

1 Intro

We will continue to broaden the class of models that we can fit to our data. Graphical models allowed us to fit joint distributions that are compact to represent, and efficient for learning and inference. The complexity was adaptable, in the sense that structure search could try different dependence models and select one that was expected to be a good model of future data.

We now turn to models that are not directly interpretable as fitting a fixed-dimension distribution to data. The name *non-parametric methods* is misleading: it is really a class of methods that does not have a fixed parameterization in advance. Some non-parametric models, such as trees and boosting, which we might call *semi-parametric methods*, can be seen as dynamically constructing something that ends up looking like a more traditional parametric model, but where the actual training data affects exactly what the form of the model will be. Other non-parametric methods, such as nearest-neighbor and Bayesian non-parametric methods, rely directly on the data to make predictions and do not compute a model that summarizes the data.

The semi-parametric methods tend to have the form of a composition of simple models. We'll look at:

- *Tree models*: partition the input space and use different simple predictions on different regions of the space; this increases the hypothesis space.
- *Additive models*: train several different classifiers on the whole space and average the answers; this decreases the variance.

Boosting is a way to construct an additive model that both increases hypothesis space and decreases variance.

2 Trees

The idea here is that we would like to find a partition of the input space and then fit very simple models to predict the output in each piece. The partition is described using a (typically binary) "decision tree," which recursively splits the space.

These methods differ by:

- The class of possible ways to split the space at each node; these are generally linear splits, either aligned with the axes of the space, or more general.
- The class of predictors within the partitions; these are often simply constants, but may be probability distribution or more general classification or regression models.
- The way in which we control the complexity of the hypothesis: it would be within the capacity of these methods to have a separate partition for each individual training example.
- The algorithm for making the partitions and fitting the models.

The primary advantage of tree models is that they are understandable by humans. This is important in application domains, such as medicine, where there are human experts who think they know what they're doing and where decisions are critically important.

We'll concentrate on the CART/ID3 family of algorithms, which were invented independently in the statistics and the artificial intelligence communities. They work by greedily constructing a partition, where the splits are *axis aligned* and by fitting a *constant* model in the leaves. The interesting questions are how to select the splits and how to control capacity. The regression and classification versions are very similar.

2.1 Regression

	No model	Prediction rule	Prob model	Dist over models
Regression		*		

The classifier is made up of

- A partition function, π , mapping elements of the input space into exactly one of M regions, R_1, \dots, R_M .
- A collection of M output values, O_m , one for each region.

If we already knew a division of the space into regions, we would set \hat{y}_m , the constant output for region R_m to be the average of the output values in that region; that is:

$$O_m = \text{average}_{\{i | x^{(i)} \in R_m\}} y^{(i)} .$$

Define the error in a region as

$$E_m = \sum_{\{i | x^{(i)} \in R_m\}} (y^{(i)} - O_m)^2 .$$

Ideally, we would select the partition to minimize

$$cM + \sum_{m=1}^M E_m ,$$

for some regularization constant c . It is enough to search over all partitions of the training data (not all partitions of the input space) to optimize this, but the problem is NP-complete.

2.1.1 Building a tree

So, we'll be greedy. We establish a criterion, given a set of data, for finding the best single split of that data, and then apply it recursively to partition the space. We will select the partition of the data that *minimizes the sum of the mean squared errors of each partition*.

Given a data set D , let

- $R_{j,s}^+(D) = \{x \in D \mid x_j \geq s\}$
- $R_{j,s}^-(D) = \{x \in D \mid x_j < s\}$
- $\hat{y}_{j,s}^+ = \text{average}_{\{i | x^{(i)} \in R_{j,s}^+(D)\}} y^{(i)}$
- $\hat{y}_{j,s}^- = \text{average}_{\{i | x^{(i)} \in R_{j,s}^-(D)\}} y^{(i)}$

BuildTree(D):

- If $|D| \leq k$: return **Leaf**(D)
- Find the variable j and split point s that minimizes:

$$E_{R_{j,s}^+(D)} + E_{R_{j,s}^-(D)} .$$

- Return **Node**($j, s, \text{BuildTree}(R_{j,s}^+(D)), \text{BuildTree}(R_{j,s}^-(D))$)

Each call to **BuildTree** considers $O(dn)$ splits (only need to split between each data point in each dimension); each requires $O(n)$ work.

2.1.2 Pruning

It might be tempting to regularize by stopping for a somewhat large k , or by stopping when splitting a node does not significantly decrease the error. One problem with short-sighted stopping criteria is that they might not see the value of a split that is, essentially, two-dimensional. So, we will tend to build a tree that is much too large, and then prune it back.

Define *cost complexity* of a tree T , where m ranges over its leaves as

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

For a fixed α , find a T that (approximately) minimizes $C_\alpha(T)$ by “weakest-link” pruning. Create a sequence of trees by successively removing the bottom-level split that minimizes the increase in overall error, until the root is reached. Return the T in the sequence that minimizes the criterion.

Pick α using cross validation.

2.2 Classification

	No model	Prediction rule	Prob model	Dist over models
Classification		*		

The strategy for building and pruning classification trees is very similar to the one for regression trees.

The output is now the majority of the output values in the leaf:

$$O_m = \text{majority}_{\{i | x^{(i)} \in R_m\}} y^{(i)} .$$

Define the error in a region as the number of data points that do not have the value O_m :

$$E_m = \left| \{i \mid x^{(i)} \in R_m \text{ and } y^{(i)} \neq O_m\} \right| .$$

Define the *empirical probability* of an item from class k occurring in region m as:

$$\hat{p}_{mk} = \hat{p}(R_m)(k) = \frac{|\{i \mid x^{(i)} \in R_m \text{ and } y^{(i)} = k\}|}{N_m}$$

We’ll define the empirical probabilities of feature values, as well, for later use.

$$\hat{p}_{mjv} = \hat{p}(R_m)(v) = \frac{|\{i \mid x^{(i)} \in R_m \text{ and } x_j^{(i)} = v\}|}{N_m}$$

Splitting criteria Minimize “impurity” in child nodes. Some measures include:

- *Misclassification error*:

$$Q_m(T) = \frac{E_m}{N_m} = 1 - \hat{p}_{mO_m}$$

- *Gini index*:

$$Q_m(T) = \sum_k \hat{p}_{mk}(1 - \hat{p}_{mk})$$

- *Entropy:*

$$Q_m(T) = H(R_m) = - \sum_k \hat{P}_{mk} \log \hat{P}_{mk}$$

Choosing the split that minimizes the entropy of the children is equivalent to maximize the *information gain* of the test $X_j = v$, defined by

$$\text{infoGain}(X_j = v, R_m) = H(R_m) - (\hat{P}_{mjv} H(R_{j,v}^+) + (1 - \hat{P}_{mjv}) H(R_{j,v}^-))$$

Consider the two-class case. All have value

$$\begin{cases} 0.0 & \text{when } \hat{P}_{m0} = 0.0 \\ 0.5 & \text{when } \hat{P}_{m0} = 0.5 \\ 0.0 & \text{when } \hat{P}_{m0} = 1.0 \end{cases}$$

There used to be endless haggling about which one to use. It seems to be traditional to use:

- Entropy to select which node to split while growing the tree
- Misclassification error in the pruning criterion

Points about trees There are many variations on this theme:

- Linear regression or other regression or classification method in each leaf
- Non-axis-parallel splits: e.g., run a perceptron for a while to get a split.

What's good about trees:

- People understand them
- Easy to handle multi-class classification
- Easy to handle different loss functions (just change predictor in the leaves)

What's bad about trees:

- High variance: small changes in the data can result in very big changes in the hypothesis.
- Usually not the best predictions

Hierarchical mixture of experts Make a “soft” version of trees, in which the splits are probabilistic (so every point has some degree of membership in every leaf). Can be trained with a form of EM.

3 Bagging

Bootstrap aggregation is a technique for reducing the variance of a non-linear predictor, or one that is adaptive to the data.

- Construct B new data sets of size n by sampling them with replacement from \mathcal{D}
- Train a predictor on each one: \hat{f}^b

- *Regression case*: bagged predictor is

$$\hat{f}_{\text{bag}}(x) = \frac{1}{B} \sum_{b=1}^B \hat{f}^b(x)$$

- *Classification case*: majority bagged predictor: let $\hat{f}^b(x)$ be a vector with a single 1 and $K - 1$ zeros, so that $\hat{y}^b(x) = \arg \max_k \hat{f}^b(x)_k$. Then

$$\hat{f}_{\text{bag}}(x) = \frac{1}{B} \sum_{b=1}^B \hat{f}^b(x),$$

which is a vector containing the proportion of classifiers that predicted each class k for input x ; and the predicted output is

$$\hat{y}_{\text{bag}}(x) = \arg \max_k \hat{f}_{\text{bag}}(x)_k .$$

Alternatively, we can average the class probabilities from the individual classifier, which gives us an even lower-variance estimate.

In the case of regression and *squared error*, we can show that expected squared error of a classifier trained on a single data set of size n is bounded below by the squared error of a classifier that is an average over classifiers trained on infinitely many data sets of size n drawn from the population. This suggests that bagging (in which new training sets are drawn from the data set, not population) will also decrease error.

For classification under 0-1 loss, bagging can improve a good classifier, but it can also make a bad classifier worse. But, here's a way to understand its advantage:

- Let the Bayes optimal decision at x be $Y(x) = 1$ in a two-class example.
- Suppose each "committee member" Y_b has an error rate $e_b < e < 0.5$
- Let $S_1(x) = \sum_{b=1}^B I(G_b(x) = 1)$ be the number of votes for class 1 given input x
- If the committee members are independent, $S_1(x) \sim \text{Bin}(B, 1 - e)$ and so $\Pr(S_1 > B/2) \rightarrow 1$ as B gets large.

From Dietterich, via Hastie, Tibshirani, and Friedman.

This is the "Wisdom of Crowds."

The main issue with this analysis is that it assumes the committee members are independent, which they definitely are not in this case.

Also, when we bag a model, any simple predictability is lost.

3.1 Random Forests

Random forests are collections of trees that are constructed to be de-correlated, so that using them to vote gives maximal advantage.

For $b = 1..B$

- Draw a bootstrap sample \mathcal{D}_b of size n from \mathcal{D}
- Grow a tree on data \mathcal{D}_b by recursively repeating these steps:
 - Select m variables at random from the d variables
 - Pick the best variable and split point among them
 - Split the node
- return tree T_b

Given the ensemble of trees, vote to make a prediction on a new x .