Business Club

Decision Trees

Rituparna Mandal | Aman Sharma

23 December 2017

Introduction

1. Motivation- A Case Study
2. The Trees
    a. What is a decision tree
    b. Representation
3. Regression v/s Classification
4. Building a model
    a. Split Criteria
        i. Gini Index
        ii. Information Gain
        iii. Chi Squared
    b. Result Criteria
    c. Tree Pruning
5. Implementation
    a. Python Implementation on our case
    b. Some theoretical insights

Decision Trees- Module 2

A. Bagging
B. Boosting

Motivation:

Classification techniques for analysts have majorly been Logistic Regression, Decision Trees and SVM. Due to some of the following limitations of Logistic Regression, Decision Trees prove to be very useful:

1. Doesn't perform well when the feature space is too large
2. Doesn't handle large number of variables/features well
3. Linear decision boundary layers

We will try to work out the implementation of Decision Trees using the Banknote dataset. You can find the data on (https://archive.ics.uci.edu/ml/datasets/banknote+authentication)

The data was extracted from specimen of forged and genuine banknotes. The data columns contain details about various continuous properties of the image scanned. The aim of the problem set is to develop a classifier which, given a set of properties of the data classify whether the note is genuine or forged.

Tree

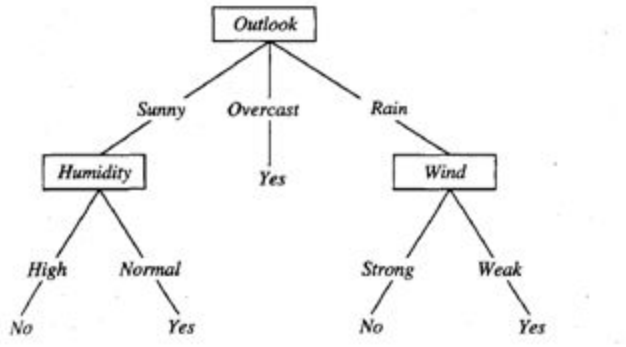
What are decision trees?

Decision trees are a powerful prediction method and extremely popular.

They are popular because the final model is so easy to understand by practitioners and domain experts alike. The final decision tree can explain exactly why a specific prediction was made, making it very attractive for operational use.

Representation

Given below is an example of a decision tree classifying whether a day is suitable for playing tennis .



A decision tree consist of three main parts - **root nodes , leaf nodes , branches** .It starts with a single node called “root node ” which then branches into “child nodes” .

At each node , a specific attribute is tested and the training data is classified accordingly.
In the above example , the root node tests the attribute “Outlook” and splits into 3 branches according to the 3 values taken by the attribute - “Sunny” , “Overcast ”and “Rain” .

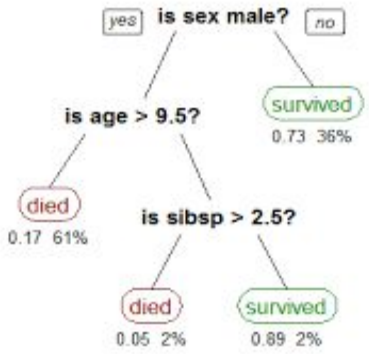
Regression vs Classification

Classification trees are used when the target variable is categorical in nature whereas Regression trees are used when the target variable is continuous or numerical in nature.

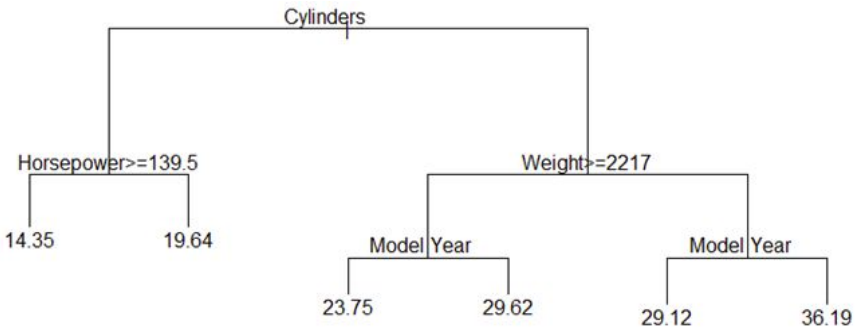
In case of linear regression , a single formula is used for prediction in the entire training data set. But if the number of features are more , they can interact in non-linear ways and modelling a single formula becomes difficult .Hence regression trees are used.

A regression tree is built through a process known as binary recursive partitioning, in which the dataset in partitioned into smaller regions until we obtain data sets where we can fit simpler models.

Classification trees works in the same manner, only the predictor variables are categorical in nature.



Classification Tree



Regression Tree

Building a Model:

Choosing a split criteria to to start forming child nodes is the first step in a building our decision trees. There are many index or measures available to test the homogeneity of a sample after a split, some of the popular ones are mentioned below

Popular cost functions :

1. Information gain :

Information gain measures the reduction in entropy caused by classifying the training data on the basis of an attribute. Higher the information gain ,better is the attribute in classifying the data.

Mathematically ,

Entropy (S) = − p1 log p1− p2 log p2 ...− pn log pn

Entropy(S | A) = sum (P(c) x E(c))

Gain (S,A) =Entropy(S) −Entropy(S | A)

Where S is the Training set and A is the attribute Entropy(S | A) is the weighted entropy of the child nodes after splitting using attribute A.

2. Gini index :

For classification the Gini index function is used which provides an indication of how “pure” the leaf nodes are (how mixed the training data assigned to each node is).

G = sum(pk * (1 – pk))

Where G is the Gini index over all classes, pk are the proportion of training instances with class k in the rectangle of interest.

For a binary classification problem, this can be re-written as:

$$G = 2 \times p_1 \times p_2$$

or

$$G = 1 - (p_1^2 + p_2^2)$$

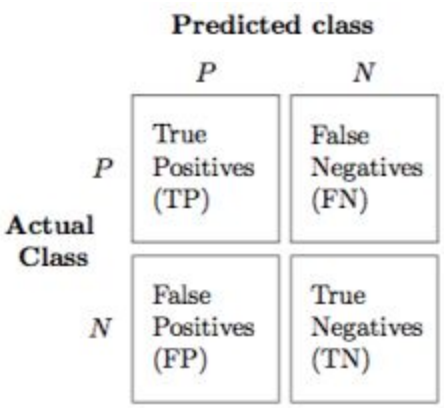
3. Chi - square test : Chi-square is another test used to determine the statistical significance between the parent node and the child-nodes after the split.

$$\text{Chi-square} = ((\text{Actual} - \text{Expected})^2 / \text{Expected})$$

Higher the value of Chi-Square higher the statistical significance of differences between parent node and the child-nodes and thus better is the attribute for classification.

Evaluating the performance of Decision trees :

Confusion Matrix : The performance of the decision tree over the test data set can be summarised in a confusion matrix .

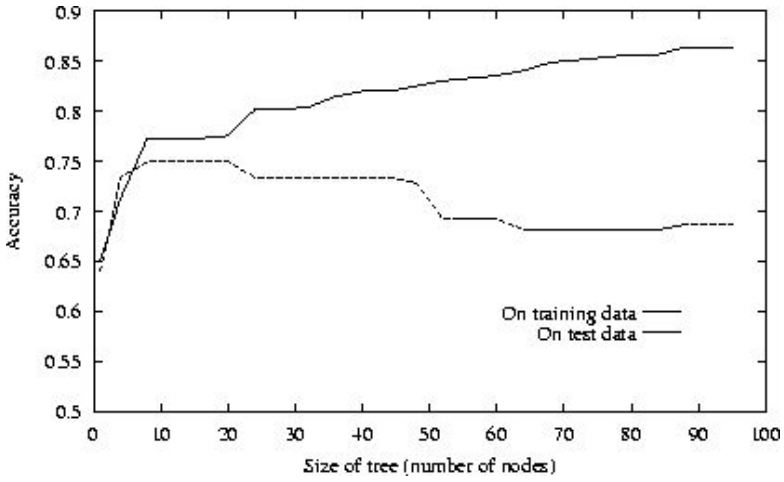


The each training example will fall into one of the four categories -

1. True Positives (TP) : All positive instances correctly classified
2. False Positives (FP) : All negative instances incorrectly classified as positive.
3. False Negatives (FN) : All positive instances incorrectly classified as negative.
4. True Negatives (TN) : All negative instances correctly classified

Accuracy can be calculated from the table as the ratio of correct predictions to total predictions . However , calculating accuracy alone is not very reliable if the number of datasets belong to each class are very different .

Decision trees are prone to overfitting . If they are grown too deep , they lose generalisation capability .



Tree pruning :

Pruning is a general technique to avoid overfitting by elimination the branches of the tree that provide little power to classify data .

Some of the popular pruning strategies are :

1. Cost Complexity pruning : A series of pruned trees are generated by removing a sub-tree and replacing it with a leaf node . The optimally pruned tree among these is the one with the least cross-validated error .
2. Reduced error pruning : In this method , a sub-tree is replaced by a leaf-node and if the prediction accuracy is not affected significantly , the changes are kept .

Random forests :

As discussed earlier , when working with large sets of data decision trees often run into the problem of overfitting. Hence, an ensemble learning technique called Random Forests is used.

Ensemble Learning refers to using multiple learning multiple learning algorithms to make better predictive models. Some of the common ensemble learning types is bagging, boosting, bayesian parameter average etc.

As the name suggests random “forest” is a combination of large number of decision “trees”. A random forest grows multiple trees.

To classify a new object based on attributes, each tree gives a classification which is then aggregated into one result .The forest chooses the classification by the number of votes from all the trees and in case of regression, it takes the average of outputs by different trees.

Random forests are considered more popular than single decision tree due to the fact that they can minimise overfitting without bringing in much error of bias .

Despite being more accurate , the random forest algorithms are computationally expensive and hard to implement .

Implementation

We’ll break down our implementation on the banknote case study into the following for steps:

- Data Exploring and Pre-processing
- Data Splitting
- Building Model
- Making Predictions

Data Exploring and Pre-processing:

This is how a snapshot of our data looks like:

```
df=pd.read_csv("banknote.csv",names=['var','skewness','kurtoisis','entropy','class'])
df.head()
```

	var	skewness	kurtoisis	entropy	class
0	3.62160	8.6661	-2.8073	-0.44699	0.0
1	4.54590	8.1674	-2.4586	-1.46210	0.0
2	3.86600	-2.6383	1.9242	0.10645	0.0
3	3.45660	9.5228	-4.0112	-3.59440	0.0
4	0.32924	-4.4552	4.5718	-0.98880	0.0

To give a brief background of the data and the column headers:

Data was extracted from images that were taken from genuine and forged banknote-like specimens. For digitization, an industrial camera usually used for print inspection was used. The final images have 400 x 400 pixels. Due to the object lens and distance to the investigated object grayscale pictures with a resolution of about 660 dpi were gained. Wavelet Transform tool were used to extract features from images.

The variables and their respective denotion goes as follows:

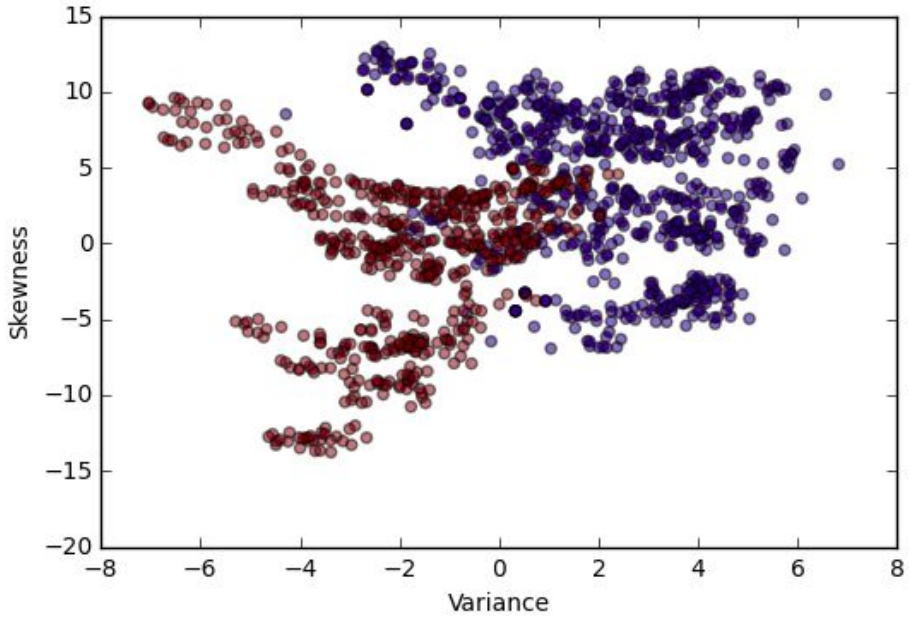
- Var: variance of Wavelet Transformed image (continuous)
- Skewness: skewness of Wavelet Transformed image (continuous)
- Kurtosis: kurtosis of Wavelet Transformed image (continuous)
- Entropy: Entropy of image(continuous)
- Class: DIscrete variable accounting for the class in which the banknote lies

The is taken from the UCI Repository http://archive.ics.uci.edu/ml/datasets/banknote+authentication
The data has no null values and any other cleaning are also not needed, only column headers need to be added and converting the txt file to csv with a comma delimiter is needed. The implementation for the same is:

```
import csv

txt_file = r"banknote.txt"
csv_file = r"banknote.csv"

in_txt = csv.reader(open(txt_file, "rb"), delimiter = ',')
out_csv = csv.writer(open(csv_file, 'wb'))

out_csv.writerows(in_txt)
```

```
import pandas as pd
from matplotlib import pyplot as plt

df=pd.read_csv("banknote.csv",names=['var','skewness','kurtosis','entropy','class'])
df.head()
```

For data exploration we start with multivariate analysis to get an idea of how the decision boundary might look like. The r




```
%matplotlib inline
fig = plt.figure()
plt.scatter(df['var'],df['skewness'],c=df['class'],alpha=0.5)
plt.xlabel('Variance')
plt.ylabel('Skewness')

plt.show()
```

A little idea about the decision boundary can be gathered from the above visualization, where the Red dots represent class 0 and blue represent class 1.

Building the Model:

We'll be using two splitting techniques and then compare the better model. In each case we would need to split the data into train and test set, done as below: (As of now we have limited the maximum tree depth, we would discuss the reasoning behind later)

```
X=df.drop(['class'],axis=1)
Y=df['class']
```

```
from sklearn.cross_validation import train_test_split
X_train, X_test, y_train, y_test = train_test_split( X, Y, test_size = 0.3, random_state = 100)
```

Using gini index as splitter:

```
from sklearn import tree
from sklearn.tree import DecisionTreeClassifier
```

```
clf_gini = DecisionTreeClassifier(criterion = "gini",max_depth=3, min_samples_leaf=5)
clf_gini.fit(X_train, y_train)
```

```
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=3,
            max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=5, min_samples_split=2,
            min_weight_fraction_leaf=0.0, presort=False, random_state=None,
            splitter='best')
```

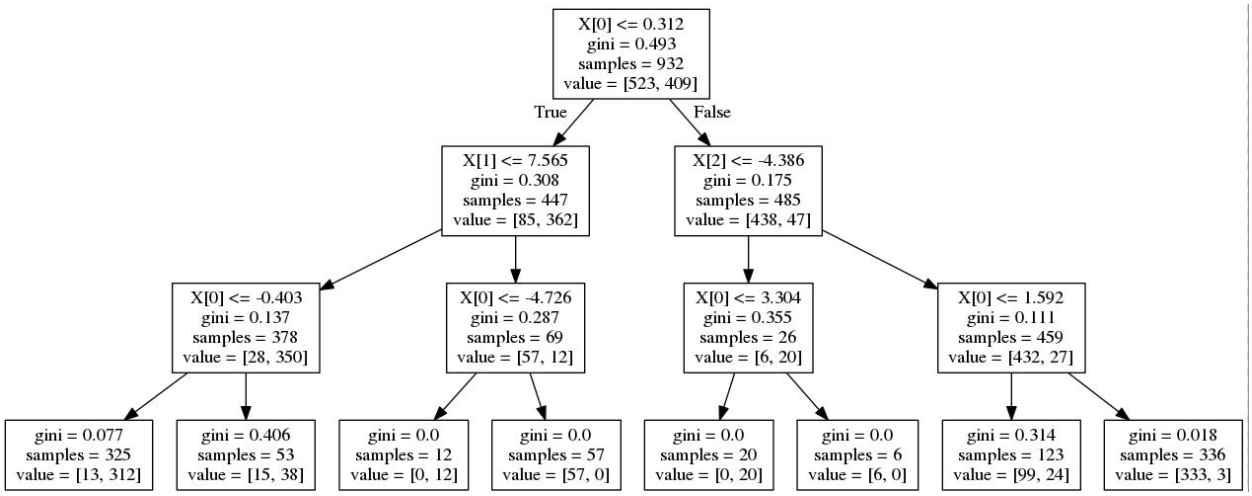
We can visualize our decision trees using:

```
tree.export_graphviz(clf_gini, out_file='tree.dot')
```

And converting the output into png format

```
aman@aman-Inspiron-15-3567:~/Python$ dot -Tpng tree.dot -o tree.png
aman@aman-Inspiron-15-3567:~/Python$
```

You would require graphviz for the above operation, it can be easily installed using the command prompt. This is how the png output of our decision tree would look like:

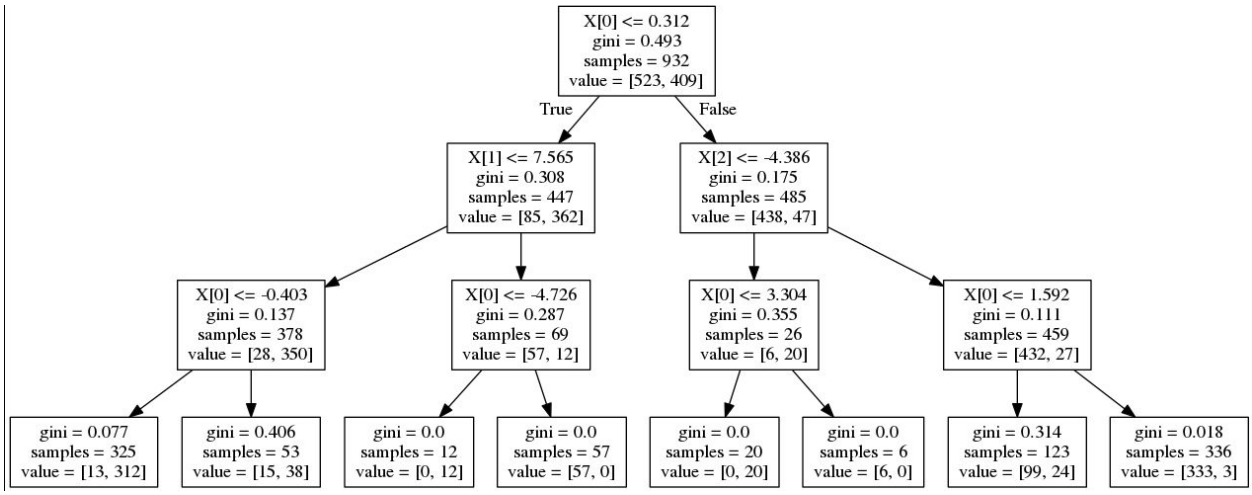


Similarly, we can use entropy as a parameter:

```
clf_entropy = DecisionTreeClassifier(criterion = "entropy", random_state = 100, max_depth=3, min_samples_leaf=5)
clf_entropy.fit(X_train, y_train)
```

```
DecisionTreeClassifier(class_weight=None, criterion='entropy', max_depth=3,
            max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=5, min_samples_split=2,
            min_weight_fraction_leaf=0.0, presort=False, random_state=100,
            splitter='best')
```

```
tree.export_graphviz(clf_gini, out_file='tree1.dot')
```



Performance of the Model

Sklearn has a module for accuracy metrics, we would fit both the models and check their accuracy

```
from sklearn.metrics import accuracy_score
print "Accuracy is ", accuracy_score(y_test,y_pred)*100
```

```
 Accuracy is  93.75
```

```
print "Accuracy is ", accuracy_score(y_test,y_pred_en)*100
```

```
Accuracy is  95.5
```

Random Forests

```
from sklearn.ensemble import RandomForestClassifier
clf_forest_gini = RandomForestClassifier(criterion='gini')
clf_forest_entropy = RandomForestClassifier(criterion='entropy')
```

```
clf_forest_gini.fit(X_train,y_train)
clf_forest_entropy.fit(X_train,y_train)
```

```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='entropy',
            max_depth=None, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators=10, n_jobs=1,
            oob_score=False, random_state=None, verbose=0,
            warm_start=False)
```

```
y_pred_gini=clf_forest_gini.predict(X_test)
y_pred_entropy=clf_forest_entropy.predict(X_test)
```

```
print "Accuracy is ", accuracy_score(y_test,y_pred_gini)*100
```

```
Accuracy is  98.75
```

```
print "Accuracy is ", accuracy_score(y_test,y_pred_entropy)*100
```

```
Accuracy is  99.25
```

As you can see the accuracy of both the models have improved significantly.

A new dimension to the study of Decision Trees comes after learning about bagging and boosting. These topics have been covered in advanced modules