15-388/688 - Practical Data Science: Decision trees and boosting

J. Zico Kolter
Carnegie Mellon University
Fall 2016

Outline

Decision trees

Training (classification) decision trees

Boosting

Examples

Announcements

HW4 due tonight

Misplaced parenthesis in Python SVM code in slides (pg 26), apologies

We will send feedback on tutorial mid-way report by tomorrow (Thursday) night (this will be a *brief* review)

Tutorial submission for K-means is due by Monday

Outline

Decision trees

Training (classification) decision trees

Boosting

Examples

Overview

Decision trees and boosted decision trees are some of the most ubiquitous algorithms in data science

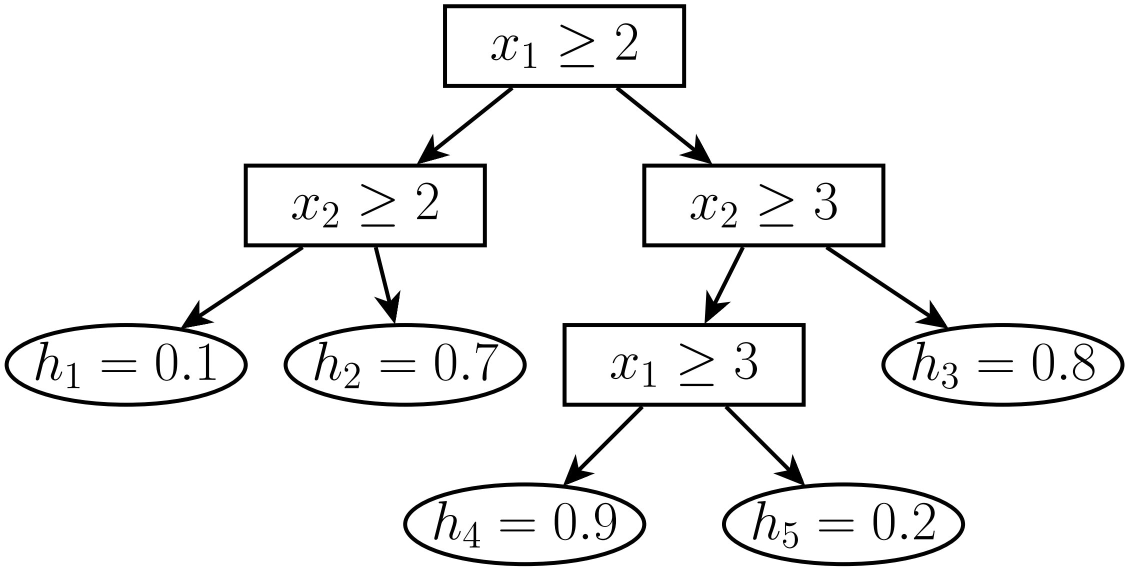
Boosted decision trees typically perform very well without much tuning (the majority of Kaggle contests, for instance, are won with boosting methods)

Decision trees, while not as powerful from a pure ML standpoint, are still one of the canonical examples of an "understandable" ML algorithm

Decision trees

Decision trees were one of the first machine learning algorithms

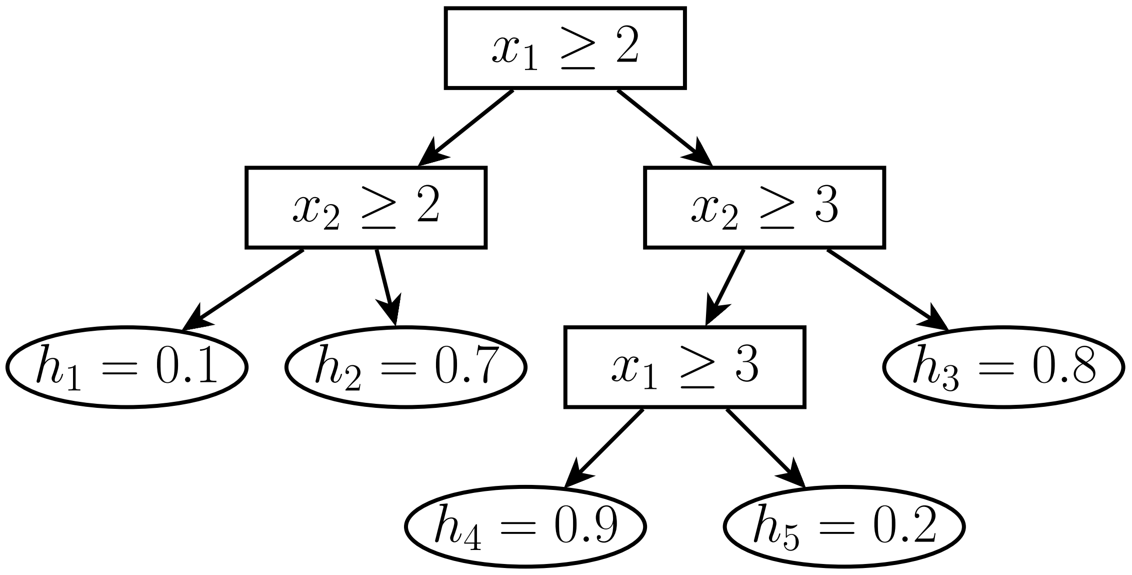
Basic idea: make classification/regression predictions by tracing through rules in a tree, with a constant prediction at each leaf node

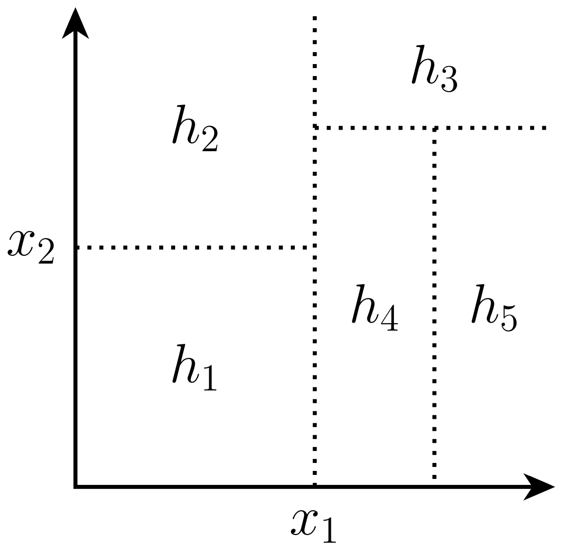


Partitioning the input space

You can think of the hypothesis function of decision trees as partitioning the input space with axis-aligned boundaries

In each partition, predict a constant value

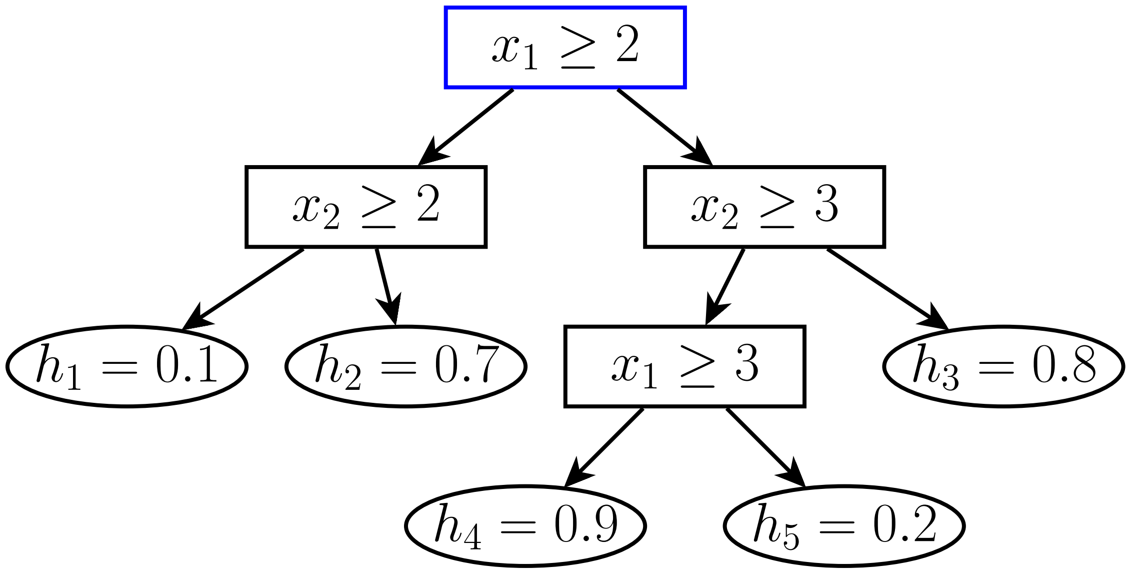


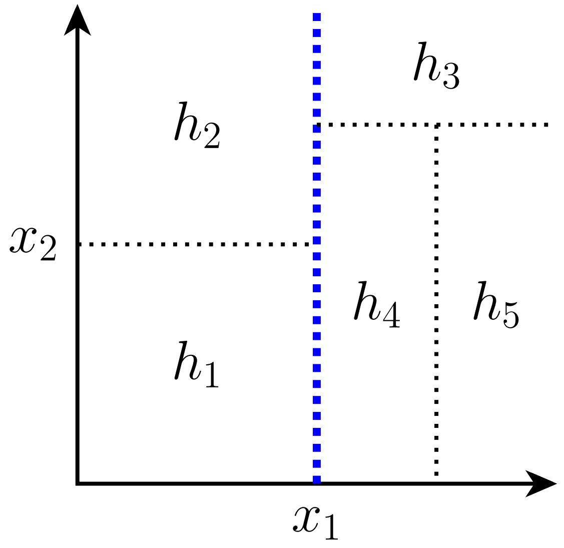


Partitioning the input space

You can think of the hypothesis function of decision trees as partitioning the input space with axis-aligned boundaries

In each partition, predict a constant value

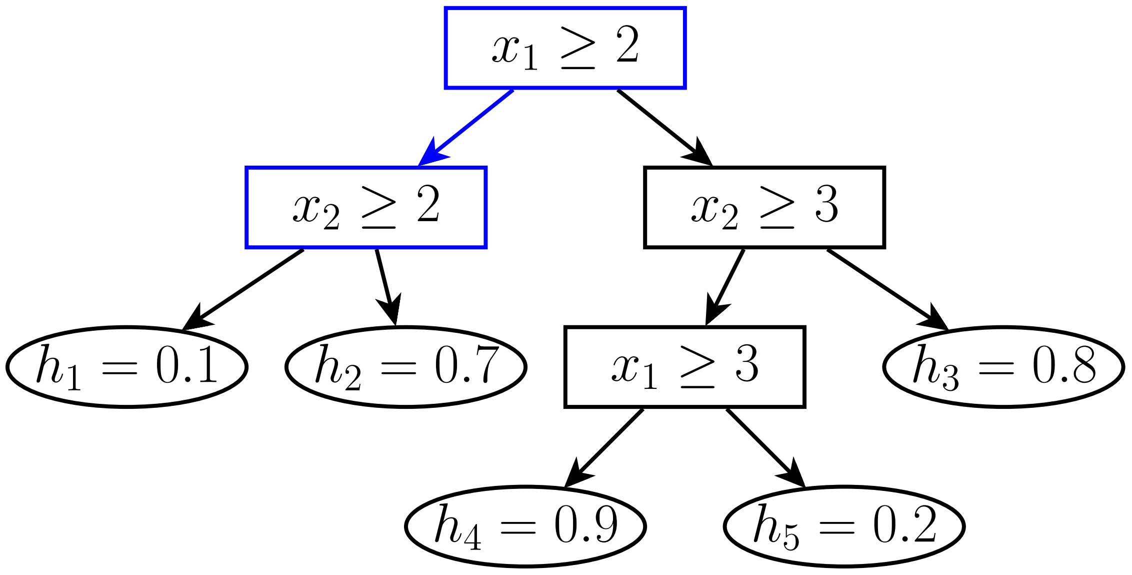


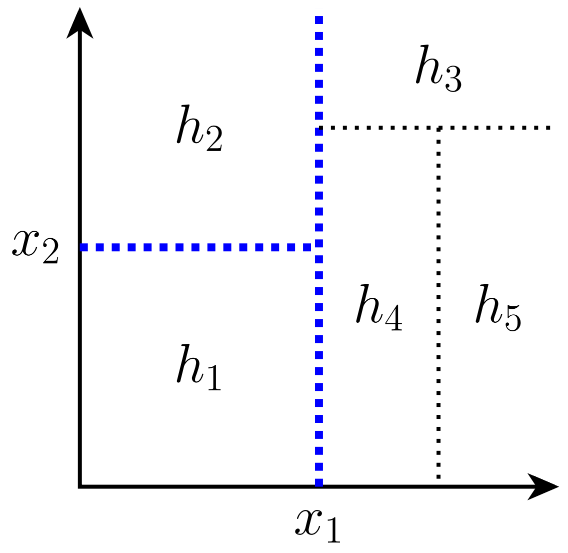


Partitioning the input space

You can think of the hypothesis function of decision trees as partitioning the input space with axis-aligned boundaries

In each partition, predict a constant value

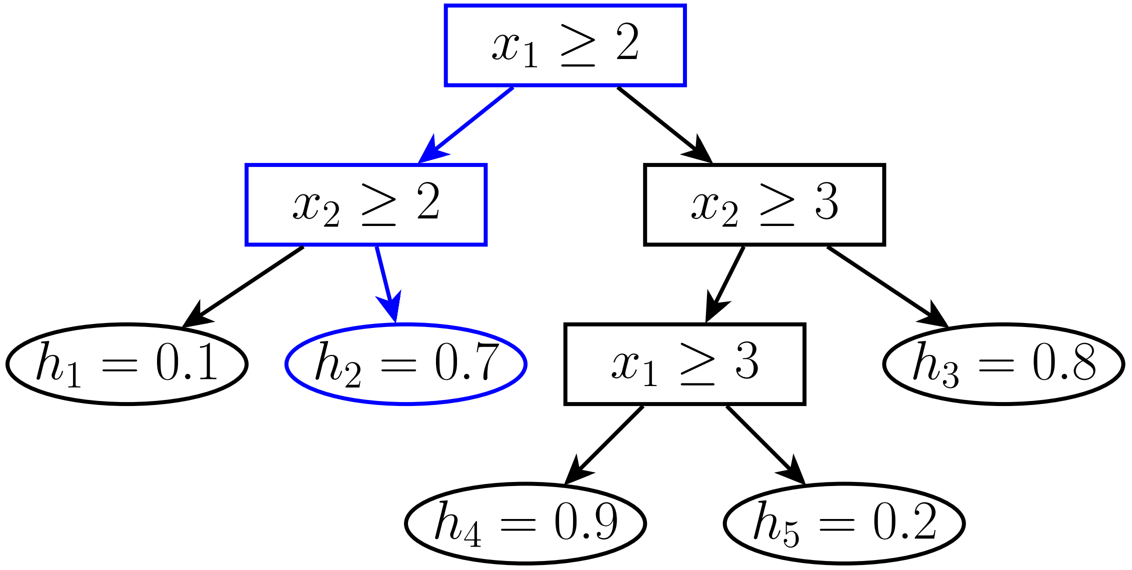


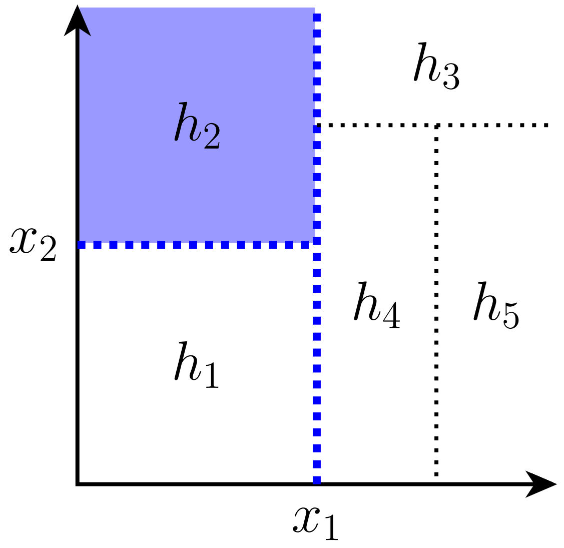


Partitioning the input space

You can think of the hypothesis function of decision trees as partitioning the input space with axis-aligned boundaries

In each partition, predict a constant value





Outline

Decision trees

Training (classification) decision trees

Boosting

Examples

Decision trees as ML algorithms

To specify the decision trees from a machine learning standpoint, we need to specify

1. What is the hypothesis function $h_\theta(x)$?

2. What is the loss function $\ell(h_\theta(x), y)$?

3. How do we minimize the loss function?

$$\underset{\theta}{\text{minimize}} \; \frac{1}{m}\sum_{i=1}^{m} \ell(h_\theta(x^{(i)}), y^{(i)})$$

Decision trees as ML algorithms

To specify the decision trees from a machine learning standpoint, we need to specify

1. **What is the hypothesis function $h_\theta(x)$?**
2. What is the loss function $\ell(h_\theta(x), y)$?
3. How do we minimize the loss function?

$$\underset{\theta}{\text{minimize}} \; \frac{1}{m}\sum_{i=1}^{m} \ell(h_\theta(x^{(i)}), y^{(i)})$$

…a decision tree (θ is shorthand for all the parameters that define the tree: tree structure, values to split on, leaf predictions, etc)

Decision trees as ML algorithms

To specify the decision trees from a machine learning standpoint, we need to specify

1. What is the hypothesis function $h_\theta(x)$?

2. **What is the loss function $\ell(\boldsymbol{h_\theta}(\boldsymbol{x}), \boldsymbol{y})$?**

3. How do we minimize the loss function?

$$\operatorname*{minimize}_{\theta} \ \frac{1}{m}\sum_{i=1}^{m} \ell(h_\theta(x^{(i)}), y^{(i)})$$

Loss functions in decision trees

Let’s assume the output is binary for now (classification task, we will deal with regression shortly), and assume $y \in \{0,1\}$

The typical decision tree algorithm using a probabilistic loss function that considers y to be a Bernoulli random variable with probability $h_\theta(x)$

$$p(y|h_\theta(x)) = h_\theta(x)^y \big(1 - h_\theta(x)\big)^{1-y}$$

The loss function is just the negative log probability of the output (like in maximum likelihood estimation)

$$\begin{aligned}\ell(h_\theta(x), y) &= -\log p(y|h_\theta(x)) \\ &= -y \log h_\theta(x) - (1-y)\log\big(1 - h_\theta(x)\big)\end{aligned}$$

Decision trees as ML algorithms

To specify the decision trees from a machine learning standpoint, we need to specify

1. What is the hypothesis function $h_\theta(x)$?

2. What is the loss function $\ell(h_\theta(x), y)$?

3. **How do we minimize the loss function?**

$$\underset{\boldsymbol{\theta}}{\mathbf{minimize}} \frac{1}{m} \sum_{i=1}^{m} \ell(h_{\theta}(x^{(i)}), y^{(i)})$$

Optimizing decision trees

Key challenge: unlike models we have considered previously, discrete tree structure means there are no gradients

Additionally, even if we assume binary inputs i.e., $x \in \{0,1\}^n$, there are 2^{2^n} possible decision trees: $n = 7$ means 3.4×10^{38} possible trees

Instead, we're going to use *greedy* methods to incrementally build the tree (i.e., minimize the loss function) one node at a time

Optimizing a single leaf

Consider a single leaf in a decision tree (could be root of initial tree)

Let $\mathcal{X}$ denote the examples at this leaf (e.g., in this partition), where $\mathcal{X}^+$ denotes the positive examples and $\mathcal{X}^-$ denotes negative (zero) examples

What should we choose as the (constant) prediction h at this leaf?

$$\underset{h}{\text{minimize}} \; \frac{1}{|\mathcal{X}|} \sum_{x,y \in \mathcal{X}} \ell(h, y) \;\; = -\frac{|\mathcal{X}^+|}{|\mathcal{X}|} \log h - \frac{|\mathcal{X}^-|}{|\mathcal{X}|} \log(1-h)$$

$$\Longrightarrow h = \frac{|\mathcal{X}^+|}{|\mathcal{X}|},$$

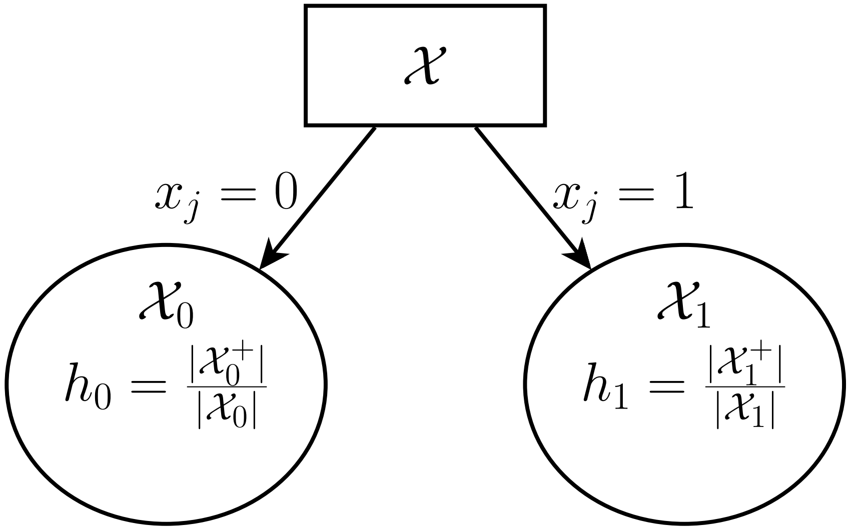
Which achieves loss:

$$\ell = -h \log h - (1-h) \log(1-h)$$

Optimizing splits

Now suppose we want to split this leaf into two leaves, assuming for the time being that $x \in \{0{,}1\}^n$ is binary

If we split on a given feature j, this will separate $\mathcal{X}$ into two sets: $\mathcal{X}_0$ and $\mathcal{X}_1$ (with $\mathcal{X}_{0/1}^{+/-}$ and defined similarly to before), and we would choose optimal prediction $h_0 = |\mathcal{X}_0^+|/|\mathcal{X}_0|, h_1 = |\mathcal{X}_1^+|/|\mathcal{X}_1|$


$\mathcal{X}$
$x_j = 0$
$x_j = 1$
$\mathcal{X}_0$
$h_0 = \frac{|\mathcal{X}_0^+|}{|\mathcal{X}_0|}$
$\mathcal{X}_1$
$h_1 = \frac{|\mathcal{X}_1^+|}{|\mathcal{X}_1|}$

Loss of split

The new leafs will each now suffer loss

$$\ell_0 = -h_0 \log h_0 - (1 - h_0) \log(1 - h_0)$$
$$\ell_1 = -h_1 \log h_1 - (1 - h_1) \log(1 - h_1)$$

Thus, if we split the original leaf on feature j, we no longer suffer our original loss ℓ, but we do suffer losses $\ell_1 + \ell_2$, i.e., we have decreased the overall loss function by $\ell_0 + \ell_1 - \ell$ (this quantity is called ***information gain***)

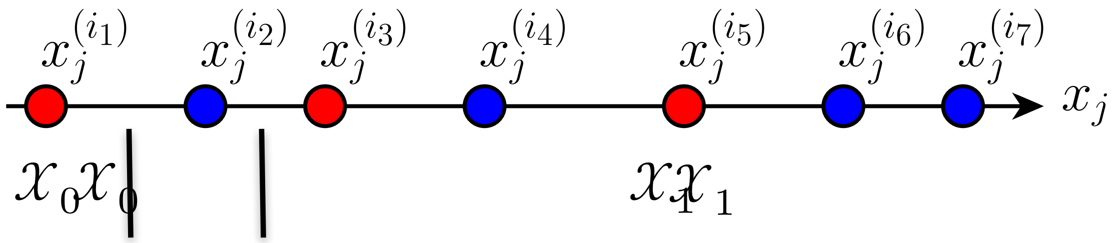
Greedy decision tree learning – repeat:

- For all leaf nodes, evaluate information gain (i.e., decrease in loss) when splitting on each feature j
- Split the node/feature that minimizes loss the most
- (Run cross-validation to determine when to stop, or after N nodes)

Continuous features

What if x_j's are continuous?

Solution: sort the examples by their x_j values, compute information gain at each possible split point



Regression trees

Regression trees are the same, except that the hypothesis h are real-valued instead of probabilities, and we use squared loss

$$\ell(h, y) = (h - y)^2$$

This means that the loss a node is given by

$$\operatorname*{minimize}_h \frac{1}{|\mathcal{X}|} \sum_{x,y \in \mathcal{X}} (h - y)^2 \implies h = \frac{1}{|\mathcal{X}|} \sum_{x,y \in \mathcal{X}} y \ \ (\mathrm{i.e.\,mean})$$

and suffers loss

$$\ell = \frac{1}{|\mathcal{X}|} \sum_{x,y \in \mathcal{X}} (y - h)^2 \quad (\mathrm{i.e.\,variance})$$

Outline

Decision trees

Training (classification) decision trees

Boosting

Examples

Ensembles of trees

Decision trees have notable advantages (they are relatively easy to interpret, usually fast to train, insensitive to scale of input features)

But, they are also quite limited in their representation power (require axis-aligned splits, don’t model probabilities very smoothly)

Basic idea of tree *ensemble* methods is to combine *multiple* tree models together to form a better predictor

Two of the most popular ensemble methods: random forests and boosting

Boosting

Boosting originated as an idea in theoretical machine learning, for “boosting” the performance of weak classifiers (i.e., combining many classifiers that each had modest accuracy to one that had high accuracy)

After some initial success in theory, during the 90s there came to be several practical applications of boosting methods

There are *many* interpretations of boosting (experts still disagree on the “right” one!), and I’m going to highlight one:

We focus on the Gradient Boosted Regression Trees (GBRT) algorithm

Machine learning with general predictions

Let's consider the basic machine learning optimization problem (could be any loss function, classification or regression)

$$\underset{\theta}{\text{minimize}} \sum_{i=1}^{m} \ell(\hat{y}^{(i)}, y^{(i)})$$

where $\hat{y}^{(i)}$ denotes our prediction for the ith example

The gradient with respect to these *predictions* themselves to determine the best way to adjust our predictions (ignoring whether we have any hypothesis function that could actually change the predictions in this way)

$$\frac{\partial}{\partial \hat{y}^{(i)}} \ell(\hat{y}^{(i)}, y^{(i)})$$

Some examples of prediction gradients

We can compute gradients with respect to the predictions for all our usually loss functions (they are actually *easier* to compute than gradients with respect to parameters)

Logistic loss:

$$\ell_{\text{logistic}}(\hat{y}, y) = \log(1 + \exp(-\hat{y} \cdot y))$$
$$\frac{\partial}{\partial \hat{y}} \ell_{\text{logistic}}(\hat{y}, y) = \frac{-y}{1 + \exp(\hat{y} \cdot y)}$$

Squared loss:

$$\ell_{\text{squared}}(\hat{y}, y) = \frac{1}{2}(\hat{y} - y)^2$$
$$\frac{\partial}{\partial \hat{y}} \ell_{\text{squared}}(\hat{y}, y) = (\hat{y} - y)$$

Basic idea of GBRTs

Basic idea: GBRTs are effectively performing *gradient descent* on our loss function, using *regression trees* to approximate the gradient

```
Given: Data set $(x^{(i)}, y^{(i)})_{i=1,\ldots,m}$, # trees $T$, loss $\ell$, step size $\alpha$
Initialize:
    $\hat{y}^{(i)} \leftarrow 0, \quad \forall\, i = 1, \ldots, m$
For $t = 1, \ldots, T$:
    Compute gradients:
        $g_t^{(i)} \leftarrow \frac{\partial}{\partial \hat{y}} \ell(\hat{y}^{(i)}, y^{(i)}), \quad \forall\, i = 1, \ldots, m$
    $h_t \leftarrow \mathrm{Train-Regression-Tree}\left(x^{(1,\ldots,m)}, g_t^{(1,\ldots,m)}\right)$
    Update predictions:
        $\hat{y}^{(i)} \leftarrow \hat{y}^{(i)} - \alpha h_t(x^{(i)})$
For new data point $x$:
    Predict: $\hat{y} = -\alpha \sum_{i=1}^{T} h_t(x^{(i)})$
```

GBRTs a bit more practically

In practice, fitting the trees is slow, so we actually do a line search to determine how large of a gradient step to take

Can take different gradient steps at different leaves in tree

Here you probably want to use a library (you *could* write your own implementation, but it would be about ~100 lines of code, not ~5 like for an SVM)

Outline

Decision trees

Training (classification) decision trees

Boosting

Examples

Decision trees and GBRTs in scikit-learn

Interface for decision trees and GBRTs in scikit-learn is just like any other classifier

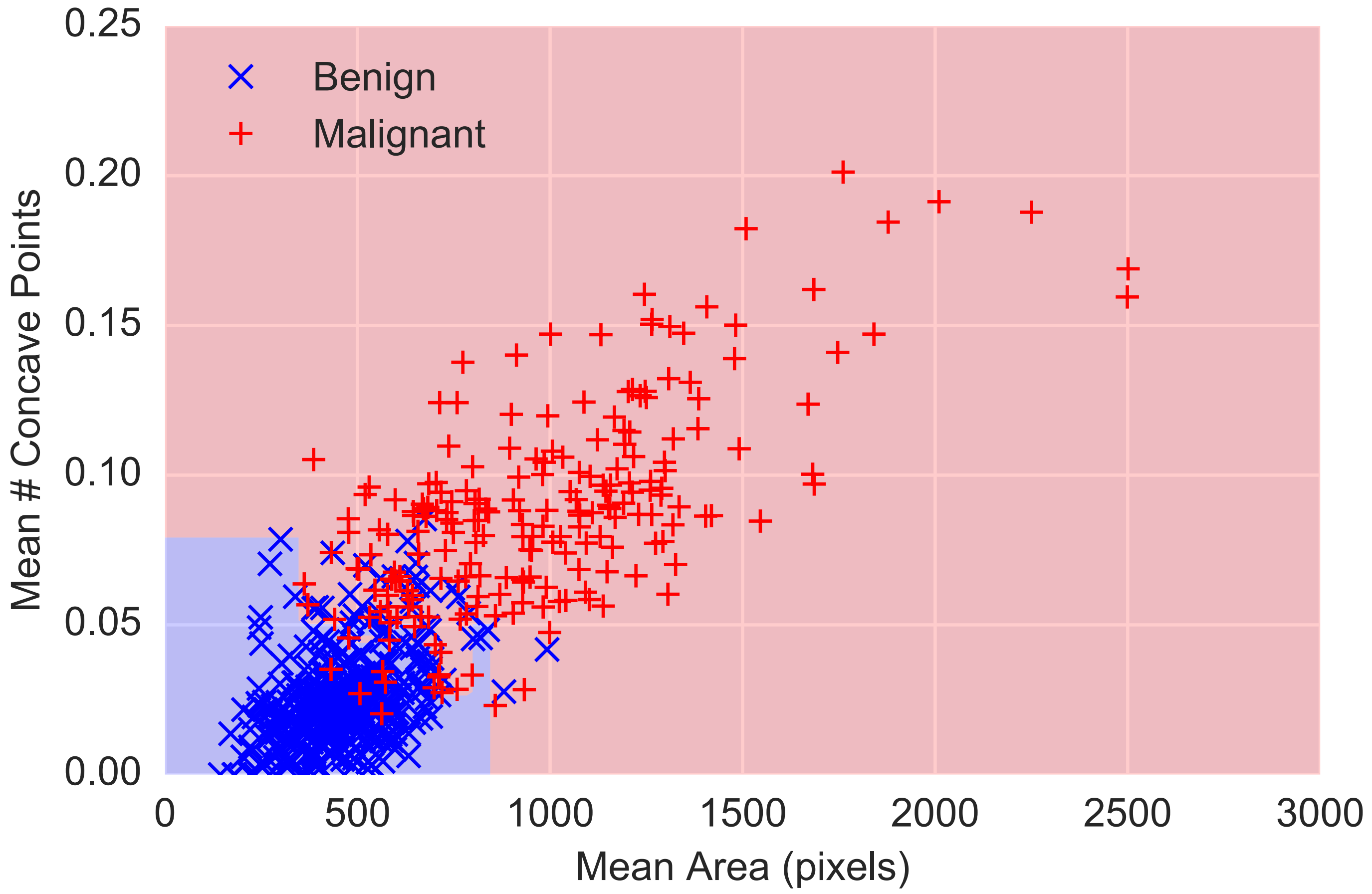
```
from sklearn.tree import DecisionTreeClassifier

clf = DecisionTreeClassifier(criterion='entropy', max_depth=5)
clf.fit(X, y)
```

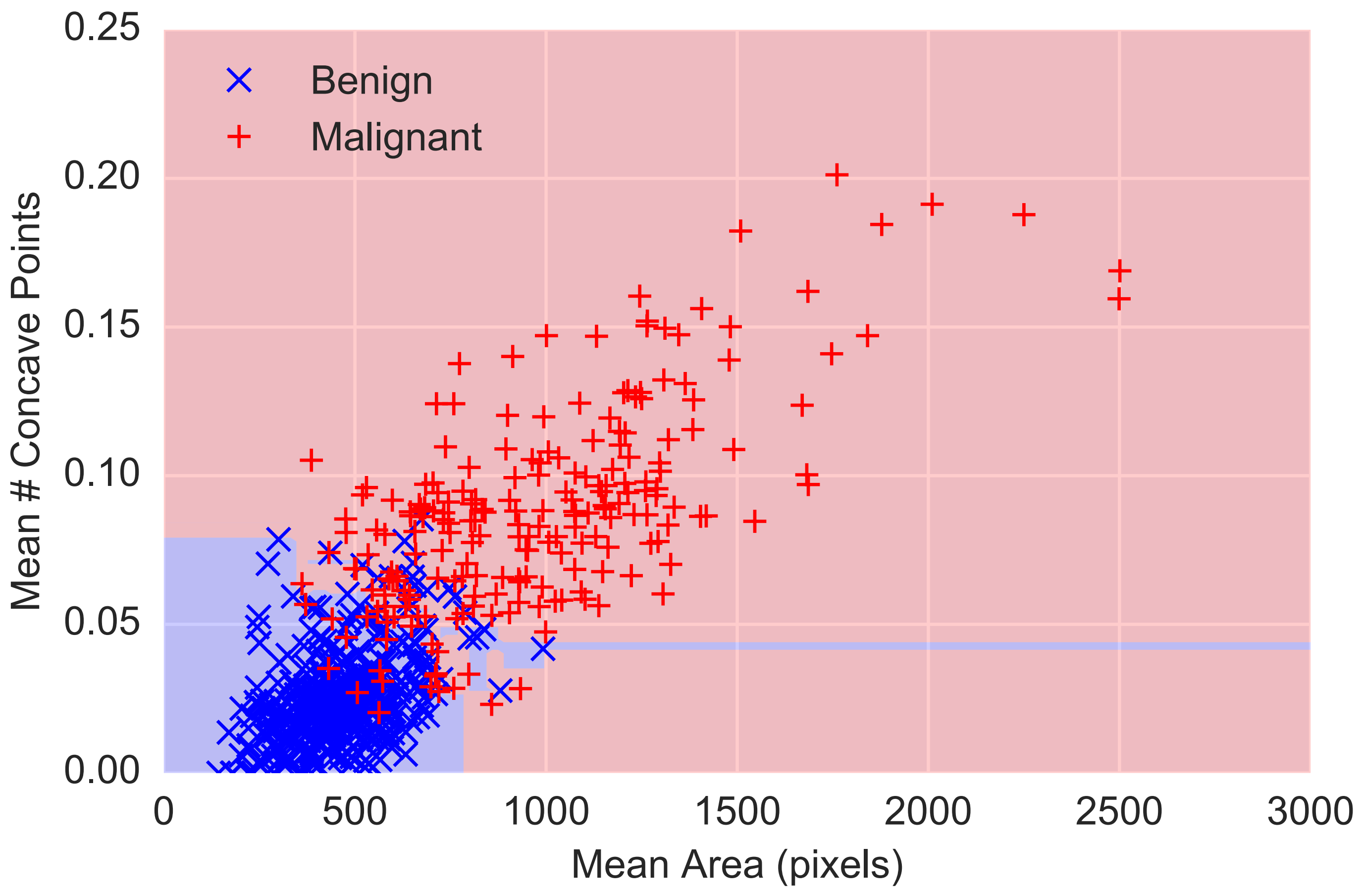
```
from sklearn.ensemble import GradientBoostingClassifier

clf = GradientBoostingClassifier(loss='deviance',
                                 max_depth=3, n_estimators=100)
clf.fit(X, y)
```

Decision tree surface for cancer prediction



GBRT surface for cancer prediction


Benign
Malignant
Mean # Concave Points
Mean Area (pixels)