

Decisions, Trees, Forests

Daniel Lawson University of Bristol

Lecture 06.1 (v2.0.0)

Signposting

- ▶ We've covered some key classification tools:
 - ▶ SVM, logistic regression, LDA,
 - ▶ meta-learning algorithms of boosting, bagging, and stacking.
- ▶ This is the final set of key classification tools: Decision Trees and Random Forests.
 - ▶ We'll also cover regression trees.
- ▶ The Workshop covers using them in practice.

Questions

- ▶ Which splits should we make in a tree?
- ▶ When should we stop splitting?
- ▶ How can we combine multiple trees?

Trees, Forests, Decisions

- ▶ **Decision trees:** are extremely flexible and can fit highly non-linear spaces.
 - ▶ They can capture arbitrary complexity in the training data
 - ▶ They tend to overfit
 - ▶ They have overly-regular shapes, aligned to features
- ▶ **Random Forests:** Combining many random, decision trees
 - ▶ Randomization fights overfitting
 - ▶ Averaging creates smoother decision boundaries
 - ▶ Remarkable predictive performance.

Important note on programming tools

- ▶ We are now moving into Machine Learning from Statistics, though there is no hard boundary.
- ▶ R has a more complete and more robustly documented toolset for statistics than does python.
- ▶ The ML toolsets start to look more cutting-edge in python than in R.
- ▶ Everything can be completed in either language, but we will switch to the most convenient tool for the job.
- ▶ There are two reasons for this:
 1. Community momentum: **sklearn is the de-facto standard**, and so new methods are incorporated into it, making it the de-facto standard. . .
 2. Native constructs. R has a good data.frame interface. python has a good list/hash interface. Both have extensions to handle everything, but working native is nicer.
- ▶ We use R and Python this week, then switch to Python.

Some data, before we start

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(
    features_pd, labels, train_size=0.8, test_size=0.2)
```

Decision Tree

- ▶ A decision tree is a sequence of conditionally evaluated tests:
- ▶ If c_1 then:
 - ▶ If c_{11} then:
 - ▶ ...
 - ▶ Else $!c_{11}$ so:
 - ▶ ...
- ▶ Else $!c_1$ so:
 - ▶ If c_{21} then:
 - ▶ ...
 - ▶ Else $!c_{21}$ so:
 - ▶ ...
- ▶ The conditions need to be chosen appropriately. How to decide?

Decision Tree Algorithm: CART

- ▶ **Classification and Regression Trees (CART)**¹
- ▶ Consider a decision tree for C classes.
- ▶ For each **feature** $j \in [1, \dots, J]$, we evaluate the **best-split** location in the feature space...
 - ▶ i.e. the one that **Minimises** the “Gini impurity”:

$$G_j = 1 - \sum_{c=1}^C p_{jc}^2 = \sum_{c=1}^C p_{jc}(1 - p_{jc}).$$

- ▶ The assignment probability p_{jc} is the probability that class c is found at this leaf of the tree, if we use feature j .
- ▶ We choose the “best” (induces many $p \rightarrow 0$ or $p \rightarrow 1$) feature as the next split.

¹CART = Classification and Regression Trees. Breiman, Leo; Friedman, J. H.; Olshen, R. A.; Stone, C. J. (1984). Classification and regression trees. See [Wei-Yin Loh's Review](#)

Gini index example

```
def sq(y):  
    return y * y  
sq = np.vectorize(sq)  
def gini(x):  
    return 1-sq(x/x.sum()).sum()
```

Gini index example

```
test=pd.DataFrame()  
test["size0"]=np.array([100,100,100])  
test["size1.1"]=np.array([50,50,50])  
test["size1.2"]=np.array([50,50,50])  
test["size2.1"]=np.array([100,0,0])  
test["size2.2"]=np.array([0,100,100])
```

Gini index example

```
test=pd.DataFrame()
test["size0"]=np.array([100,100,100])
test["size1.1"]=np.array([50,50,50])
test["size1.2"]=np.array([50,50,50])
test["size2.1"]=np.array([100,0,0])
test["size2.2"]=np.array([0,100,100])

print("Gini index initial value      =",gini(test["size0"]))
print("Gini reduction from split 1 =",gini(test["size0"]) -
      (gini(test["size1.1"])/2+gini(test["size1.2"])/2))
print("Gini reduction from split 2 =",gini(test["size0"]) -
      (gini(test["size2.1"])*1/3+gini(test["size2.2"])*2/3))
```

Gini index example

```
test=pd.DataFrame()
test["size0"]=np.array([100,100,100])
test["size1.1"]=np.array([50,50,50])
test["size1.2"]=np.array([50,50,50])
test["size2.1"]=np.array([100,0,0])
test["size2.2"]=np.array([0,100,100])

print("Gini index initial value      =",gini(test["size0"]))
print("Gini reduction from split 1 =",gini(test["size0"]) -
      (gini(test["size1.1"])/2+gini(test["size1.2"])/2))
print("Gini reduction from split 2 =",gini(test["size0"]) -
      (gini(test["size2.1"])*1/3+gini(test["size2.2"])*2/3))

Gini index initial value      = 0.66666666666667
Gini reduction from split 1 = 0.0
Gini reduction from split 2 = 0.33333333333334
```

Decision Tree Algorithm: ID3

- ▶ But is Gini Index *right*?
- ▶ ID3² (Iterative Dichotomiser 3) **Maximises** the “information gain”, by **Minimising**:

$$H_j = - \sum_{c=1}^C p_{jc} \log(p_{jc})$$

- ▶ The difference is how each probability is weighted.
- ▶ We still define “best” feature as one that makes many $p \rightarrow 0$ and $p \rightarrow 1$.
- ▶ Gini punishes large absolute-value mistakes whilst information punishes large log-scale mistakes.
- ▶ The difference is important for a tree but usually unimportant for the classifier.

²ID = Iterative Dichotomiser. Quinlan, J. R. 1986. **Induction of Decision Trees**. Mach. Learn. 1, 1 (Mar. 1986), 81-106.

ID3

```
def ylogy(y): # CAREFUL!
    ty=[max(1e-10,x) for x in y]
    return y * np.log(ty)
def id3(x):
    return -ylogy(x/x.sum()).sum()
```

ID3

```
def ylogy(y): # CAREFUL!
    ty=[max(1e-10,x) for x in y]
    return y * np.log(ty)
def id3(x):
    return -ylogy(x/x.sum()).sum()

print("ID3 index initial value      =",id3(test["size0"]))
print("ID3 reduction from split 1 =",id3(test["size0"]) -
      (id3(test["size1.1"])/2+id3(test["size1.2"])/2))
print("ID3 reduction from split 2 =",id3(test["size0"]) -
      (id3(test["size2.1"])*1/3+id3(test["size2.2"])*2/3))
```

ID3

```
def ylogy(y): # CAREFUL!
    ty=[max(1e-10,x) for x in y]
    return y * np.log(ty)
def id3(x):
    return -ylogy(x/x.sum()).sum()

print("ID3 index initial value      =",id3(test["size0"]))
print("ID3 reduction from split 1 =",id3(test["size0"]) -
      (id3(test["size1.1"])/2+id3(test["size1.2"])/2))
print("ID3 reduction from split 2 =",id3(test["size0"]) -
      (id3(test["size2.1"])*1/3+id3(test["size2.2"])*2/3))

ID3 index initial value      = 1.0986122886681096
ID3 reduction from split 1 = 0.0
ID3 reduction from split 2 = 0.6365141682948128
```


Decision Tree pruning (Information Criteria)

- ▶ Decision trees **overfit** to the data.
- ▶ Penalisation is often used:
 - ▶ Minimise $\mathcal{L}' = \mathcal{L}(\mathbf{y}, \hat{\mathbf{y}}) + \alpha|T|$
 - ▶ Where $|T|$ is the number of bipartitions in the tree, and \mathcal{L} is a log-loss.
 - ▶ All the usual caveats of Information Criteria apply.
 - ▶ Tree search is usually performed by a greedy brute force approach:
 - ▶ Evaluate \mathcal{L}' for every branch in the tree
 - ▶ Choose the sub-tree with the lowest value
 - ▶ Repeat until no cuts improve the loss
 - ▶ Alternative search approaches exist for large trees

Decision Tree pruning (Cross Validation)

- ▶ To avoid the problems with Information Criteria, Cross-validation can be used
- ▶ Choose the **sub-tree** that has the best **out of sample** predictive power.
 - ▶ With a single left-out dataset (risks overfitting),
 - ▶ Or random sets (higher variance)
- ▶ Now we have to re-compute the entire model each pruning
- ▶ Search is a computational concern

Regression Trees (simple version)

- ▶ Regression trees are constructed identically to classification trees
- ▶ Decision can follow information or squared loss
- ▶ Prediction is the average $\hat{y}_i | \{i \in c_j\} = \bar{y}_{c_j}$ inside the leaf j
- ▶ Comparing to classification: if $y \in \{0, 1\}$ this reduces to:

$$R_j = \frac{1}{N_j} \sum_{i \in c_j} (y_i - \hat{y}(x_i))^2 \quad (1)$$

$$= \frac{1}{N_j} \sum_{i \in c_j} (y_i^2 - 2y_i\hat{y}(x_i) + \hat{y}(x_i)^2) \quad (2)$$

- ▶ Which since $\hat{y} = \bar{y}$, simplifies to:

$$R = \bar{y} - 2\bar{y}^2 + \bar{y}^2 \quad (3)$$

$$= \bar{y}(1 - \bar{y}) \quad (4)$$

Comparing Regression Trees to Classification Trees

- ▶ Similarly the Gini index is:

$$G = \sum_{j=1}^J p_j(1 - p_j) \quad (5)$$

$$= \bar{y}(1 - \bar{y}) \quad (6)$$

- ▶ So regression trees and classification trees use equivalent loss functions in CART.

General Regression Trees

- ▶ Within each decision node (leaf) we fit a model
- ▶ This is typically a **constant** model, i.e. the average
- ▶ Why?
 - ▶ The piecewise constant model fit can be arbitrarily good (number of splits scales with data volume)
 - ▶ Making non-constant models consistent is computationally costly
- ▶ Why not?
 - ▶ Computational cost grows with tree depth
 - ▶ Often locally the structure is linear
- ▶ Leads to Local Regression Trees ³
 - ▶ Complex if we ensure that the local regressions meet up

³Karalic A, "Employing Linear Regression in Regression Tree Leaves" (1992)

Decision tree notes

- ▶ In practice, **bagging** (bootstrapping the data) is important, to prevent overfitting and for smoothing the output
- ▶ The choice of feature space directly affects the decisions that are examined. So **LDA** or similar could usefully be applied to obtain “orthogonal” feature space, **reducing depth**.
- ▶ There are parameters, e.g. depth/stopping criterion/split rule, which could be chosen by cross-validation.

Fitting a Decision Tree in Python

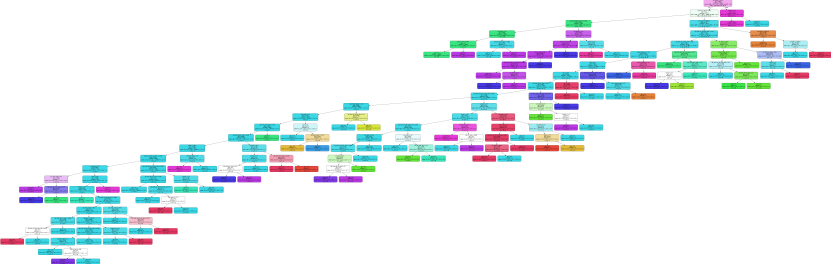
```
from sklearn import tree
cldt = tree.DecisionTreeClassifier()

trained_model_d= cldt.fit(X_train, y_train)
y_pred_d = cldt.predict(X_test)
error_d = zero_one_loss(y_test, y_pred_d)
```

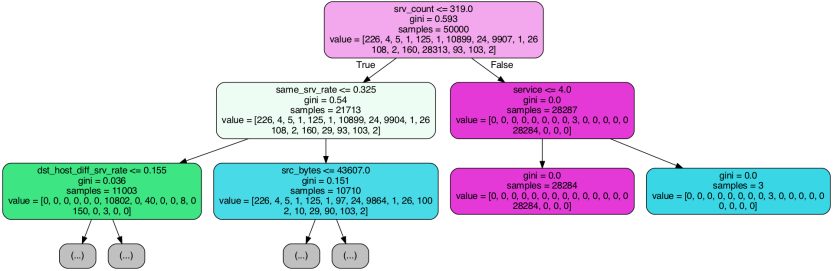
Plot a Decision Tree in Python

```
from sklearn.externals.six import StringIO
from IPython.display import Image
from sklearn.tree import export_graphviz
import pydotplus
dot_data = StringIO()
export_graphviz(cldt, out_file=dot_data,max_depth=2,
                feature_names=X_train.columns.values,
                filled=True, rounded=True)
graph = pydotplus.graph_from_dot_data(
    dot_data.getvalue())
Image(graph.create_png())
```

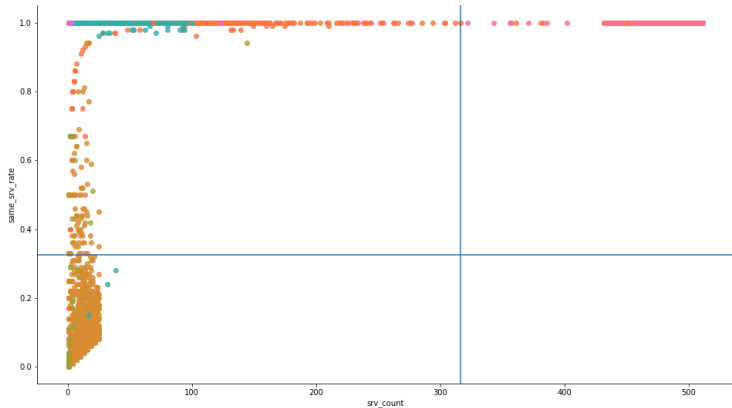

Decision Tree (whole)



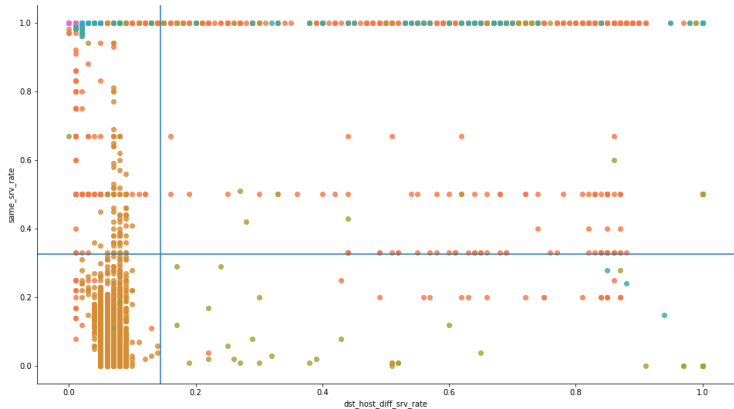
Decision Tree (root)



Decision Tree in feature space (1)



Decision Tree in feature space (2)



Boosted decision tree

- ▶ As noted previously, adaboost is using a **sequence of decisions** to make a boosted classifier.
- ▶ By default it uses a boosted depth=1 decision tree, i.e. classifiers were just the features. This is called a **decision stump**.
- ▶ You can use deeper trees, eg. with xgboost⁴; usually the depth is limited to control learning cost and complexity
- ▶ Boosting in theory doesn't need trees so the difference is about **learning rate** and **computational complexity**

⁴J. Elith, J. Leathwick, and T. Hastie “A working guide to boosted regression trees” (2008). British Ecological Society.

Random Forest

- ▶ A **random forest** is a set of **decision trees** that are combined together to perform classification.
- ▶ For each of T trees, the following steps are run:
 - a) Choose which of J variables to include:
 - ▶ Choose m_f **random features**. The canonical choice is $m_f = \sqrt{J}$.
 - ▶ Like **bagging for features**? (downsampling **without** replacement)
 - b) Learn a Tree classifier independently as above.

Random Forest outputs

- ▶ The Random Forest **combines decision trees** into a classification by:
 - ▶ Weighting each tree according to its performance
 - ▶ Report the weighted vote
- ▶ It is also possible to extract **feature importance**:
 - ▶ How much a feature decreases the score, averaged over all trees
 - ▶ Features that are never used will get a score of 0
 - ▶ Features that are important in every tree in which they appear will get a high score
 - ▶ Features that are correlated will often split their importance

Random Forest vs boosted decision tree

- ▶ Gradient Boosting Machine (GBM) is the go-to boosted decision tree
- ▶ GBM and RF differ in the way the trees are built, the order, and the way the results are combined
- ▶ RF can be trivially parallellized
- ▶ GBMs seem to outperform RFs under competition conditions, but do worse when their parameters are untuned⁵

⁵<http://fastml.com/what-is-better-gradient-boosted-trees-or-random-forest/>

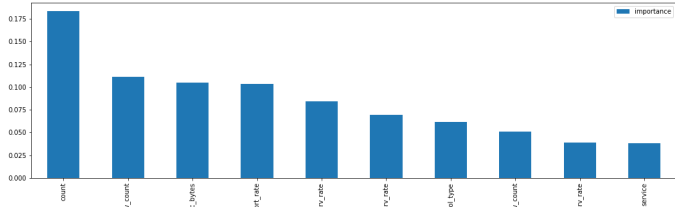
Random Forest algorithm

```
from sklearn.ensemble import RandomForestClassifier
clf= RandomForestClassifier(n_jobs=-1,
    random_state=3, n_estimators=102)
trained_model= clf.fit(X_train, y_train)
clf_score=trained_model.score(X_train, y_train)
y_pred = clf.predict(X_test)
```

Random Forest Feature Importance

```
feature_importances = pd.DataFrame(clf.feature_importances_,  
    index = X_train.columns,  
    columns=['importance']).sort_values('importance',  
    ascending=False)  
  
feature_importances.nlargest(10,  
    columns=['importance']).plot(kind='bar',figsize=(18, 5))
```

Random Forest Feature Importance



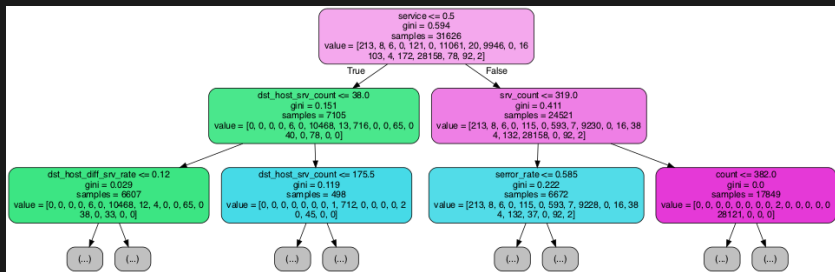
Random Forest Extract single trees

```
estimator5 = clf.estimators_[5]

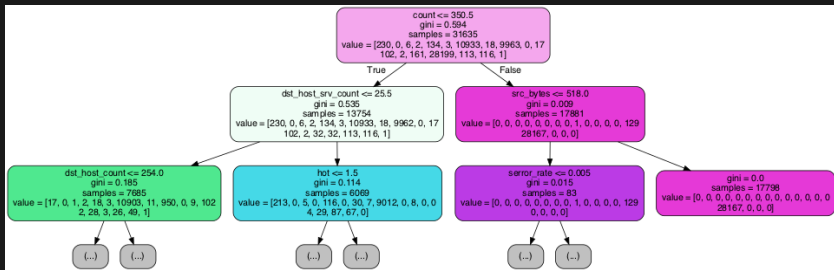
dot_data = StringIO()
export_graphviz(estimator5, out_file=dot_data,max_depth=2,
                feature_names=X_train.columns.values,
                filled=True, rounded=True)

graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
Image(graph.create_png())
```

Random Forest Feature Trees



Random Forest Feature Trees



Final thoughts

- ▶ Random Forests are typically better than bagged decision trees
- ▶ There are theoretical examples where either dominates
- ▶ Boosting changes things but isn't a magic bullet
- ▶ Usually worth being open minded; the differences could be seen as tuning parameters of a more general algorithm

Reflection

- ▶ Why are Random Forests considered to be important?
- ▶ What variations on Random Forests can you think of? Under what circumstances would you expect them to work?
- ▶ What does a Random Forest decision boundary look like? How dependent are they on the specific choice of features?

Signposting

- ▶ **References:**
- ▶ Tree methods:
 - ▶ Chapter 9.2 of **The Elements of Statistical Learning: Data Mining, Inference, and Prediction** (Friedman, Hastie and Tibshirani).
 - ▶ Penn State U Applied Data Mining and Statistical Learning **How to prune trees**
 - ▶ **Decision Tree Algorithms: Deep Math ML**
- ▶ Regression Trees:
 - ▶ Karalic A, **"Employing Linear Regression in Regression Tree Leaves"** (1992) ECAI-92

Signposting

- ▶ **References (2):**
- ▶ Boosted Decision Trees:
 - ▶ J. Elith, J. Leathwick, and T. Hastie “**A working guide to boosted regression trees**” (2008). British Ecological Society.
- ▶ CART:
 - ▶ CART = Classification and Regression Trees. Breiman, Leo; Friedman, J. H.; Olshen, R. A.; Stone, C. J. (1984). Classification and regression trees.
 - ▶ **Wei-Yin Loh's 2011 Review** is popular.
- ▶ ID3: Quinlan, J. R. 1986. **Induction of Decision Trees**. Mach. Learn. 1, 1 (Mar. 1986), 81-106.

Signposting

- ▶ In the practical we'll implement these models in R and Python; compare implementations, and to previous results.
- ▶ Next semester we'll start with the “other” LDA (Latent Dirichlet Allocation), Topic Modelling, and Modelling Documents.
- ▶ **References:**
 - ▶ Chapter 15 of **The Elements of Statistical Learning: Data Mining, Inference, and Prediction** (Friedman, Hastie and Tibshirani).
 - ▶ **Implement a Random Forest From Scratch in Python**
 - ▶ **A Gentle Introduction to Random Forests at CitizenNet**
 - ▶ **DataDive on Selecting good features**
 - ▶ **Cosma Shalizi on Regression Trees**
 - ▶ **Gilles Louppe PhD Thesis: Understanding Random Forests**