Machine Learning Coms-4771

Decision Tree Learning
Lecture 2

January 28, 2008

Two Types of Supervised Learning Problems (recap)

Feature (input) space X, label (output) space Y. Unknown distribution D over $X \times Y$.

Classification: Given a set of labeled training examples $(x_1, y_1), \ldots, (x_n, y_n)$ from D, find a mapping $f : X \to Y$ minimizing

$$e_D(f) = \mathbf{Pr}_{(x,y)\sim D}[f(x) \neq y] \qquad \text{(zero-one loss)}$$

Regression: Same but find f minimizing

$$L_D(f) = \mathbf{E}_{(x,y)\sim D}(f(x) - y)^2 \qquad \text{(squared loss)}$$

or

$$\ell_D(f) = \mathbf{E}_{(x,y)\sim D}|f(x) - y| \qquad \text{(absolute loss)}$$

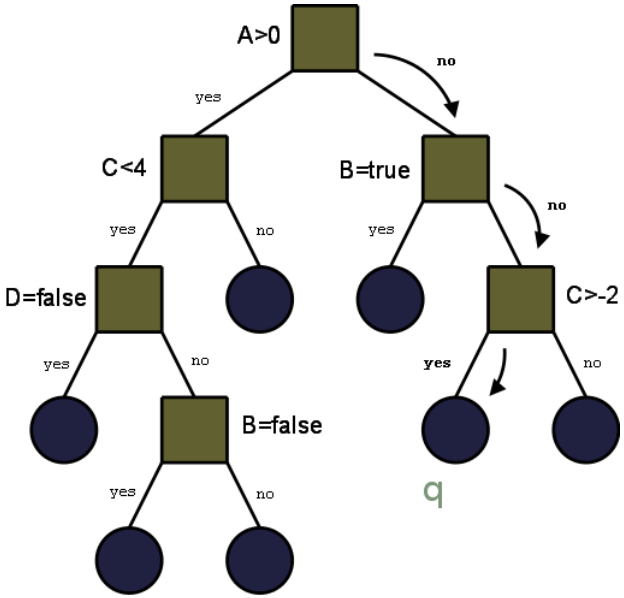
Note: If $Y = \{0, 1\}$ (binary classification), $e(f) = \ell(f)$. (If D is clear from context, we drop the subscript.)

Training error (also called empirical risk or empirical loss):

$$\widehat{\ell}(f) = \frac{1}{n}\sum_{i=1}^{n}|f(x_i) - y_i|, \quad \widehat{L}(f) = \frac{1}{n}\sum_{i=1}^{n}(f(x_i) - y_i)^2$$

The hat notation is used for empirical estimates based on limited samples.

Decision Tree Learning



Each node in T corresponds to a subset of X (defined by the set of constraints on the path from the root). In a similar way, T defines a partition of the training set.

Each internal node corresponds to a test typically involving a single feature (also called attribute).

- Learning = process of cutting up the input space X into a set of cells, and assigning (typically constant) predictions to each cell.
- Tree T = partition of X.
- Prediction on each cell q depends on the loss function:
  Zero-one loss: most frequent class

$$\hat{y}_q = \text{argmax}_k \sum_{(x_i,y_i)\in R_q} \mathbf{1}(y_i = k),$$

  where R_q is the set of training examples in q.
  $\mathbf{1}(\cdot)$ is the indicator function, which is 1 when its argument is 1 and 0 otherwise.
  Absolute loss: $\hat{y}_q$ = sample median of training data (sort the labels of examples in R_q and take the median label)
  Squared loss: sample mean

$$\hat{y}_q = \frac{1}{|R_q|} \sum_{(x_i,y_i)\in R_q} y_i.$$

- The final classifier is given by $\hat{f}_T(x) = \sum_q \mathbf{1}(x \in R_q) \cdot \hat{y}_q$

Emprical Risk Minimization

Natural goal: Find T (in some tree class $\mathcal{F}$, e.g., the set of all trees with at most a certain number of leaves) minimizing the empirical loss $\frac{1}{n}\sum_{i=1}^{n}|\hat{f}_T(x_i) - y_i|^\alpha$ over the training set ($\alpha = 1$ for absolute loss and $\alpha = 2$ for squared loss).

Two problems with this:

- We can still overfit the training set (due to noise in the data and training set not being large enough for $\mathcal{F}$). In the extreme case, each training example can end up is in its own cell: empirical loss $= 0$, but we haven't learned anything. (A bit more formally, a hypothesis $T \in \mathcal{F}$ is said to *overfit* the training set if there exists some $T' \in \mathcal{F}$ such that h has a smaller empirical loss than h', but h' has a smaller expected loss on the entire distribution of instances.)
- The optimization problem is computationally infeasible for general $\mathcal{F}$.

We can avoid both problems by constructing T **greedily, top down**:

- Choose a test A.
- Create a child for each outcome of A and sort training examples into subtrees according to the outcome.
- Recurse on each subtree (stop early if a stopping criteria is satisfied).
- Post-prune the tree.

Choosing tests: Entropy (as a measure of node impurity)

- Let R be the set of training examples reaching some cell (current leaf in the tree). Entropy of R = uncertainty still remaining about the class of instance x knowing that x is in the cell:

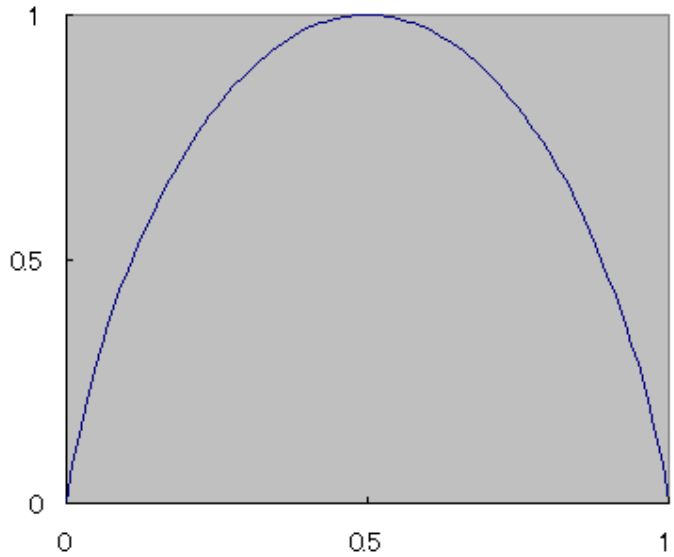
$$H(R) = -\sum_{k \in Y} D(k \mid R) \log_2 D(k \mid R),$$

  where $D(k \mid R)$ is the probability that a random example from D is in class k, conditioned on being in R. Since we don't know D, we estimate $D(k \mid R)$ as

$$\frac{1}{|R|} \sum_{(x_i, y_i) \in R} \mathbf{1}(y_i = k).$$

- Entropy is the expected number of bits needed to encode class of a randomly drawn member of R (under optimal, shortest length code which assigns $-\log_2 p$ bits to message with probability p).

Entropy and Information Gain



binary case (x axis = probability that the class is 1)

- $H(R)$ measures the "impurity" of R. If all examples in R belong to the same class, $H(R) = 0$.
- Sanity check: As we travel down the tree, the uncertainty should decrease.
- Imagine splitting R by performing a test A with K possible outcomes, resulting in respective subsets $R_1, \ldots, R_K$.
- If the outcome of A turned out to be j, the reduction of uncertainty about the class would be $H(R) - H(R_j)$.

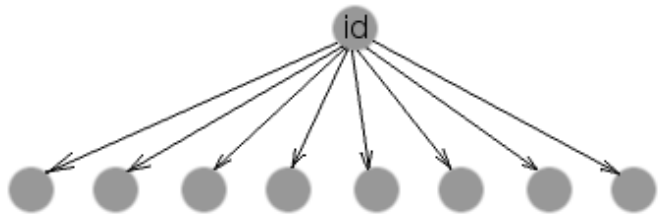
Since we don't know the outcome of A, we can only talk about the *expected* reduction of uncertainty due to performing A:

$$\text{Gain}(R, A) = H(R) - \frac{1}{K}\left[\sum_{j=1}^{K} p(R_j)H(R_j)\right],$$

where $p(R_j)$ is the probability, according to D, that the outcome of A (conditioned on being in R) is j, estimated using $|R_j|/|R|$.

Information Gain

- Choose test A to maximize Gain(R, A).
- Problem: Entropy is always decreased by having more classes, so Gain prefers attributes with many values (extreme cases: unique ids, no attribute can do better since the infogain is $H(R)$).



- Quinlan's C4.5 solution: Divide Gain(R, A) by a correction factor

$$-\sum_j p(R_j) \log p(R_j),$$

with $p(R_j)$ estimated as $|R_j|/|R|$ as before.

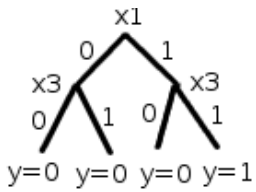
- Correction factor = "entropy" of A's partition of R: the higher the factor, the lower the normalized gain.

Example

Training set R:

(A_0, A_1, A_2, A_3)	label y
$(1, 0, 0, 0)$	0
$(2, 0, 0, 1)$	0
$(3, 0, 1, 0)$	0
$(4, 0, 1, 1)$	0
$(5, 1, 0, 0)$	0
$(6, 1, 0, 1)$	1
$(7, 1, 1, 0)$	0
$(8, 1, 1, 1)$	1

Resulting tree:



Which single test should be performed first?

- The initial uncertainty about y is

$$H(R) = \underbrace{-(6/8)\log_2(6/8)}_{\text{6 examples with label 0}} \quad \underbrace{-(2/8)\log_2(2/8)}_{\text{2 with label 1}} = 0.81$$

- Next, we calculate the reduction of uncertainty due to performing test A_1 first. The two outcomes, $A_1 = 0$ and $A_1 = 1$, result in the partition $R_i^{A_1} = \{(x, y) \in R : A_1 = i\}, i \in \{0, 1\}$ with

$H(R_0^{A_1}) = -1\log_2 1 - 0\log_2 0 = 0$, $H(R_1^{A_1}) = -(2/4)\log_2(2/4) - (2/4)\log_2(2/4) = 1$. Since $|R_0^{A_1}| = |R_1^{A_1}| = |R|/2$, the average uncertainty after performing A_1 is $0.5H_{A_1}(R_0^{A_1}) + 0.5H_{A_1}(R_1^{A_1}) = 0.5$. Thus $\text{Gain}(R, A_1) = 0.81 - 0.5 = 0.31$. Similar calculations show that $\text{Gain}(R, A_3) = \text{Gain}(R, A_1)$, but $\text{Gain}(R, A_2) = 0$. The three corresponding correction factors are all 1.

- Suppose that attribute A_0 can have 8 possible outcomes, and we are entertaining an 8-way split. We have $\text{Gain}(R, A_0) = H(R) - 0 = H(R)$, but we need to divide by the correction factor $-(8/8)\log_2(1/8) = 3$, so the normalized gain is $0.81/3 = 0.27$. Thus our greedy algorithm would select either A_1 or A_3. Suppose that we select A_1. Now we recurse on the two subtrees corresponding to $R_0^{A_1}$ and $R_1^{A_1}$, selecting A_3 on both sides. The decision tree implements A_1 AND A_3.

Handling different types of attributes

- How to handle a continuous valued attribute A? For a split point s, define a boolean attribute B_A, which is 1 if $A < s$ and 0 otherwise. To find the best split point s for A, find s maximizing $\text{Gain}(R, B_A)$.
  Sort the examples in R based on the value of A. Identify adjacent examples that differ in their labels. Splits that are midway between the corresponding values of A define a sufficient set of candidates. Compute the information gain of each candidate to find the best split.
- Concern with multiway splits: the data is fragmented too quickly (leading to overfitting). Sometimes, it's better to use boolean indicator variables for each value of a multi-valued attribute A. Thus instead of using $A \in \{1, \ldots, k\}$, use variables B_A^j (with $B_A^j = 1$ when $A = j$ and 0 otherwise), for $j \in \{1, \ldots, k\}$.
- Handling missing values for attribute A (being evaluated for infogain): Simple strategy: estimate the missing value as the most common among training examples R reaching the node (perhaps conditioned on having the same label).

  More advanced strategy (used in C4.5): Get the empirical probability distribution on the values of A based on examples in R. Say, the fraction of examples in R with $A(x) = 1$ is 0.6, and with $A(x) = 0$ is 0.4. Then a fractional 0.6 of instance x with a missing value of A goes into R_1^A and 0.4 goes into R_0^A. Same fractioning strategy is used at test time if the value of A is missing: label = most probable label computed by summing the weights of the instance fragments classified at different leaf nodes.

Bias-Variance Tradeoff: Estimation and Approximation Errors

- Given a set of n training examples from D, let $\hat{f}_T$ denote the tree with empirical median fit on each cell (sort the training examples in each cell according to their labels, and use the label of the median example to label any test example falling into the cell).
- Let f_T denote the tree with true median fit on each cell, if we knew D.

Decompose the expected loss of $\hat{f}_T$ (where the expectation is over the draw of the training set form D):

$$\mathbf{E}[\ell(\hat{f}_T)] = \underbrace{\{\mathbf{E}[\ell(\hat{f}_T)] - \ell(f_T)\}}_{\text{Estimation error "variance"}} + \underbrace{\{\ell(f_T) - f^*\}}_{\text{Approximation error, "bias"}} + \underbrace{f^*}_{\text{noise}}$$

Here $f^* = \min_f \ell(f)$ is the smallest achievable ℓ on D.

Estimation and Approximation Errors

Recap: Given a set of n data points from the underlying distribution D, we want to find a function $\hat{f} \in \mathcal{F}$ which minimizes the expected loss ℓ on D (where the expectation is over the draw of the training set). (On the previous slide, $\mathcal{F}$ is the set of all classifiers corresponding to tree structure T. It can also be the set of all tree classifiers with at most a certian number of leaves.)

The decomposition on the previous slide says that the expected loss of a learning algorithm on a training set of size n has three components:

- A *approximation error* measuring the loss due to restricting the search to $\mathcal{F}$, i.e., it measures the mismatch between $\mathcal{F}$ and D.
- A *estimation error* measuring how much the chosen $\hat{f}$ varies with the draw of the training set of size n. In other words, how well the learned $\hat{f}$ performs on average, compared to the best possible f which could have been chosen from the class $\mathcal{F}$.
- A term measuring the smallest achievable loss on D (this term is sometimes referred to as the intrinsic target noise).
- By allowing $\mathcal{F}$ to be complex, we can decrease the approximation error, but at the cost of incurring a large estimation error (overfitting the data). If $\mathcal{F}$ is very simple, the approximation error will be large but we can make the estimation error small.

Estimation Error

- Let's look at the estimation error:

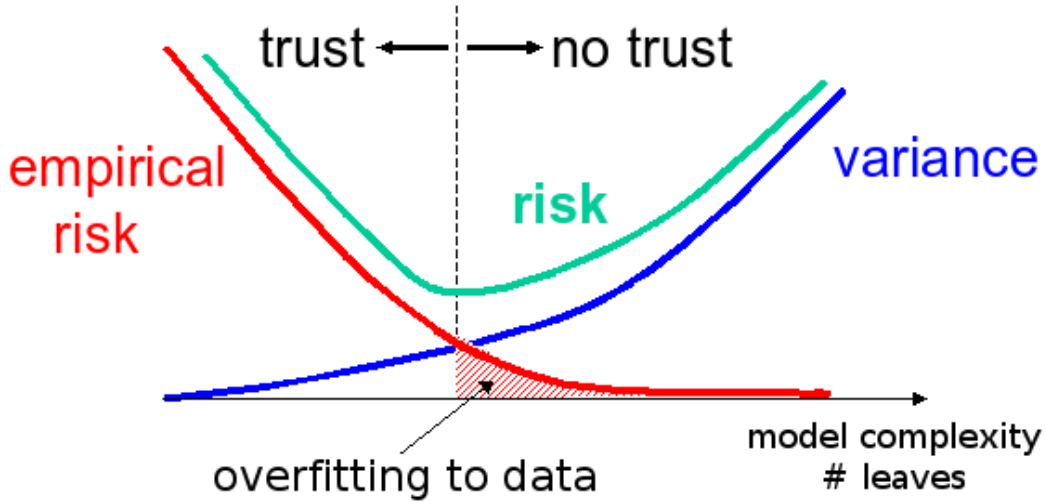
$$\begin{aligned}\mathbf{E}_S[\ell(\hat{f}_T)] - \ell(f_T) &= \mathbf{E}_{S,(x,y)}|\hat{f}_T(x) - y| - \mathbf{E}_{(x,y)}|f_T(x) - y| \\ &\leq \mathbf{E}_{S,(x,y)}|\hat{f}_T(x) - y - f_T(x) + y| = \mathbf{E}_{S,(x,y)}|\hat{f}_T(x) - f_T(x)|\end{aligned}$$

  using the triangle inequality $|a - b| \geq |a| - |b|$ and linearity of expectation. (In the $\mathbf{E}$ notation, (x, y) is a random example drawn from D and S is a set of n training examples drawn independently from D.)

- Thus the estimation error ("variance") of $\hat{f}_T$ is at most $\mathbf{E}|\hat{f}_T(x) - f_T(x)|$

- Similarly, for regression we have $\mathrm{var}(\hat{f}_T) = \mathbf{E}(\hat{f}_T(x) - f_T(x))^2$.

Overfitting



- Bias-variance tradeoff:
  - detailed/flexible partition = small bias, but high variance (too many degrees of freedom)
  - coarse partition = large bias, poor approximation but lower variance (more stable)
- Since expected loss is lower bounded by variance, it is unreasonable to drive empirical error lower than variance

Controlling Overfitting

- Select a tree/partition which minimizing the empirical error + variance:
$$\hat{f} = \operatorname{argmin}_T [\hat{\ell}(\hat{f}_T) + \textbf{var}(\hat{f}_T)]$$
- Variance is proportional to the complexity of the partition, so the strategy is called complexity regularization
- Empirical error is easy to compute, but we don't know the variance. Use bounds on the variance.
- For leaf q, $\mathbf{E}|\hat{f}_q(x) - f_q(x)| = O(1/\sqrt{n_q})$. (For regression, variance scales as $O(1/n_q)$.)

Controlling Overfitting: Simpler strategies

- Pre-pruning (early stopping): Don't split if the split is insignificant. Can use cross-validation: if the cross-validation error increases as a consequence of a node split, then don't split!

  Problems: Hard to evaluate split without seeing which splits it can lead to (lookahead). Some attributes are useful only in combination with other attributes (XOR).
- Post-pruning: grow the tree to its full size and then prune away subtrees until cross-validation accuracy no longer increases.

Conclusion

- Decision tree algorithms are well automated, quite fast and easy to implement.
- There are learning problems which can not be easily solved by splitting on single attributes, but which are solvable (diagonal splits like "Is $x_i < x_j$?")
- May not be the best performer, but consistently good and very popular. Bagged or boosted decision trees work pretty well. Bagging: train many trees with different random samples and average predictions.
- Often said to be interpretable, but trees grown in practice (especially when bagged or boosted to gain performance) are not so easy to interpret.