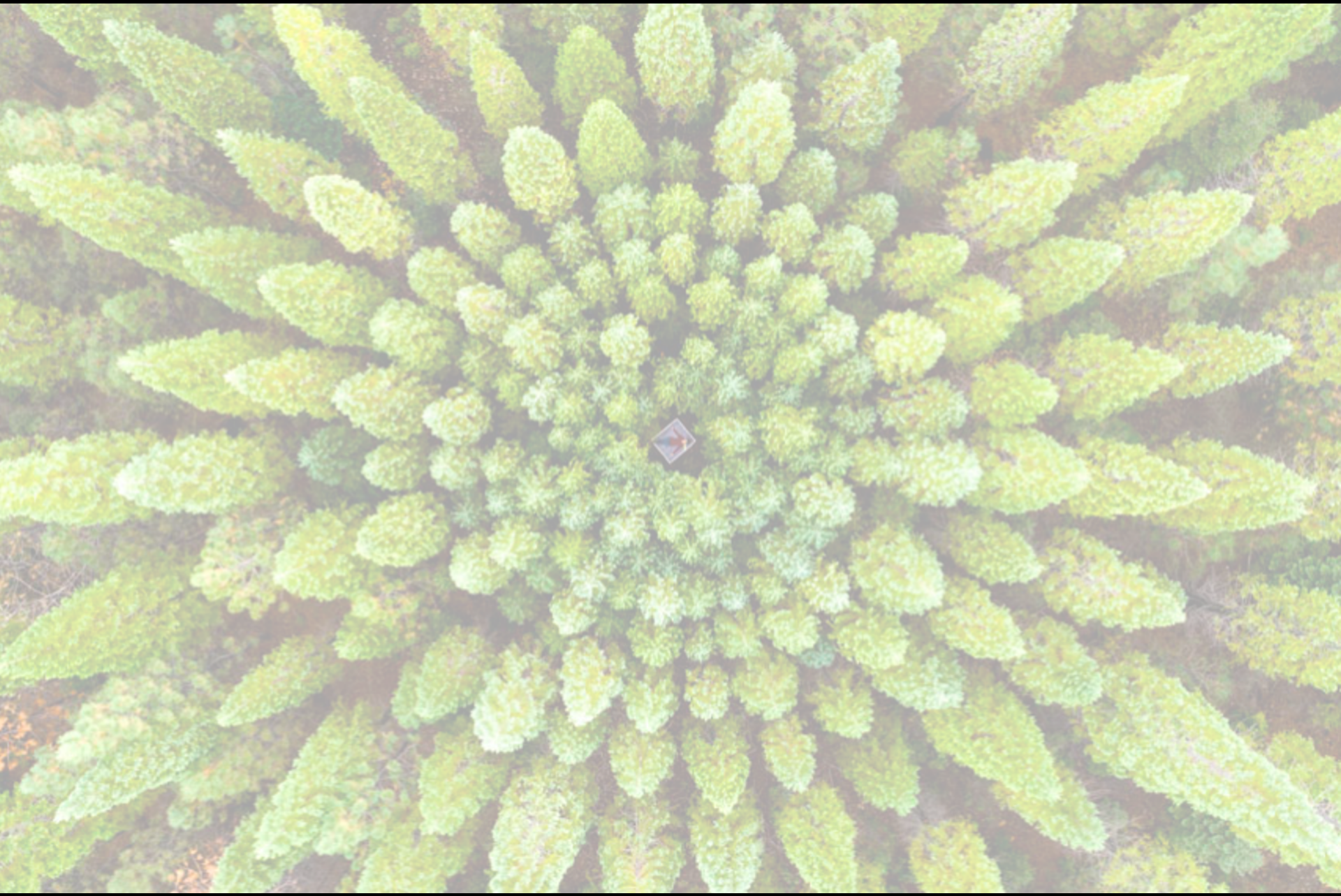
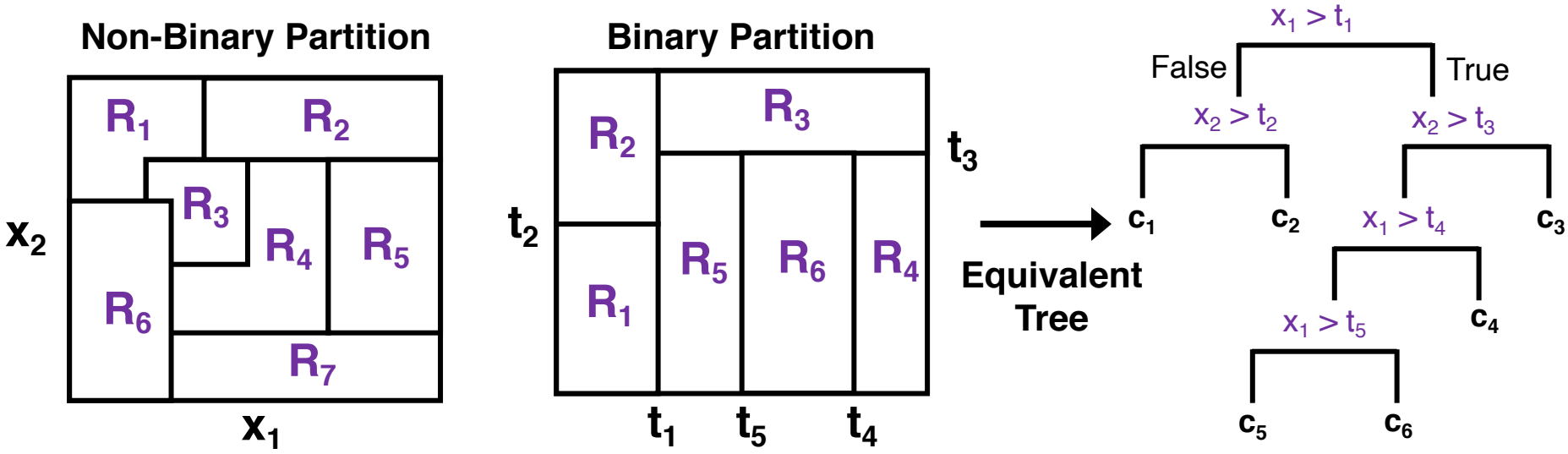
Lecture 15: Decision Trees and Random Forests



Decision Tress - Introduction

• We'll consider a variety known as classification and regression trees (**CART**) originally developed by Brieman (1984), or more commonly just "decision Trees".

• Consider data consisting of a response, **y**, and inputs x_1 and x_2. The idea is to split up the data into regions based on x_1 and x_2. In each region $x \in R_i$ we can model **y** as a constant or category, c_i, depending on the type of **y**.

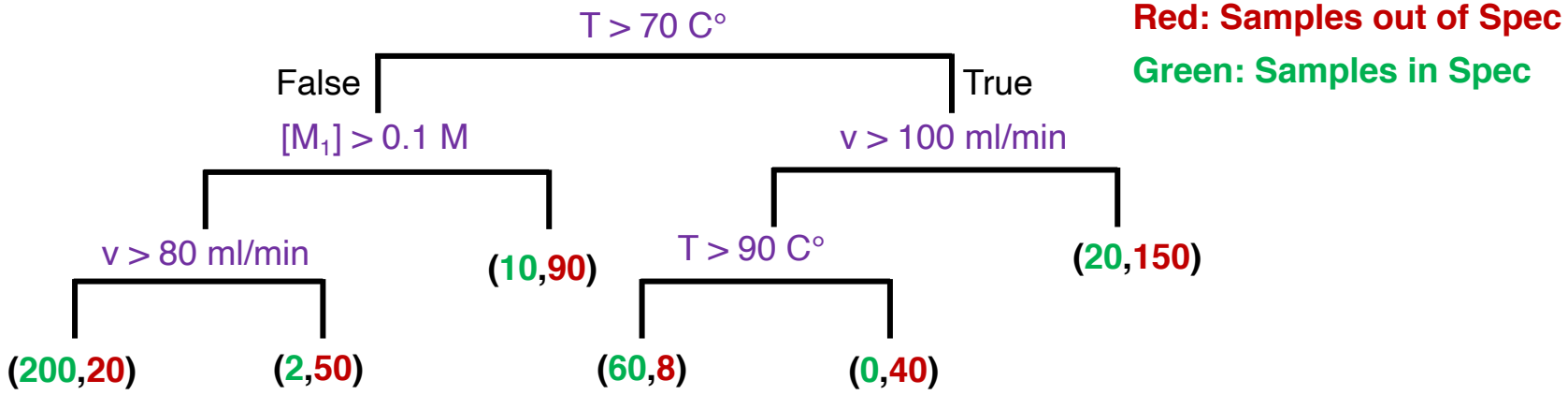


• Even though the model is simple, non-binary splits (left) are difficult to describe computationally. **CART** restricts itself to binary splits of each region with respect to a x_i

Decision Tress - Introduction

• For more than two features, drawing partitions as we did on the previous slide isn't feasible, but the binary tree representation works the same way.

• Let's say our inputs included three continuous features: reactor temperature (T), Reactant concentration (M_1), and volumetric flow rate (v), for a classification task:



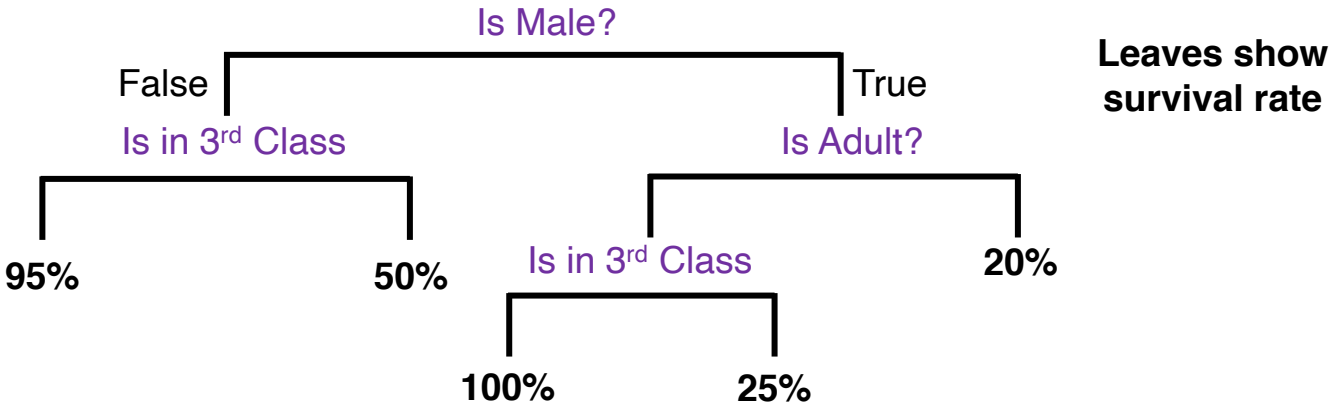
• With our tree in hand, prediction on unseen samples is straightforward: run the **x** for the new sample through the tree starting at the base until you terminate in a leaf. The leaf will tell you your class/value depending on the proportion (classification) or mean (regression).

Decision Tress – Titanic Example

• This is a famous machine learning example that is commonly used to highlight the strength of decision trees and by extension random forests (discussed later).

• Dataset can be downloaded from Kaggle and consists of 891 passengers 11 features (x) and whether or not each survived (y).

Example of decision tree utilizing 3 features:



The example highlights the high interpretability of decision trees. You can identify which features are important, which sub-populations were likely and unlikely to survive, and for what reasons.

Decision Trees – Definitions

Kingsford, C. & Salzberg, S. L. "What are decision trees?" *Nature Biotech.* 26(9), 2008.

Node: A point in DT where a question is asked & a decision must be made

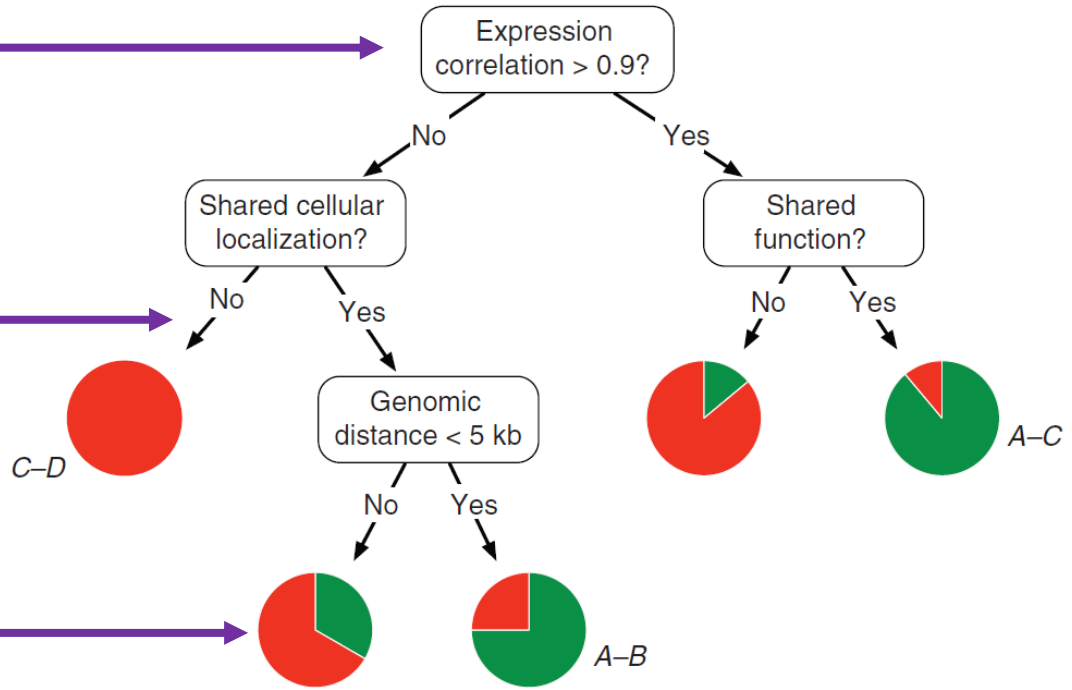
Branch: Represents the decision

Leaf: Final node which yields classification (via proportion)

a

Gene Pair	Interact?	Expression correlation	Shared localization?	Shared function?	Genomic distance
A-B	Yes	0.77	Yes	No	1 kb
A-C	Yes	0.91	Yes	Yes	10 kb
C-D	No	0.1	No	No	1 Mb

b



Decision Trees – Training

• Consider N data, each consisting of a response, y, and k inputs x. That is, (x_i,y_i) for i=1,2…,N with $x_i = (x_1,x_2,\ldots,x_k)$.

• During training we'll need to decide which variables to split on and their split value.

• Suppose we want M regions, $R_1,R_2,\ldots,R_M$ and we model the response in each region as a constant, c_m:

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

Where I() is an indicator function (1 or 0 for condition)

• If our error is defined as the sum of squares, then the best c_m choice is just the average of y_i in each region:

$$\bar{c}_m = \text{ave}\left(y_i | x_i \in R_m\right)$$

• So far so simple. However, simultaneously optimizing R_m and c_m is infeasible in this way. Instead, we need an algorithm that starts with a single region, tries to optimally split it along some xi, and recursively continues until some termination criteria.

Decision Trees – Training (Regression)

• In the typical greedy **CART** algorithm, we start with all of the data and consider a splitting variable *j* and split point *s*. These will define a pair of half planes:

$$R_1(j,s) = \{X | X_j \leq s\} \qquad R_2(j,s) = \{X | X_j > s\}$$

• We want to find the *j* and *s* that minimize the least-squared error:

$$\min_{j,s} \left[\min_{c_1} \sum_{x_i \in R_1(j,s)} (y_i - c_1)^2 + \min_{c_2} \sum_{x_i \in R_2(j,s)} (y_i - c_2)^2 \right]$$

• The inner minimizations can be solved by taking the average of the y_i in each region:

$$\bar{c}_1 = \text{ave}(y_i | x_i \in R_1(j,s)) \qquad \bar{c}_2 = \text{ave}(y_i | x_i \in R_2(j,s))$$

- The outer minimizations over all *j* and *s* are likewise feasible (ordering tricks can help).
- After finding the best split variable and split value, we recursively repeat on R_1 and R_2.

Decision Trees – Training (Classification)

• For classification, the only changes necessary are to our loss function. We will consider loss functions based on the proportion of observations of a given class, *k*, in R_m:

$$p_{mk} = \frac{1}{N_m} \sum_{x_i \in R_m} I\left(y_i = k\right)$$

• We classify the observations in a node based on the majority proportion:

$$k(m) = \underset{k}{\operatorname{argmax}}\left(p_{mk}\right)$$

• We seek to perform the split such that we minimize the impurity Q of the child nodes:

$$\min_{j,s}\left[\frac{N_1}{N_\mathrm{p}} Q\left(x_i \in R_1(j,s)\right) + \frac{N_2}{N_\mathrm{p}} Q\left(x_i \in R_2(j,s)\right)\right]$$

Where N_1 and N_2 are the samples in the binary child regions and N_p is the number of samples in the parent node.

$$Q = 1 - p_{mk(m)}$$

Misclassification Error

$$Q = \sum_{k=1}^{K} p_{mk}(1 - p_{mk})$$

Gini Index

$$Q = -\sum_{k=1}^{K} p_{mk} \log\left(p_{mk}\right)$$

Cross-Entropy/Deviance

Decision Trees – Training (Classification)

- Misclassification error is not generally desirable for generating trees.

Compare the following splits:

Split A

$$Q = 1 - p_{mk(m)} \rightarrow \frac{400}{800}(1-0.75) + \frac{400}{800}(1-0.75) = 0.25$$

$$Q = \sum_{k=1}^{K} p_{mk}(1-p_{mk}) \rightarrow \frac{400}{800}(0.25(1-0.25)+0.75(1-0.75)) + \frac{400}{800}(0.25(1-0.25)+0.75(1-0.75)) = 0.375$$

$$Q = -\sum_{k=1}^{K} p_{mk}\log(p_{mk}) \rightarrow \frac{400}{800}(-0.75\log(0.75)-0.25\log(0.25)) + \frac{400}{800}(-0.25\log(0.25)-0.75\log(0.75)) = 0.56$$

Split B

$$Q = 1 - p_{mk(m)} \rightarrow \frac{600}{800}(1-0.66) + \frac{200}{800}(1-1) = 0.25$$

$$Q = \sum_{k=1}^{K} p_{mk}(1-p_{mk}) \rightarrow \frac{600}{800}(0.33(1-0.33)+0.66(1-0.66)) + \frac{200}{800}(0.0+0.0) = 0.333$$

$$Q = -\sum_{k=1}^{K} p_{mk}\log(p_{mk}) \rightarrow \frac{600}{800}(-0.33\log(0.33)-0.66\log(0.66)) + \frac{200}{800}(-1.0\log(1.0)-0) = 0.48$$

Both splits exhibit the same misclassification error, but B is intuitively preferable because it produces a “pure” node. Both Gini and Cross-Entropy prefer B (in general, these perform similarly).

Decision Trees – Training (Stopping)

• It is obvious that if we let our tree grow indefinitely, eventually every sample will occupy a pure leave and the data will be perfectly memorized. Several stopping heuristics are variously employed to determine when to stop:

- Is the reduction in error from a split above a threshold? (*not preferred, sometimes split with low reduction will lead to an important split below it*)
- Is the node already pure enough? (*preferred to above, essentially inverse*)
- Has the tree reached a maximum depth?
- Is the number of samples in the children nodes too small?

Decision Trees – Additional Details

• **Why binary splits?** We might consider splitting into more than two child nodes. In general, multiway splits fragment data too quickly and don't leave enough data for lower layers. Also, since a multiway split can be decomposed into a series of binary splits, the latter is preferable.

• Since splits are based on a single feature, XOR type relationships cannot be effectively learned with basic implementations.

Decision Trees – Python Details

.DecisionTreeClassifier() from sklearn.tree

Important hyperparameters:

max_depth: depth to grow tree (important for overfitting)

max_features: number to features to search for split (important for computational efficiency)

min_samples_split: number of samples required to attempt a split

min_samples_leaf: minimum number of samples required to define a leaf.

min_impurity_decrease: minimum impurity decrease to perform a split (Hastie et al say don't use for reasons given above)

Important attributes:

.classes_, **feature_importances_**, <-- useful but not unique

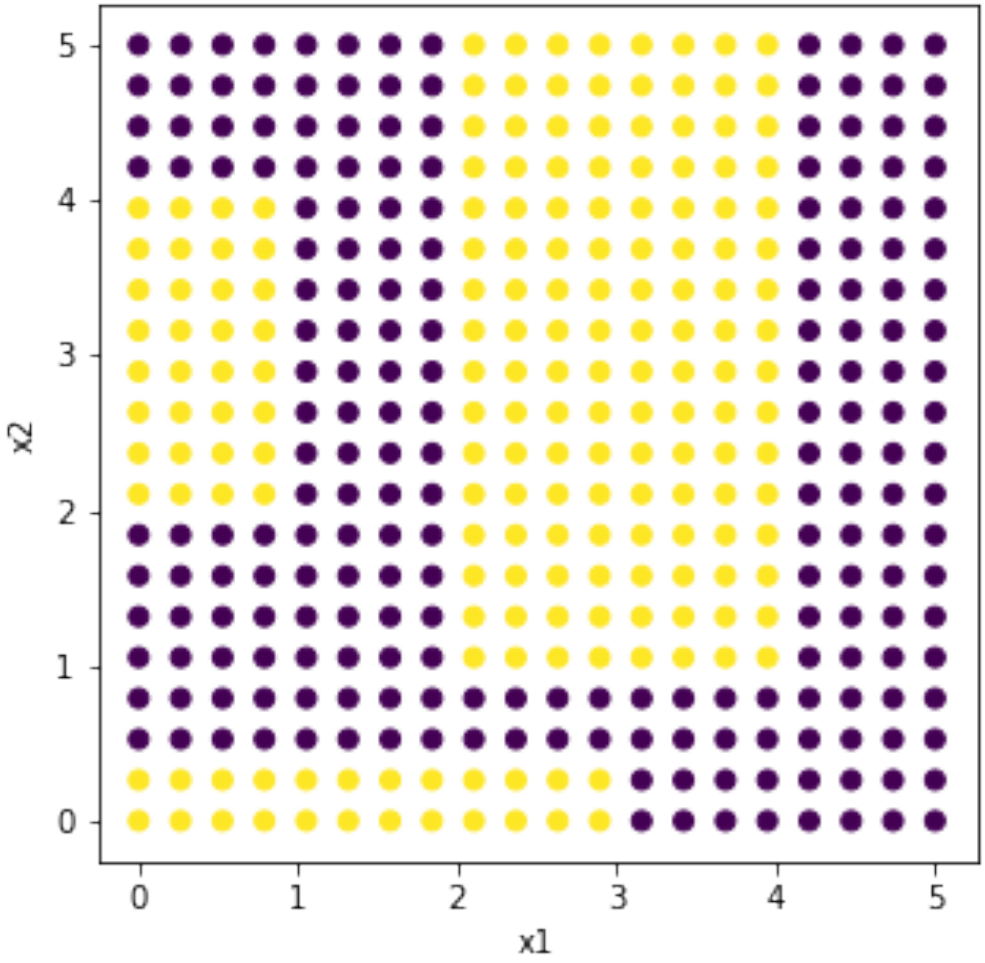
.tree_ <-- stores tree structure

Decision Trees – Python Details

To illustrate some of the details of how decision trees behave, we'll use the following classification problem:

Two features (x1 and x2), one classification (color) with two values (purple, yellow). This could easily map on to a pass/fail type classification

To start we have no noise (this influences hyperparameter selection).



Decision Trees – Python Details

Train the model:

```
from sklearn import tree
model = tree.DecisionTreeClassifier() # default depth and purity criteria
model.fit(X,Y)
```

Illustrate predictions:

```
# Plot decision boundaries
x_min, x_max = X[:,0].min() - 1, X[:,0].max() + 1
y_min, y_max = X[:,1].min() - 1, X[:,1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.01),
np.arange(y_min, y_max, 0.01))
plt.figure(figsize=(5,5))
Z = model.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)
cs = plt.contourf(xx, yy, Z)

# Plot predictions
plt.scatter(X[:,0],X[:,1],edgecolors="k",c=model.predict(X))
plt.xlabel("x1")
plt.ylabel("x2")
plt.show()
```

Note: `np.c_()` is a helper function for reorganizing arrays, `np.r_()` is also a common helper. `plt.contourf()` and `np.meshgrid()` always confuse me personally and I have to experiment with them almost every time that I use them.

Decision Trees – Python Details

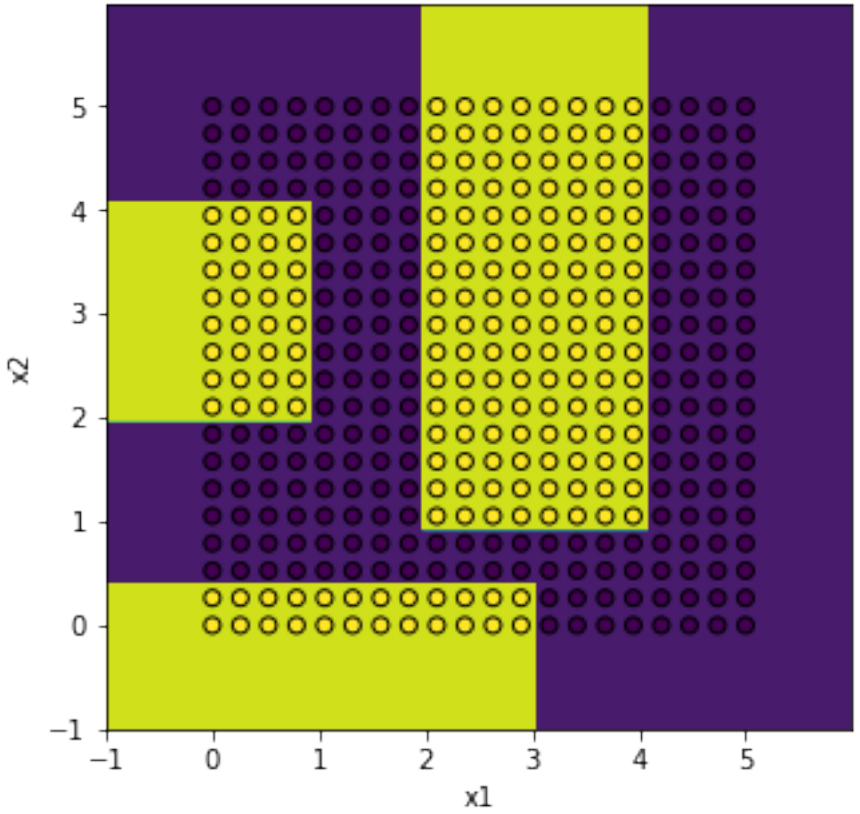
Train the model:

```
from sklearn import tree
model = tree.DecisionTreeClassifier() # default depth and purity criteria
model.fit(X,Y)
```

Illustrate predictions:

Perfect predictions!

It is obvious that if we let our tree grow indefinitely we can always get perfect training predictions, but the model won't generalize



Common stopping criteria:

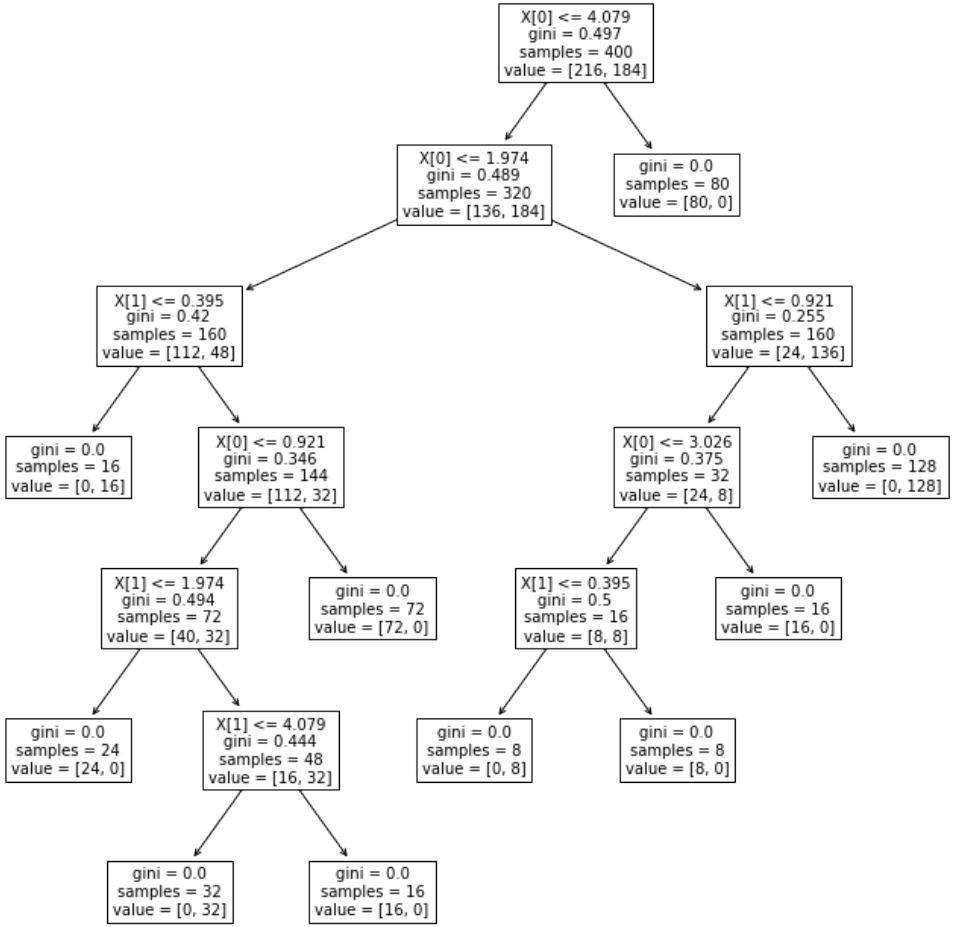
- *Is the node pure enough?*
- *Has the tree reached max depth?*
- *Is the number of samples in the split too small?*

Decision Trees – Python Details

Visualizing the model:

```
# Showing the resulting decision tree
plt.figure(figsize=(12,12))
tree.plot_tree(model, fontsize=10)
plt.show()
```

Note: first split occurs along x1 rather than x2, because it is the only split that allows a pure child leaf:



Decision Trees – Python Details

Varying depth of the tree:

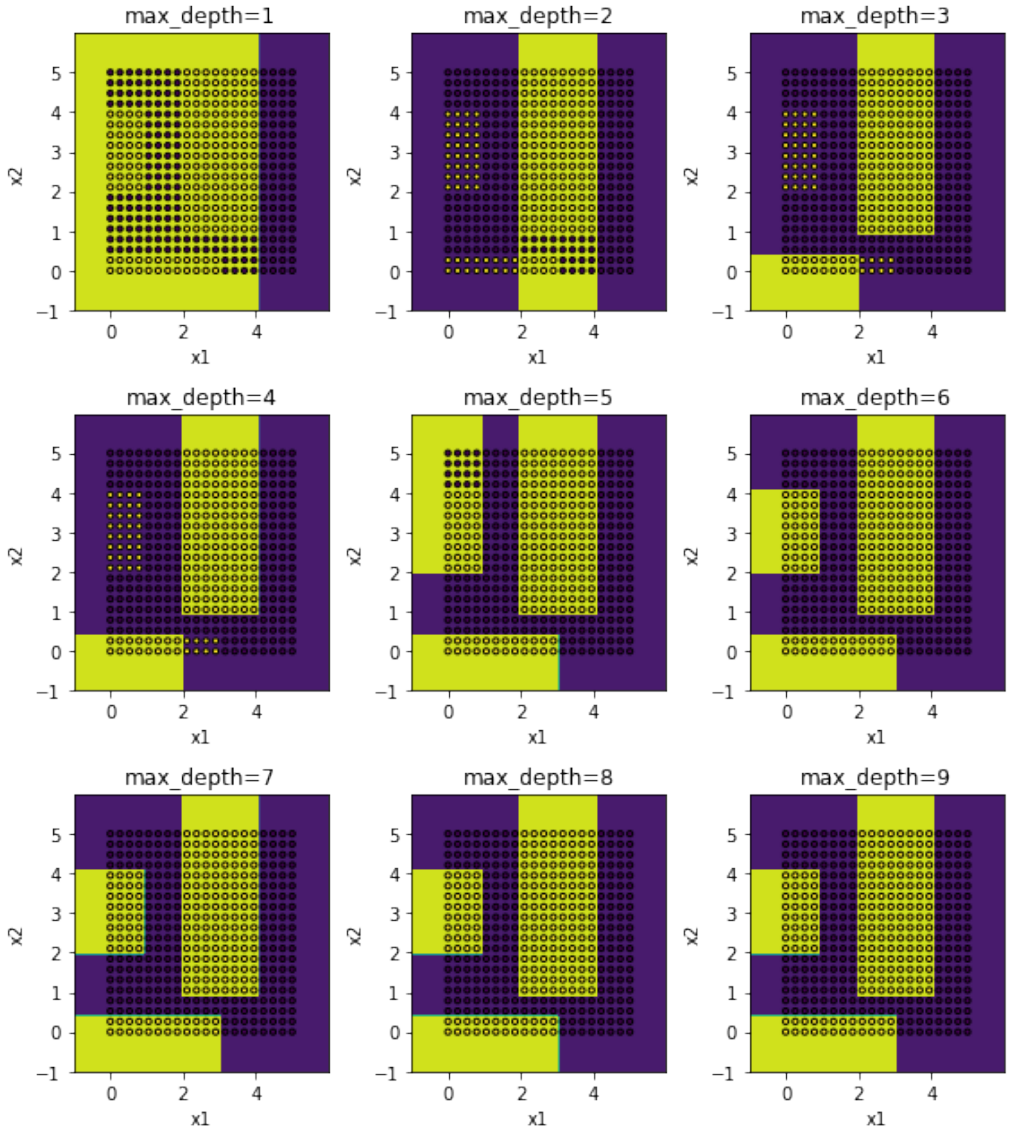
```
# Loop over max_depth and evaluate the results
plt.figure(figsize=(8,9))
for i in range(1,10):
    model = tree.DecisionTreeClassifier(max_depth=i) model.fit(X,Y)
    ax = plt.subplot(3,3,i)
    Z = model.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)
    ax.contourf(xx, yy, Z)

    # Plot original data
    ax.scatter(X[:,0],X[:,1],s=10,edgecolors="k",c=Y)
    ax.set_xlabel("x1")
    ax.set_ylabel("x2")
    ax.set_title("max_depth={}".format(i))
plt.tight_layout()
plt.show()
```

Decision Trees – Python Details

Varying depth of the tree:

Clearly, adding more complexity after max_depth=6 adds no explanatory power to the model



Decision Trees – Limits of Validation Data

Using validation data to select max_depth:

```
import random
from sklearn.metrics import accuracy_score

# Using validation data to determine an optimal tree depth
inds = list(range(len(X)))
random.shuffle(inds)
train_inds = inds[:int(len(X)*0.8)]
val_inds = inds[int(len(X)*0.8):]
X_train = X[train_inds]
Y_train = Y[train_inds]
X_val = X[val_inds]
Y_val = Y[val_inds]

# Loop over max_depth and evaluate the results
plt.figure(figsize=(8,9))
for i in range(1,10):
    model = tree.DecisionTreeClassifier(max_depth=i)
    model.fit(X_train,Y_train)
    Y_pred = model.predict(X_val)
    acc = accuracy_score(Y_val,Y_pred)
    ax = plt.subplot(3,3,i)
    Z = model.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)
    ax.contourf(xx, yy, Z)

    # Plot original data
    ax.scatter(X[:,0],X[:,1],s=10,edgecolors="k",c=Y)
    ax.set_xlabel("x1")
    ax.set_ylabel("x2")
    ax.set_title("max_depth={}\nAccuracy:{}".format(i,acc))
plt.tight_layout()
plt.show()
```

Decision Trees – Limits of Validation Data

Using validation data to select max_depth:

Why don't we our validation error eventually increase as we increase the complexity of the tree?

Why don't we our validation error eventually increase as we increase the complexity of the tree?