Random Forest

Akshaya Rane, Mukut Ranjan Kalita, Tom Park

Feb 20th, 2019

Decision Tree overview

Supervised learning approach and can be used for classification/regression.

Idea is to ask question about features and make decisions at each step.

In practice this is how it is done:

- In training, separate the feature space into a number of smaller regions.
- To predict, find out which region your test sample belongs and use median or mode of that region to make a prediction.

So, we need a set of rules to separate the feature space

Classification tree

$$E = 1 - \max_k(\hat{p}_{mk}).$$

Classification error rate, m goes over region, k goes over classes

- ❖ Measures fraction of samples that do not belong to the most common class

$$G = \sum_{k=1}^{K} \hat{p}_{mk}(1 - \hat{p}_{mk}),$$

Gini index, m goes over region, k goes over classes

- ❖ Measures node purity, pure node G is small

$$D = -\sum_{k=1}^{K} \hat{p}_{mk} \log \hat{p}_{mk}.$$

Entropy, m goes over region, k goes over classes

- ❖ Measures node purity, pure node D is small

Since $0 \leq \hat{p}_{mk} \leq 1$, it follows that $0 \leq -\hat{p}_{mk} \log \hat{p}_{mk}$.

- ➔ G and D mostly used to evaluate quality of a particular split.
- ➔ All three can be used in pruning the tree, E is used when prediction accuracy of the final pruned tree is the goal.

Advantage/disadvantage of trees

- Easy to explain
- Closely mirrors human decision making
- Can be displayed graphically
- Can handle qualitative predictors without need to create dummy variables

➢ Do not have very good predictive accuracy

➢ Can be non robust, small change in data can lead to a completely different tree

➢ High variance, low bias, use this to improve

Random forest

From same sample build multiple trees (forest), idea is to insert randomness

For each tree:

First, randomness in samples

- ➔ Take N samples uniformly at random from dataset, some may get picked multiple times, some may not even get picked (Bootstrap).

Second, randomness in features

- ➔ At each node take sqrt(features) to build tree.

Result is a forest with T trees, each tree on a different subset of data, can use different machine

To predict

- ➔ Send test point to all trees in the forest
- ➔ Result is combination of predictions from T trees (Bagging), can be mean, median, mode, some probability.

Decision Tree Algorithm

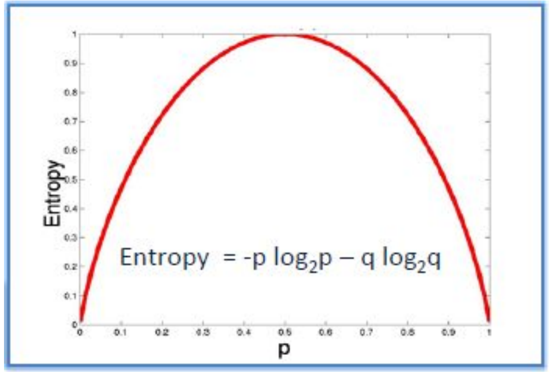
1. Start with all training data initially
2. Apply a decision to split the data using the best attribute
3. Next choose the best attribute from the remaining attributes
4. Repeat until all data is classified

https://github.com/AQM-Repos/2019RandomForest/blob/master/DT_from_scratch.ipynbtch.ipynb

Entropy and Information Gain (IG)

$$Entropy = -\sum_i p_i log_2 p_i$$

p_i --> the probability of the target class i



Entropy = -0.5 log$_2$0.5 – 0.5 log$_2$0.5 = 1

$$IG = Entropy_{parent} - Entropy_{child}$$

Decision Tree Algorithm with example

1. Start with all training data and calculate the entropy of the parent branch

In [3]:
```
train_data
```

Out[3]:

	Pclass	Gender	Survived
10	3	0	1
17	2	1	1
6	1	1	0
13	3	1	0
4	3	1	0
2	3	0	1
5	3	1	0
14	3	0	0
9	2	0	1
7	3	1	0
16	3	1	0
11	1	0	1
3	1	0	1
0	3	1	0
15	2	0	1
12	3	1	0

The features considered here are

Passenger class (Pclass): 1-->upper, 2-->middle, 3-->lower Gender: 0-->female, 1-->male

Target label: 0-->not survived, 1-->survived

At root node, before splitting

$$\text{Entropy\_parent} = -p_0 \log_2 p_0 - p_1 \log_2 p_1 = -\frac{9}{16}\log_2\left(\frac{9}{16}\right) - \frac{7}{16}\log_2\left(\frac{7}{16}\right) = 0.99$$

```python
def entropy_parent(data):
    """
    Calculate entropy of the training set using the target labels.
    The parameter data = training dataset
    """
    en = 0
    target = data.keys()[-1]
    target_labels = data[target].unique()
    for t_label in target_labels:
        p_class = data[target].value_counts()[t_label]/len(data[target])
        en += - (p_class * np.log2(p_class))
    return en
```

Decision Tree Algorithm with example

2. Calculate the entropy of each attribute (Pclass, Gender)

```
def entropy_child(data,child):
    """
    Calculate the weighted entropy for the feature in the training data. This function takes two parameters:
    1. data = The dataset for which we need to calculate the entropy of each feature
    2. child = The feature in the dataset for which the entropy should be calculated
    """
    en_child_weighted = 0
    target = data.keys()[-1]
    target_labels = data[target].unique()
    child_labels = data[child].unique()  # get the unique labels from the feature column selected as child
    for c_label in child_labels:
        en_child_feature = 0
        for t_label in target_labels:
            p_class_num = len(data[child][data[child]==c_label][data[target] == t_label])
            p_class_den = len(data[child][data[child]==c_label])
            if (p_class_num) != 0:
                p_class = p_class_num/p_class_den
                en_child_feature += - p_class * np.log2(p_class)
            else:
                en_child_feature = 0.0

        weight = p_class_den/len(data)
        en_child_weighted += (weight*en_child_feature)
    return en_child_weighted
```

3. Subtract this entropy from the parent entropy to get the information gain

4. Determine which attribute gives the maximum information gain

```
def select_max_IG_feature(data):
    """
    Calculate the information gain for each feature in the dataset and determine the feature with maximum gain.
    1. data = The dataset for which we need to determine the feature which has maximum gain
    """
    ig = []
    for child in data.keys()[:-1]:
        en_parent = entropy_parent(data)  # Calculate parent entropy
        en_child = entropy_child(data,child) # Calculate weighted entropy for each feature in the training data.
        ig_child = en_parent - en_child
        ig.append(ig_child)
    return data.keys()[:-1][np.argmax(ig)]
```

Decision Tree Algorithm with example

```
In [3]: train_data
```

Out[3]:

	Pclass	Gender	Survived
10	3	0	1
17	2	1	1
6	1	1	0
13	3	1	0
4	3	1	0
2	3	0	1
5	3	1	0
14	3	0	0
9	2	0	1
7	3	1	0
16	3	1	0
11	1	0	1
3	1	0	1
0	3	1	0
15	2	0	1
12	3	1	0

1. Using Pclass

$$\text{Entropy\_Pclass\_1} = -\frac{2}{3}\log_2\left(\frac{2}{3}\right) - \frac{1}{3}\log_2\left(\frac{1}{3}\right) = 0.92$$

$$\text{Entropy\_Pclass\_2} = -\frac{3}{3}\log_2\left(\frac{3}{3}\right) - \frac{0}{3}\log_2\left(\frac{0}{3}\right) = 0$$

$$\text{Entropy\_Pclass\_3} = -\frac{2}{10}\log_2\left(\frac{2}{10}\right) - \frac{8}{10}\log_2\left(\frac{8}{10}\right) = 0.72$$

The weighted entropy for this attribute is,

$$\text{Entropy\_Pclass} = \frac{3}{16}\text{Pclass\_1} + \frac{3}{16}\text{Pclass\_2} + \frac{10}{16}\text{Pclass\_3} = 0.62$$

And the information gain,

$$\text{IG\_Pclass} = \text{Entropy\_parent} - \text{Entropy\_Pclass} = 0.37$$

1. Using Gender

$$\text{Entropy\_Gender\_0} = -\frac{6}{7}\log_2\left(\frac{6}{7}\right) - \frac{1}{7}\log_2\left(\frac{1}{7}\right) = 0.59$$

$$\text{Entropy\_Gender\_1} = -\frac{1}{9}\log_2\left(\frac{1}{9}\right) - \frac{8}{9}\log_2\left(\frac{8}{9}\right) = 0.50$$

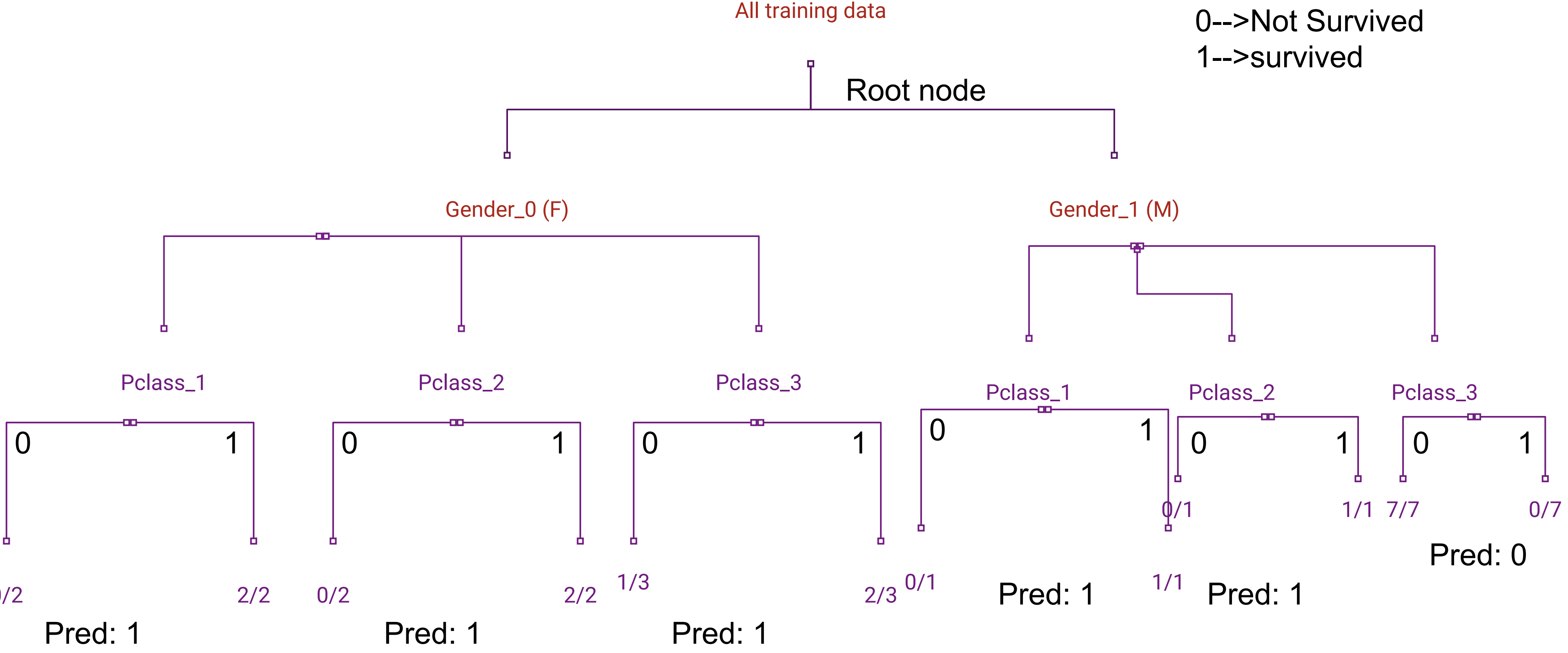
The weighted entropy for this attribute is,

$$\text{Entropy\_Gender} = \frac{7}{16}\text{Gender\_0} + \frac{9}{16}\text{Gender\_1} = 0.54$$

And the information gain,

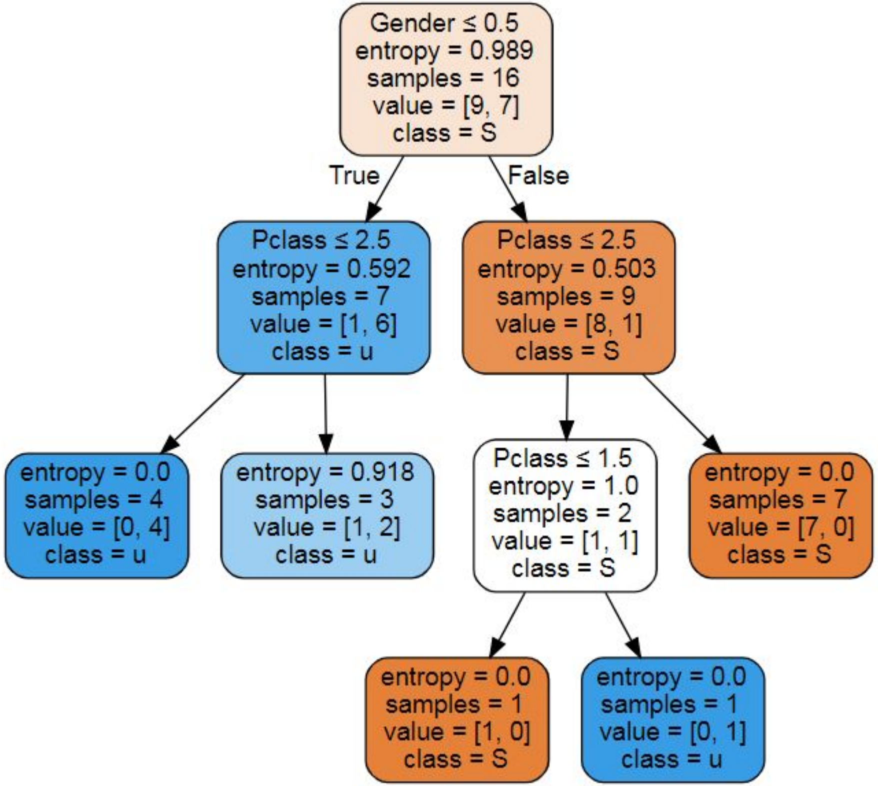
$$\text{IG\_Gender} = \text{Entropy\_parent} - \text{Entropy\_Gender} = 0.45$$

Decision Tree Algorithm with example



Decision Tree Algorithm Sklearn

Out[143]:



Hyperparameters

- Criterion
- Maximum depth
- Minimum number of samples per split
- Minimum samples per leaf

Random Forest Algorithm

- Bootstrapping

```
def RandomForest_train(data,number_of_trees):
    """Train the random forest model, this takes data and number of trees that one wants to
build
    """
    # Make a list in which single trees are stored
    random_forest_sub_tree=[]
    # Make a number of n models
    for i in range(number_of_trees):
    #Make a number of bootstrap sampled data sets from original dataset
    #Keep in mind some samples may be taken multiple times and some may
    #not even get picked
        bootstrap_sample=data.sample(frac=1,replace=False)
    #Make a tree for each of the bootstraped sample and append to the list
        random_forest_sub_tree.append(build_DTree(bootstrap_sample,bootstrap_sample,\
                                      features_col=bootstrap_sample.keys()[:-1],\
                                      target_col=bootstrap_sample.keys()[-1],\
                                      parent_node_label=None))

    return random_forest_sub_tree
```

- Hyperparameters: Number of trees, Maximum depth, Minimum number of samples per split
- Feature selection

https://github.com/AQM-Repos/2019RandomForest/blob/master/RForest_fromScratch.ipynb

Result from scratch Comparison with the package

- Prediction accuracy: Titanic data
  - Same result from implemented decision tree, random forest, and the package.
    - Accuracy score on training data: 0.9375
    - F-score on training data: 0.8974
    - Random Forest: categorical variables only, entropy
- Implemented Random Forest
  - Default: ‘entropy’ as criterion, sqrt of number of features as random feature selection
- Sklearn Package
  - n_estimator=’10’, max_features=sqrt(n_features), criterion=’entropy’
  - Other options can be used. (eg. criterion=’gini’)

Clean data, the features that we selected

1. Removed NA values
2. Generated 'Month' and 'Date' variables from 'Operation Date'
3. Dropped redundant or unnecessary variables
4. Changed the type of variables from Nominal to Ratio
5. One Hot Encoding was done for categorical variables

https://github.com/AQM-Repos/2019RandomForest/blob/master/BusBunchingClassification_clean_final.ipynb

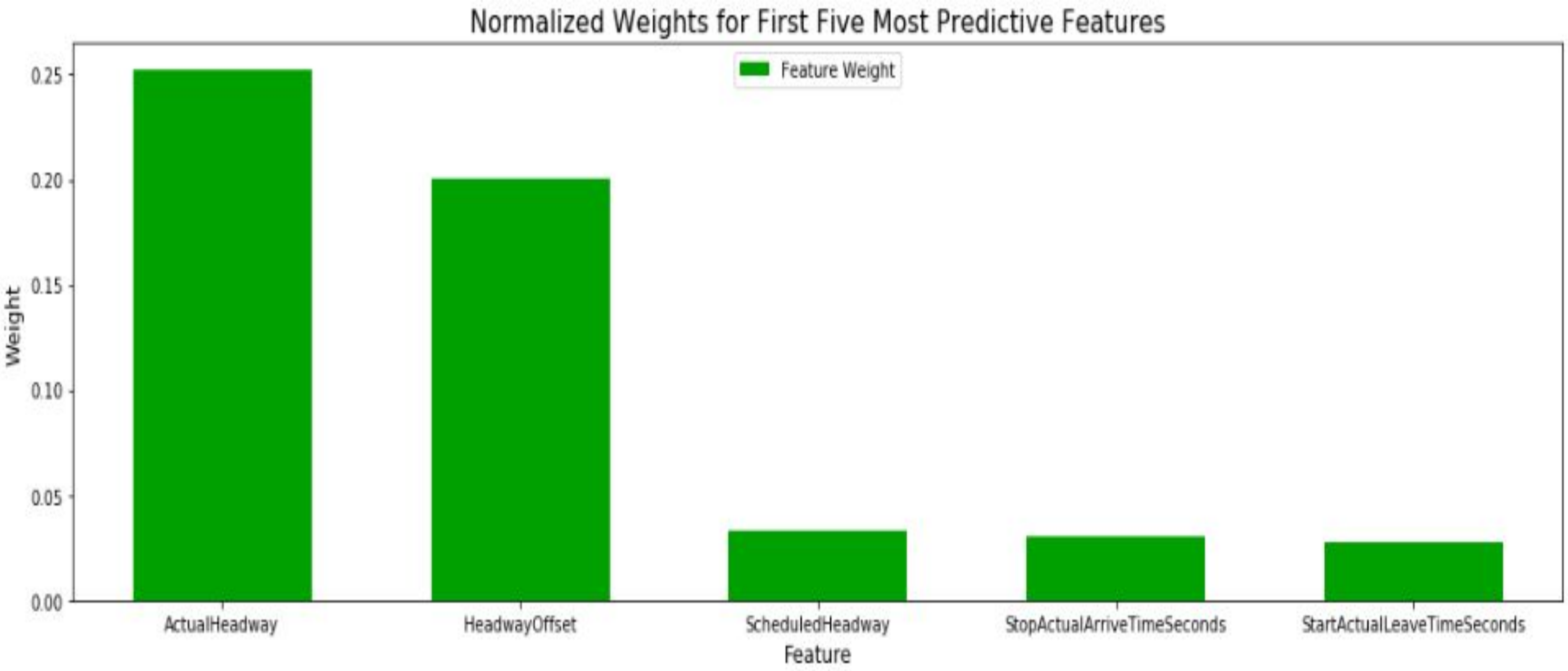
BusBunching Classification Result: Decision Tree

- Using only **one target variable** 'NextLegBunchingFlag' with Decision Tree
  - Accuracy score on training data: 1.0000
  - Accuracy score on test data: 0.982
- Grid search for Decision Tree
  - Unoptimized model
    - Accuracy score on testing data: 0.982
    - F-score on testing data: 0.798
  - Optimized Model
    - Final accuracy score on the testing data: 0.985
    - F-score on the testing data: **0.828**

BusBunching Classification Result: Random Forest

- Random Forest with one target variable
  - Accuracy score on training data: 0.9988
  - Fbeta score on training data: 0.9916
  - Accuracy score on test data: 0.9868
  - Fbeta score on test data: 0.8715
- Grid search for Random Forest
  - Optimized Model
    - Final accuracy score on the testing data: 0.9836
    - Final F-score on the testing data: 0.8329

Results



Normalized Weights for First Five Most Predictive Features
Feature Weight
0.25
0.20
0.15
0.10
0.05
0.00
Weight
ActualHeadway
HeadwayOffset
ScheduledHeadway
StopActualArriveTimeSeconds
StartActualLeaveTimeSeconds
Feature

BusBunching Classification Result

- Optimized Random Forest
  - Final accuracy score on the testing data: 0.9836
  - Final F-score on the testing data: 0.8329

- Using the top 5 important features
  - Accuracy score on training data for Random Forest: 0.9982
  - F score on training data for Random Forest: 0.9844
  - Accuracy score on test data for Random Forest: 0.9765
  - F score on test data for Random Forest: **0.7514**

BusBunching Classification Result: All Three Targets

- Random Forest with all three target variables
  - Accuracy score on training data: 0.99687
  - Fbeta score on training data for Random Forest: 0.9857
  - Accuracy score on test data: 0.9867
  - Fbeta score on test data for Random Forest: 0.8473

Reference

- https://en.wikipedia.org/wiki/Decision_tree
- https://en.wikipedia.org/wiki/Random_forest
- An Introduction to Statistical Learning with Applications in R
- Criminesi 2018, microsoft.
- Lectures from UBC prof.