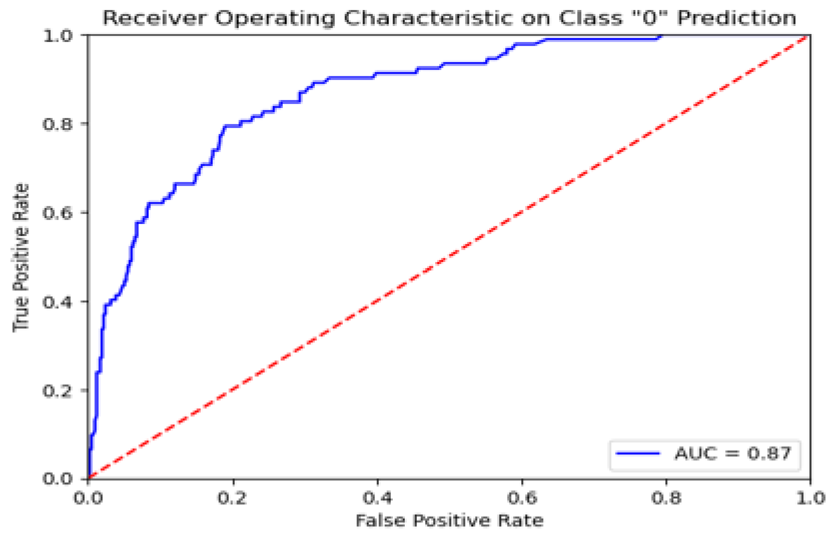
```
[22]: from sklearn.metrics import classification_report
      print(classification_report(y_test, y_pred_RF))

                    precision    recall  f1-score   support

                 0       0.75      0.39      0.51        92
                 1       0.68      0.88      0.76       284
                 2       0.73      0.51      0.60       132

          accuracy                           0.69       508
         macro avg       0.72      0.59      0.63       508
      weighted avg       0.70      0.69      0.68       508
```

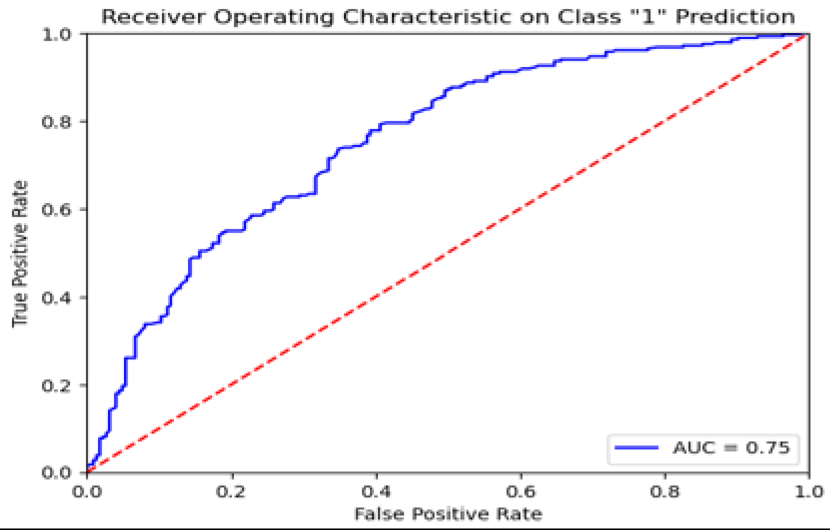
The first, second, and third row (from top to bottom) in the confusion matrix denotes actual class 0, 1, and 2 test instances. The first, second, and third columns (from left to right) denotes the predicted class 0, 1, and 2 respectively. The recall for class 0 is 36/(36+54+2) = 39.1%. The precision for class 0 is 36/(36+12+0) = 75%. The recall for class 2 is 67/(67+65+0) = 50.75%. The precision for class 2 is 67/(67+23+2) = 72.8%.

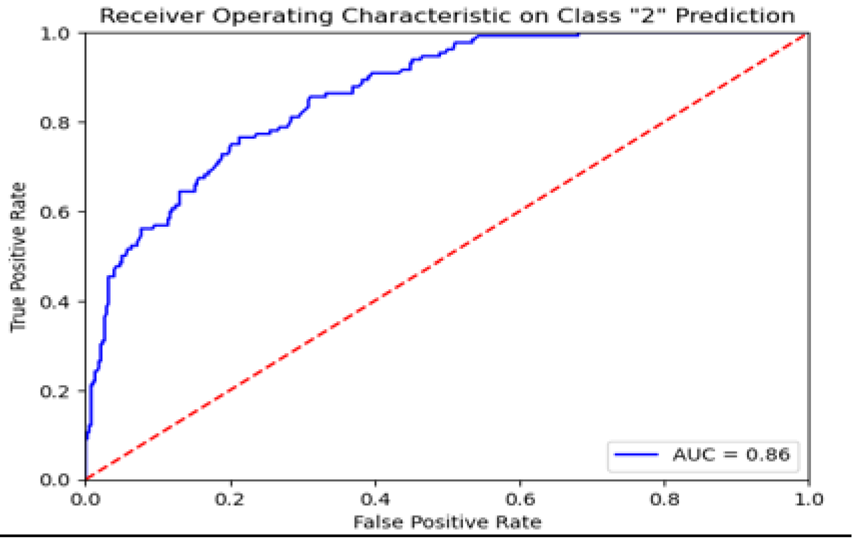
The recall for class 0 appears low and can be improved via more effective algorithms.

MULTI-CLASS CLASSIFICATION

For multi-class classification, it is not a natural step to construct a single ROC curve since the ROC curve involves the true positive rate plot versus the false positive rate and these come more naturally from a binary classification table. However, we can construct 3 separate ROCs, one for each class as the positive case.







Summary

- 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.
- 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.

Homework 3 Graded Exercise:

Chapter5-3.ipynb (this file is provided only after grading)

The data is related with direct marketing campaigns of a Portuguese banking institution. The marketing campaigns were based on phone calls. Often, more than one contact to the same client was required, in order to access if the product (bank term deposit) would be (or not) subscribed. Data set is public – acknowledgement: S. Moro, R. Laureano and P. Cortez. Using Data Mining for Bank Direct Marketing: An Application of the CRISP-DM Methodology. In P. Novais et al. (Eds.), Proceedings of the European Simulation and Modelling Conference - ESM'2011, pp. 117-121, Guimarães, Portugal, October, 2011. EUROSIS.

Download from 'banking3.csv'. This is a simplified version of the original data set. There are 18 features and one label variable y. y = 1 denotes success in getting client to put in a term deposit. y=0 denotes failure to do so.

The features are self-explanatory as in Homework 2. Note that Euribor3m is the 3-month Euribor interest rate prevailing in the particular campaign for the client when client's response was y=1 or y=0.

Homework 3 Graded Exercise:

Use Random Forest method to predict if a client would subscribe to a bank term deposit. Use the following specifications to construct training and test data sets. Assume optimal hyperparameters have been determined in the following specifications.

```
"from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=0)"
```

Use the following specifications in the RandomForestClassifier in sklearn.

```
"from sklearn.ensemble import RandomForestClassifier
  RF_model = RandomForestClassifier(n_estimators=1000, random_state=1, max_features=None, max_depth=None)"
```

Record the % of cases of y=1 in the given 'banking3.csv'. Find the confusion matrix, classification report and area under ROC.

What are the importance features. How do you compare with logistic regression?

(Follow format of the MCQ in answering the above.)

End of Class