Course Outline

- Fundamentals (2 weeks)
  - Bayes Decision Theory
  - Probability Density Estimation
- Discriminative Approaches (5 weeks)
  - Linear Discriminant Functions
  - Statistical Learning Theory & SVMs
  - Ensemble Methods & Boosting
  - Randomized Trees, Forests & Ferns
- Generative Models (4 weeks)
  - Bayesian Networks
  - Markov Random Fields

Topics of This Lecture
- Decision Trees
- 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

Recap: Decision Trees

- Elements
  - Each node specifies a test for some attribute.
  - Each branch corresponds to a possible value of the attribute.

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?

CART - 2. Picking a Good Splitting Feature

- Goal
  - Want a tree that is as simple/small as possible (Occam’s razor).
  - But: Finding a minimal tree is an NP-hard optimization problem.
- Greedy top-down search
  - Efficient, but not guaranteed to find the smallest tree.
  - Seek a property \( T \) at each node \( N \) that makes the data in the child nodes as pure as possible.
  - For formal reasons more convenient to define impurity \( i(N) \).
  - Several possible definitions explored.
Picking a Good Splitting Feature

- **Goal**
  - Select the query (=split) that decreases impurity the most
  \[ \Delta i(N) = i(N) - P_Li(N_L) - (1 - P_L)i(N_R) \]
- **Impurity measures**
  - Entropy impurity (information gain):
    \[ i(N) = -\sum_{j} p(C_j|N) \log_2 p(C_j|N) \]
  - Gini impurity:
    \[ i(N) = \sum_{j} p(C_j|N)p(C_j|N) = \frac{1}{2} \left[ 1 - \sum_{j} p^2(C_j|N) \right] \]

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: Decision Trees - Summary

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

Decision Trees - Computational Complexity

- **Given**
  - Data points \([x_1, \ldots, x_N]\)
  - 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.
      \(O(DN \log N)\)

Topics of This Lecture

- Decision Trees
  - 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

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):
    \[
    \Delta E = \sum_{l=1}^{K} \sum_{j=1}^{N} p_l \log_2(p_l)
    \]

Randomized Decision Trees

- Randomized splitting
  - Faster training: $O(KN^2 \log N)$ with $K \ll D$.
  - Use very simple binary feature tests.
  - Typical choice
    - $K = 10$ for root node.
    - $K = 100d$ for node at level $d$.
- Effect of random split
  - Of course, the tree is no longer as powerful as a single classifier...
  - But we can compensate by building several trees.

Ensemble Combination

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

Applications: Character Recognition

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

Applications: Character Recognition

- Image patches ("Tags")
  - Randomly sampled 4x4 patches
  - Construct a randomized tree based on binary single-pixel tests
  - Each leaf node corresponds to a "patch class" and produces a tag
- Representation of digits ("Queries")
  - Specific spatial arrangements of tags
  - An image answers "yes" if any such structure is found anywhere
  - How do we know which spatial arrangements to look for?

Applications: Character Recognition

- Answer: Create a second-level decision tree!
  - Start with two tags connected by an arc
  - Search through extensions of confirmed queries (or rather through a subset of them, there are lots!)
  - Select query with best information gain
  - Recurse...
- Classification
  - Average estimated posterior distributions stored in the leaves.
Applications: Fast Keypoint Detection

- 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
  - 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

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".

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 $K = \sqrt{N}$).
  - 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???

A Graphical Interpretation

Different trees induce different partitions on the data.

...which at the same time also better reflects the uncertainty due to the bootstraped 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
    - http://www.stat.berkeley.edu/users/breiman/RandomForests/
    - Papers, documentation, and code...
    - ...in Fortran 77.
  - But also newer version available in Fortran 90!
  - Fast Random Forest implementation for Java (Weka)
    - http://code.google.com/p/fast-random-forest/

Topics of This Lecture

- Randomized Decision Trees
  - Randomized attribute selection
- Recap: 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

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: \( O(KN^2 \log N) \)
    - 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: \( O(N) \)
  - \textbf{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 (e.g. for keypoint detection)
  - 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!).

Topics of This Lecture
- Randomized Decision Trees
  - Randomized attribute selection
- Recap: 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

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
    - \( f_j \): Binary feature
    - \( N_f \): Total number of features in the model.
    - \( C_k \): Target class
  - Given \( f_1, \ldots, f_N_f \), we want to select class \( C_k \) such that
    \[ k = \arg \max_k p(C_k|f_1, \ldots, f_N_f) \]
  - Assuming a uniform prior over classes, this is the equal to
    \[ k = \arg \max_k p(f_1, \ldots, f_N_f|C_k) \]
  - Main issue: How do we model the joint distribution?

From Trees to Ferns...
- Observation
  - If we select the node queries randomly anyway, what is the point of choosing different ones for each node?
  \[ \Rightarrow \text{Keep the same query for all nodes at a certain level.} \]
  - This effectively enumerates all \( 2^M \) possible outcomes of the \( M \) tree queries.
  - Tree can be collapsed into a fern-like structure.

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

Modeling the Joint Distribution
- Ferns
  - A fern \( F \) is defined as a set of \( S \) binary features \( \{f_0, \ldots, f_{S-1}\} \).
  - \( M \): number of ferns, \( N_f = S \cdot M \).
  - This represents a compromise:
    \[ p(f_1, \ldots, f_N_f|C_k) \approx \prod_{m=1}^M p(F_m|C_k) \]
    \[ = p(f_1, \ldots, f_{S-1}|C_k) \cdot p(f_S, \ldots, f_{2S}|C_k) \cdot \ldots \]
  \[ \Rightarrow \text{Model with } M \cdot 2^S \text{ parameters ("Semi-Naïve").} \]
  - Flexible solution that allows complexity/performance tuning.

Modeling the Joint Distribution
- Naïve Bayes classifier
  - Assumption: all features are independent.
    \[ p(f_1, \ldots, f_N_f|C_k) = \prod_{j=1}^{N_f} p(f_j|C_k) \]
  \[ \Rightarrow \text{Too simplistic, assumption does not really hold!} \]
  \[ \Rightarrow \text{Naïve Bayes model ignores correlation between features.} \]
Modeling the Joint Distribution

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

- **Interpretation**
  - Combine the tests \( f_1, \ldots, f_n \) into a binary number.
  - Update the “fern leaf” corresponding to that number.

\[ \begin{array}{c|c|c}
0 & 0 & 1 \\
\end{array} \]

Update leaf \( 100_2 = 4 \)

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:

\[ P(f_1, f_2, \ldots, f_n \mid C = c_j) \]

Slide credit: Vincent Lepetit
### Ferns - Training

![Training Ferns](image1)

### Ferns - Training Results

![Training Results](image2)

### Ferns - Recognition

![Recognition](image3)

### 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

```java
1: for (int i = 0; i < H; i++) P[i] = 0.;
2: for (int k = 0; k < M; k++) {
3:   int index = 0, * d = D + k * 2 * S;
4:   for (int j = 0; j < S; j++) {
5:     index <<= 1;
6:     if (*(K + d[0]) < *(K + d[1]))
7:       index++;
8:     d += 2;
9:   }
10:   p = PF + k * shift2 + index * shift1;
11:   for (int i = 0; i < H; i++) P[i] += p[i];
}
```

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

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

- The original papers for Randomized Trees

- The original paper for Random Forests:

- The papers for Ferns: