Machine Learning III: Beyond Decision Trees
AI Class 15 (Ch. 20.1–20.2)

Today’s Class
- Extensions to Decision Trees
- Sources of error
- Evaluating learned models
- Bayesian Learning
- MLA, MLE, MAP
- Bayesian Networks I

Extensions of the Decision Tree Learning Algorithm
- Using gain ratios
- Real-valued data
- Noisy data and overfitting
- Generation of rules
- Setting parameters
- Cross-validation for experimental validation of performance
- C4.5 is an extension of ID3 that accounts for unavailable values, continuous attribute value ranges, pruning of decision trees, rule derivation, and so on

Using Gain Ratios
- Information gain favors attributes with a large number of values
  - If we have an attribute D that has a distinct value for each record, then Info(D,T) is 0, thus Gain(D,T) is maximal
  - To compensate, use the following ratio instead of Gain:
    \[ \text{GainRatio}(D,T) = \frac{\text{Gain}(D,T)}{\text{SplitInfo}(D,T)} \]
  - \(\text{SplitInfo}(D,T)\) is the information due to the split of T on the basis of value of categorical attribute D
  \[ \text{SplitInfo}(D,T) = I(\frac{|T_1|}{|T|}, \frac{|T_2|}{|T|}, \ldots, \frac{|T_m|}{|T|}) \]
  where \(\{T_1, T_2, \ldots, T_m\}\) is the partition of T induced by value of D

Real-Valued Data
- Select a set of thresholds defining intervals
  - Each interval becomes a discrete value of the attribute
- How?
  - Use simple heuristics…
  - Always divide into quartiles
  - Use domain knowledge…
  - Divide age into infant (0-2), toddler (3 - 5), school-aged (5-8)
  - Or treat this as another learning problem
  - Try a range of ways to discretize the continuous variable and see which yield “better results” w.r.t. some metric
  - E.g., try midpoint between every pair of values

Noisy Data
- Many kinds of “noise” can occur in the examples:
  - Two examples have same attribute/value pairs, but different classifications
  - Some values of attributes are incorrect
  - Errors in the data acquisition process, the preprocessing phase, //
  - Classification is wrong (e.g., + instead of -) because of some error
  - Some attributes are irrelevant to the decision-making process, e.g., color of a die is irrelevant to its outcome
  - Some attributes are missing (are pangolins bipedal?)
Overfitting

- Overfitting: coming up with a model that is TOO specific to your training data
  - Does well on training set but not new data
  - How can this happen?
- Too little training data
  - Irrelevant attributes
    - high-dimensional (many attributes) hypothesis space → meaningless regularity in the data irrelevant to important, distinguishing features
    - Fix by pruning lower nodes in the decision tree
    - For example, if Gain of the best attribute at a node is below a threshold, stop and make this node a leaf rather than generating children nodes

Pruning Decision Trees

- Replace a whole subtree by a leaf node
- If a decision rule establishes that he expected error rate in the subtree is greater than in the single leaf. E.g.,
  - Training: one training red success and two training blue failures
  - Test: three red failures and one blue success
  - Consider replacing this subtree by a single Failure node (leaf)
- After replacement we will have only two errors instead of five:

Converting Decision Trees to Rules

- It is easy to derive a rule set from a decision tree:
  - Write a rule for each path in the decision tree from the root to a leaf
  - Left-hand side is label of nodes and labels of arcs
- The resulting rules set can be simplified:
  - Let LHS be the left hand side of a rule
  - Let LHS' be obtained from LHS by eliminating some conditions
  - We can replace LHS by LHS' in this rule if the subsets of the training set that satisfy respectively LHS and LHS' are equal
- A rule may be eliminated by using metaconditions such as “if no other rule applies”

Measuring Model Quality

- How good is a model?
  - Predictive accuracy
  - False positives / false negatives for a given cutoff threshold
  - Loss function (accounts for cost of different types of errors)
  - Area under the (ROC) curve
- Minimizing loss can lead to problems with overfitting

Measuring Model Quality

- Training error
  - Train on all data; measure error on all data
  - Subject to overfitting (of course we’ll make good predictions on the data on which we trained!)
- Regularization
  - Attempt to avoid overfitting
  - Explicitly minimize the complexity of the function while minimizing loss
  - Tradeoff is modeled with a regularization parameter

Cross-Validation

- Holdout cross-validation:
  - Divide data into training set and test set
  - Train on training set; measure error on test set
  - Better than training error, since we are measuring generalization to new data
  - To get a good estimate, we need a reasonably large test set
  - But this gives less data to train on, reducing our model quality!
Cross-Validation, cont.

- **k-fold cross-validation:**
  - Divide data into $k$ folds
  - Train on $k-1$ folds, use the $k$th fold to measure error
  - Repeat $k$ times, use average error to measure generalization accuracy
  - Statistically valid and gives good accuracy estimates

- **Leave-one-out cross-validation (LOOCV)**
  - $k$-fold cross validation where $k=N$ (test data = 1 instance!)
  - Quite accurate, but also quite expensive, since it requires building $N$ models

Naïve Bayes

- Use Bayesian modeling
- Make the simplest possible independence assumption:
  - Each attribute is independent of the values of the other attributes, given the class variable
  - In our restaurant domain: Cuisine is independent of Patrons, given a decision to stay (or not)

Bayesian Formulation

- The probability of class $C$ given $F_1, ... , F_i$
  \[
  p(C | F_1, ..., F_i) = \frac{p(C) \prod p(F_i | C)}{P(F_1, ..., F_i)}
  \]

  - Assume that each feature $F_i$ is conditionally independent of the other features given the class $C$. Then:
    \[
    p(C | F_1, ..., F_i) = \alpha \frac{p(C) \prod p(F_i | C)}{P(F_1, ..., F_i)}
    \]
  - We can estimate each of these conditional probabilities from the observed counts in the training data:
    - $p(F_i | C) = N(F_i \wedge C) / N(C)$
  - One subtlety of using the algorithm in practice: When your estimated probabilities are zero, ugly things happen
  - The fix: Add one to every count (aka "Laplacian smoothing")

Naive Bayes: Example

- $p(Wait | Cuisine, Patrons, Rainy?)$
  \[
  = \alpha \frac{p(Wait)\prod p(Cuisine | Wait) p(Patrons | Wait)}{p(Rainy? | Wait)}
  \]

Naive Bayes: Analysis

- Naïve Bayes is amazingly easy to implement (once you understand the bit of math behind it)
- Naïve Bayes can outperform many much more complex algorithms—it’s a baseline that should pretty much always be used for comparison
- Naïve Bayes can’t capture interdependencies between variables (obviously)—for that, we need Bayes nets!