Foundations of Machine Learning
CentraleSupélec — Fall 2017

9. Tree-based approaches

Chloé-Agathe Azencott
Centre for Computational Biology, Mines ParisTech
chloe-agathe.azencott@mines-paristech.fr

Learning objectives

- Build **decision trees:**
  - Decide how to grow a tree
  - Decide when to stop growing a tree
- Explain why they are examples of **non-metric learning** and **hierarchical learning.**
- **Combine decision trees** (or other **weak learners**) to make more powerful classifiers.

Decision trees

Hierarchical learning

- **Single-stage** classifiers:
  - Assign a class to object **x** using a single operation.
  - Use a single set of features for all classes.
  - Difficulties:

    - when classes have **multi-modal** distributions

    - when features are **nominal.**
- **Hierarchical** classifiers:

  Multiple successive tests.

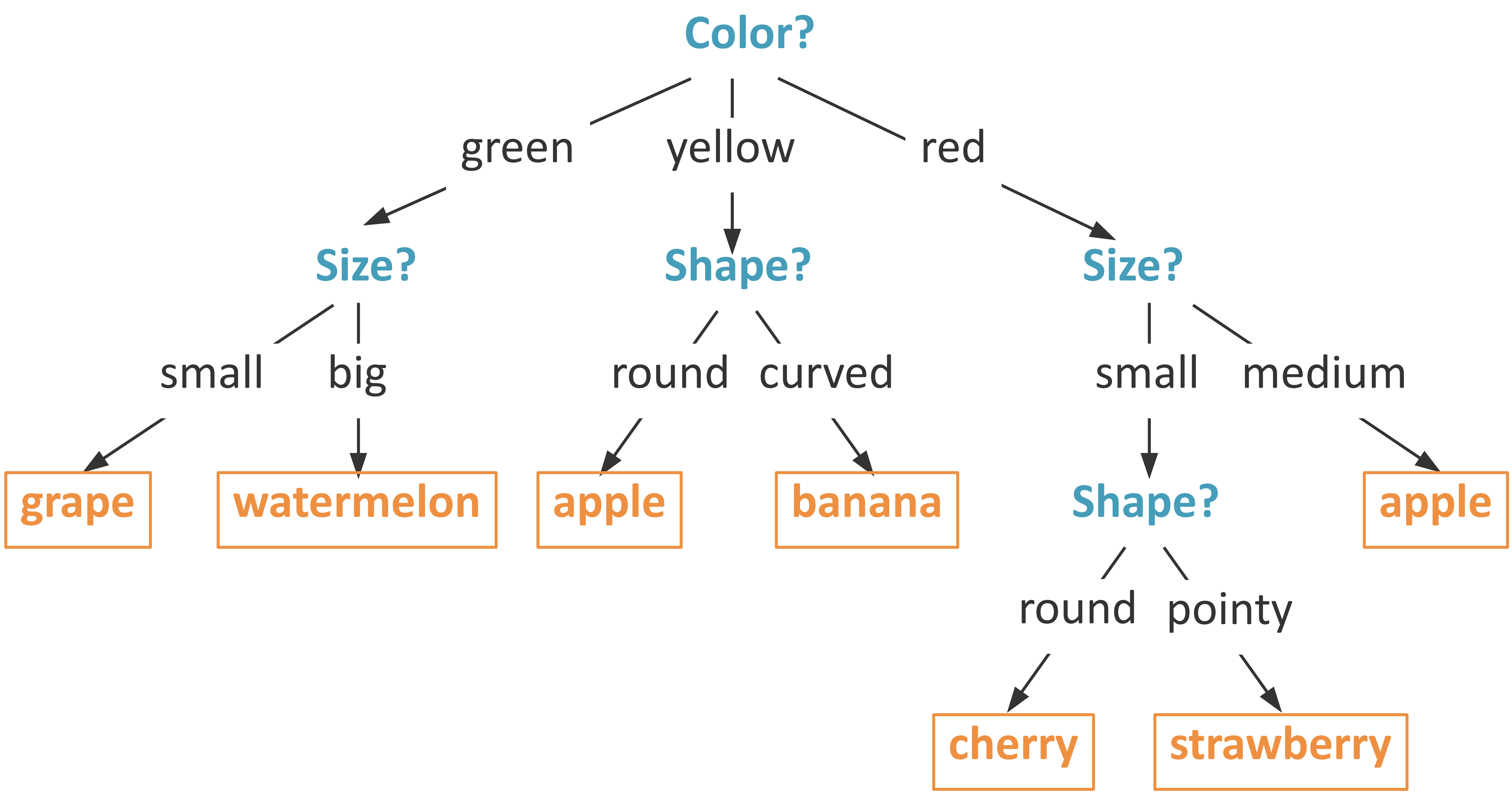
Nominal data

- Attributes that are
  - **Discrete**
  - Without any natural notion of **similarity/ordering**

    → **Non-metric learning.**
- Example:

  Classify fruit from {color, shape, texture, size}.

Decision trees: The 20Q game



Multiclass classification

- **One-versus-all**

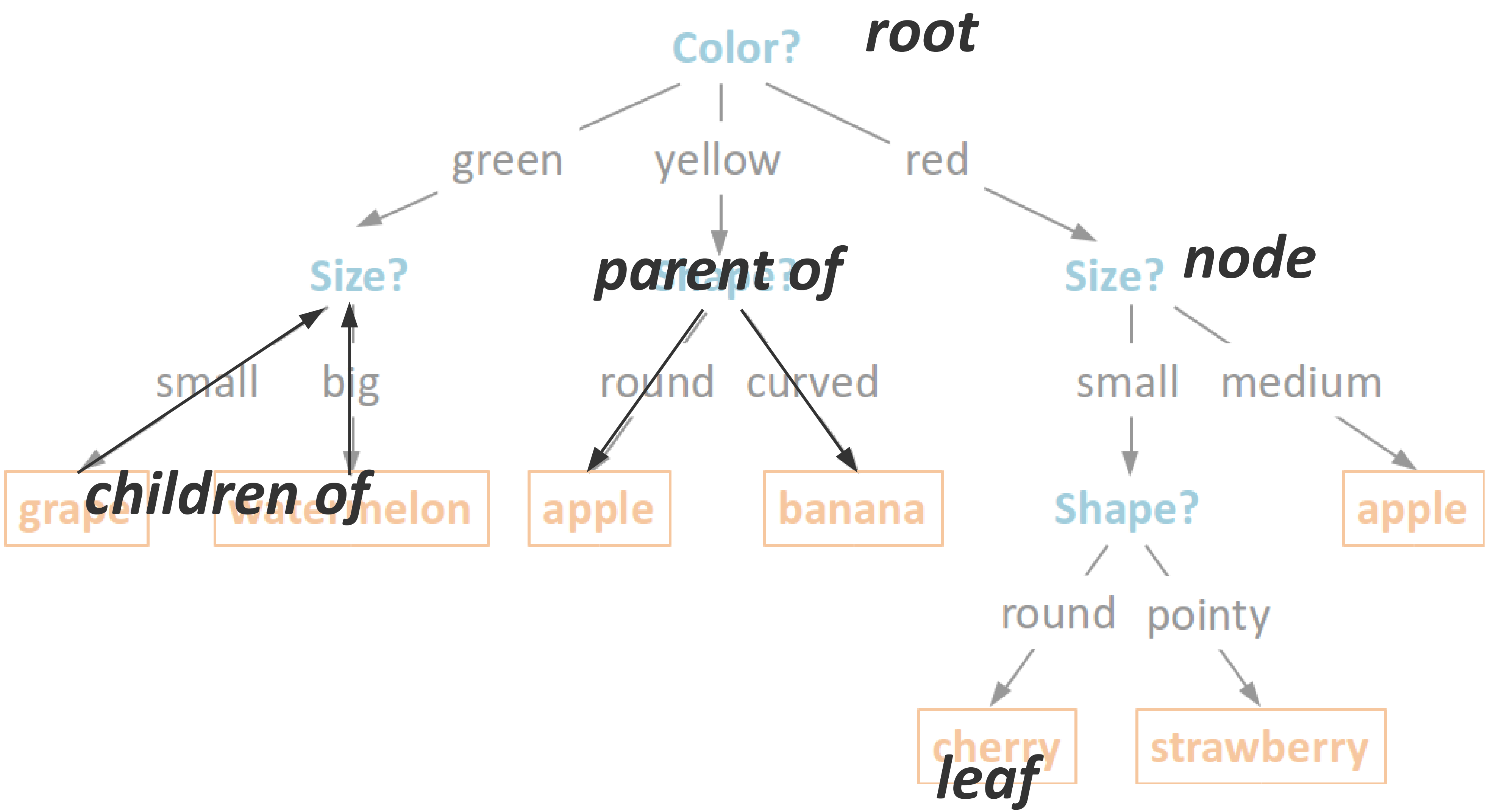
  Build K classifiers, make them vote.

- **One-versus-one**

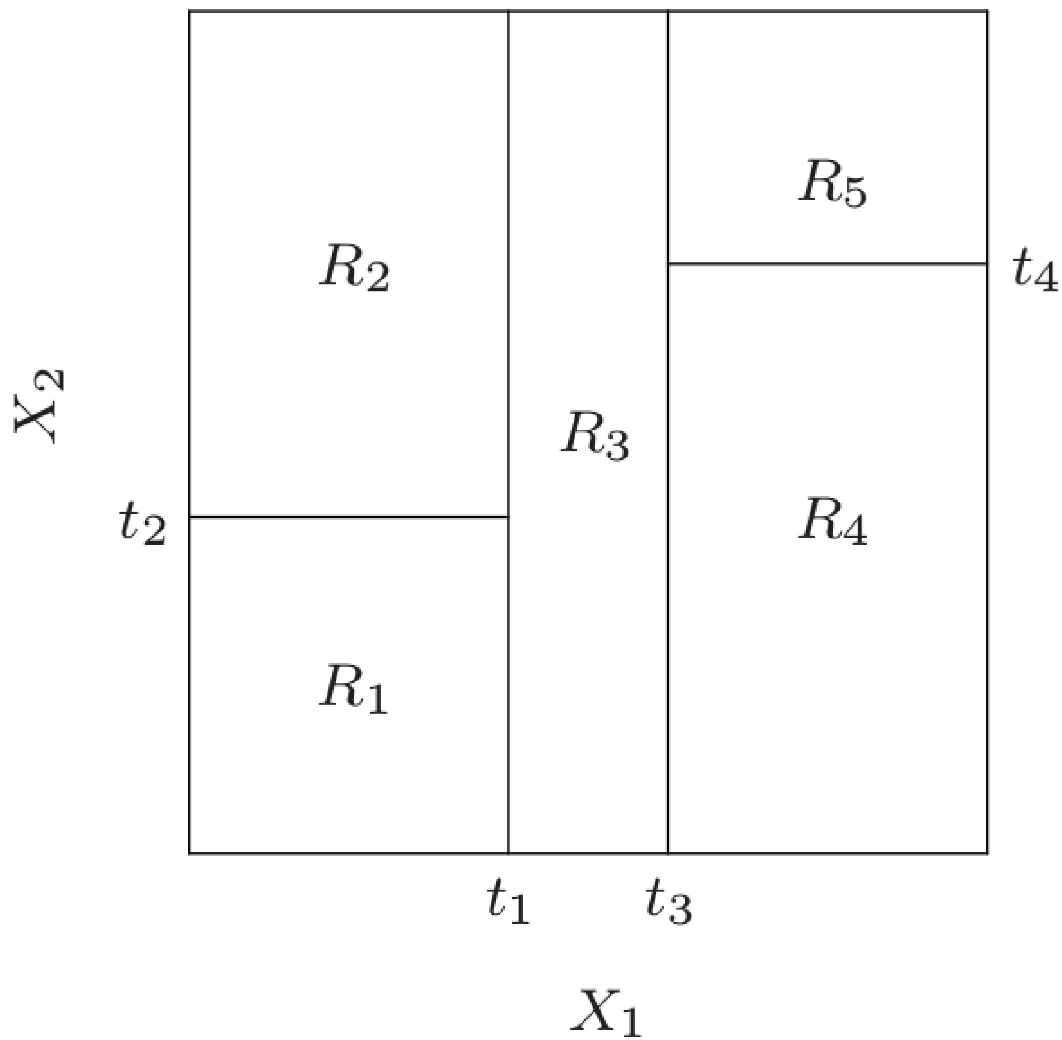
  Build K (K-1) / 2 classifiers, make them vote.

- Use an algorithm that **naturally handles multiple classes.**
  - Decision trees and their variants
  - Neural networks (Chap. 11)

Decision trees: The 20Q game



Partition of the feature space



$$f(x) = \sum_{m=1}^{M} c_m I(x \in R_m)$$

indicator

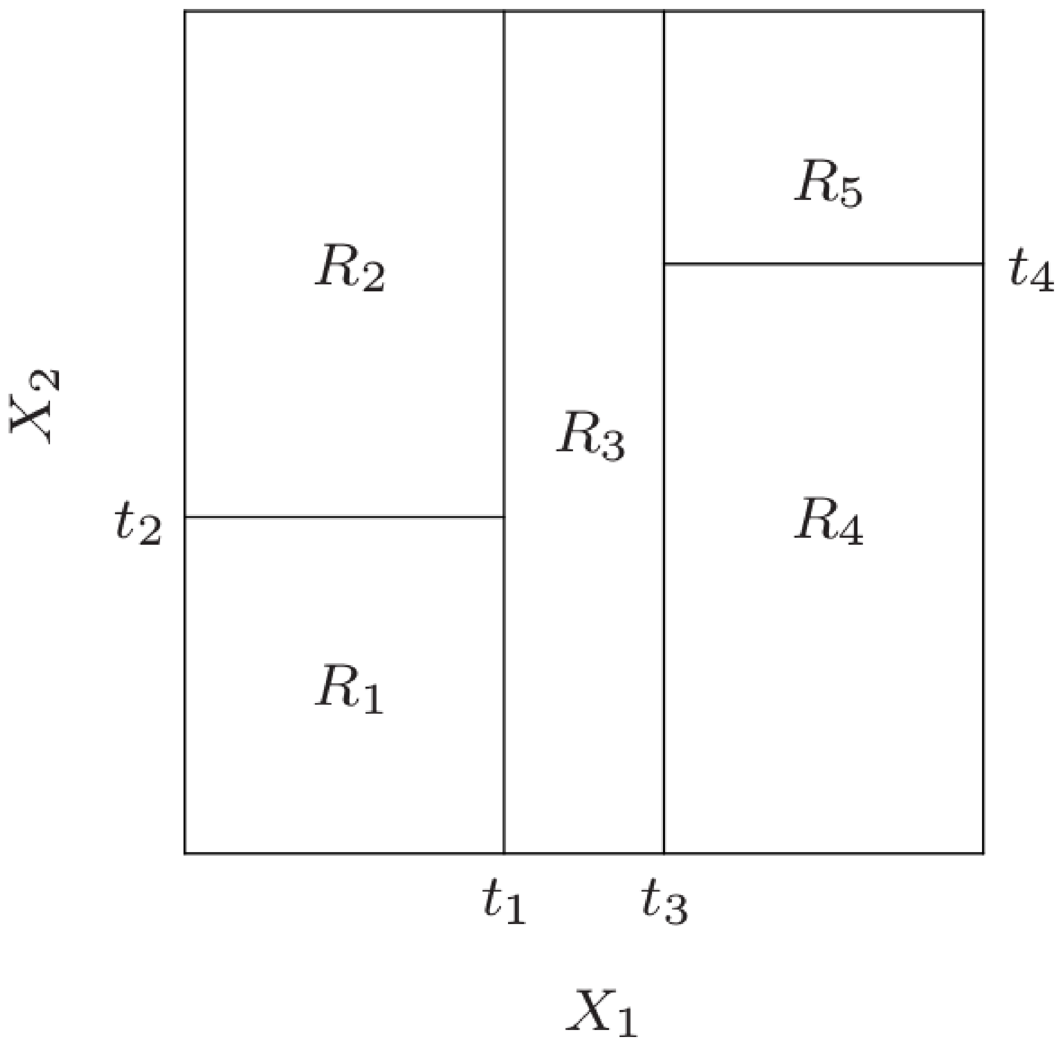
- **Classification:**

  c_m =

- **Regression:**

  c_m =

Partition of the feature space



$$f(x) = \sum_{m=1}^{M} c_m I(x \in R_m)$$

- **Classification:**

  c_m = majority vote in region.

- **Regression:**

  c_m = average value in region.

Partition of the feature space

- A decision tree is a **recursive partition** of the training set into smaller and smaller subsets.
- **Purity:** a node is **pure** if all samples at that node have the same class label.
- **CART: Classification And Regression Trees**

  Recursive procedure to split a training set and organize it into a tree.

CART: Design choices

- **Binary** or **multi-way** splits?
- Which **feature(s)** to use at each node?

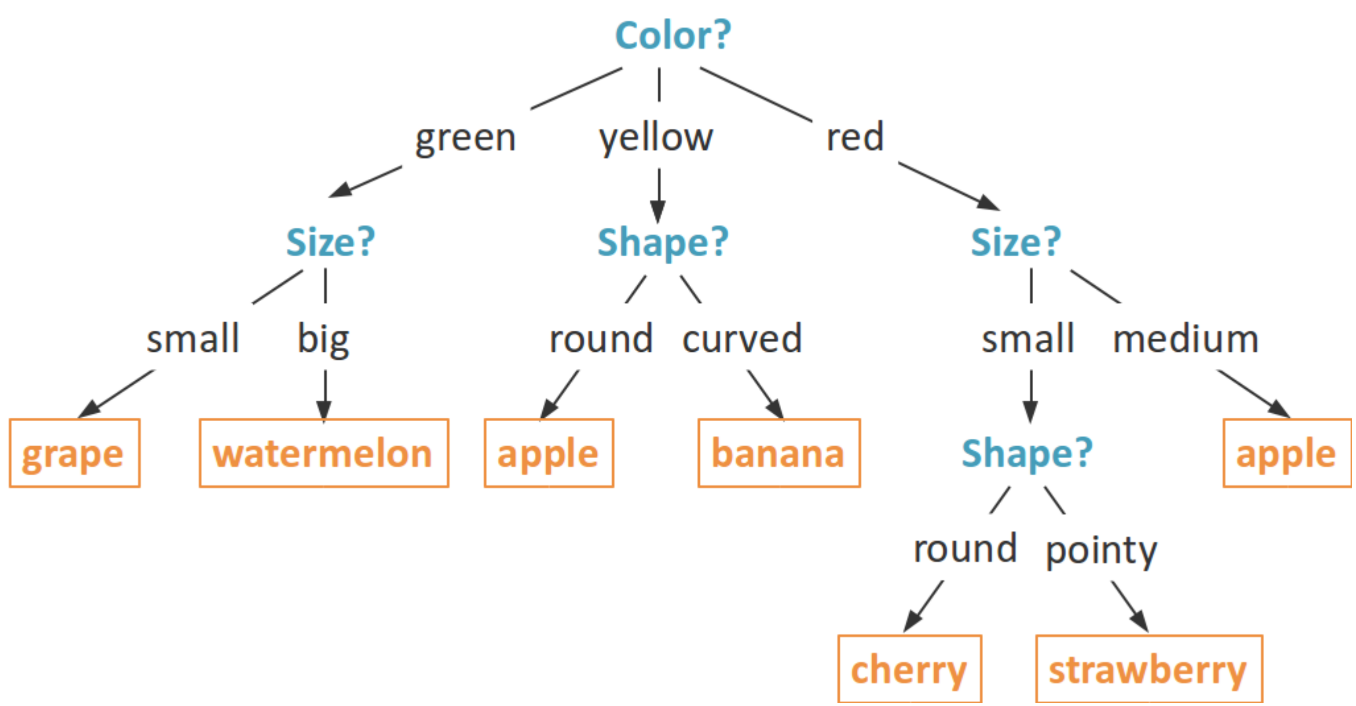
  i.e. how to split?
- When to **stop** growing a tree?

Binary vs. non-binary trees

Binary & multi-value splits

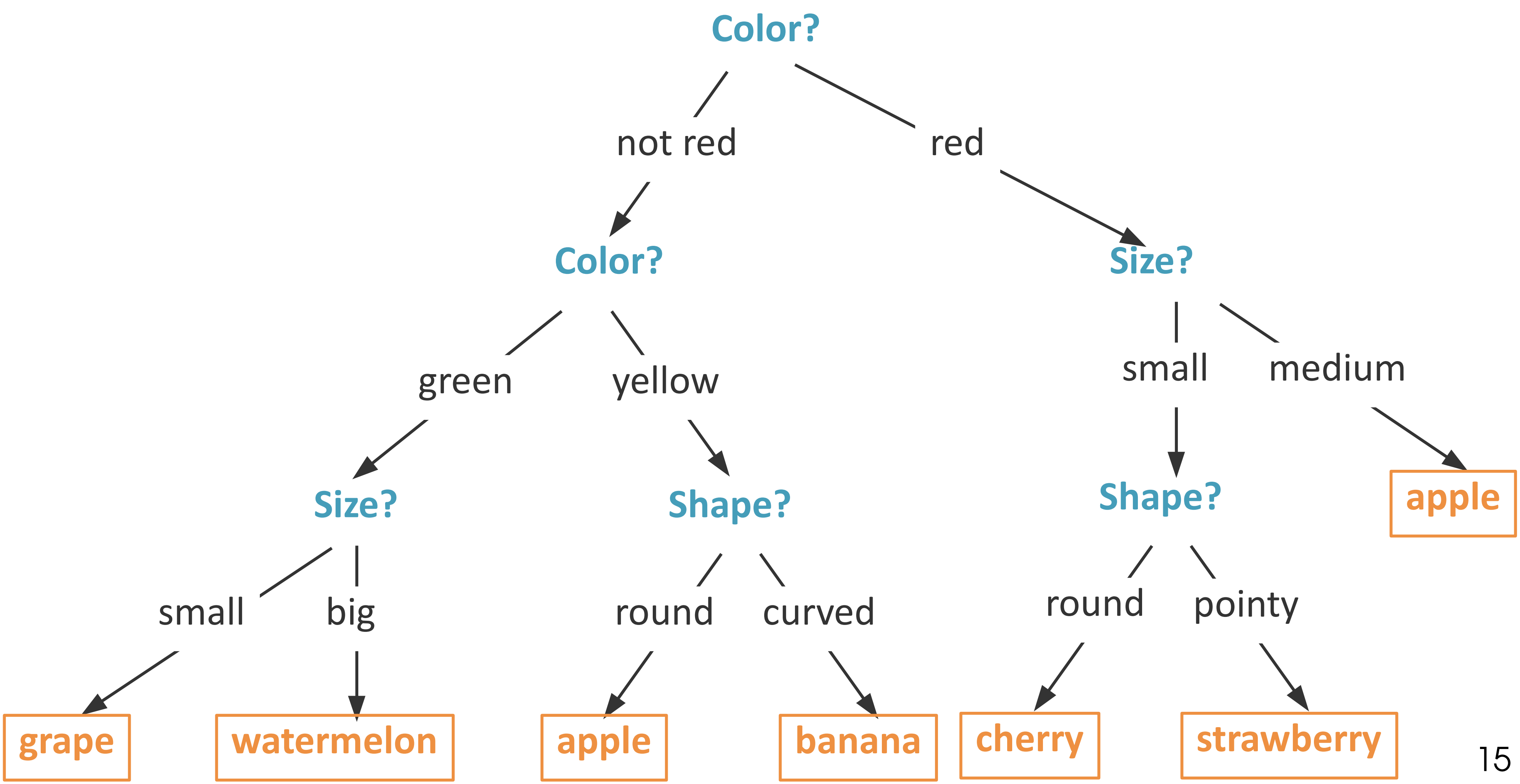
A tree with arbitrary branching factor can always be **equivalently represented** by a binary tree.

Find a binary tree that's equivalent to:



Binary & multi-value splits

A tree with arbitrary branching factor can always be **equivalently represented** by a binary tree.



How to grow a tree

How to grow a tree

- **Monothetic trees**
  - Use 1 feature per node
  - The decision boundary is

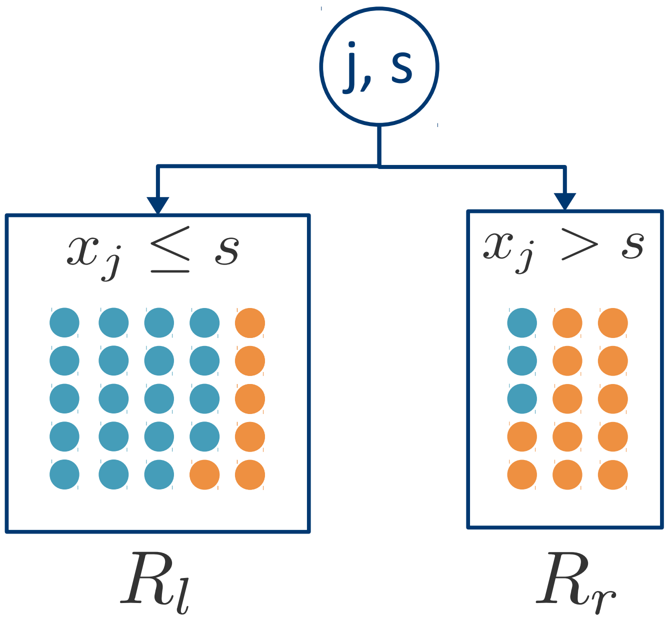
How to grow a tree

- **Monothetic trees**
  - Use 1 feature per node
  - The decision boundary is orthogonal to the axes.

How to grow a tree

- **Splitting variable** (j) and **splitting point** (s) define 2 regions:

$$R_l = \{x : x_j \leq s\} \text{ and } R_r = \{x : x_j > s\}$$



How to grow a tree

- **Splitting variable** (j) and **splitting point** (s) define 2 regions:

$$R_l = \{x : x_j \leq s\} \text{ and } R_r = \{x : x_j > s\}$$

- **Regression tree:** choose j and s to **minimize SE:**

$$\min_{j,s} \left(\sum_{i:x_i \in R_l(j,s)} (y_i - c_l)^2 + \sum_{i:x_i \in R_r(j,s)} (y_i - c_r)^2 \right)$$

How to grow a tree

- **Splitting variable** (j) and **splitting point** (s) define 2 regions:

$$R_l = \{x : x_j \leq s\} \text{ and } R_r = \{x : x_j > s\}$$

- **Classification tree:** choose j and s to **minimize impurity.**

$$\min_{j,s} \left(\frac{|R_l(j,s)|}{n} \times \text{Imp}(R_l(j,s)) + \frac{|R_r(j,s)|}{n} \times \text{Imp}(R_r(j,s)) \right)$$

- Greedy algorithm / local optimization.

How to grow a tree

- **Splitting variable** (j) and **splitting point** (s) define 2 regions:

$$R_l = \{x : x_j \leq s\} \text{ and } R_r = \{x : x_j > s\}$$

- **Classification tree:** choose j and s to **maximize the drop in impurity.**
- **Impurity:**
  - Classification error
  - Entropy
  - Gini impurity.

Impurity: Classification error

- Minimum probability that a training point will be misclassified at node (s,j)

$$\mathrm{Imp}(R_m) = 1 - \max_k \hat{p}_{mk}$$

proportion of training instances from class k in Rm

- If all examples from one class belong to Rm, then Imp(Rm) =

Impurity: Classification error

- Minimum probability that a training point will be misclassified at node (s,j)

proportion of training instances from class k in Rm

$$\mathrm{Imp}(R_m) = 1 - \max_k \hat{p}_{mk}$$

- If all examples from one class belong to Rm, then $\boxed{\text{Imp(Rm) = 0}}$

Impurity: Classification error

- Minimum probability that a training point will be misclassified at node (s,j)

proportion of training instances from class k in Rm

$$\mathrm{Imp}(R_m) = 1 - \max_k \hat{p}_{mk}$$

- If all examples from one class belong to Rm, then $\boxed{\text{Imp(Rm) = 0}}$
- If we have 2 balanced classes, and instances are randomly split at (s, j), then Imp(Rm) = ?

Impurity: Classification error

- Minimum probability that a training point will be misclassified at node (s,j)

$$\mathrm{Imp}(R_m) = 1 - \max_k \hat{p}_{mk}$$

proportion of training instances from class k in Rm

- If all examples from one class belong to Rm, then $\boxed{\text{Imp(Rm) = 0}}$
- If we have 2 balanced classes, and instances are randomly split at (s, j), then $\boxed{\text{Imp(Rm) = 0.5}}$

Impurity: Entropy

- Information theory: Shannon's entropy

proportion of training instances from class k in Rm

$$\mathrm{Imp}(R_m) = -\sum_k \hat{p}_{mk} \log_2 \hat{p}_{mk}$$

- If all examples from one class belong to Rm, then Imp(Rm) = ?

Impurity: Entropy

- Information theory: Shannon's entropy

proportion of training instances from class k in Rm

$$\mathrm{Imp}(R_m) = -\sum_k \hat{p}_{mk} \log_2 \hat{p}_{mk}$$

- If all examples from one class belong to Rm, then Imp(Rm) = 0

Impurity: Entropy

- Information theory: Shannon's entropy

proportion of training instances from class k in Rm

$$\mathrm{Imp}(R_m) = -\sum_k \hat{p}_{mk} \log_2 \hat{p}_{mk}$$

- If all examples from one class belong to Rm, then Imp(Rm) = 0
- If we have 2 balanced classes, and instances are randomly split at (s, j), then Imp(Rm) = ?

Impurity: Entropy

- Information theory: Shannon's entropy

proportion of training instances from class k in Rm

$$\mathrm{Imp}(R_m) = -\sum_k \hat{p}_{mk} \log_2 \hat{p}_{mk}$$

- If all examples from one class belong to Rm, then $\boxed{\text{Imp(Rm) = 0}}$
- If we have 2 balanced classes, and instances are randomly split at (s, j), then $\boxed{\text{Imp(Rm) = 1}}$

Gini impurity

$$\mathrm{Imp}(R_m) = \sum_{k=1}^{K} \hat{p}_{mk}(1 - \hat{p}_{mk})$$

proportion of training instances from class k in Rm

- If all examples from one class belong to Rm, then Imp(Rm) = ?

Gini impurity

$$\mathrm{Imp}(R_m) = \sum_{k=1}^{K} \hat{p}_{mk}(1 - \hat{p}_{mk})$$

proportion of training instances from class k in Rm

- If all examples from one class belong to Rm, then $\boxed{\text{Imp(Rm) = 0}}$

Gini impurity

$$\text{Imp}(R_m) = \sum_{k=1}^{K} \hat{p}_{mk}(1 - \hat{p}_{mk})$$

proportion of training instances from class k in Rm

- If all examples from one class belong to Rm, then $\boxed{\text{Imp(Rm) = 0}}$
- If we have 2 balanced classes, and instances are randomly split at (s, j), then Imp(Rm) = ?

Gini impurity

$$\mathrm{Imp}(R_m) = \sum_{k=1}^{K} \hat{p}_{mk}(1 - \hat{p}_{mk})$$

proportion of training instances from class k in Rm

- If all examples from one class belong to Rm, then $\boxed{\text{Imp(Rm) = 0}}$
- If we have 2 balanced classes, and instances are randomly split at (s, j), then $\boxed{\text{Imp(Rm) = 0.5}}$

Gini impurity

$$\text{Imp}(R_m) = \sum_{k=1}^{K} \hat{p}_{mk}(1 - \hat{p}_{mk})$$

$$GI(j,s) = \frac{|R_l(j,s)|}{n} \times \text{Imp}(R_l(j,s)) + \frac{|R_r(j,s)|}{n} \times \text{Imp}(R_r(j,s))$$

- If the **split respects the overall distribution:** $\hat{p}_{mk} = \dfrac{n_k}{n} \quad \forall k$
- All regions are identically distributed and have Gini impurity:

$$\text{Imp}(R_m) = \sum_{k=1}^{K} \frac{n_k}{n}\left(1 - \frac{n_k}{n}\right) = 1 - \sum_{k=1}^{K} \frac{n_k^2}{n^2}$$

- If the dataset is **balanced** $n_k = \dfrac{n}{K} \quad \forall k$

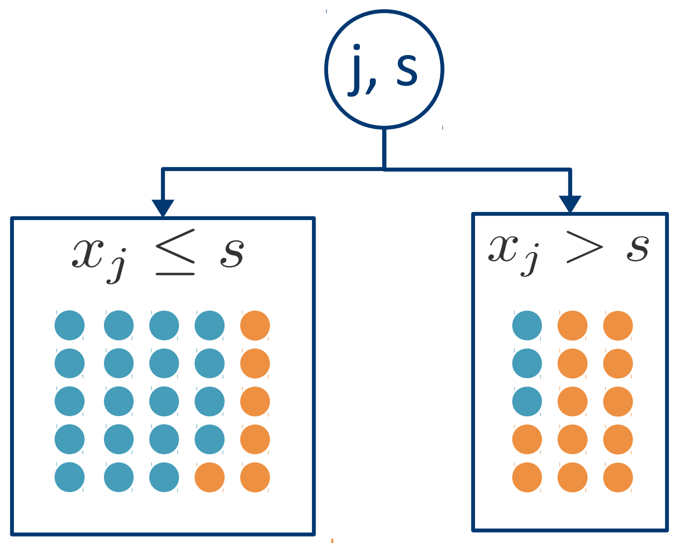
  then

$$\text{Imp}(R_m) = 1 - \frac{1}{K}$$

Gini impurity (K=2)

$$\mathrm{Imp}(R_m) = \sum_{k=1}^{K} \hat{p}_{mk}(1 - \hat{p}_{mk})$$

$$GI(j,s) = \frac{|R_l(j,s)|}{n} \times \mathrm{Imp}(R_l(j,s)) + \frac{|R_r(j,s)|}{n} \times \mathrm{Imp}(R_r(j,s))$$



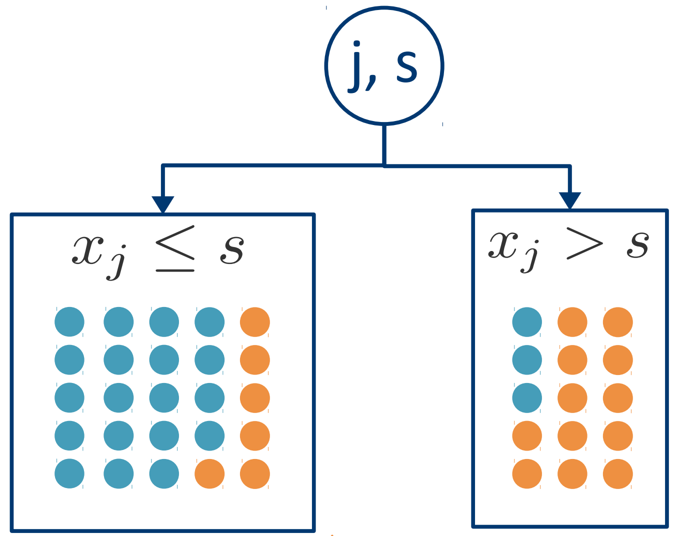
$n_l = n_l^- + n_l^+$ $\qquad$ $n_r = n_r^- + n_r^+$

- Rewrite GI(j, s) using $n_l, n_l^+, n_l^-, n_r, n_r^+, n_r^-$

Gini impurity (K=2)

$$\mathrm{Imp}(R_m) = \sum_{k=1}^{K} \hat{p}_{mk}(1-\hat{p}_{mk})$$

$$GI(j,s) = \frac{|R_l(j,s)|}{n} \times \mathrm{Imp}(R_l(j,s)) + \frac{|R_r(j,s)|}{n} \times \mathrm{Imp}(R_r(j,s))$$



$$n_l + n_r = n$$

$$n_l^- + n_r^- = n^-$$

$$n_l^+ + n_r^+ = n^+$$

$$n_l = n_l^- + n_l^+ \qquad n_r = n_r^- + n_r^+$$

$$GI(j,s) = \frac{n_l}{n}\left(\frac{n_l^-}{n_l}\left(1-\frac{n_l^-}{n_l}\right) + \frac{n_l^+}{n_l}\left(1-\frac{n_l^+}{n_l}\right)\right) + \frac{n_r}{n}\left(\frac{n_r^-}{n_r}\left(1-\frac{n_r^-}{n_r}\right) + \frac{n_r^+}{n_r}\left(1-\frac{n_r^+}{n_r}\right)\right)$$

$$\boxed{GI(j,s) = \frac{2}{n}\left(\frac{n_l^- n_l^+}{n_l} + \frac{n_r^- n_r^+}{n_r}\right)}$$

When to stop growing a tree

When to stop growing a tree

- **Large tree** might overfit
- **Small tree** might underfit
- Strategy:
  - grow the tree until a **minimum node size** (# training points in the region) is reached;
  - prune the tree: **cost-complexity pruning**.

When to stop growing a tree

– prune the tree: **cost-complexity pruning**.

number of regions in T

$$C_\alpha(T) = \sum_{m=1}^{|T|} N_m Q_m(T) + \alpha|T|$$

pruned tree

number of training instances in Rm

Error on Rm

α: trade-off between model complexity and goodness of fit.

Advantages & drawbacks of trees

- :-) Trees are **easy to explain.**
- :-) Trees seem to **mirror human-decision making.**
- :-) Trees can be **displayed graphically** and **easily interpreted.**
- :-) Trees can easily handle **quantitative variables.**
- :-) Trees naturally handle **multi-class problems.**

- :-(Trees generally do not have very good **predictive accuracy.**

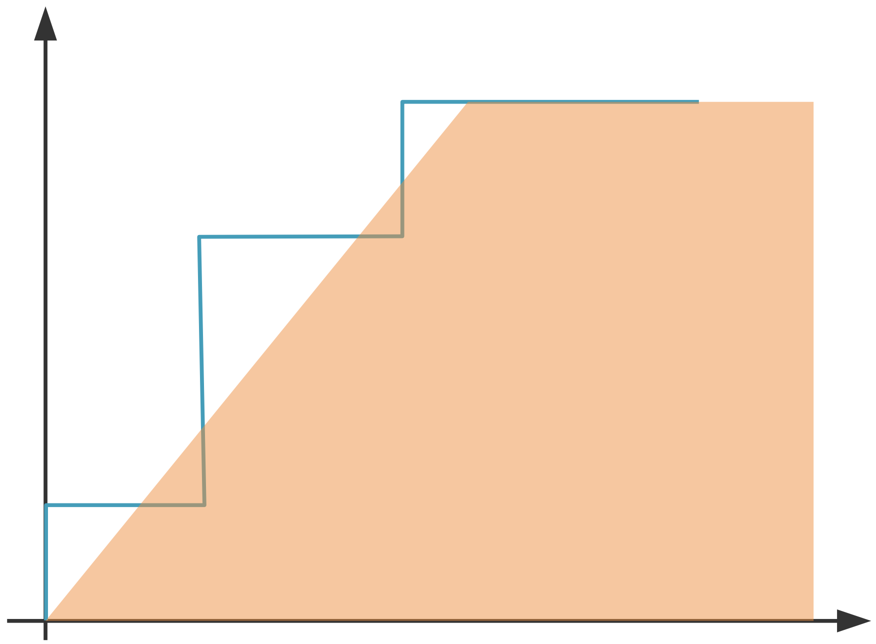
Forests

Building forests

- Idea: **Aggregating many weak learners can substantially increase their performance.**
- **Ensemble learning**

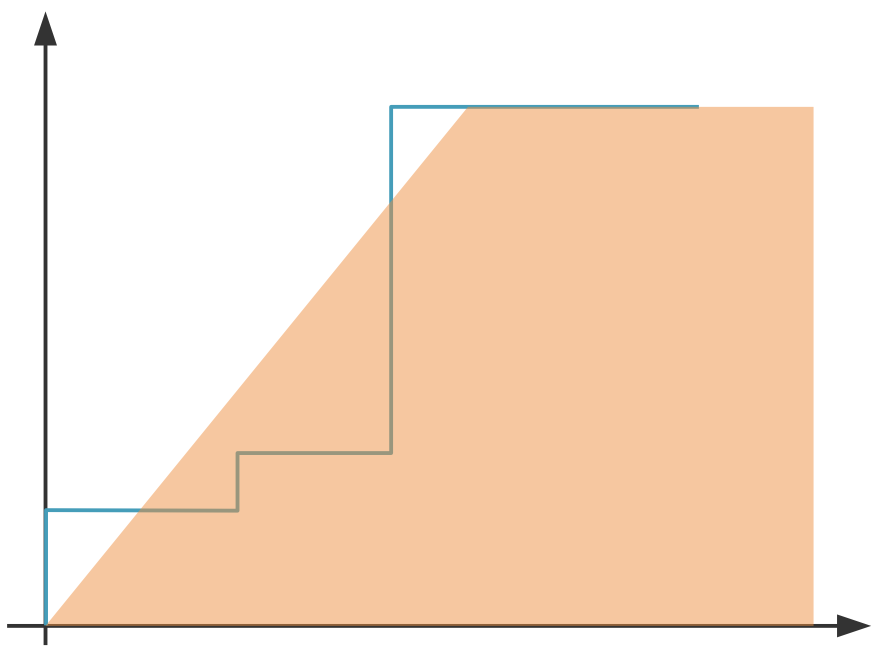
Ensemble learning

- **Wisdom of crowds:** Average out the uncorrelated errors of individual classifiers.
- **Example:** Learn a diagonal separation from “staircase” decision boundaries.



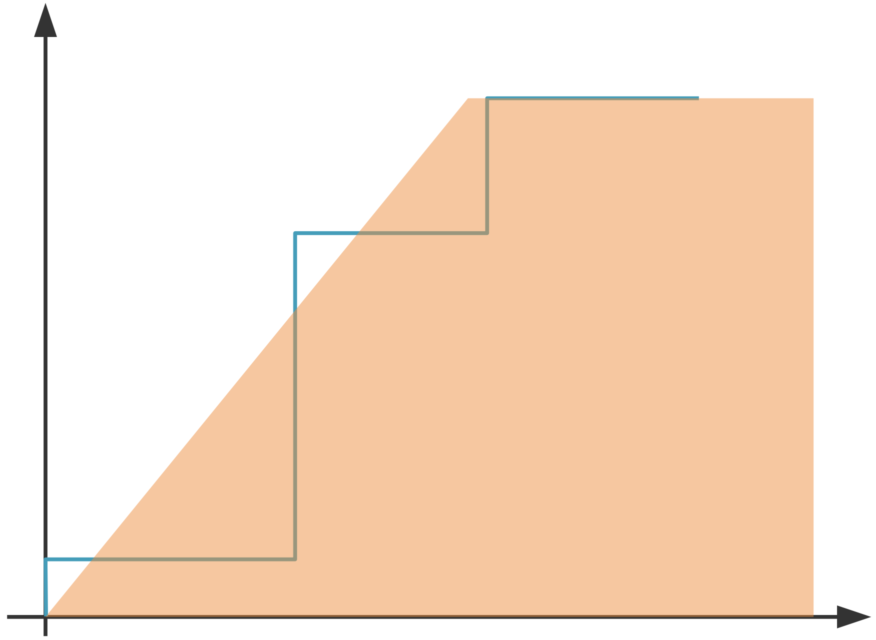
Ensemble learning

- **Wisdom of crowds:** Average out the uncorrelated errors of individual classifiers.
- **Example:** Learn a diagonal separation from "staircase" decision boundaries.



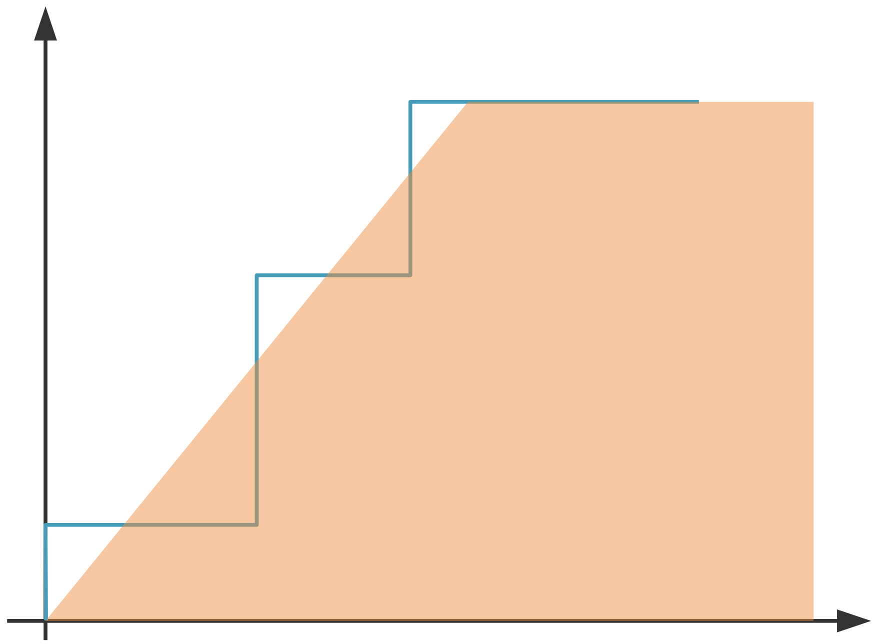
Ensemble learning

- **Wisdom of crowds:** Average out the uncorrelated errors of individual classifiers.
- **Example:** Learn a diagonal separation from "staircase" decision boundaries.



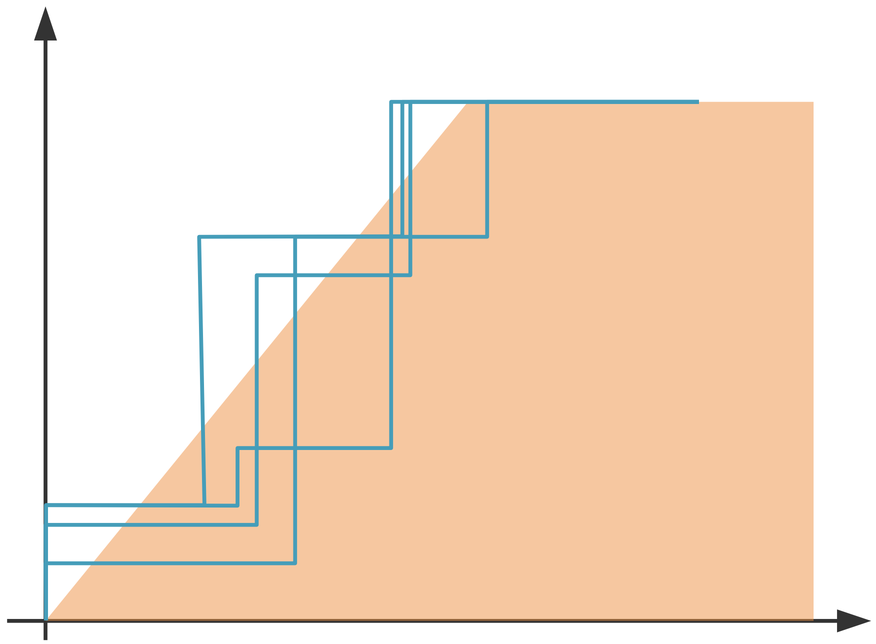
Ensemble learning

- **Wisdom of crowds:** Average out the uncorrelated errors of individual classifiers.
- **Example:** Learn a diagonal separation from "staircase" decision boundaries.



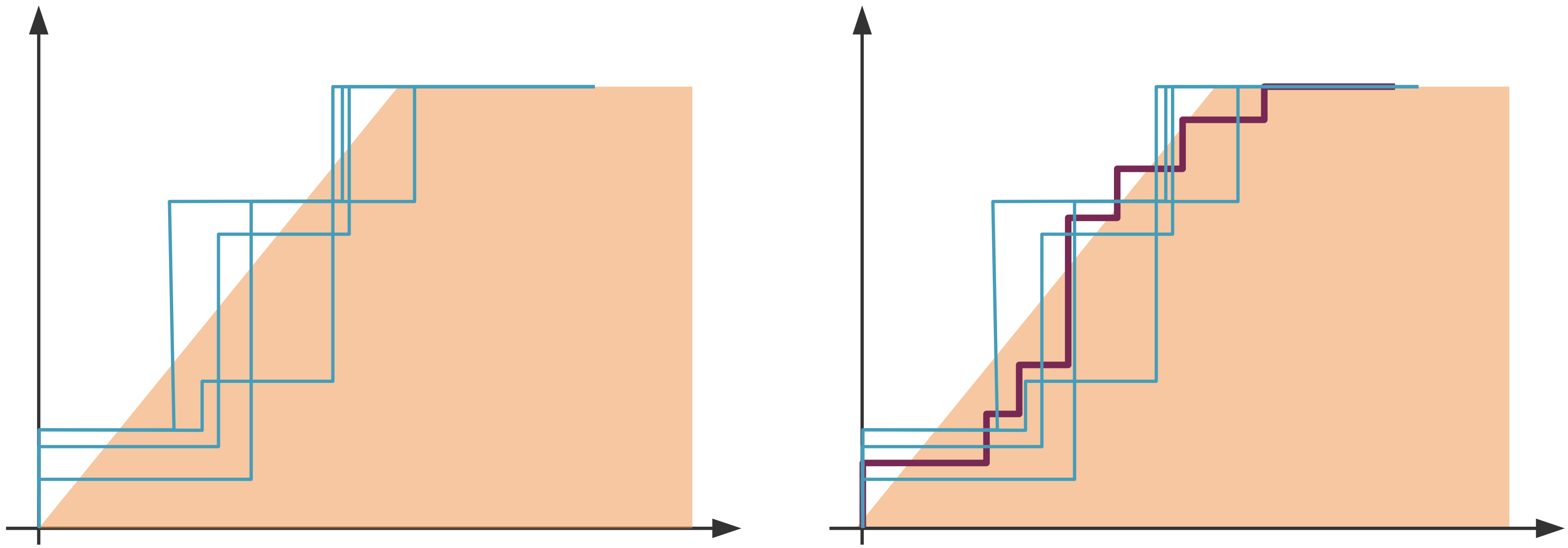
Ensemble learning

- **Wisdom of crowds:** Average out the uncorrelated errors of individual classifiers.
- **Example:** Learn a diagonal separation from “staircase” decision boundaries.



Ensemble learning

- **Wisdom of crowds:** Average out the uncorrelated errors of individual classifiers.
- **Example:** Learn a diagonal separation from "staircase" decision boundaries.



Building ensembles

- **Subsample** the training data
  - **Bagging** [Breiman 1996]: bootstrap resampling
  - **Boosting** [Schapire 1990]: resample based on performance
- Use different **features**
  - Multiple input representations
  - Feature selection (Chap 11)
- Use different **parameters** of the learning algorithm.

Combining learners

- **Non-trainable combination:**
  - **Voting** (classification)
  - **Averaging** (regression)
- **Trainable combination:**
  - **Weighted averaging:** based on performance on a validation set.
  - **Meta-learner:** the outputs of the individual learners are features for another learning algorithm.

Bagging trees

- **Bagging:**
  - Take repeated samples from the training data (bootstrap)
  - Build one predictor from each of these samples
  - **Final prediction:** average (regression) or majority vote (classification)

Random forests

[Breiman 2001]

- Similar to bagging trees
- One trick to **decorrelate** the trees:

  Before splitting, first **randomly sample q** (out of p) **variables** among which the one over which to split must be chosen.

  Typically $q = \sqrt{p}$.
- Very good predictive power in practice!

$y \in \{-1, +1\}$

AdaBoost

[Schapire & Freund 1997]

- weak learners = **stumps** (only one split)
- Give more weight to the more difficult samples
- At iteration m:
  - learn f_m from data weighted by $\{w_i^{m-1}\}_{i=1,\dots n}$

$$\boxed{\epsilon_m = \sum_{i=1}^{n} w_i^m \delta_{f_m(\boldsymbol{x}^i) \neq y^i}}$$

weighted error

$$\alpha_m = \frac{1}{2} \log \frac{1-\epsilon_m}{\epsilon_m}$$

  - update weights: $w_i^m = \frac{1}{Z_m} w_i^{m-1} \boxed{\exp\left(-\alpha_m y^i f_m(\boldsymbol{x}^i)\right)}$

exponential loss

such that the weights sum to 1

- **Final decision function:**

$$f : \boldsymbol{x} \mapsto \sum_{m=1}^{M} \alpha_m f_m(\boldsymbol{x})$$

$y \in \{-1, +1\}$

Gradient Boosting [Friedman 2001]

- At iteration m:
  - learn f_m that minimizes a loss function for predictor
$$F_m = \sum_{l=1}^{m} \alpha_l f_l = F_{m-1} + \alpha_m f_m$$
    by gradient descent
  - **Exponential loss:** equivalent to **AdaBoost**
$$\mathcal{L}(y, f(\boldsymbol{x})) = \exp(-y f(\boldsymbol{x}))$$
  - Other possible losses:
    - **cross-entropy:**

$y \in \{0, 1\}$

$$\mathcal{L}(y, f(\boldsymbol{x})) = \underbrace{\boxed{\log(1 + \exp^{-y f(\boldsymbol{x})})}}_{\text{= logistic loss}} = \underbrace{\boxed{-\sum_{k=0}^{K-1} y \log(f_k(\boldsymbol{x}))}}_{\text{multiclass}}$$

= binomial divergence loss

    - **least-squares:**
$$\mathcal{L}(y, f(\boldsymbol{x})) = \frac{1}{2}(y - f(\boldsymbol{x}))^2$$

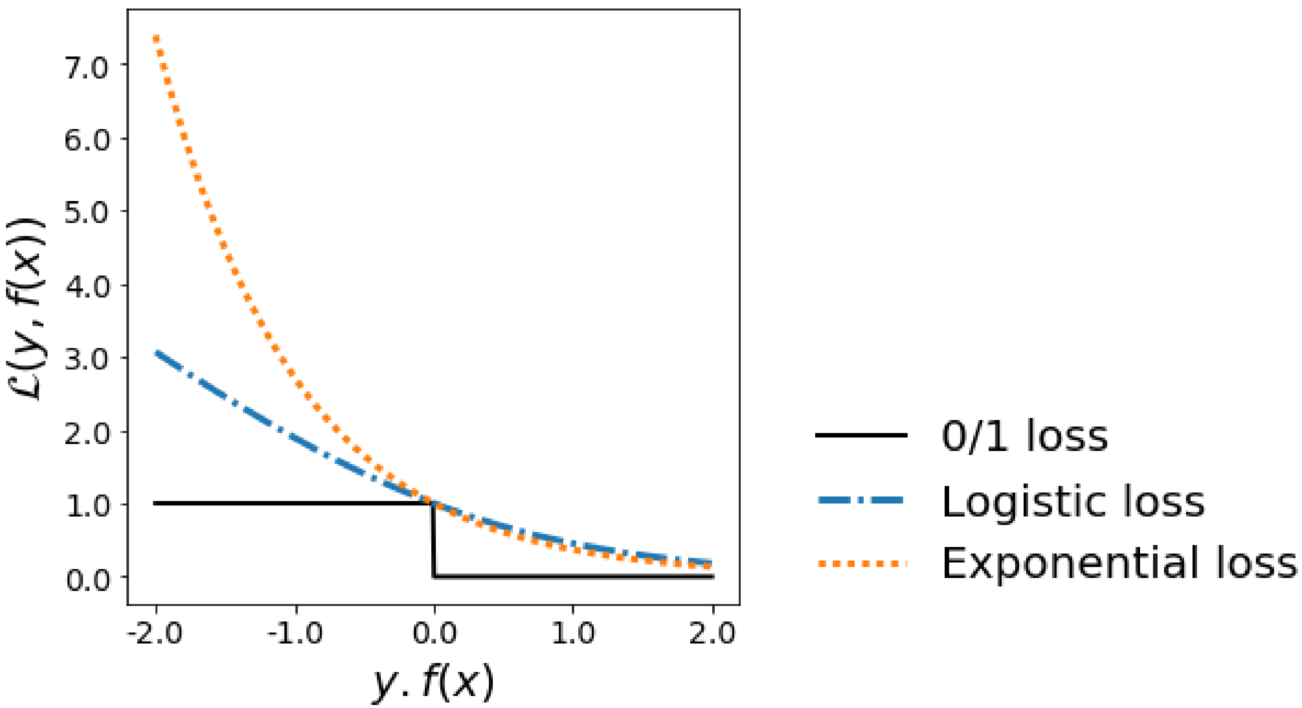
$y \in \{-1, +1\}$

Gradient Boosting [Friedman 2001]

- At iteration m:
  - learn f_m that minimizes a loss function for predictor

$$F_m = \sum_{l=1}^{m} \alpha_l f_l = F_{m-1} + \alpha_m f_m$$

by gradient descent



Summary

- Decision trees are **easy to interpret.**
- Decision trees elegantly deal with
  - **Quantitative variables**
  - **Multiple classes**
  - **Multimodal distributions.**
- Decision trees have **limited predictive power,** but this can be addressed thanks to **ensemble methods**
  - **Boosting**
  - **Bagging**
  - **Random forests.**

References

- *A Course in Machine Learning.* `http://ciml.info/dl/v0_99/ciml-v0_99-all.pdf`
  - **Decision trees:** Chap 1.3
  - **Boosting :** Chap 13.2
  - **Random forests:** Chap 13.3
- *The Elements of Statistical Learning.* `http://web.stanford.edu/~hastie/ElemStatLearn/`
  - **Decision trees:** Chap 9.2
  - **Boosting:** Chap 10.1 – 10.10
  - **Random forests:** Chap 15.1 – 15.2
- **A complete tutorial on tree-based modeling**

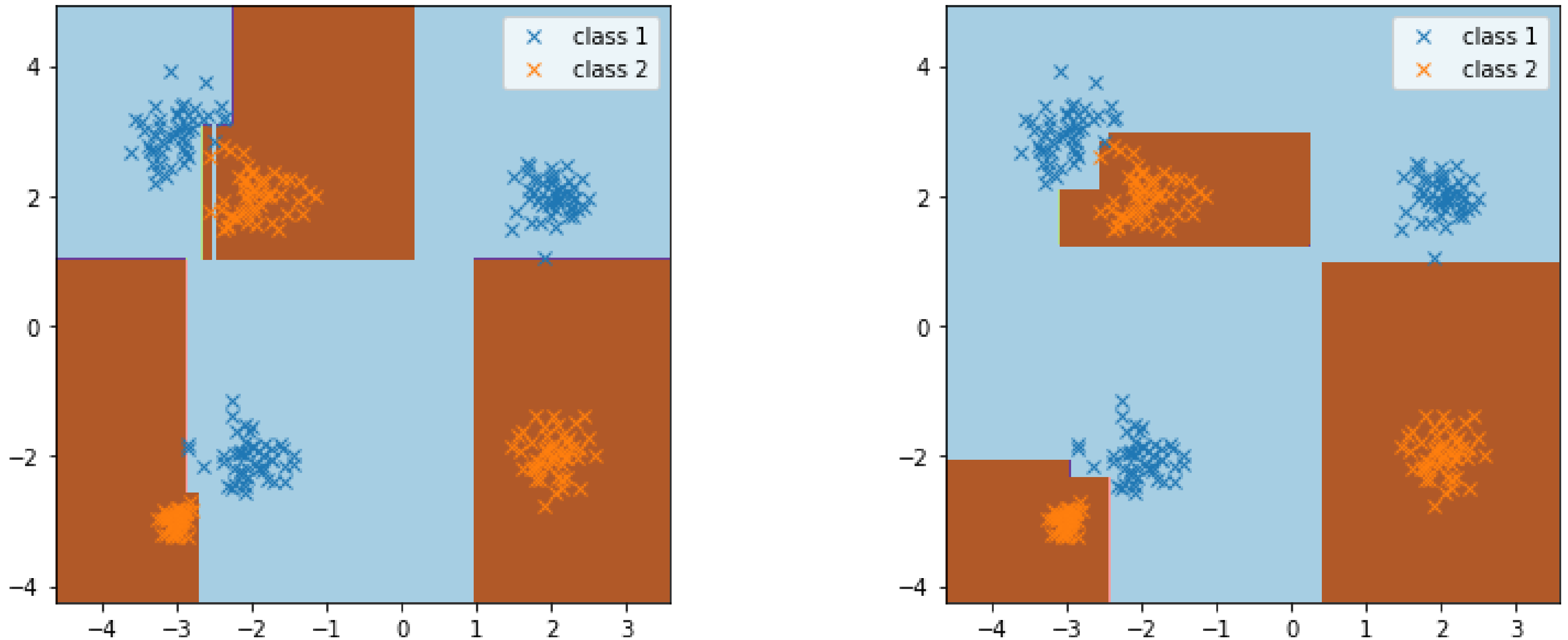
  `https://www.analyticsvidhya.com/blog/2016/04/complete-tutorial-tree-based-modeling-scratch-in-python/#ten`

Lab 7

```python
# Training data
X_demo = np.concatenate((X1, X2), axis=0)
y_demo = np.concatenate((np.zeros(X1.shape[0]), np.ones(X2.shape[0])))

# Train a DecisionTreeClassifier on the training data
clf = tree.DecisionTreeClassifier(splitter='random').fit(X_demo, y_demo)
```

splitter : string, optional (default="best")

> The strategy used to choose the split at each node. Supported strategies are "best" to choose the best split and "random" to choose the best random split.



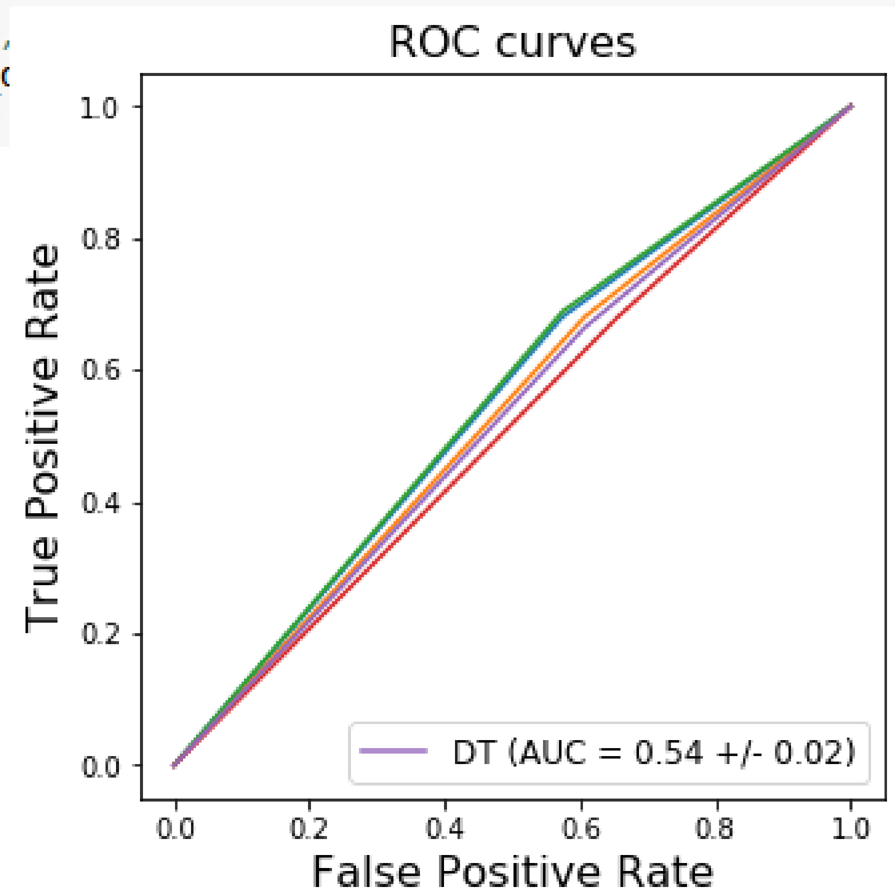
```python
ypred_dt = [] # will hold the 5 arrays of predictions (1 per tree)
for tree_index in range(5):
    # Initialize a DecisionTreeClassifier
    clf = tree.DecisionTreeClassifier()

    # Cross-validate this DecisionTreeClassifier on the toy data
    pred_proba = cross_validate_clf(X, y, clf, folds)

    # Append the prediction to ypred_dt
    ypred_dt.append(pred_proba)

    # Print the accuracy of DecisionTreeClassifier
    print("%.3f" % metrics.accuracy_score(y, np.where(pred_proba > 0.5, 1, 0)))
```

The features are always randomly permuted at each split. Therefore, the best found split may vary, even with the same training data and `max_features=n_features`, if the improvement of the criterion is identical for several splits enumerated during the search of the best split. To obtain a deterministic behaviour during fitting, `random_state` has to be fixed.

Decision trees

```
for tree_index in range(5):
    # Compute the ROC curve of the current tree
    fpr_dt_tmp, tpr_dt_tmp, thresholds =  metrics.roc_curve(y, ypred_dt[tree_index], pos_label=1)
    # Compute the area under the ROC curve of the current tree
    auc_dt_tmp = metrics.auc(fpr_dt_tmp, tpr_dt_tmp)
    fpr_dt.append(fpr_dt_tmp)
    tpr_dt.append(tpr_dt_tmp)
    auc_dt.append(auc_dt_tmp)

# Plot the first 4 ROC curves
for tree_index in range(4):
    plt.plot(fpr_dt[tree_index], tpr_dt[tree_index], '-')

# Plot the last ROC curve,
plt.plot(fpr_dt[-1], tpr_c
         label='DT (AUC =                                                    uc_dt)))
```

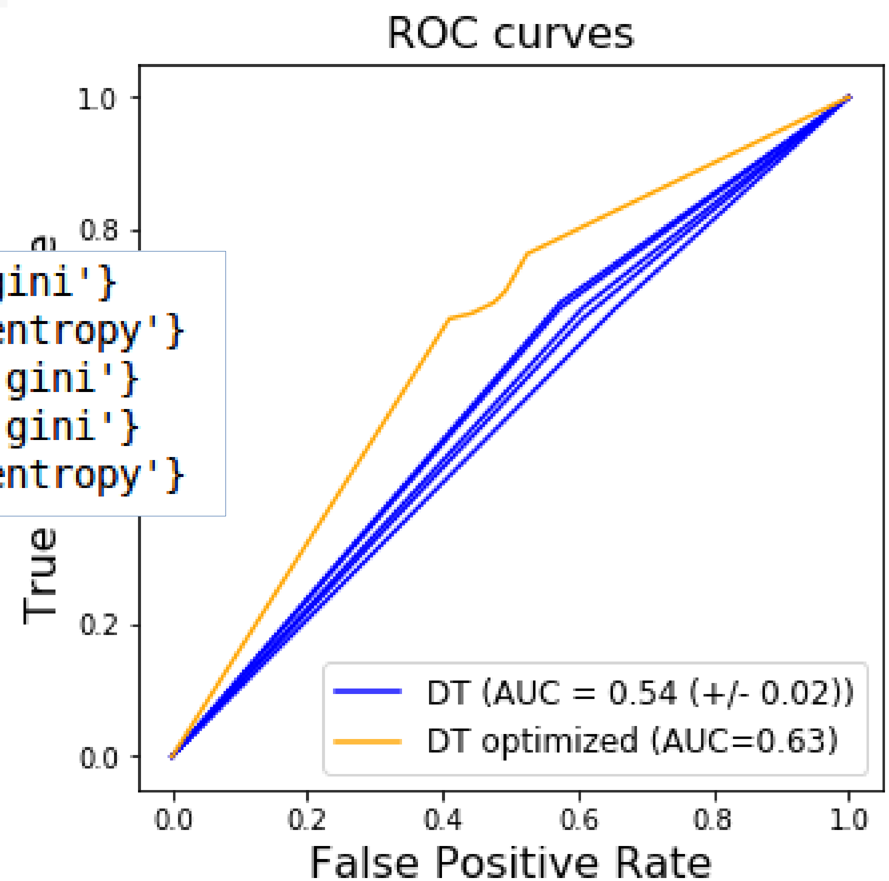
Decision trees perform poorly on this data



Optimized decision tree

```python
# Define the grid of parameters to test
param_grid = {'criterion': ['gini', 'entropy'], 'min_samples_split': [2, 4, 16, 256]}

# Initialize a GridSearchCV object that will be used to cross-validate
# a DecisionTreeClassifier with these parameters.
# What scoring function do you want to use?
clf = model_selection.GridSearchCV(tree.DecisionTreeClassifier(), param_grid,
                                   scoring='roc_auc')

# Cross-validate the GridSearchCV object
ypred_dt_opt = cross_validate_clf_optimize(X, y, clf, folds)
```

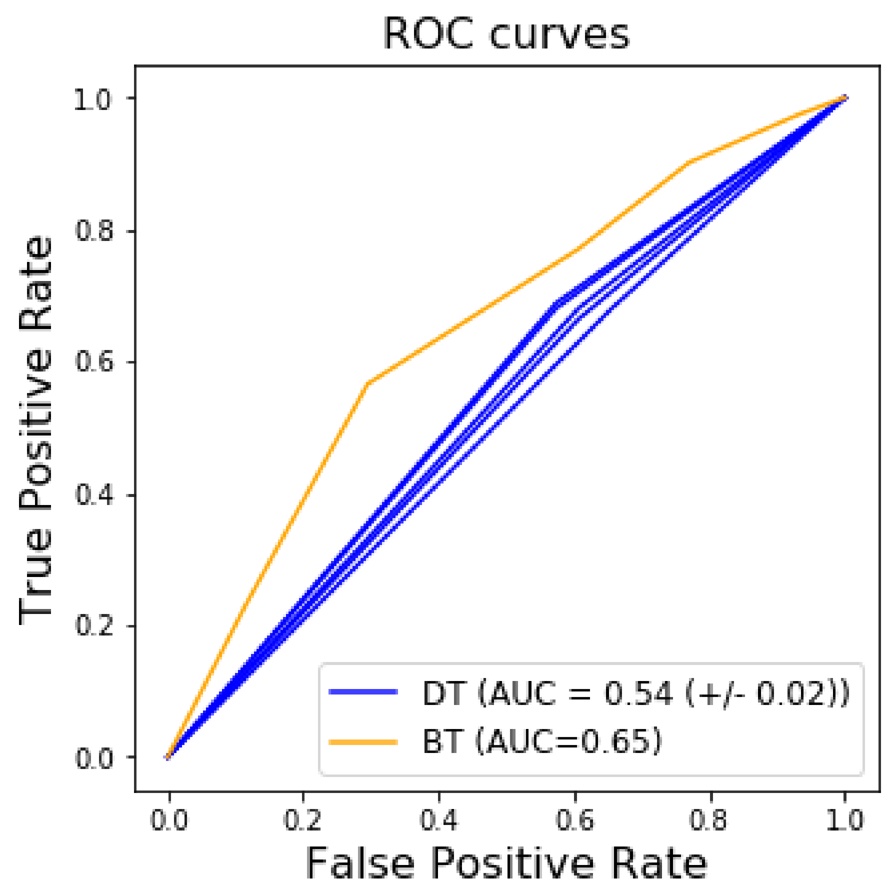
```
{'min_samples_split': 2, 'criterion': 'gini'}
{'min_samples_split': 2, 'criterion': 'entropy'}
{'min_samples_split': 16, 'criterion': 'gini'}
{'min_samples_split': 16, 'criterion': 'gini'}
{'min_samples_split': 2, 'criterion': 'entropy'}
```



No clear optimal parameters, and the performance is still not very good.

Bagging trees

```python
from sklearn import ensemble

# Initialize a bag of trees
clf = ensemble.BaggingClassifier(n_estimators=5, max_features=X.shape[1])

# Cross-validate the bagging trees on the tumor data
ypred_bt = cross_validate_clf(X, y, clf, folds)
```

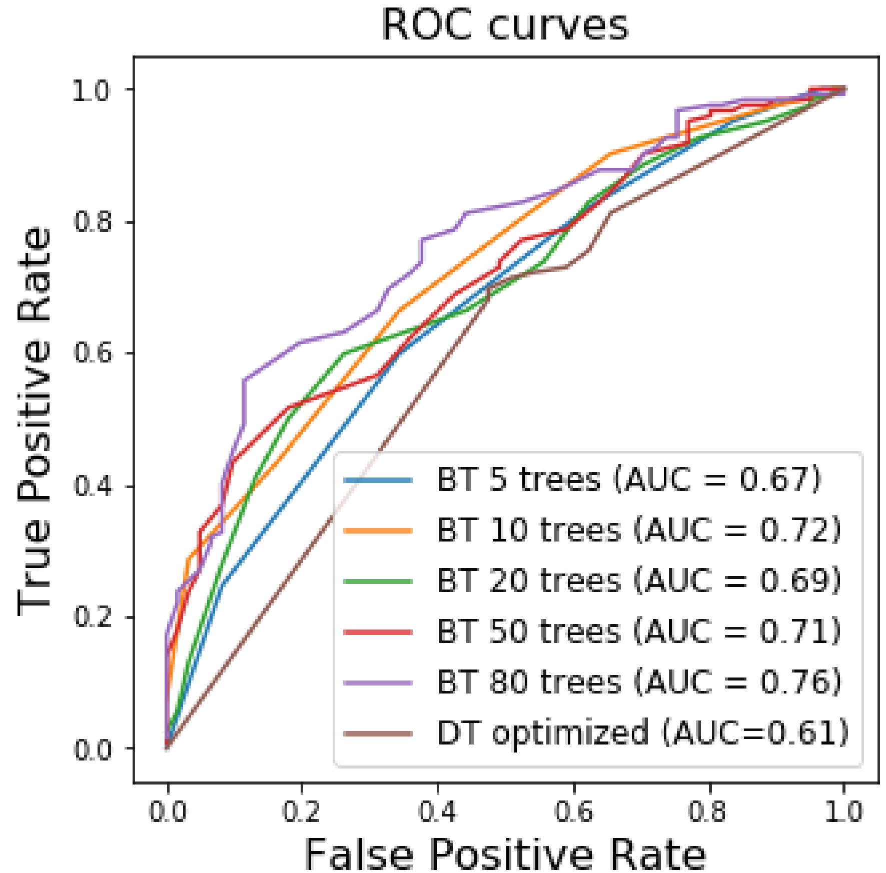


Combining 5 trees gives a much better performance than any of the 5 trees!

Bagging trees

```
# Number of trees to use
list_n_trees = [5, 10, 20, 50, 80]

# Start a ROC curve plot
fig = plt.figure(figsize=(5, 5))

for idx, n_trees in enumerate(list_n_trees):
    # Initialize a bag of trees with n_trees trees
    clf = ensemble.BaggingClassifier(n_estimators=n_trees)

    # Cross-validate the bagging trees on the tumor data
    ypred_bt_tmp = cross_validate_clf(X, y, clf, folds)
```

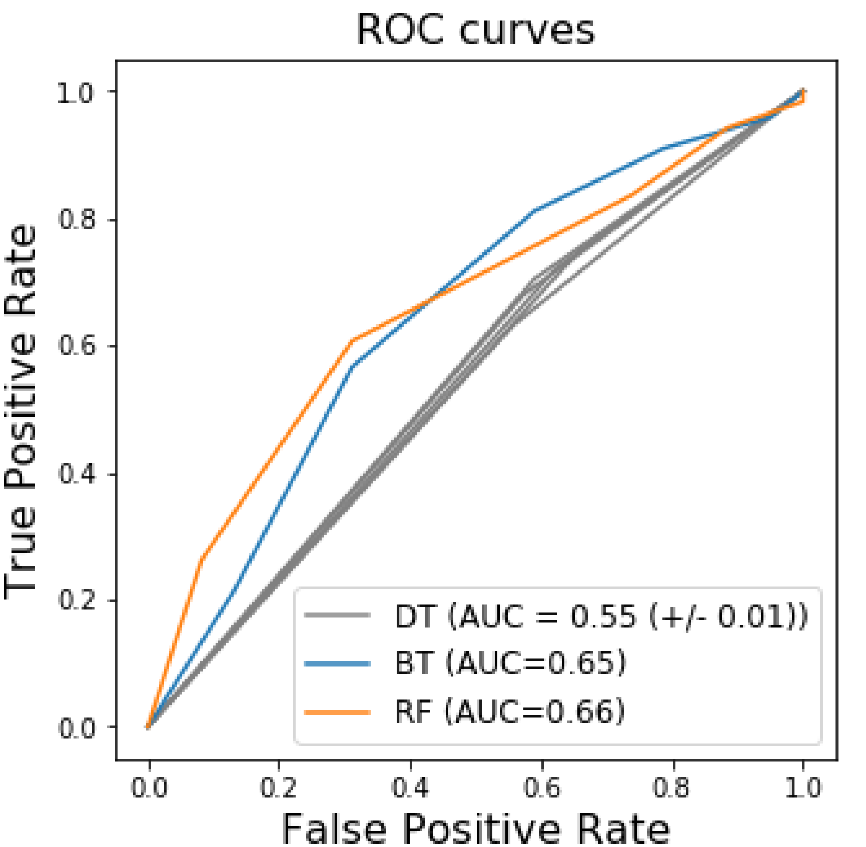
Increasing the number of trees increases the performance, but quickly does not make much of a difference.



Random forest

```
# Initialize a random forest with 5 trees
clf = ensemble.RandomForestClassifier(n_estimators=5)

# Cross-validate the random forest on the tumor data
ypred_rf = cross_validate_clf(X, y, clf, folds)

# Compute the ROC curve of the random forest
fpr_rf, tpr_rf, thresholds = metrics.roc_curve(y, ypred_rf, pos_label=1)
auc_rf = metrics.auc(fpr_rf, tpr_rf)

# Plot the ROC curve of the 5 decision trees from earlier
fig = plt.figure(figsize=(5, 5))

for tree_index in range(4):
    plt.plot(fpr_dt[tree_index], tpr_dt[tree_index], '-', color='grey')
plt.plot(fpr_dt[-1], tpr_dt[-1], '-', color='grey',
         label='DT (AUC = %0.2f (+/- %0.2f))' % (np.mean(auc_dt), np.std(auc dt)))

# Plot the ROC curve of the bagging trees (5 trees)
plt.plot(fpr_bt, tpr_bt, label='BT (AUC=%0.2f)' % auc_bt)

# Plot the ROC curve of the random forest (5 trees)
plt.plot(fpr_rf, tpr_rf, label='RF (AUC=%0.2f)' % auc_rf)
```

With few trees, we observe a similar behavior for the random forest as for the bagging trees.

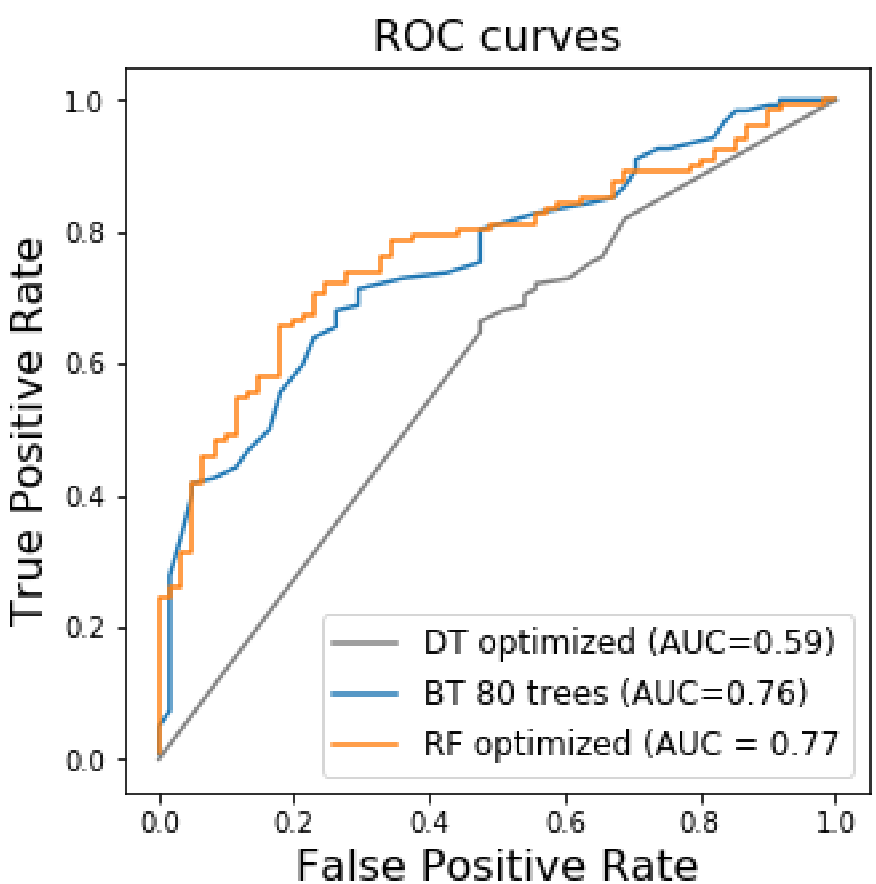


Optimized random forest

```
# Define the grid of parameters to test
param_grid = {'criterion': ['gini', 'entropy'],
              'max_features': ['auto', 'log2'],
              'min_samples_leaf': [1, 2, 5],
               'n_estimators': [5, 10, 20, 50, 80]}

# Initialize a GridSearchCV object that will be used to cross-validate
# a random forest with these parameters.
# What scoring function do you want to use?
clf = model_selection.GridSearchCV(ensemble.RandomForestClassifier(), param_grid,
                                   scoring='roc_auc')

# Cross-validate the GridSearchCV object
ypred_rf_opt = cross_validate_clf_optimize(X, y, clf, folds)
```

On this data set, the random forest and the bagging trees perform very similarly.

If you have time, try increasing the number of trees to several hundreds.



Comparison with linear models

On this data set, the random forest slightly outperforms the l1-regularized logistic regression.

On data sets with fewer features and more samples, random forests can be much more powerful than linear methods.