Decision Tree and Related Techniques for Classification in ScalaTion

By,

Susan George(susangeorge@uga.edu)

(Under the direction of Dr. John A. Miller)

December 9,2019

Abstract

ScalaTion is a scala based framework for analytics, simulation and optimization. The 'analytics' package in ScalaTion contains tools for performing data analytics. Several models are already implemented within the package with the aim to accurately predict the response for future observations and for understanding the relationship between the predictors and response. Our work involved developing DecisionTree algorithms to handle discrete as well as continuous data. We further developed a pruning algorithm to reduce the size of the decision tree by removing sub-trees that have little power to classify instances. All the models developed were compared and evaluated against corresponding *sklearn* packages in Python.

1. Introduction

Vast amount of data is analyzed and understood by statistical learning. Supervised learning is a paradigm of statistical learning which involves building a prediction model to predict the output based on one or more inputs.

Consider a dataset of input variables(X) and an output variable(Y). In general, relationship between X and Y is

$$Y = f(X) + \epsilon$$

where f is a fixed but unknown function of X and ϵ is the irreducible error.

Supervised learning algorithms learn the mapping function f and try to approximate it so well with the aim of accurately predicting the response for future observations. The response variable Y can be characterized as either quantitative or qualitative. Quantitative variables have numerical values (a person's age) while qualitative variables are categorical (person's gender: male or

female). Problems with quantitative response are called regression problems and the ones with qualitative response are referred as classification problems.

Decision-tree learning algorithm is one of the best and preferred supervised-learning algorithms. They can be applied to both classification and regression problems. The tree-based models can be used to map linear as well as non-linear relationships in data and are characterized by their ease to interpret, high accuracy and stability.

Decision Trees involve the technique of partitioning the predictor space into several simple regions. The name 'Decision Tree' is attributed to the fact that the splitting rules used for partitioning data could be summarized in a tree. The 3 main elements of a decision tree are

- *Decision Node/Interior node* – test the value of an attribute
- *Edge* – represents the outcome of a test
- *Leaf Node* - terminal nodes used to predict outcome

We use the principle of recursive partitioning to build trees which is an iterative process of splitting the data into partitions and then splitting it up further on each branch. We can have both classification and regression trees. In a classification tree, the decision variable is discrete while in regression trees the decision variable is continuous.

In this report, we primarily focus on Classification trees. The next few sections give an overview of Decision Trees and tree related ensemble methods in ScalaTion and then we proceed to the evaluation section wherein tree-based ScalaTion tools are compared with their counterparts from sklearn package in python.

2. Classification trees – DecisionTreeID3

ID3(iterative Dichotomiser 3) algorithm is used to generate a decision tree from dataset. Let us consider we have the original dataset as S. On each iteration of the algorithm, we iterate through every unused attribute of dataset S and compute the entropy/Information gain of that attribute. Attribute with the smallest entropy or largest information gain is selected. The set S is then partitioned by the selected attribute to produce subsets of data and we recursively iterate on each subset, considering the attributes never selected earlier. Recursion stops when

- Every element in the subset belongs to the same class. In such a scenario, the node is replaced by a leaf node and labelled with the class of the examples.
- No more attributes to be selected, but the remaining data samples still do not belong to the same class. In this case, the node is made a leaf node and labelled with the mode (most common class of the examples in the subset).
- No sample data in the subset

The decision tree consists of decision/interior node which represents the selected attribute on which the data was split, and leaf nodes which represent the class label of the final subset of this branch.

Approach:

I. Calculate the initial entropy of the tree
II. Find the attribute with best Information Gain (minimum entropy)
III. Given attribute with best Information Gain, partition the dataset into subsets where each subset corresponds to an attribute value
IV. Create a decision tree node containing the attribute
V. Using the remaining attributes, recursively iterate through the subsets

Following are the input parameters to the DecisionTreeID3:

- *X* - input matrix
- *Y* - output vector
- *fn* - feature names
- *k* - number of classes
- *cn* - class names

Important terms:

- ***Entropy*** - measures degree of uncertainty and the aim of a machine learning model is to reduce this uncertainty. Entropy values range from 0 to 1. Value 0 indicates minimum uncertainty and value 1 indicates maximum uncertainty. The mathematical formula to calculate entropy is

$$E(Y) = \sum_{c}^{i=1} -pi \log pi$$

  Where c indicates the total number of outcomes.

- ***Information Gain*** – is a metric to measure the reduction of uncertainty in the target variable given additional information. It is obtained by subtracting the entropy of Y given X from the entropy of Y. Greater the reduction in this uncertainty, the more information is gained about Y from X.

$$IG(Y,X) = E(Y) - E(Y|X)$$

3. Classification trees – DecisionTreeC45

DecisionTreeC45 algorithm is an extension of DecisionTreeID3 algorithm. While DecisionTreeID3 can handle only discrete attributes, DecisionTreeC45 can handle both discrete and continuous attributes. Discrete attributes are handled in the same manner as ID3 algorithm described above.

For continuous attributes, C4.5 creates a split threshold and then splits the dataset into those whose attribute value is greater than threshold and those that are less than or equal to it. The optimum split threshold divides the feature values into low (<= threshold) and high (> threshold) values such that the weighted entropy is minimized.

Approach:

I. Calculate the initial entropy of the tree
II. Find the attribute A with best Information Gain (minimum entropy)
III. Create a decision tree node that splits on A at the best split threshold
IV. Recursively iterate through the subsets obtained by splitting on A

Following are the input parameters to the DecisionTreeC45:

- *X* - input matrix
- *Y* - output vector
- *fn* - feature names
- *k* - number of classes
- *cn* - class names
- *conts* - the set of feature indices for variables that are treated as continuous

4. Hyper-parameter tuning in DecisionTree

ScalaTion also supports hyperparameter tuning in decision trees. Below are the 2 tuning parameters:

- *height* - recursively build the tree until the edges in longest path is at the specified tree height
- *cut-off* - recursively build the tree until the entropy drops to the 'cut-off' threshold

5. Pruning

Pruning is a machine learning technique to reduce the size of the decision tree by removing sub-trees that have little power to classify instances. A tree that is too large has the risk of overfitting the training data and this problem can be overcome by pruning. Pruning reduces the tree size without altering the prediction accuracy.

Different techniques are present to prune a tree which involves a top-down or bottom-up approach. In top-down, pruning happens at nodes down from the root and, and in bottom-up approach we start at the leaf nodes. These approaches could be more formerly described as 2 algorithms:

- Reduced Error Pruning
- Cost complexity Pruning

Currently, we will be implementing Reduced Error Pruning technique which uses bottom-up approach. DecisionTreeId3wp and DecisionTreeC45wp handle pruning for ID3 and C45 trees in ScalaTion.

Approach:

I. Consider each node from bottom for pruning
II. Find the subtree with the least information gain
III. Remove the subtree only if the Information gain is less than the threshold
IV. The subtree is replaced by a leaf node with the majority class assigned to that node
V. Repeat this procedure until the information gain of a subtree exceeds threshold

Pruning could be fine-tuned using 2 parameters:

- *nPrune* - number of nodes to be pruned
- *threshold* – remove sub-tree only if information gain < 'threshold'

6. Bagging

Bagging is a very powerful ensemble method. We use ensemble methods because it combines the predictions from multiple machine learning models to make more accurate predictions rather than any individual model.

Decision trees suffer from high variance. To overcome this, we use bagging or bootstrap aggregation. Bagging uses bootstrap technique wherein we create several subsets of the data from the training sample by randomly choosing data with replacement. Then each data subset is used to train their own decision trees thereby creating an ensemble of different models. In the case of regression trees, average of predictions from all the trees is used which is more robust than a single decision tree. For classification trees, we take the majority vote to arrive at a decision. Bootstrapping procedure leads to better model performance because it decreases the variance of the model, without increasing the bias.

Approach:

Assume we have a sample dataset of 500 instances. Bagging algorithm for decision trees is:

I. Create many (e.g. 50) random subsets of dataset with replacement.
II. Train a decision tree model on each subset.
III. Given a new dataset, calculate the average prediction/majority vote from each model.

7. Random Forest

Random Forest is an extension over bagging. Random Forest utilizes both Bootstrap aggregation and Feature randomness. Apart from taking a random subset of data, random forests consider only a random subset of features rather than taking all features to build trees. This approach creates several random trees, hence called Random Forest.

In a normal decision tree, when it is time to split a node, we consider every possible feature and pick the one that produces the most optimal split-point. In contrast, each tree in a random forest consider only a random subset of features. Unlike bagging, this causes more variation amongst the trees in the ensemble model and lower correlation across trees. Random forests are very useful in handling high dimensional data.

Parameter-tuning

a. nTrees: specifies the number of trees required to build a model. The optimal number of trees can be found using cross-validation.
b. bRatio: is the Bagging ratio which determines the subset size. It gives the fraction of data to be used while building trees

Following are the input parameters to the ensemble model:

- *X* - input matrix
- *Y* - output vector
- *fn* - feature names
- *k* - number of classes
- *cn* - class names
- *conts* - the set of feature indices for variables that are treated as continuous
- *nTrees* – number of trees
- *bRatio* – Bagging ratio
- *fS* – number of features for building trees(only for random forest)

8. Evaluation

We performed an evaluation of the API's offered by ScalaTion against standard packages(sklearn) in Python. For the experiments, we used datasets like Wisconsin Breast Cancer dataset and Pima Indian Diabetes dataset which are commonly used for classification problems.

Wisconsin Breast Cancer Dataset has 683 rows of data with 9 features. The target variable determines if cancer is benign or malignant. All the features in the dataset are discrete. *Pima Indian Diabetes Dataset* has 768 rows and 8 features. The target variable determines if a patient has diabetes or not. Some of the features in this dataset are continuous.

The split criterion for the tree is Entropy.

DecisionTreeID3

With the Breast cancer dataset, ScalaTion *DecisionTreeID3* model fetched an accuracy of 94.7% with 5-fold cross-validation and 94.9% with 10-fold cross validation. The *DecisionTreeClassifier* from the python sklearn package gave an accuracy of 94.1% and 94.7% for 5-fold and 10-fold cross validation respectively, which was slightly lower than ScalaTion models. In both the packages, the trees could grow to its full depth.

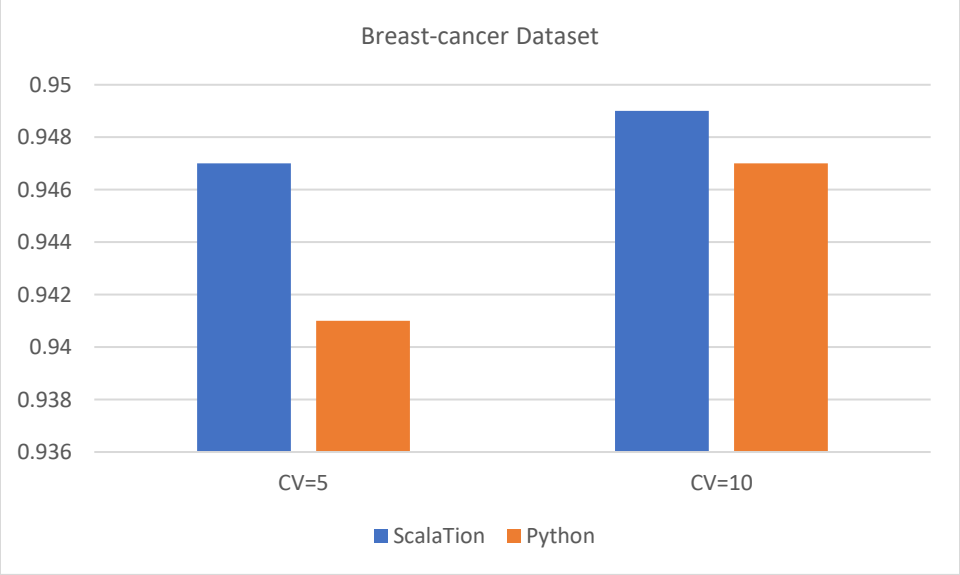

Fig 1. DecisionTreeID3(ScalaTion) vs DecisionTreeClassifier(sklearn)

DecisionTreeC45

We used *Pima Indian Diabetes Dataset* which has which 768 data and 8 features. *DecisionTreeClassifier* model from python package gave an accuracy of 70.1% and 70.4% for 5-fold and 10-fold cross-validations respectively. *DecisionTreeC45* model from ScalaTion gave higher accuracy of 72.9% and 73.3% for 5-fold and 10-fold cross-validation. Both the tree models could grow to its full depth.

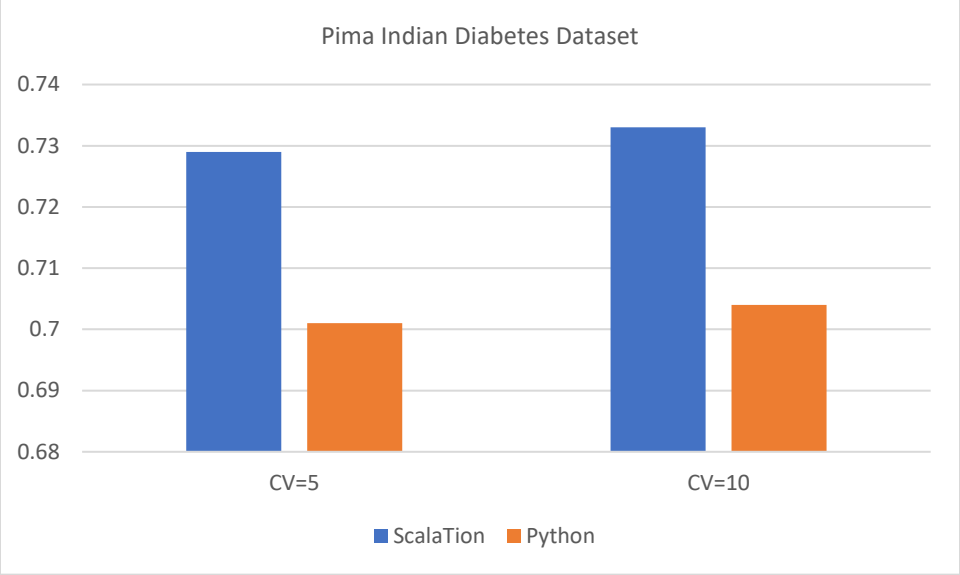

Fig 2. DecisionTreeC45(ScalaTion) vs DecisionTreeClassifier(sklearn)

Limiting maximum depth of tree

If we allow the decision tree to grow to its full depth, the model could get complex because of too many splits and it captures a lot of information about the data. This could potentially lead to overfitting and the resulting tree would not generalize well on the test dataset. We could avoid this by reducing the depth of the tree. While building tree model, we can specify the tree depth. We trained decision trees on the Diabetes dataset and set the tree-depth to 5.

In the below graph, we see that accuracy of the model gets better when the depth is limited in both ScalaTion and python and the overall accuracy of scalation tree model is higher than python.

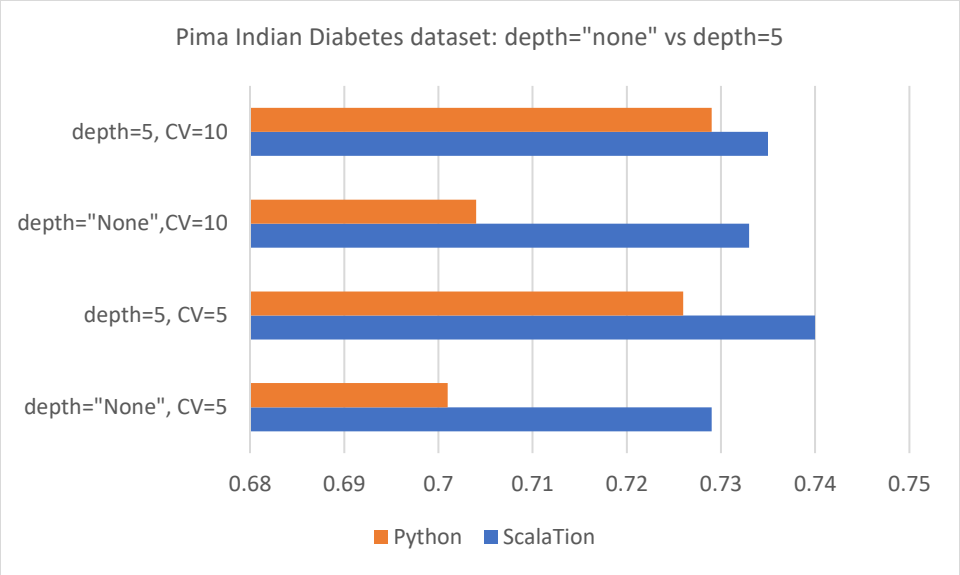

Fig 3. Unpruned vs pruned tree (depth=5)

Pruning

This experiment was done in ScalaTion. To evaluate pruning, we trained a decision tree model on the Brain cancer dataset. We split the data into 70% training data and 30% test data. We trained the unpruned decision tree on the training data and the trained model was used to classify the test dataset. The accuracy score of the unpruned tree model was 92.6%. Next, we pruned the tree, setting *nPrune(number of nodes to prune)* = '5' and the pruned tree model classified the test dataset with an accuracy of 94.6%.

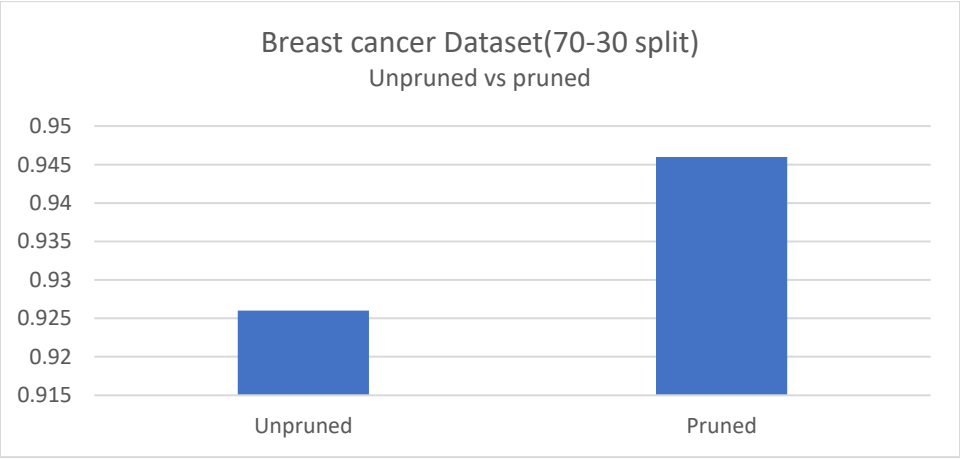

Fig 4. Unpruned vs pruned tree (nPrune=5)

Bagging

We used the Breast Cancer dataset to evaluate Bagging. We trained a decision tree model on the dataset and then used bagging which involves combining the predictions of multiple trees using the bootstrap technique.

Following parameters were set for bagging:

- No of trees – 5
- Bagging ratio – 0.7
- 5-fold cross-validation
-

The trees could grow up to full depth. Bagging fetched a higher accuracy of 98.7% in ScalaTion and 96.4% in sklearn. We observe that the behavior of the 2 models was consistent across the dataset with Bagging giving a better accuracy.

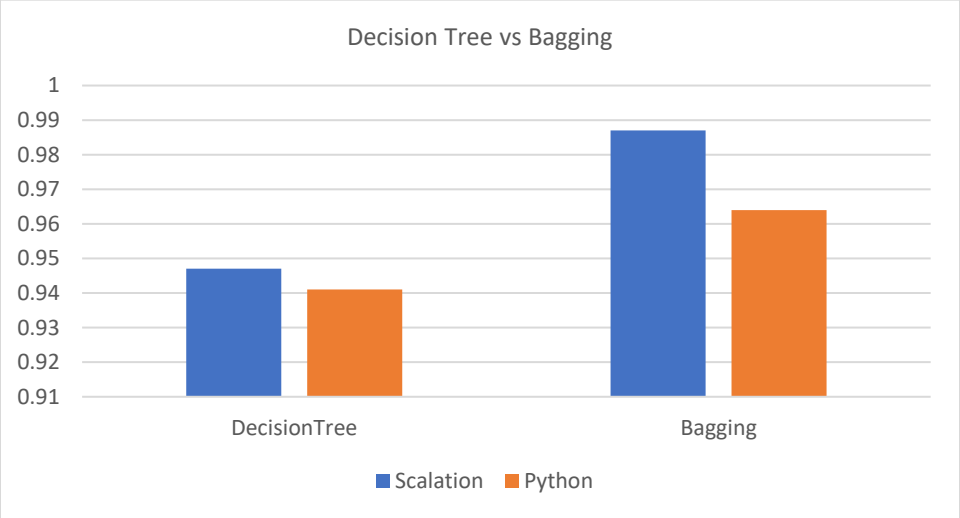

Fig 5. DecisionTree vs Bagging for Breast-cancer dataset

Random Forest

We used both the Breast cancer dataset and Diabetes dataset to evaluate Random Forest. Random Forests in general gave a higher accuracy than Decision Trees for both the datasets.

Following parameters were set for the RF:

- Number of trees = 10
- Bagging Ratio = 0.7
- No of features included = 8(breast cancer) and 7(Diabetes)

Fig 6. shows a comparison of 3 models for breast cancer dataset. Bagging gave the highest accuracy followed by Random Forest and DecisionTree. Behavior of the models are consistent across the 2 packages.

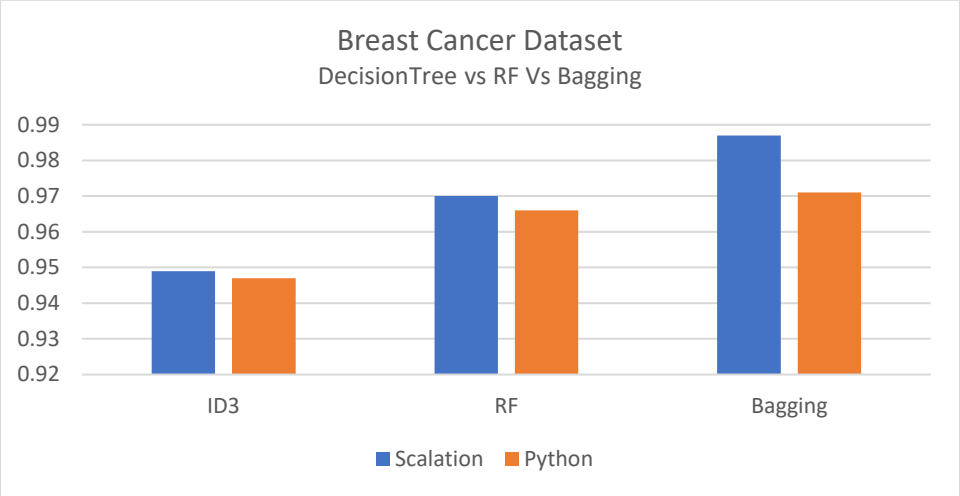

Fig 6. DecisionTreeID3 vs RF vs Bagging for Breast-cancer dataset

Fig 7. compares the C45 and Random Forest model for the Diabetes dataset.

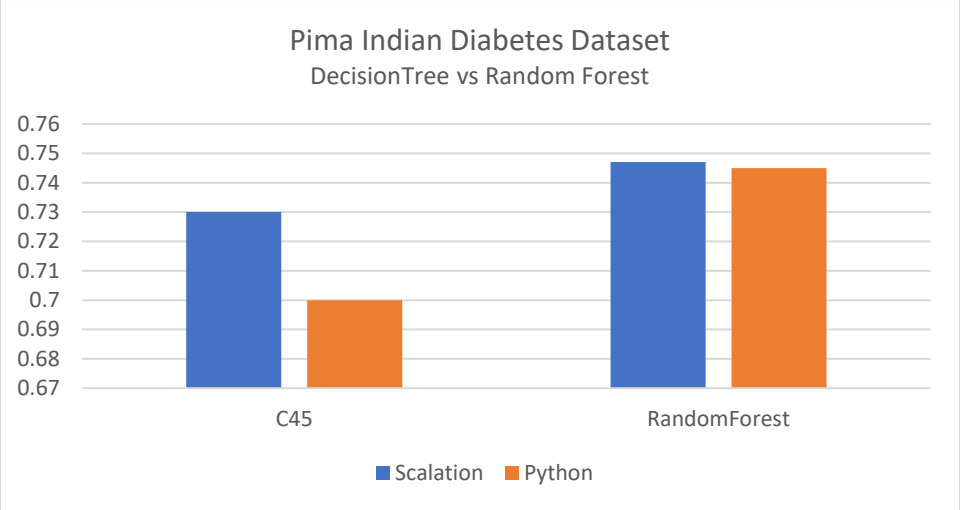

Fig 7. DecisionTreeC45 vs RF vs Bagging for Diabetes dataset

9. References

1. http://cobweb.cs.uga.edu/~jam/scalation_1.6/README.html
2. [Online] https://en.wikipedia.org/wiki/Decision_tree
3. [Online[http://faculty.marshall.usc.edu/gareth-james/ISL/ISLR%20Seventh%20Printing.pdf
4. [Online] https://web.stanford.edu/~hastie/ElemStatLearn/printings/ESLII_print12.pdf
5. [Online] https://machinelearningmastery.com/ensemble-machine-learning-algorithms-python-scikit-learn/
6. [Online] https://towardsdatascience.com/understanding-random-forest-58381e0602d2
7. [Online] https://www.datacamp.com/community/tutorials/random-forests-classifier-python
8. [Online] https://en.wikipedia.org/wiki/ID3_algorithm
9. [Online] https://en.wikipedia.org/wiki/C4.5_algorithm
10. [Online] https://scikit-learn.org/stable/
11. [Online] https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.KFold.html
12. [Online] https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.cross_val_score.html
13. [Online] https://en.wikipedia.org/wiki/Information_gain_in_decision_trees
14. [Online] https://towardsdatascience.com/entropy-how-decision-trees-make-decisions-2946b9c18c8