Decisions, Trees, Forests, (Part 1, Trees)

Daniel Lawson University of Bristol

Lecture 06.1.1 (v1.0.1)

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.
- Lecture 6.1 is split into two parts:
  - 6.1.1 Trees
  - 6.1.2 Forests
- The Workshop covers using them in practice.

ILOs

- Primarily:
  - ILO2 Be able to use and apply basic machine learning tools

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 that 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, and switch to Python for Semester 2.

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** $i \in [1, \cdots, M]$, we evaluate the **best-split** location in the feature space…
  - i.e. the one that minimises the "Gini impurity":

$$G_i = 1 - \sum_{j=1}^{J} p_{ij}^2 = \sum_{j=1}^{J} p_{ij}(1 - p_{ij}).$$

  - The assignment probability p_{ij} is the probability that class j is found at this leaf of the tree, if we use feature i.
  - We choose the best feature as the next split.

[1] CART = 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    = 0.666666666667
Gini reduction from split 1 = 0.0
Gini reduction from split 2 = 0.333333333334
```

Decision Tree Algorithm: ID3

- But is Gini Index *right*?
- ID3[2] instead maximises the "information gain", by minimising:

$$H = -\sum_{i=1}^{J} p_i \log(p_j)$$

  - The only difference is how each probability is weighted.
  - Gini punishes large absolute-value mistakes whilst information punishes large log-scale mistakes.
  - The difference is rarely important.

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

ID3

```
def ylogy(y):
    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):
    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):
    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:

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

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

- 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:

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

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

- 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

[3] Karalic A, "Employing Linear Regression in Regression Tree Leaves" (1992) ECAI-92

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

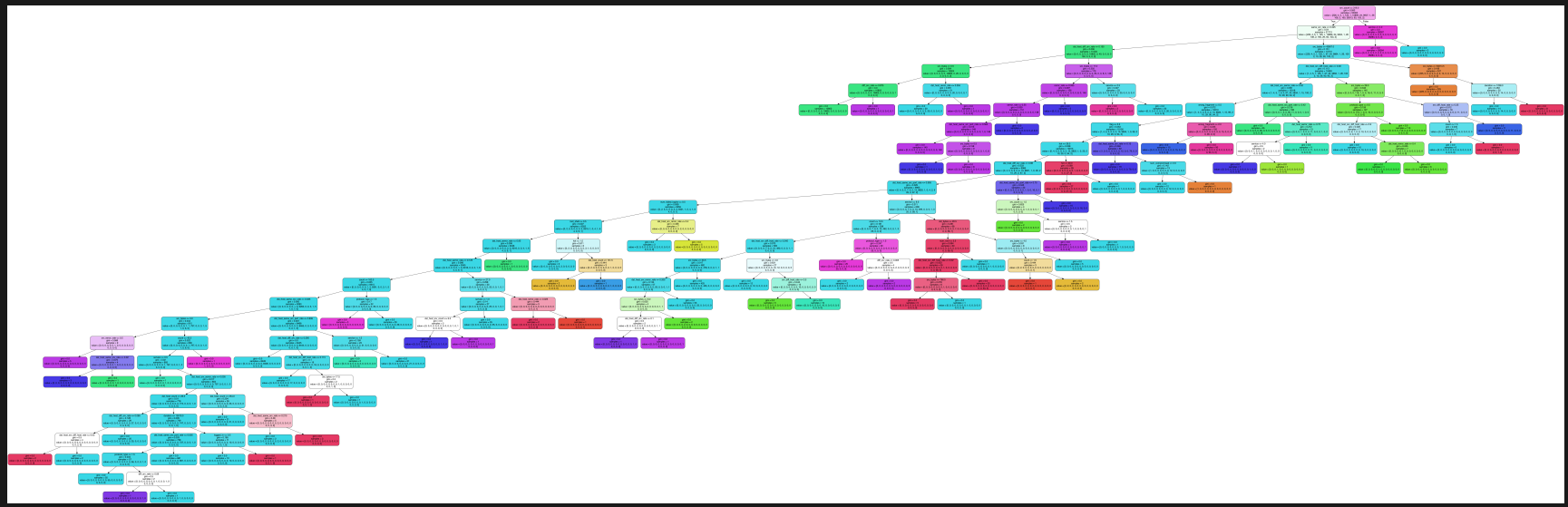
```
from sklearn import tree
cldt = tree.DecisionTreeClassifier()

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)
```

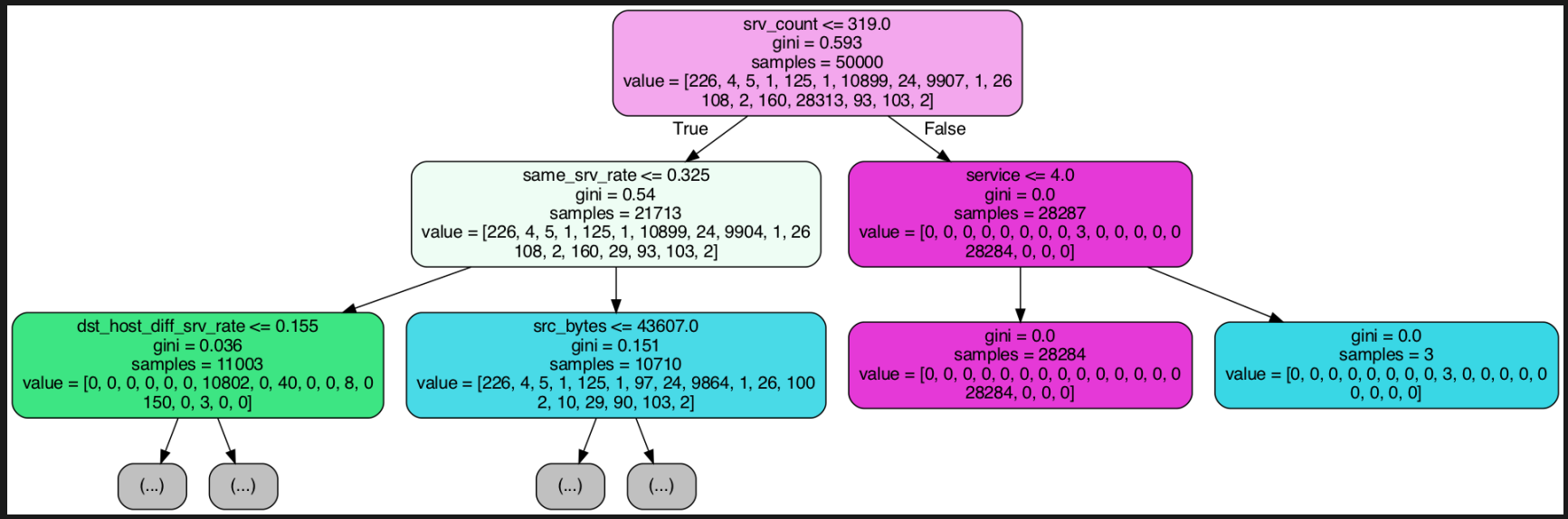
Plot a Decision Tree in Python

```
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())
```

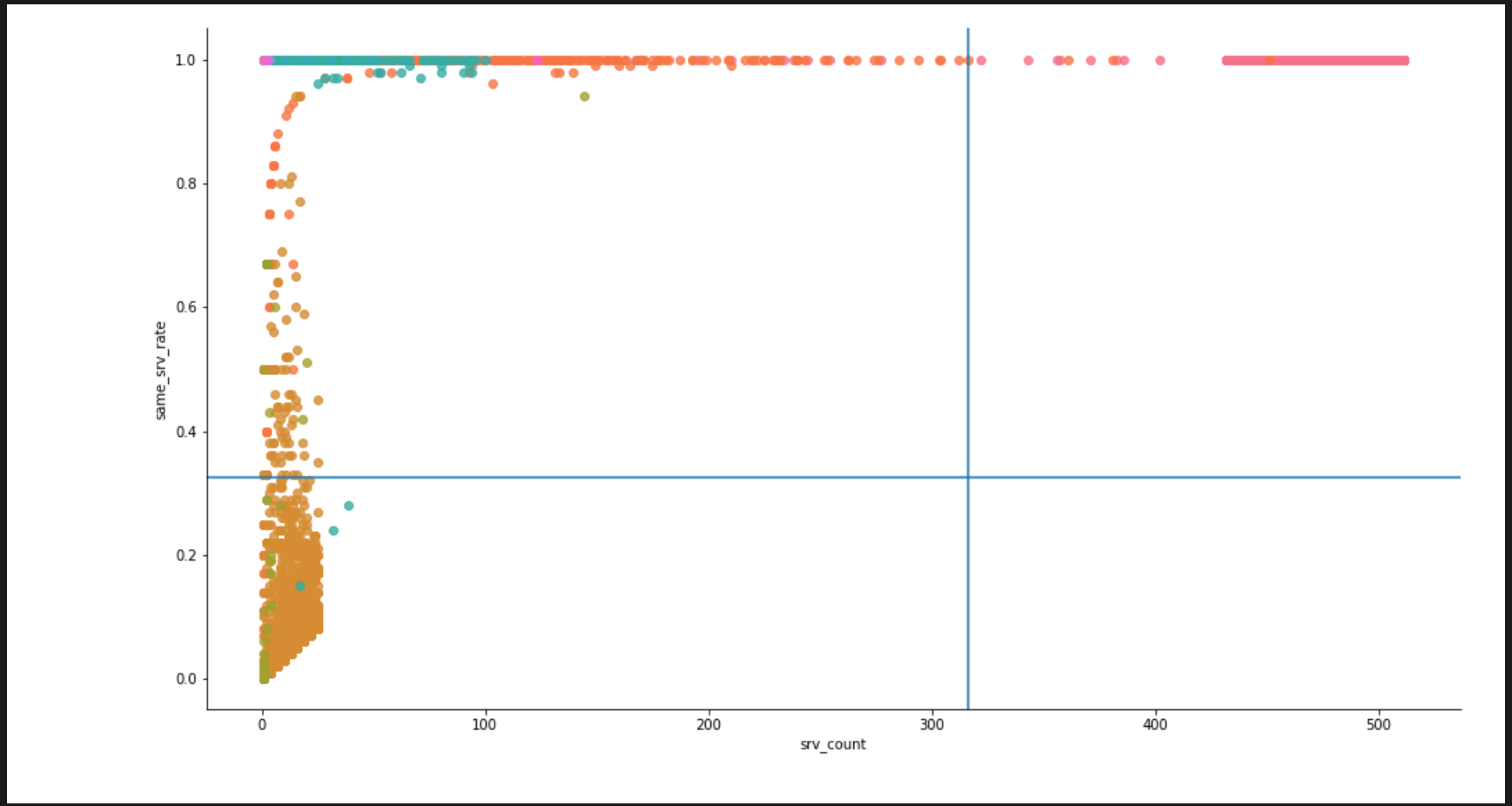
Decision Tree (whole)



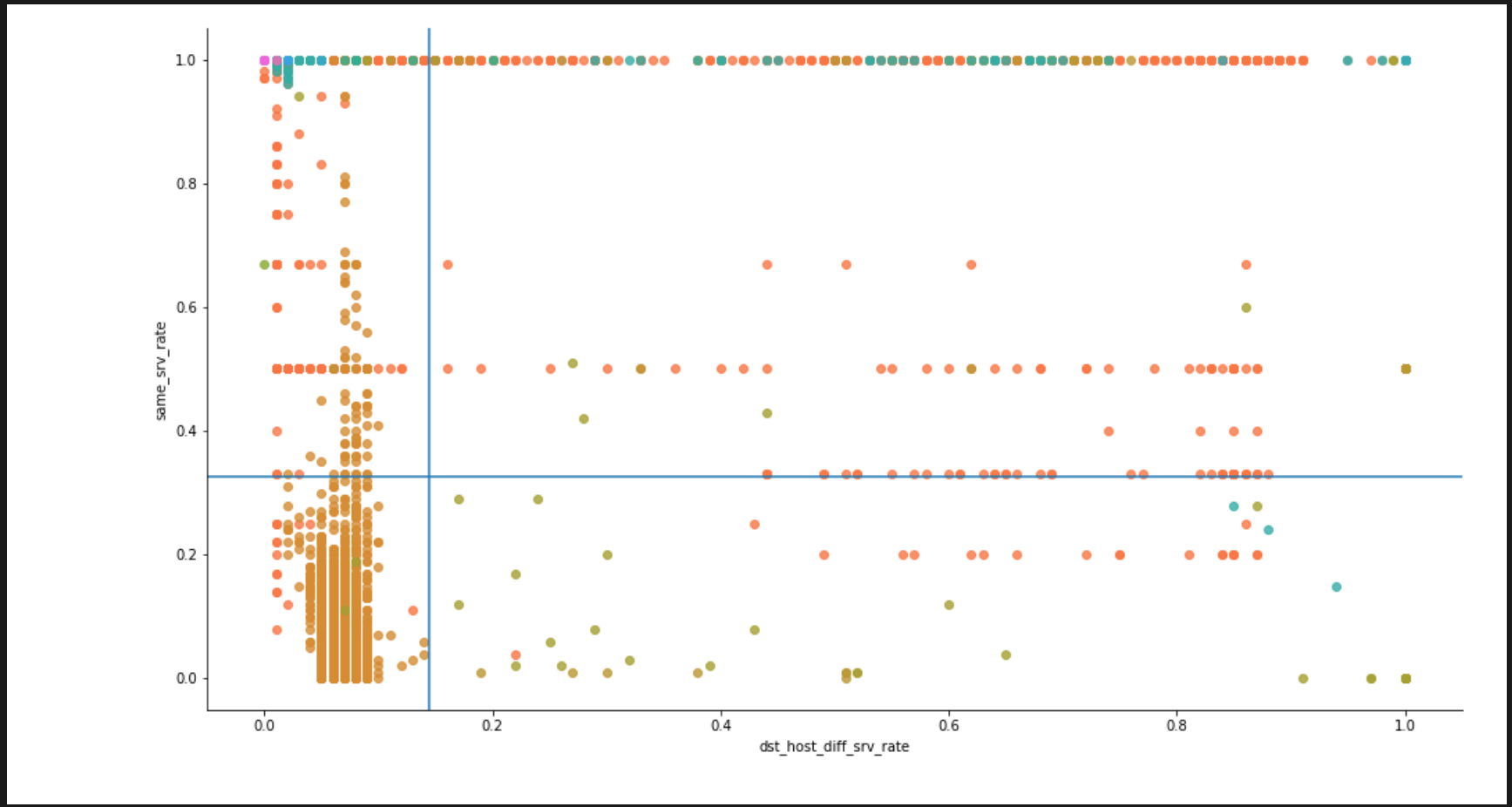
Decision Tree (root)



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 xgboost[4]; 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**

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

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.