Recap: Boosting (Schapire 1989)

- Algorithm: (3-component classifier)
  1. Sample $N_1 < N$ training examples (without replacement) from training set $\mathcal{D}$ to get set $\mathcal{D}_1$.
  2. Sample $N_2 < N$ training examples (without replacement), half of which were misclassified by $C_1$, to get set $\mathcal{D}_2$.
  3. Choose all data in $\mathcal{D}$ on which $C_1$ and $C_2$ disagree to get set $\mathcal{D}_3$.
  4. Get the final classifier output by majority voting of $C_1$, $C_2$, and $C_3$.

Recap: Bayesian Model Averaging

- Model Averaging
  - Suppose we have $H$ different models $h = 1, \ldots, H$ with prior probabilities $p(h)$.
  - Construct the marginal distribution over the data set
    
    $p(X) = \sum_{h=1}^{H} p(X|h)p(h)$
  - Average error of committee
    
    $\overline{E}_{COM} = \frac{1}{M} \overline{E}_{AV}$

- This suggests that the average error of a model can be reduced by a factor of $M$ simply by averaging $M$ versions of the model!
- Unfortunately, this assumes that the errors are all uncorrelated. In practice, they will typically be highly correlated.

Course Outline

- Fundamentals (2 weeks)
  - Bayes Decision Theory
  - Probability Density Estimation
- Discriminative Approaches (5 weeks)
  - Linear Discriminant Functions
  - Statistical Learning Theory
  - Support Vector Machines
  - Boosting, Decision Trees
- Generative Models (5 weeks)
  - Bayesian Networks
  - Markov Random Fields
- Regression Problems (2 weeks)
  - Gaussian Processes

Recap: Stacking

- Idea
  - Learn $L$ classifiers (based on the training data)
  - Find a meta-classifier that takes as input the output of the $L$ first-level classifiers.
- Example
  - Learn $L$ classifiers with leave-one-out.
  - Interpret the prediction of the $L$ classifiers as $L$-dimensional feature vector.
  - Learn "level-2" classifier based on the examples generated this way.

- Why can this be useful?
  - Simplicity
    - We may already have several existing classifiers available.
    - No need to retrain those, they can just be combined with the rest.
  - Correlation between classifiers
    - The combination classifier can learn the correlation.
  - Better results than simple naive Bayes combination.
  - Feature combination
    - E.g. combine information from different sensors or sources (vision, audio, acceleration, temperature, radar, etc.).
    - We can get good training data for each sensor individually, but data from all sensors together is rare.
    - Train each of the $L$ classifiers on its own input data.
    - Only combination classifier needs to be trained on combined input.
Recap: AdaBoost - “Adaptive Boosting”

- **Main idea** [Freund & Schapire, 1996]
  - Instead of resampling, reweight misclassified training examples.
  - Increase the chance of being selected in a sampled training set.
  - Or increase the misclassification cost when training on the full set.

- **Components**
  - $h_m(x)$: “weak” or base classifier
  - Condition: <50% training error over any distribution
  - $H(x)$: “strong” or final classifier

- **AdaBoost:**
  - Construct a strong classifier as a thresholded linear combination of the weighted weak classifiers:
    $$ H(x) = \text{sign} \left( \sum_{m=1}^{M} \alpha_m h_m(x) \right) $$

Recap: AdaBoost – Intuition

Consider a 2D feature space with positive and negative examples.

Each weak classifier splits the training examples with at least 50% accuracy.

Examples misclassified by a previous weak learner are given more emphasis at future rounds.

Recap: AdaBoost – Algorithm

1. Initialization: Set $w_i^{(1)} = \frac{1}{N}$ for $n = 1, \ldots, N$.
2. For $m = 1, \ldots, M$ iterations
   a) Train a new weak classifier $h_m(x)$ using the current weighting coefficients $W^{(m)}$ by minimizing the weighted error function
      $$ J_m = \sum_{n=1}^{N} w_n^{(m)} I(h_m(x) \neq t_n) = \frac{1}{2} \text{ if } A \text{ is true} \quad \text{else}$$
   b) Estimate the weighted error of this classifier on $X$:
      $$ \epsilon_m = \frac{1}{2} \sum_{n=1}^{N} w_n^{(m)} I(h_m(x) \neq t_n) $$
   c) Calculate a weighting coefficient for $h_m(x)$:
      $$ \alpha_m = \ln \left( \frac{1}{\epsilon_m} \right) $$
   d) Update the weighting coefficients:
      $$ w_i^{(m+1)} = w_i^{(m)} \exp \left\{ \alpha_m I(h_m(x_i) \neq t_i) \right\} $$

Recap: Comparing Error Functions

- Ideal misclassification error function
- “Hinge error” used in SVMs
- Exponential error function
  - Continuous approximation to ideal misclassification function.
  - Sequential minimization leads to simple AdaBoost scheme.
  - Disadvantage: exponential penalty for large negative values!
  - Less robust to outliers or misclassified data points!
Recap: Comparing Error Functions

- Ideal misclassification error function
- “Hinge error” used in SVMs
- Exponential error function
- “Cross-entropy error”
  - Similar to exponential error for $z \geq 0$.
  - Only grows linearly with large negative values of $z$.
  - Make AdaBoost more robust by switching to “GentleBoost”

Topics of This Lecture

- Decision Trees
  - CART
  - Impurity measures
  - Stopping criterion
  - Pruning
  - Extensions
  - Issues
  - Historical development: ID3, C4.5

- Random Forests
  - Basic idea
  - Bootstrap sampling
  - Randomized attribute selection
  - Applications

Decision Trees

- Very old technique
  - Origin in the 60s, might seem outdated.
- But...
  - Can be used for problems with nominal data
    - E.g. attributes color $\in \{\text{red, green, blue}\}$ or weather $\in \{\text{sunny, rainy}\}$.
    - Discrete values, no notion of similarity or even ordering.
  - Interpretable results
    - Learned trees can be written as sets of if-then rules.
  - Methods developed for handling missing feature values.
  - Successfully applied to broad range of tasks
    - E.g. Medical diagnosis
    - E.g. Credit risk assessment of loan applicants
  - Some interesting novel developments building on top of them...

Decision Trees

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

Decision Trees

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

- Assumption
  - Links must be mutually distinct and exhaustive
  - I.e. one and only one link will be followed at each step.

- Interpretability
  - Information in a tree can then be rendered as logical expressions.
  - In our example:
    - $(\text{Outlook} = \text{sunny} \land \text{Humidity} = \text{Normal})$
    - $(\text{Outlook} = \text{Overcast})$
    - $(\text{Outlook} = \text{Rain} \land \text{Wind} = \text{Weak})$
Training Decision Trees

- Finding the optimal decision tree is NP-hard...
- Common procedure: Greedy top-down growing
  - Start at the root node.
  - Progressively split the training data into smaller and smaller subsets.
  - In each step, pick the best attribute to split the data.
  - If the resulting subsets are pure (only one label) or if no further attribute can be found that splits them, terminate the tree.
  - Else, recursively apply the procedure to the subsets.
- CART framework
  - Classification And Regression Trees (Breiman et al. 1993)
  - Formalization of the different design choices.

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 - 1. Number of Splits

- Each multi-valued tree can be converted into an equivalent binary tree:

⇒ Only consider binary trees here...

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 \( \text{i}(N) \).
  - Several possible definitions explored.

CART - Impurity Measures

- Misclassification impurity
  \[
  i(N) = 1 - \max_j \frac{|C_j|}{|N|}
  \]
  “Fraction of the training patterns in category \( C_j \) that end up in node \( N \).”

- Entropy impurity
  \[
  i(N) = - \sum_j \frac{|C_j|}{|N|} \log_2 \frac{|C_j|}{|N|}
  \]
  “Reduction in entropy = gain in information.”
CART – Impurity Measures

- Gini impurity (variance impurity)

\[ i(N) = \sum_{j} p(C_{j}|N)p(C_{j}|N) \]
\[ = \frac{1}{2}(1 - \sum_{j} p^2(C_{j}|N)) \]

"Expected error rate at node N if the category label is selected randomly."

CART – Picking a Good Splitting Feature

- Application
  - Select the query that decreases impurity the most

\[ \Delta i(N) = i(N) - P_k i(N_k) - (1 - P_k) i(N_R) \]

- Multiway generalization (gain ratio impurity):
  - Maximize

\[ \Delta i(s) = \frac{1}{2} \left( i(N) - \sum_{k=1}^{K} P_k i(N_k) \right) \]

  where the normalization factor ensures that large K are not inherently favored:

\[ Z = -\sum_{k=1}^{K} P_k \log_2 P_k \]

CART - 2. Picking a Good Splitting Feature

- For efficiency, splits are often based on a single feature
- "Monothetic decision trees"

CART - 3. When to Stop Splitting

- Problem: Overfitting
  - Learning a tree that classifies the training data perfectly may not lead to the tree with the best generalization to unseen data.
  - Reasons
    - Noise or errors in the training data.
    - Poor decisions towards the leaves of the tree that are based on very little data.

- Typical behavior

\[ \text{Accuracy} \]
\[ \text{hypothesis complexity} \]

on training data

on test data

CART - 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.
  - Postpruning: Grow the full tree, then remove subtrees that do not have sufficient evidence.

- Label leaf resulting from pruning with the majority class of the remaining data, or a class probability distribution.

\[ C_N = \arg \max_k p(C_k|N) \]
\[ p(C_k|N) \]
CART - Stopping Criterion

- Determining which subtrees to prune:
  - Cross-validation: Reserve some training data as a hold-out set (validation set, tuning set) to evaluate utility of subtrees.
  - Statistical test: Determine if any observed regularity can be dismissed as likely due to random chance.
    - Determine the probability that the outcome of a candidate split could have been generated by a random split.
    - Chi-squared statistic (one degree of freedom)
      \[ \chi^2 = \sum_{i=1}^{4} \frac{(\hat{N}_{i, obs} - \hat{N}_{i, corr})^2}{\hat{N}_{i, corr}} \]
    - Compare to critical value at certain confidence level (table lookup).
    - Minimum description length (MDL): Determine if the additional complexity of the hypothesis is less complex than just explicitly remembering any exceptions resulting from pruning.

(Post-)Pruning Strategies

- Common strategies
  - Merging leaf nodes
    - Consider pairs of neighboring leaf nodes.
    - If their elimination results only in small increase in impurity, prune them.
    - Procedure can be extended to replace entire subtrees with leaf node directly.
  - Rule-based pruning
    - Each leaf has an associated rule (conjunction of individual decisions).
    - Full tree can be described by list of rules.
    - Can eliminate irrelevant preconditions to simplify the rules.
    - Can eliminate rules to improve accuracy on validation set.
    - Advantage: can distinguish between the contexts in which the decision rule at a node is used \( \Rightarrow \) can prune them selectively.

Decision Trees - Handling Missing Attributes

- During training
  - Calculate impurities at a node using only the attribute information present.
    - E.g. 3-dimensional data, one point is missing attribute \( x_1 \).
    - Compute possible splits on \( x_1 \) using all \( N \) points.
    - Compute possible splits on \( x_{2} \) using all \( N \) points.
    - Compute possible splits on \( x_{3} \) using \( N-1 \) non-deficient points.
    - Choose split which gives greatest reduction in impurity.

- During test
  - Cannot handle test patterns that are lacking the decision attribute.
  - In addition to primary split, store an ordered set of surrogate splits that try to approximate the desired outcome based on different attributes.

Decision Trees - Feature Choice

- Best results if proper features are used
  - Preprocessing to find important axes often pays off.
Decision Trees - Non-Uniform Cost

- Incorporating category priors
  - Often desired to incorporate different priors for the categories.
  - Solution: weight samples to correct for the prior frequencies.

- Incorporating non-uniform loss
  - Create loss matrix $\lambda_{ij}$
  - Loss can easily be incorporated into Gini impurity
    $$ i(N) = \sum_{j} \lambda_{ij} p(C_j) \Delta p(C_j) $$

Historical Development

- C4.5 (Quinlan 1993)
  - Improved version with extended capabilities.
  - Ability to deal with real-valued variables.
  - Multway splits are used with nominal data.
  - Using gain ratio impurity based on entropy (information gain) criterion.
  - Heuristics for pruning based on statistical significance of splits.
  - Rule post-pruning

- Main difference to CART
  - Strategy for handling missing attributes.
  - When missing feature is queried, C4.5 follows all $|I|$ possible answers.
  - Decision is made based on all $|I|$ possible outcomes, weighted by decision probabilities at node $N$.

Summary: Decision Trees

- 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: $O(\lambda N)$
  - Training runtime: $O(DN^2 \log N)$

Summary: Decision Trees

- 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.
  - Expensive learning step
    - Mostly due to costly selection of optimal split.
Topics of This Lecture

• Decision Trees
  - CART
  - Impurity measures
  - Stopping criteria
  - Pruning
  - Extensions
  - Issues
  - Historical development: ID3, C4.5
• Random Forests
  - Basic idea
  - Bootstrap sampling
  - Randomized attribute selection
  - Applications

Random Forests (Breiman 2001)

• 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 use very little effort in this.
  - CART algorithm with Gini coefficient, no pruning.
  - Each split 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 reweigh 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 \( \approx 63\% \) of the original training data.
  - Remaining \( 37\% \) “out-of-bag” (OOB) data used for validation.
    - Provides ongoing assessment of model performance.
    - Allows fitting to small data sets without explicitly holding back any data for testing.

Random Forests - Injecting Randomness (2)

• Random attribute selection
  - For each node, randomly choose subset of \( \sqrt{\mathcal{M}} \) attributes on which the split is based (typically square root of number available).
  - Evaluate splits only on OOB data (out-of-bag estimate).
  - Very fast training procedure
    - Need to test few attributes.
    - Evaluate only \( \approx 37\% \) of the data.
  - 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.

Big Question

How can this ever possibly 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.
  - Very rapid training
    - Also often used for online learning.
  - 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/
    - Code + documentation
      - 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/

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

Application: Fast Keypoint Detection

References and Further Reading

- More information on Decision Trees can be found in Chapters 8.2-8.4 of Duda & Hart.

R.O. Duda, P.E. Hart, D.G. Stork
Pattern Classification
2nd Ed., Wiley-InterScience, 2000

- The original paper for Random Forests: