Multiclass Classification, Structured Prediction, & Decision Trees

Mengye Ren

NYU

Nov 7, 2023

Margin for Multiclass

Binary

- Margin for $(x^{(n)}, y^{(n)})$:

$$y^{(n)} w^T x^{(n)} \tag{1}$$

- Want margin to be large and positive ($w^T x^{(n)}$ has same sign as $y^{(n)}$)

Multiclass

- Class-specific margin for $(x^{(n)}, y^{(n)})$:

$$h(x^{(n)}, y^{(n)}) - h(x^{(n)}, y). \tag{2}$$

- Difference between scores of the correct class and each other class
- Want margin to be large and positive for all $y \neq y^{(n)}$.

Multiclass SVM: separable case

Binary

$$\begin{aligned} \min_{w} \quad & \frac{1}{2}\|w\|^2 && (3) \\ \text{s.t.} \quad & \underbrace{y^{(n)} w^T x^{(n)}}_{\text{margin}} \geqslant 1 \quad \forall (x^{(n)}, y^{(n)}) \in \mathcal{D} && (4) \end{aligned}$$

Multiclass As in the binary case, take 1 as our target margin.

$$m_{n,y}(w) \stackrel{\text{def}}{=} \underbrace{\left\langle w, \Psi(x^{(n)}, y^{(n)}) \right\rangle}_{\text{score of correct class}} - \underbrace{\left\langle w, \Psi(x^{(n)}, y) \right\rangle}_{\text{score of other class}} \tag{5}$$

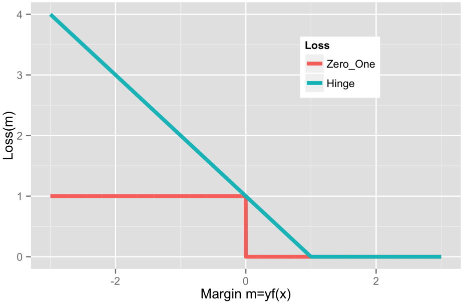
$$\begin{aligned} \min_{w} \quad & \frac{1}{2}\|w\|^2 && (6) \\ \text{s.t.} \quad & m_{n,y}(w) \geqslant 1 \quad \forall (x^{(n)}, y^{(n)}) \in \mathcal{D}, y \neq y^{(n)} && (7) \end{aligned}$$

Exercise: write the objective for the non-separable case

Recap: hingle loss for binary classification

- Hinge loss: a convex upperbound on the 0-1 loss

$$\ell_{\text{hinge}}(y, \hat{y}) = \max(0, 1 - yh(x)) \tag{8}$$



Generalized hinge loss

- What's the zero-one loss for multiclass classification?

$$\Delta(y,y') = \mathbb{I}\{y \neq y'\} \tag{9}$$

- In general, can also have different cost for each class.
- Upper bound on $\Delta(y,y')$.

$$\begin{aligned}
\hat{y} &\overset{\text{def}}{=} \operatorname*{arg\,max}_{y' \in \mathcal{Y}} \left\langle w, \Psi(x,y') \right\rangle && (10)\\
\Longrightarrow &\langle w, \Psi(x,y)\rangle \leqslant \langle w, \Psi(x,\hat{y})\rangle && (11)\\
\Longrightarrow &\Delta(y,\hat{y}) \leqslant \Delta(y,\hat{y}) - \langle w, (\Psi(x,y) - \Psi(x,\hat{y}))\rangle \qquad \text{When are they equal?} && (12)
\end{aligned}$$

- Generalized hinge loss:

$$\ell_{\text{hinge}}(y,x,w) \overset{\text{def}}{=} \max_{y' \in \mathcal{Y}} \left(\Delta(y,y') - \left\langle w, \left(\Psi(x,y) - \Psi(x,y')\right)\right\rangle\right) \tag{13}$$

Multiclass SVM with Hinge Loss

- Recall the hinge loss formulation for binary SVM (without the bias term):

$$\min_{w\in \mathrm{R}^d} \frac{1}{2}||w||^2 + C\sum_{n=1}^{N} \max\left(0, 1 - \underbrace{y^{(n)}w^T x^{(n)}}_{\text{margin}}\right).$$

- The multiclass objective:

$$\min_{w\in \mathrm{R}^d} \frac{1}{2}||w||^2 + C\sum_{n=1}^{N} \max_{y'\in\mathcal{Y}}\left(\Delta(y,y') - \underbrace{\left\langle w, \left(\Psi(x,y) - \Psi(x,y')\right)\right\rangle}_{\text{margin}}\right)$$

  - $\Delta(y,y')$ as target margin for each class.
  - If margin $m_{n,y'}(w)$ meets or exceeds its target $\Delta(y^{(n)},y')$ $\forall y \in \mathcal{Y}$, then no loss on example n.

Recap: What Have We Got?

- Problem: Multiclass classification $\mathcal{Y} = \{1, \ldots, k\}$
- Solution 1: One-vs-All
  - Train k models: $h_1(x), \ldots, h_k(x) : \mathcal{X} \to \mathrm{R}$.
  - Predict with $\arg\max_{y \in \mathcal{Y}} h_y(x)$.
  - Gave simple example where this fails for linear classifiers
- Solution 2: Multiclass loss
  - Train one model: $h(x, y) : \mathcal{X} \times \mathcal{Y} \to \mathrm{R}$.
  - Prediction involves solving $\arg\max_{y \in \mathcal{Y}} h(x, y)$.

Does it work better in practice?

- Paper by Rifkin & Klautau: "In Defense of One-Vs-All Classification" (2004)
  - Extensive experiments, carefully done
    - albeit on relatively small UCI datasets
  - Suggests one-vs-all works just as well in practice
    - (or at least, the advantages claimed by earlier papers for multiclass methods were not compelling)
- Compared
  - many multiclass frameworks (including the one we discuss)
  - one-vs-all for SVMs with RBF kernel
  - one-vs-all for square loss with RBF kernel (for classification!)
- All performed roughly the same

Why Are We Bothering with Multiclass?

- The framework we have developed for multiclass
  - compatibility features / scoring functions
  - multiclass margin
  - target margin / multiclass loss
- Generalizes to situations where k is very large and one-vs-all is intractable.
- Key idea is that we can generalize across outputs y by using features of y.

Introduction to Structured Prediction

Example: Part-of-speech (POS) Tagging

- Given a sentence, give a part of speech tag for each word:

x	$\underbrace{\text{[START]}}_{x_0}$	$\underbrace{\text{He}}_{x_1}$	$\underbrace{\text{eats}}_{x_2}$	$\underbrace{\text{apples}}_{x_3}$
y	$\underbrace{\text{[START]}}_{y_0}$	$\underbrace{\text{Pronoun}}_{y_1}$	$\underbrace{\text{Verb}}_{y_2}$	$\underbrace{\text{Noun}}_{y_3}$

- $\mathcal{V} = \{\text{all English words}\} \cup \{\text{[START]}, ''.''\}$
- $\mathcal{X} = \mathcal{V}^n$, $n = 1, 2, 3, \ldots$ [Word sequences of any length]
- $\mathcal{P} = \{\text{START}, \text{Pronoun}, \text{Verb}, \text{Noun}, \text{Adjective}\}$
- $\mathcal{Y} = \mathcal{P}^n$, $n = 1, 2, 3, \ldots$[Part of speech sequence of any length]

Multiclass Hypothesis Space

- Discrete output space: $\mathcal{Y}(x)$
    - Very large but has structure, e.g., linear chain (sequence labeling), tree (parsing)
    - Size depends on input x
- Base Hypothesis Space: $\mathcal{H} = \{h : \mathcal{X} \times \mathcal{Y} \to \mathrm{R}\}$
    - $h(x,y)$ gives compatibility score between input x and output y
- Multiclass hypothesis space

$$\mathcal{F} = \left\{ x \mapsto \arg\max_{y \in \mathcal{Y}} h(x,y) \mid h \in \mathcal{H} \right\}$$

- Final prediction function is an $f \in \mathcal{F}$.
- For each $f \in \mathcal{F}$ there is an underlying compatibility score function $h \in \mathcal{H}$.

Structured Prediction

- Part-of-speech tagging

x :	he	eats	apples
y :	pronoun	verb	noun

- Multiclass hypothesis space:

$$h(x,y) = w^T \Psi(x,y) \tag{14}$$

$$\mathcal{F} = \left\{ x \mapsto \operatorname*{arg\,max}_{y \in \mathcal{Y}} h(x,y) \mid h \in \mathcal{H} \right\} \tag{15}$$

- A special case of multiclass classification
- How to design the feature map Ψ? What are the considerations?

Unary features

- A **unary feature** only depends on
  - the label at a single position, y_i, and x
- Example:

$$\begin{aligned}\phi_1(x, y_i) &= \mathbb{1}[x_i = \text{runs}]\mathbb{1}[y_i = \text{Verb}]\\ \phi_2(x, y_i) &= \mathbb{1}[x_i = \text{runs}]\mathbb{1}[y_i = \text{Noun}]\\ \phi_3(x, y_i) &= \mathbb{1}[x_{i-1} = \text{He}]\mathbb{1}[x_i = \text{runs}]\mathbb{1}[y_i = \text{Verb}]\end{aligned}$$

Markov features

- A **markov feature** only depends on
  - two adjacent labels, y_{i-1} and y_i, and x
- Example:

$$\begin{aligned}\theta_1(x, y_{i-1}, y_i) &= \mathbb{1}[y_{i-1} = \text{Pronoun}]\mathbb{1}[y_i = \text{Verb}] \\ \theta_2(x, y_{i-1}, y_i) &= \mathbb{1}[y_{i-1} = \text{Pronoun}]\mathbb{1}[y_i = \text{Noun}]\end{aligned}$$

- Reminiscent of Markov models in the output space
- Possible to have higher-order features

Local Feature Vector and Compatibility Score

- At each position i in sequence, define the **local feature vector** (unary and markov):

$$\begin{aligned}\Psi_i(x, y_{i-1}, y_i) &= (\phi_1(x, y_i), \phi_2(x, y_i), \ldots, \\ &\quad \theta_1(x, y_{i-1}, y_i), \theta_2(x, y_{i-1}, y_i), \ldots)\end{aligned}$$

- And **local compatibility score** at position i: $\langle w, \Psi_i(x, y_{i-1}, y_i)\rangle$.

- The compatibility score for (x, y) is the sum of local compatibility scores:

$$\sum_i \langle w, \Psi_i(x, y_{i-1}, y_i)\rangle = \left\langle w, \sum_i \Psi_i(x, y_{i-1}, y_i)\right\rangle = \langle w, \Psi(x, y)\rangle, \tag{16}$$

where we define the **sequence feature vector** by

$$\Psi(x, y) = \sum_i \Psi_i(x, y_{i-1}, y_i). \qquad \text{decomposable}$$

Structured perceptron

```
Given a dataset $\mathcal{D} = \{(x, y)\}$;
Initialize $w \leftarrow 0$;
for $iter = 1, 2, \ldots, T$ do
    for $(x, y) \in \mathcal{D}$ do
        $\hat{y} = \arg\max_{y' \in \mathcal{Y}(x)} w^T \psi(x, y')$;
        if $\hat{y} \neq y$ then // We've made a mistake
            $w \leftarrow w + \Psi(x, y)$ ; // Move the scorer towards $\psi(x, y)$
            $w \leftarrow w - \Psi(x, \hat{y})$ ; // Move the scorer away from $\psi(x, \hat{y})$
        end
    end
end
```

Identical to the multiclass perceptron algorithm except the $\arg\max$ is now over the structured output space $\mathcal{Y}(x)$.

Structured hinge loss

- Recall the generalized hinge loss

$$\ell_{\text{hinge}}(y, \hat{y}) \overset{\text{def}}{=} \max_{y' \in \mathcal{Y}(x)} \left(\Delta(y, y') + \left\langle w, \left(\Psi(x, y') - \Psi(x, y) \right) \right\rangle \right) \tag{17}$$

- What is $\Delta(y, y')$ for two sequences?
- **Hamming loss** is common:

$$\Delta(y, y') = \frac{1}{L} \sum_{i=1}^{L} \mathbb{1}[y_i \neq y_i']$$

where L is the sequence length.

Structured SVM

Exercise:

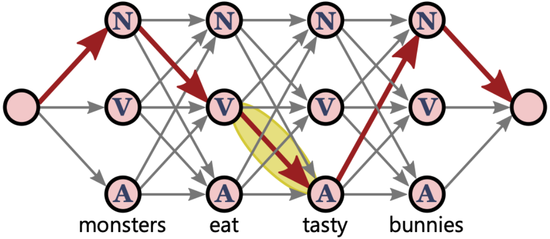
- Write down the objective of structured SVM using the structured hinge loss.
- Stochastic sub-gradient descent for structured SVM (similar to HW3 P3)
- Compare with the structured perceptron algorithm

The argmax problem for sequences

Problem To compute predictions, we need to find $\arg\max_{y \in \mathcal{Y}(x)} \langle w, \Psi(x,y) \rangle$, and $|\mathcal{Y}(x)|$ is exponentially large.

Observation $\Psi(x,y)$ decomposes to $\sum_i \Psi_i(x,y)$.

Solution Dynamic programming (similar to the Viterbi algorithm)



What's the running time?

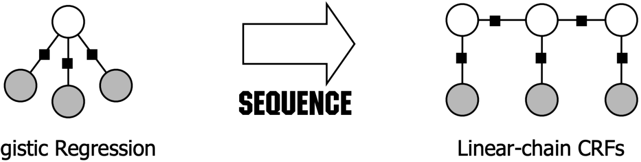
Figure by Daumé III. A course in machine learning. Figure 17.1.

Conditional random field (CRF)

- Recall that we can write logistic regression in a general form:

$$p(y|x) = \frac{1}{Z(x)} \exp(w^\top \psi(x,y)).$$

- Z is normalization constant: $Z(x) = \sum_{y \in Y} \exp(w^\top \psi(x,y))$.
- Example: linear chain $\{y_t\}$
- We can incorporate unary and Markov features: $p(y|x) = \frac{1}{Z(x)} \exp(\sum_t w^\top \psi(x, y_t, y_{t-1}))$



Conditional random field (CRF)

- Compared to Structured SVM, CRF has a probabilistic interpretation.
- We can draw samples in the output space.
- How do we learn w? Maximum log likelihood, and regularization term: $\lambda \|w\|^2$
- Loss function:

$$
\begin{aligned}
l(w) &= -\frac{1}{N}\sum_{i=1}^{N} \log p(y^{(i)} | x^{(i)}) + \frac{1}{2}\lambda \|w\|^2 \\
&= -\frac{1}{N}\sum_{i}\sum_{t}\sum_{k} w_k \psi_k(y_t^{(i)}, y_{t-1}^{(i)}) + \frac{1}{N}\sum_{i} \log Z(x^{(i)}) + \frac{1}{2}\sum_{k} \lambda w_k^2
\end{aligned}
$$

Conditional random field (CRF)

- Loss function:

$$l(w) = -\frac{1}{N}\sum_i \sum_t \sum_k w_k \psi_k(x^{(i)}, y_t^{(i)}, y_{t-1}^{(i)}) + \frac{1}{N}\sum_i \log Z(x^{(i)}) + \frac{1}{2}\sum_k \lambda w_k^2$$

- Gradient:

$$\frac{\partial l(w)}{\partial w_k} = -\frac{1}{N}\sum_i \sum_t \sum_k \psi_k(x^{(i)}, y_t^{(i)}, y_{t-1}^{(i)}) \quad (18)$$

$$+\frac{1}{N}\sum_i \frac{\partial}{\partial w_k} \log \sum_{y' \in Y} \exp(\sum_t \sum_{k'} w_{k'} \psi_{k'}(x^{(i)}, y_t', y_{t-1}')) + \sum_k \lambda w_k \quad (19)$$

Conditional random field (CRF)

- What is $\frac{1}{N}\sum_i \sum_t \sum_k \psi_k(x^{(i)}, y_t^{(i)}, y_{t-1}^{(i)})$?
- It is the expectation $\psi_k(x^{(i)}, y_t, y_{t-1})$ under the empirical distribution $\tilde{p}(x,y) = \frac{1}{N}\sum_i \mathbb{1}[x = x^{(i)}]\mathbb{1}[y = y^{(i)}]$.

Conditional random field (CRF)

- What is $\frac{1}{N}\sum_i \frac{\partial}{\partial w_k} \log \sum_{y' \in Y} \exp(\sum_t \sum_{k'} w_{k'} \psi_{k'}(x^{(i)}, y'_t, y'_{t-1}))$?

$$\frac{1}{N}\sum_i \frac{\partial}{\partial w_k} \log \sum_{y' \in Y} \exp(\sum_t \sum_{k'} w_{k'} \psi_{k'}(x^{(i)}, y'_t, y'_{t-1})) \tag{20}$$

$$= \frac{1}{N}\sum_i \left[\sum_{y' \in Y} \exp(\sum_t \sum_{k'} w_{k'} \psi_{k'}(x^{(i)}, y'_t, y'_{t-1})) \right]^{-1} \tag{21}$$

$$\left[\sum_{y' \in Y} \exp(\sum_t \sum_{k'} w_{k'} \psi_{k'}(x^{(i)}, y_t^{(i)}, y_{t-1}^{(i)})) \sum_t \psi_k(x^{(i)}, y'_t, y'_{t-1}) \right] \tag{22}$$

$$= \frac{1}{N}\sum_i \sum_t \sum_{y' \in Y} p(y'_t, y'_{t-1}|x) \psi_k(x^{(i)}, y'_t, y'_{t-1}) \tag{23}$$

- It is the expectation of $\psi_k(x^{(i)}, y'_t, y'_{t-1})$ under the model distribution $p(y'_t, y'_{t-1}|x)$.

Conditional random field (CRF)

- To compute the gradient, we need to infer expectation under the model distribution $p(y|x)$.
- Compare the learning algorithms: in structured SVM we need to compute the argmax, whereas in CRF we need to compute the model expectation.
- Both problems are NP-hard for general graphs.

CRF Inference

- In the linear chain structure, we can use the forward-backward algorithm for inference, similar to Viterbi.
- Initiate $\alpha_j(1) = \exp(w^\top \psi(y_1 = j, x_1))$
- Recursion: $\alpha_j(t) = \sum_i \alpha_i(t-1) \exp(w^\top \psi(y_t = j, y_{t-1} = i, x_t))$
- Result: $Z(x) = \sum_j \alpha_j(T)$
- Similar for the backward direction.
- Test time, again use Viterbi algorithm to infer argmax.
- The inference algorithm can be generalized to belief propagation (BP) in a tree structure (exact inference).
- In general graphs, we rely on approximate inference (e.g. loopy belief propagation).

Examples

- POS tag Relationship between constituents, e.g. NP is likely to be followed by a VP.
- Semantic segmentation
  Relationship between pixels, e.g. a grass pixel is likely to be next to another grass pixel, and a sky pixel is likely to be above a grass pixel.
- Multi-label learning
  An image may contain multiple class labels, e.g. a bus is likely to co-occur with a car.

Conclusion

Multiclass algorithms

- Reduce to binary classification, e.g., OvA, AvA
  - Good enough for simple multiclass problems
  - They don't scale and have simplified assumptions
- Generalize binary classification algorithms using multiclass loss
  - Multi-class perceptron, multi-class logistics regression, multi-class SVM
- Structured prediction: Structured SVM, CRF. Data containing structure. Extremely large output space. Text and image applications.

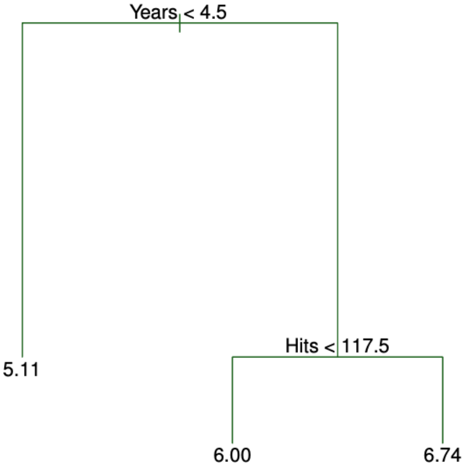
Decision Trees

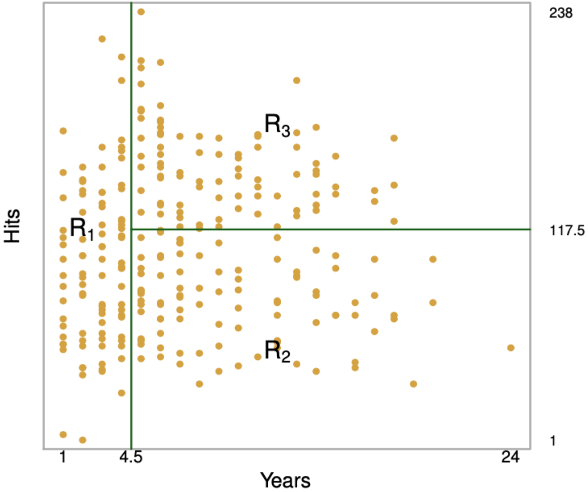
Overview: Decision Trees

- Our first inherently non-linear classifier: decision trees.
- Ensemble methods: bagging and boosting.

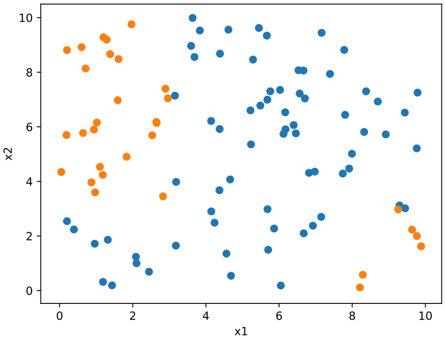
Decision Trees

Regression trees: Predicting basketball players' salaries



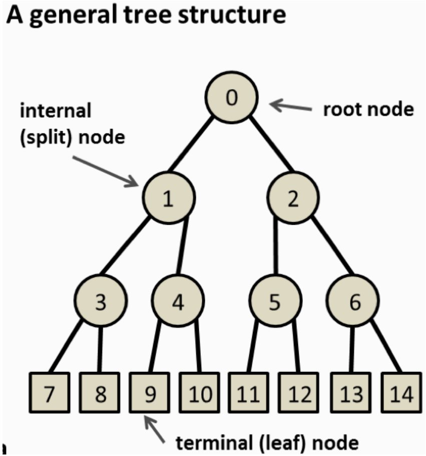


Classification trees



- Can we classify these points using a linear classifier?
- Partition the data into axis-aligned regions recursively (on the board)

Decision trees setup



- We focus on *binary* trees (as opposed to multiway trees where nodes can have more than two children)
- Each node contains a subset of data points
- The data splits created by each node involve only a *single* feature
- For continuous variables, the splits are always of the form $x_i \leqslant t$
- For discrete variables, we partition values into two sets (not covered today)
- Predictions are made in terminal nodes

From Criminisi et al. MSR-TR-2011-114, 28 October 2011.

Constructing the tree

Goal Find boxes $R_1, \ldots, R_J$ that minimize $\sum_{j=1}^{J} \sum_{i \in R_j} (y_i - \hat{y}_{R_j})^2$, subject to complexity constraints.

Problem Finding the optimal binary tree is computationally intractable.

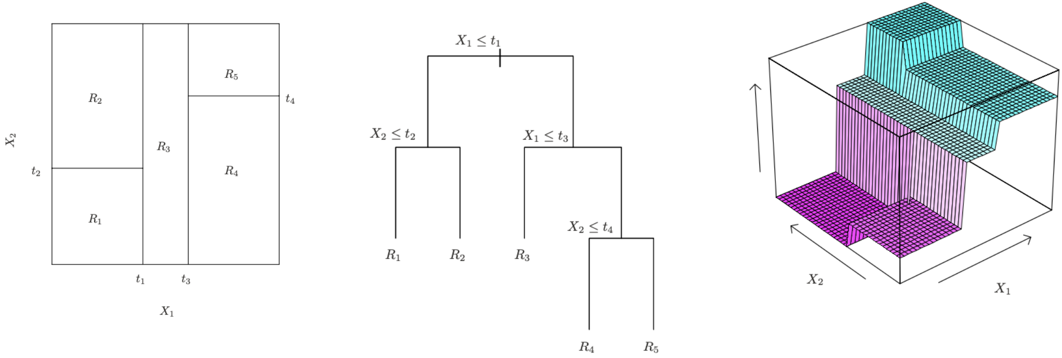
Solution Greedy algorithm: starting from the root, and repeating until a stopping criterion is reached (e.g., max depth), find the non-terminal node that results in the "best" split

- We only split regions defined by previous non-terminal nodes

Prediction Our prediction is the mean value of a terminal node: $\hat{y}_{R_m} = \text{mean}(y_i \mid x_i \in R_m)$

- A greedy algorithm is the one that make the best **local** decisions, without lookahead to evaluate their downstream consequences
- This procedure is not very likely to result in the globally optimal tree

Prediction in a Regression Tree



Finding the Best Split Point

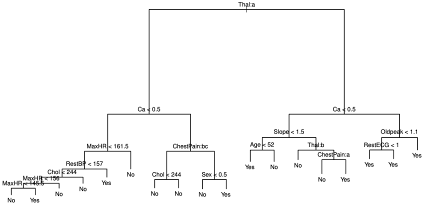
- We enumerate all features and all possible split points for each feature. There are infinitely many split points, but...
- Suppose we are now considering splitting on the j-th feature x_j, and let $x_{j(1)},\ldots,x_{j(n)}$ be the sorted values of the j-th feature.
- We only need to consider split points between two adjacent values, and any split point in the interval $(x_{j(r)}, x_{(j(r+1))})$ will result in the same loss
- It is common to split half way between two adjacent values:

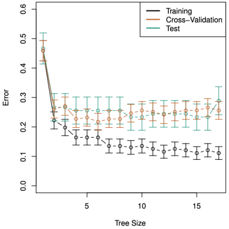
$$s_j \in \left\{ \frac{1}{2}\left(x_{j(r)} + x_{j(r+1)}\right) \mid r = 1,\ldots,n-1 \right\}. \qquad n-1 \text{ splits} \qquad (24)$$

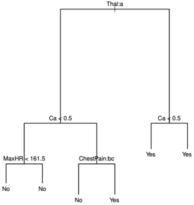
Decision Trees and Overfitting

- What will happen if we keep splitting the data into more and more regions?
  - Every data point will be in its own region—overfitting.
- When should we stop splitting? (Controlling the complexity of the hypothesis space)
  - Limit total number of nodes.
  - Limit number of terminal nodes.
  - Limit tree depth.
  - Require minimum number of data points in a terminal node.
  - **Backward pruning** (the approach used in **CART**; Breiman et al 1984):
    1. Build a really big tree (e.g. until all regions have $\leqslant 5$ points).
    2. *Prune* the tree back greedily, potentially all the way to the root, until validation performance starts decreasing.

Pruning: Example







What Makes a Good Split for Classification?

Our plan is to predict the **majority label** in each region.

Which of the following splits is better?

Split 1	$R_1 : 8+/2-$	$R_2 : 2+/8-$
Split 2	$R_1 : 6+/4-$	$R_2 : 4+/6-$

How about here?

Split 1	$R_1 : 8+/2-$	$R_2 : 2+/8-$
Split 2	$R_1 : 6+/4-$	$R_2 : 0+/10-$

Intuition: we want to produce *pure* nodes, i.e. nodes where most instances have the same class.

Misclassification error in a node

- Let's consider the multiclass classification case: $\mathcal{Y} = \{1, 2, \ldots, K\}$.
- Let node m represent region R_m, with N_m observations
- We denote the proportion of observations in R_m with class k by

$$\hat{p}_{mk} = \frac{1}{N_m} \sum_{\{i : x_i \in R_m\}} \mathbb{1}[y_i = k].$$

- We predict the majority class in node m:

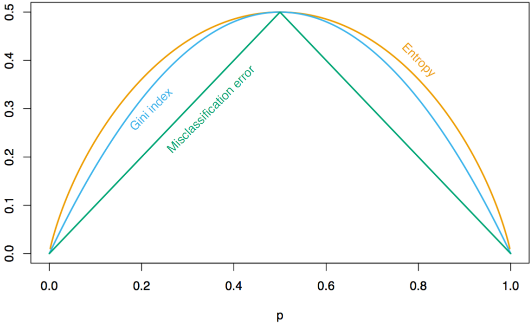
$$k(m) = \arg\max_k \hat{p}_{mk}$$

Node Impurity Measures

- Three measures of **node impurity** for leaf node m:
  - Misclassification error
    $$1 - \hat{p}_{mk(m)}.$$
  - The Gini index encourages $\hat{p}_{mk}$ to be close to 0 or 1
    $$\sum_{k=1}^{K} \hat{p}_{mk}(1 - \hat{p}_{mk}).$$
  - Entropy / Information gain
    $$-\sum_{k=1}^{K} \hat{p}_{mk} \log \hat{p}_{mk}.$$
- The Gini index and entropy are numerically similar to each other, and both work better in practice than the misclassification error.

Impurity Measures for Binary Classification

(p is the relative frequency of class 1)



HTF Figure 9.3

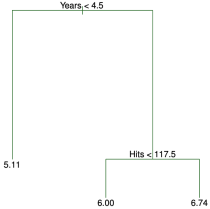
Quantifying the Impurity of a Split

Scoring a potential split that produces the nodes R_L and R_R:

- Suppose we have N_L points in R_L and N_R points in R_R.
- Let $Q(R_L)$ and $Q(R_R)$ be the node impurity measures for each node.
- We aim to find a split that minimizes the *weighted average of node impurities*:

$$\frac{N_L Q(R_L) + N_R Q(R_R)}{N_L + N_R}$$

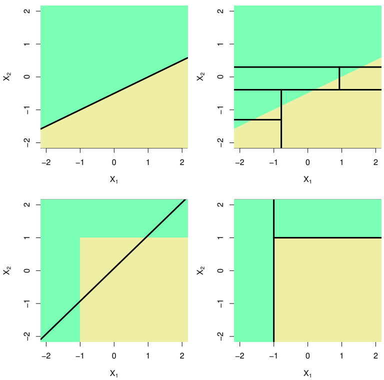
Discussion: Interpretability of Decision Trees



- Trees are easier to visualize and explain than other classifiers (even linear regression)
- Small trees are interpretable – large trees, maybe not so much

Discussion: Trees vs. Linear Models

Trees may have to work hard to capture linear decision boundaries, but can easily capture certain nonlinear ones:



Discussion: Review

Decision trees are:

- Non-linear: the decision boundary that results from splitting may end up being quite complicated
- Non-metric: they do not rely on the geometry of the space (inner products or distances)
- Non-parametric: they make no assumptions about the distribution of the data

Additional pros:

- Interpretable and simple to understand

Cons:

- Struggle to capture linear decision boundaries
- They have high variance and tend to overfit: they are sensitive to small changes in the training data (The ensemble techniques we discuss next can mitigate these issues)

Bagging and Random Forests

Recap: Statistics and Point Estimators

- We observe data $\mathcal{D} = (x_1, x_2, \ldots, x_n)$ sampled i.i.d. from a parametric distribution $p(\cdot \mid \theta)$
- A **statistic** $s = s(\mathcal{D})$ is any function of the data:
    - E.g., sample mean, sample variance, histogram, empirical data distribution
- A statistic $\hat{\theta} = \hat{\theta}(\mathcal{D})$ is a **point estimator** of θ if $\hat{\theta} \approx \theta$

Recap: Bias and Variance of an Estimator

- Statistics are random, so they have probability distributions.
- The distribution of a statistic is called a **sampling distribution**.
- The standard deviation of the sampling distribution is called the **standard error**.
- Some parameters of the sampling distribution we might be interested in:

$$\begin{aligned} \text{Bias} \quad \text{Bias}(\hat{\theta}) &\stackrel{\text{def}}{=} \mathbb{E}\left[\hat{\theta}\right] - \theta. \\ \text{Variance} \quad \text{Var}(\hat{\theta}) &\stackrel{\text{def}}{=} \mathbb{E}\left[\hat{\theta}^2\right] - \mathbb{E}^2\left[\hat{\theta}\right]. \end{aligned}$$

- Why does variance matter if an estimator is unbiased?
  - $\hat{\theta}(\mathcal{D}) = x_1$ is an unbiased estimator of the mean of a Gaussian, but would be farther away from θ than the sample mean.

Variance of a Mean

- Let $\hat{\theta}(\mathcal{D})$ be an unbiased estimator with variance σ^2: $\mathbb{E}\left[\hat{\theta}\right] = \theta$, $\text{Var}(\hat{\theta}) = \sigma^2$.
- So far we have used a single statistic $\hat{\theta} = \hat{\theta}(\mathcal{D})$ to estimate θ.
- Its standard error is $\sqrt{\text{Var}(\hat{\theta})} = \sigma$
- Consider a new estimator that takes the average of i.i.d. $\hat{\theta}_1, \ldots, \hat{\theta}_n$ where $\hat{\theta}_i = \hat{\theta}(\mathcal{D}^i)$.
- The average has the same expected value but smaller standard error (recall that $Var(cX) = c^2 Var(X)$, and that the $\hat{\theta}_i$-s are uncorrelated):

$$\mathbb{E}\left[\frac{1}{n}\sum_{i=1}^{n}\hat{\theta}_i\right] = \theta \qquad \text{Var}\left[\frac{1}{n}\sum_{i=1}^{n}\hat{\theta}_i\right] = \frac{\sigma^2}{n} \tag{25}$$

Averaging Independent Prediction Functions

- Suppose we have B independent training sets, all drawn from the same distribution $(\mathcal{D} \sim p(\cdot \mid \theta))$.
- Our learning algorithm gives us B prediction functions: $\hat{f}_1(x), \hat{f}_2(x), \ldots, \hat{f}_B(x)$
- We will define the average prediction function as:

$$\hat{f}_{\text{avg}} \overset{\text{def}}{=} \frac{1}{B} \sum_{b=1}^{B} \hat{f}_b \tag{26}$$

Averaging Reduces Variance of Predictions

- The average prediction for x_0 is

$$\hat{f}_{\text{avg}}(x_0) = \frac{1}{B}\sum_{b=1}^{B}\hat{f}_b(x_0).$$

- $\hat{f}_{\text{avg}}(x_0)$ and $\hat{f}_b(x_0)$ have the same expected value, but
- $\hat{f}_{\text{avg}}(x_0)$ has smaller variance:

$$\mathrm{Var}(\hat{f}_{\text{avg}}(x_0)) = \frac{1}{B}\mathrm{Var}\left(\hat{f}_1(x_0)\right)$$

- Problem: in practice we don't have B independent training sets!

The Bootstrap Sample

How do we simulate multiple samples when we only have one?

- A **bootstrap sample** from $\mathcal{D}_n = (x_1, \ldots, x_n)$ is a sample of size n drawn *with replacement* from $\mathcal{D}_n$
- Some elements of $\mathcal{D}_n$ will show up multiple times, and some won't show up at all
- Each x_i has a probability of $(1 - 1/n)^n$ of not being included in a given bootstrap sample
- For large n,

$$\left(1 - \frac{1}{n}\right)^n \approx \frac{1}{e} \approx .368. \tag{27}$$

- So we expect ~63.2% of elements of $\mathcal{D}_n$ will show up at least once.

The Bootstrap Method

Definition

A **bootstrap method** simulates B independent samples from P by taking B bootstrap samples from the sample $\mathcal{D}_n$.

- Given original data $\mathcal{D}_n$, compute B bootstrap samples $D_n^1, \dots, D_n^B$.
- For each bootstrap sample, compute some function

$$\phi(D_n^1), \dots, \phi(D_n^B)$$

- Use these values as though $D_n^1, \dots, D_n^B$ were i.i.d. samples from P.
- This often ends up being very close to what we'd get with independent samples from P!

Independent Samples vs. Bootstrap Samples

- Point estimator $\hat{\alpha} = \hat{\alpha}(\mathcal{D}_{100})$ for samples of size 100, for a synthetic case where the data generating distribution is known

- Histograms of $\hat{\alpha}$ based on
  - 1000 independent samples of size 100 (left), vs.
  - 1000 bootstrap samples of size 100 (right)

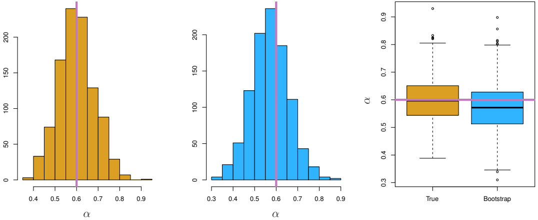


Figure 5.10 from *ISLR* (Springer, 2013) with permission from the authors: G. James, D. Witten, T. Hastie and R. Tibshirani.

Ensemble Methods

Key ideas:

- In general, **ensemble methods** combine multiple weak models into a single, more powerful model
- Averaging i.i.d. estimates reduces variance without changing bias
- We can use bootstrap to simulate multiple data samples and average them
- Parallel ensemble (e.g., bagging): models are built independently
- Sequential ensemble (e.g., boosting): models are built sequentially
    - We try to find new learners that do well where previous learners fall short

Bagging: Bootstrap Aggregation

- We draw B bootstrap samples $D^1, \ldots, D^B$ from original data $\mathcal{D}$
- Let $\hat{f}_1, \hat{f}_2, \ldots, \hat{f}_B$ be the prediction functions resulting from training on $D^1, \ldots, D^B$, respectively
- The **bagged prediction function** is a *combination* of these:

$$\hat{f}_{\text{avg}}(x) = \text{Combine}\left(\hat{f}_1(x), \hat{f}_2(x), \ldots, \hat{f}_B(x)\right)$$

Bagging: Bootstrap Aggregation

- Bagging is a general method for variance reduction, but it is particularly useful for decision trees
- For classification, averaging doesn't make sense; we can take a **majority vote** instead
- Increasing the number of trees we use in bagging does not lead to overfitting
- Is there a downside, compared to having a single decision tree?
- Yes: if we have many trees, the bagged predictor is much less interpretable

Aside: Out-of-Bag Error Estimation

- Recall that each bagged predictor was trained on about 63% of the data.
- The remaining 37% are called **out-of-bag (OOB)** observations.
- For ith training point, let

$$S_i = \left\{ b \mid D^b \text{ does not contain } i\text{th point} \right\}$$

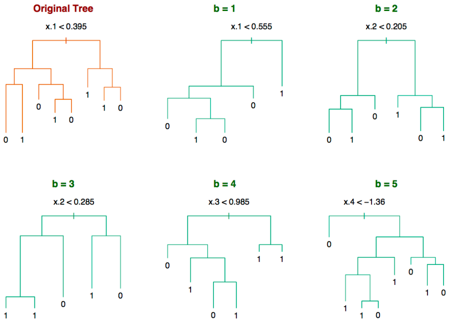
- The **OOB prediction** on x_i is

$$\hat{f}_{\text{OOB}}(x_i) = \frac{1}{|S_i|} \sum_{b \in S_i} \hat{f}_b(x_i)$$

- The OOB error is a good estimate of the test error
- Similar to cross validation error: both are computed on the training set

Applying Bagging to Classification Trees

- Input space $\mathcal{X} = \mathrm{R}^5$ and output space $\mathcal{Y} = \{-1, 1\}$. Sample size $n = 30$.



- Each bootstrap tree is quite different: different splitting variable at the root!
- **High variance**: small perturbations of the training data lead to a high degree of model variability
- Bagging helps most when the base learners are relatively unbiased but have high variance (exactly the case for decision trees)

From HTF Figure 8.9

Motivating Random Forests: Correlated Prediction Functions

Recall the motivating principle of bagging:

- For $\hat{\theta}_1, \ldots, \hat{\theta}_n$ *i.i.d.* with $\mathbb{E}\left[\hat{\theta}\right] = \theta$ and $\operatorname{Var}\left[\hat{\theta}\right] = \sigma^2$,

$$\mathbb{E}\left[\frac{1}{n}\sum_{i=1}^{n}\hat{\theta}_i\right] = \mu \qquad \operatorname{Var}\left[\frac{1}{n}\sum_{i=1}^{n}\hat{\theta}_i\right] = \frac{\sigma^2}{n}.$$

- What if $\hat{\theta}$'s are correlated?
- For large n, the covariance term dominates, limiting the benefits of averaging
- Bootstrap samples are
  - independent samples from the training set, but
  - not independent samples from $P_{\mathcal{X}\times\mathcal{Y}}$
- Can we reduce the dependence between $\hat{f}_i$'s?

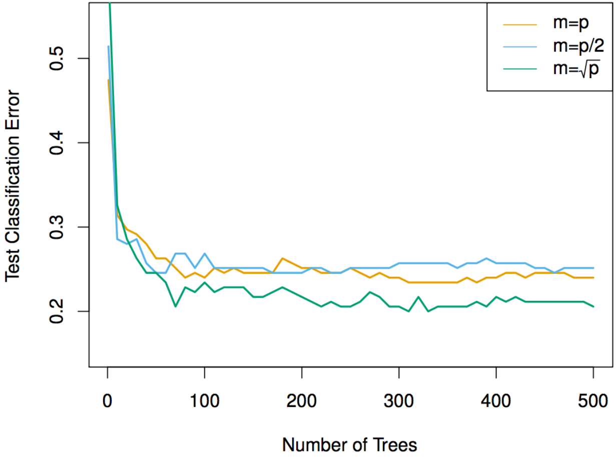
Random Forests

Key idea

Use bagged decision trees, but modify the tree-growing procedure to reduce the dependence between trees.

- Build a collection of trees independently (in parallel), as before
- When constructing each tree node, restrict choice of splitting variable to a randomly chosen subset of features of size m
  - This prevents a situation where all trees are dominated by the same small number of strong features (and are therefore too similar to each other)
- We typically choose $m \approx \sqrt{p}$, where p is the number of features (or we can choose m using cross validation)
- If $m = p$, this is just bagging

Random Forests: Effect of *m*



From *An Introduction to Statistical Learning, with applications in R* (Springer, 2013) with permission from the authors: G. James, D. Witten, T. Hastie and R. Tibshirani.

Review

- The usual approach is to build very deep trees—low bias but high variance
- Ensembling many models reduces variance
  - Motivation: Mean of i.i.d. estimates has smaller variance than single estimate
- Use bootstrap to simulate many data samples from one dataset
  - $\implies$ Bagged decision trees
- But bootstrap samples (and the induced models) are correlated
- Ensembling works better when we combine a diverse set of prediction functions
  - $\implies$ Random forests: select a random subset of features for each decision tree