Recap: Decision Trees

- Example:
  - “Classify Saturday mornings according to whether they’re suitable for playing tennis.”

Recap: CART Framework

- Six general questions
  1. Binary or multi-valued problem?
     - I.e. how many splits should there be at each node?
  2. Which property should be tested at a node?
     - I.e. how to select the query attribute?
  3. When should a node be declared a leaf?
     - I.e. when to stop growing the tree?
  4. How can a grown tree be simplified or pruned?
     - Goal: reduce overfitting.
  5. How to deal with impure nodes?
     - I.e. when the data itself is ambiguous.
  6. How should missing attributes be handled?

Recap: Picking a Good Splitting Feature

- Goal
  - Select the query (=split) that decreases impurity the most
  - \[ \Delta i(N) = i(N) - P_L i(N_L) - (1 - P_L) i(N_R) \]

- Impurity measures
  - Entropy impurity (information gain):
    - \[ i(N) = - \sum_{j} p(C_j|N) \log_2 p(C_j|N) \]
  - Gini impurity:
    - \[ i(N) = \sum_{j} p(C_j|N)p(C_j|N) = \frac{1}{N} \left[ 1 - \sum_{j} p^2(C_j|N) \right] \]

Recap: Overfitting Prevention (Pruning)

- Two basic approaches for decision trees
  - Prepruning: Stop growing tree as some point during top-down construction when there is no longer sufficient data to make reliable decisions.
    - Cross-validation
    - Chi-square test
    - MDL
  - Postpruning: Grow the full tree, then remove subtrees that do not have sufficient evidence.
    - Merging nodes
    - Rule-based pruning

- In practice often preferable to apply post-pruning.
Recap: Computational Complexity

- Given
  - Data points \( \{x_1, ..., x_N\} \)
  - Dimensionality \( D \)

- Complexity
  - Storage: \( O(N) \)
  - Test runtime: \( O(\log N) \)
  - Training runtime: \( O(DN^2 \log N) \)
    - Most expensive part.
      - Critical step: selecting the optimal splitting point.
      - Need to check \( D \) dimensions, for each need to sort \( N \) data points. \( O(DN \log N) \)

Summary: Decision Trees

- Properties
  - Simple learning procedure, fast evaluation.
  - Can be applied to metric, nominal, or mixed data.
  - Often yield interpretable results.

- Limitations
  - Often produce noisy (bushy) or weak (stunted) classifiers.
  - Do not generalize too well.
  - Training data fragmentation: as tree progresses, splits are selected based on less and less data.
  - Overtraining and undertraining:
    - Deep trees: fit the training data well, will not generalize well to new test data.
    - Shallow trees: not sufficiently refined.
  - Stability
    - Trees can be very sensitive to details of the training points.
    - If a single data point is only slightly shifted, a radically different tree may come out!
    - Result of discrete and greedy learning procedure.
    - Expensive learning step
      - Mostly due to costly selection of optimal split.

Randomized Decision Trees (Amit & Geman 1997)

- Decision trees: main effort on finding good split
  - Training runtime: \( O(DN^2 \log N) \)
  - This is what takes most effort in practice.
  - Especially cumbersome with many attributes (large \( D \)).

- Idea: randomize attribute selection
  - No longer look for globally optimal split.
  - Instead randomly use subset of \( K \) attributes on which to base the split.
  - Choose best splitting attribute e.g. by maximizing the information gain (= reducing entropy):
    \[
    \Delta E = \sum_{k=1}^{K} \left( \frac{|S_k|}{|S|} \right) \sum_{j=1}^{N_k} p_j \log_2(p_j)
    \]

Topics of This Lecture

- Randomized Decision Trees
  - Randomized attribute selection

- Random Forests
  - Bootstrap sampling
  - Ensemble of randomized trees
  - Posterior sum combination
  - Analysis

- Extremely randomized trees
  - Random attribute selection

- Ferns
  - Fern structure
  - Semi-Naïve Bayes combination
  - Applications
Ensemble Combination

- Ensemble combination
  - Tree leaves \((l, \eta)\) store posterior probabilities of the target classes.
  - Combine the output of several trees by averaging their posteriors (Bayesian model combination)
    \[ p(C|x) = \frac{1}{L} \sum_{l=1}^{L} p_{l,\eta}(C|x) \]

Applications

- Computer Vision: Optical character recognition
  - Classify small (14x20) images of hand-written characters/digits into one of 10 or 26 classes.
- Simple binary features
  - Tests for individual binary pixel values.
  - Organized in randomized tree.

Applications

- Computer Vision: fast keypoint detection
  - Detect keypoints: small patches in the image used for matching
    - Classify into one of ~200 categories (visual words)
- Extremely simple features
  - E.g. pixel value in a color channel (CIELab)
  - E.g. sum of two points in the patch
  - E.g. difference of two points in the patch
  - E.g. absolute difference of two points
- Create forest of randomized decision trees
  - Each leaf node contains probability distribution over 200 classes
  - Can be updated and re-normalized incrementally.

Topics of This Lecture

- Randomized Decision Trees
  - Random attribute selection
- Random Forests
  - Bootstrap sampling
  - Ensemble of randomized trees
  - Posterior sum combination
  - Analysis
    - Extremely randomized trees
    - Random attribute selection
    - Ferns
    - Semi-Naïve Bayes combination
    - Applications

Random Forests (Breiman 2001)

- General ensemble method
  - Idea: Create ensemble of many (very simple) trees.
- Empirically very good results
  - Often as good as SVMs (and sometimes better)!
  - Often as good as Boosting (and sometimes better)!
- Standard decision trees: main effort on finding good split
  - Random Forests trees put very little effort in this.
  - CART algorithm with Gini coefficient, no pruning.
  - Each split is only made based on a random subset of the available attributes.
  - Trees are grown fully (important!).
- Main secret
  - Injecting the “right kind of randomness”.

Application: Fast Keypoint Detection

Random Forests - Algorithmic Goals

- Create many trees (50 - 1,000)
- Inject randomness into trees such that
  - Each tree has maximal strength
    - I.e. a fairly good model on its own
  - Each tree has minimum correlation with the other trees.
    - I.e. the errors tend to cancel out.
- Ensemble of trees votes for final result
  - Simple majority vote for category.
  - Alternative (Friedman)
    - Optimally reweight the trees via regularized regression (lasso).

Random Forests - Injecting Randomness (1)

- Bootstrap sampling process
  - Select a training set by choosing \( N \) times with replacement from all \( N \) available training examples.
  - On average, each tree is grown on only ~63% of the original training data.
  - Remaining 37% “out-of-bag” (OOB) data used for validation.
  - Provides ongoing assessment of model performance in the current tree.
  - Allows fitting to small data sets without explicitly holding back any data for testing.
  - Error estimate is unbiased and behaves as if we had an independent test sample of the same size as the training sample.

Random Forests - Injecting Randomness (2)

- Random attribute selection
  - For each node, randomly choose subset of \( K \) attributes on which the split is based (typically \( K = \sqrt{N_f} \)).
  - Faster training procedure
    - Need to test only few attributes.
  - Minimizes inter-tree dependence
    - Reduce correlation between different trees.
- Each tree is grown to maximal size and is left unpruned
  - Trees are deliberately overfit
    - Become some form of nearest-neighbor predictor.

Bet You’re Asking...

How can this possibly ever work???
Different trees induce different partitions on the data. By combining them, we obtain a finer subdivision of the feature space...

...which at the same time also better reflects the uncertainty due to the bootstrapped sampling.

Summary: Random Forests

- **Properties**
  - Very simple algorithm.
  - Resistant to overfitting - generalizes well to new data.
  - Faster training
  - Extensions available for clustering, distance learning, etc.

- **Limitations**
  - Memory consumption
  - Decision tree construction uses much more memory.
  - Well-suited for problems with little training data
    - Little performance gain when training data is really large.

You Can Try It At Home...

- Free implementations available
  - Original RF implementation by Breiman & Cutler
    - Papers, documentation, and code...
    - ...in Fortran 77.
  - But also newer version available in Fortran 90!
  - Fast Random Forest implementation for Java (Weka)


Topics of This Lecture

- Randomized Decision Trees
  - Random attribute selection
- Random Forests
  - Bootstrap sampling
  - Ensemble of randomized trees
  - Posterior sum combination
  - Applications

- Extremely randomized trees
  - Random attribute selection
    - RFME
    - Fern structure
    - Semi-Naive Bayes combination
    - Applications

A Case Study in Deconstructivism...

- What we’ve done so far
  - Take the original decision tree idea.
  - Throw out all the complicated bits (pruning, etc.).
  - Learn on random subset of training data (bootstrapping/bagging).
  - Select splits based on random choice of candidate queries.
    - So as to maximize information gain.
    - Complexity: $O(N \log N)$
    - Ensemble of weaker classifiers.

- How can we further simplify that?
  - Main effort still comes from selecting the optimal split (from reduced set of options)...
  - Simply choose a random query at each node.
    - Complexity: $O(N)$
    - Extremely randomized decision trees
Extremely Randomized Decision Trees

- Random queries at each node...
  - Tree gradually develops from a classifier to a flexible container structure.
  - Node queries define (randomly selected) structure.
  - Each leaf node stores posterior probabilities

- Learning
  - Patches are “dropped down” the trees.
  - Only pairwise pixel comparisons at each node.
  - Directly update posterior distributions at leaves
    - Very fast procedure, only few pixel-wise comparisons
    - No need to store the original patches!

Performance Comparison

- Results
  - Almost equal performance for random tests when a sufficient number of trees is available (and much faster to train!).


From Trees to Ferns...

- Observation
  - If we select the node queries randomly anyway, what is the point of choosing different ones for each node?
  - Keep the same query for all nodes at a certain level.
  - This effectively enumerates all $2^M$ possible outcomes of the $M$ tree queries.
  - Tree can be collapsed into a fern-like structure.

What Does This Mean?

- Interpretation of the decision tree
  - We model the class conditional probabilities of a large number of binary features (the node queries).

  - Notation
    - $f_i$: Binary feature
    - $N_f$: Total number of features in the model.
    - $C_k$: Target class
    - Given $f_1, ..., f_N$, we want to select class $C_k$ such that $k = \arg\max p(C_k | f_1, ..., f_N)$
    - Assuming a uniform prior over classes, this is the equal to $k = \arg\max \sum_{i=1}^{N_f} p(C_k | f_i)$
  - Main issue: How do we model the joint distribution?

Modeling the Joint Distribution

- Full Joint
  - Model all correlations between features
    $$p(f_1, ..., f_N | C_k)$$
    - Model with $2^{N_f}$ parameters, not feasible to learn.

- Naïve Bayes classifier
  - Assumption: all features are independent.
    $$p(f_1, ..., f_N | C_k) = \prod_{i=1}^{N_f} p(f_i | C_k)$$
    - Too simplistic, assumption does not really hold!
    - Naïve Bayes model ignores correlation between features.
Modeling the Joint Distribution

- Decision tree
  - Each path from the root to a leaf corresponds to a specific combination of feature outcomes, e.g. $p_{\text{leaf}}(C_k) = p(f_{m1} = 1, f_{m2} = 0, \ldots, f_{md} = 1)$. 
  - Those path outcomes are independent, therefore $p(f_1, \ldots, f_N|C_k) \approx \prod_{m=1}^{M} p_{\text{leaf}}(C_k)$. 
  - But not all feature outcomes are represented here...

Ferns

- Ferns
  - Ferns are thus semi-naïve Bayes classifiers.
  - They assume independence between sets of features (between the ferns)...
  - ...and enumerate all possible outcomes inside each set.

- Interpretation
  - Combine the tests $f_{l_1}, \ldots, f_{l+S}$ into a binary number.
  - Update the “fern leaf” corresponding to that number.

Ferns - Training

The tests compare the intensities of two pixels around the keypoint:

$$ f_i = \begin{cases} 1 & \text{if } I(x) \leq I(y) \\ 0 & \text{otherwise} \end{cases} $$

Invariant to lighting change by any raising function.

Posterior probabilities:
Perceptual and Sensory Augmented Computing

Ferns – Training

Ferns – Training Results

Ferns – Recognition

Normalize:
\[ \sum = 1 \]
Performance Comparison

- Results
  - Ferns perform as well as randomized trees (but are much faster)
  - Naïve Bayes combination better than averaging posteriors.

Keypoint Recognition in 10 Lines of Code

```c
for(int k = 0; k < M; k++) {
  int index = 0, * d = D + k * 2 * S;
  for(int j = 0; j < S; j++) {
    index <<= 1;
    if (*(K + d[0]) < *(K + d[1]))
      index++;
    d += 2;
  }
  p = PF + k * shift2 + index * shift1;
  for(int i = 0; i < H; i++) P[i] += p[i];
}
```

Properties
- Very simple to implement;
- (Almost) no parameters to tune;
- Very fast.


Application: Keypoint Matching with Ferns

Application: Mobile Augmented Reality

Practical Issues - Selecting the Tests

- For a small number of classes
  - We can try several tests.
  - Retain the best one according to some criterion.
    - E.g. entropy, Gini
- When the number of classes is large
  - Any test does a decent job.

Summary

- We started from full decision trees...
  - Successively simplified the classifiers...
- ...and ended up with very simple randomized versions
  - Ensemble methods: Combination of many simple classifiers
  - Good overall performance
  - Very fast to train and to evaluate
- Common limitations of Randomized Trees and Ferns?
  - Need large amounts of training data!
  - In order to fill the many probability distributions at the leaves.
  - Memory consumption!
    - Linear in the number of trees.
    - Exponential in the tree depth,
    - Linear in the number of classes (histogram at each leaf!)
References and Further Reading

- Very recent topics, not covered sufficiently well in books yet...

- The original papers for Randomized Trees

- The original paper for Random Forests:

- The papers for Ferns:
  - D. Wagner, G. Reitmayr, A. Mulloni, T. Drummond, D. Schmalstieg, Pose Tracking from Natural Features on Mobile Phones, In ISMAR 2008.