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.

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
    \[
    E_{COM} = \frac{1}{M} E_{AV}
    \]
  - 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.

Recap: Boosting (Schapire 1989)

- **Algorithm**: (3-component classifier)
  1. Sample $N_1 < N$ training examples (without replacement) from training set $D$ to get set $D_1$.
     - Train weak classifier $C_1$ on $D_1$.
  2. Sample $N_2 < N$ training examples (without replacement), half of which were misclassified by $C_1$, to get set $D_2$.
     - Train weak classifier $C_2$ on $D_2$.
  3. Choose all data in $D$ on which $C_1$ and $C_2$ disagree to get set $D_3$.
     - Train weak classifier $C_3$ on $D_3$.
  4. Get the final classifier output by majority voting of $C_1$, $C_2$, and $C_3$.
     (Recursively apply the procedure on $C_1$ to $C_3$)

Topics of This Lecture

- **Recap: AdaBoost**
  - Algorithm
  - Analysis
  - Extensions
- **Analysis**
  - Comparing Error Functions
- **Applications**
  - AdaBoost for face detection
- **Decision Trees**
  - CART
  - Impurity measures, Stopping criterion, Pruning
  - Extensions, Issues
  - Historical development: ID3, C4.5
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 - Algorithm

1. **Initialization:** Set $w_1(n) = \frac{1}{N}$ 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 $J_m = \frac{1}{N} \sum_{n=1}^{N} w(m)_n I(h_m(x)_n \neq t_n)$
   b) Estimate the weighted error of this classifier on $X$:
   $$ e_m = \frac{1}{N} \sum_{n=1}^{N} w(m)_n I(h_m(x)_n \neq t_n) $$
   c) Calculate a weighting coefficient for $h_m(x)$:
   $$ \alpha_m = \ln \left( \frac{1}{e_m} \right) $$
   d) Update the weighting coefficients:
   $$ w(m+1)_n = w(m)_n \exp \left( \alpha_m I(h_m(x)_n \neq t_n) \right) $$

Topics of This Lecture

- Recap: AdaBoost
  - Algorithm
  - Analysis
  - Extensions
- **Analysis**
  - Comparing Error Functions
- Applications
  - AdaBoost for face detection
- Decision Trees
  - CART
  - Impurity measures, Stopping criterion, Pruning
  - Extensions, Issues
  - Historical development: ID3, C4.5

AdaBoost - Analysis

- Result of this derivation
  - We now know that AdaBoost minimizes an exponential error function in a sequential fashion.
  - This allows us to analyze AdaBoost’s behavior in more detail.
  - In particular, we can see how robust it is to outlier data points.

Recap: Error Functions

- **Ideal misclassification error function (black)**
  - This is what we want to approximate.
  - Unfortunately, it is not differentiable.
  - The gradient is zero for misclassified points.
  - We cannot minimize it by gradient descent.

- **Squared error used in Least-Squares Classification**
  - Very popular, leads to closed-form solutions.
  - However, sensitive to outliers due to squared penalty.
  - Penalizes “too correct” data points
  - Generally does not lead to good classifiers.
Recap: Error Functions

- "Hinge error" used in SVMs
  - Zero error for points outside the margin ($z > 1$) ⇒ sparsity
  - Linear penalty for misclassified points ($z < 1$) ⇒ robustness
  - Not differentiable around $z = 1$ ⇒ Cannot be optimized directly.

Discussion: AdaBoost Error Function

- Exponential error used in AdaBoost
  - No penalty for too correct data points, fast convergence.
  - Disadvantage: exponential penalty for large negative values!
  - Less robust to outliers or misclassified data points!

Discussion: Other Possible Error Functions

- Cross-entropy error used in Logistic Regression
  - Similar to exponential error for $z > 0$.
  - Only grows linearly with large negative values of $z$.
  - Make AdaBoost more robust by switching to this error function.
    ⇒ "GentleBoost"

Summary: AdaBoost

- Properties
  - Simple combination of multiple classifiers.
  - Easy to implement.
  - Can be used with many different types of classifiers.
    - None of them needs to be too good on its own.
    - In fact, they only have to be slightly better than chance.
    - Commonly used in many areas.
    - Empirically good generalization capabilities.

- Limitations
  - Original AdaBoost sensitive to misclassified training data points.
  - Improvement by GentleBoost.
  - Single-class classifier
    - Multiclass extensions available

Topics of This Lecture

- Recap: AdaBoost
  - Algorithm
  - Analysis
  - Extensions

- Analysis
  - Comparing Error Functions

- Applications
  - AdaBoost for face detection

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

- Frontal faces are a good example of a class where global appearance models + a sliding window detection approach fit well:
  - Regular 2D structure
  - Center of face almost shaped like a "patch"/window

- Now we'll take AdaBoost and see how the Viola-Jones face detector works.

Feature extraction

"Rectangular" filters

Feature output is difference between adjacent regions

Efficiently computable with integral image: any sum can be computed in constant time
Avoid scaling images → scale features directly for same cost

Large Library of Filters

Considering all possible filter parameters: position, scale, and type:
180,000+ possible features associated with each 24 x 24 window

Use AdaBoost both to select the informative features and to form the classifier

AdaBoost for Feature+Classifier Selection

- Want to select the single rectangle feature and threshold that best separates positive (faces) and negative (non-faces) training examples, in terms of weighted error.

Outputs of a possible rectangle feature on faces and non-faces.

Resulting weak classifier:

For next round, reweight the examples according to errors, choose another filter/threshold combo.

AdaBoost for Efficient Feature Selection

- Image features = weak classifiers
- For each round of boosting:
  - Evaluate each rectangle filter on each example
  - Sort examples by filter values
  - Select best threshold for each filter (min error)
    - Sorted list can be quickly scanned for the optimal threshold
  - Select best filter/threshold combination
  - Weight on this features is a simple function of error rate
  - Reweight examples

Viola-Jones Face Detector: Results

Viola-Jones Face Detector: Results

References and Further Reading

- More information on Classifier Combination and Boosting can be found in Chapters 14.1-14.3 of Bishop’s book.

  Christopher M. Bishop
  Pattern Recognition and Machine Learning
  Springer, 2006

- A more in-depth discussion of the statistical interpretation of AdaBoost is available in the following paper:

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 $c$ (red, green, blue) or weather $w$ (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...

Topics of This Lecture

- Recap: AdaBoost
  - Algorithm
  - Analysis
  - Extensions

- Analysis
  - Comparing Error Functions

- Applications
  - AdaBoost for face detection

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

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 be rendered as logical expressions.
  - In our example:
    \[
    (\text{Outlook} = \text{Sunny} \land \text{Humidity} = \text{Normal}) \\
    \lor (\text{Outlook} = \text{Overcast}) \\
    \lor (\text{Outlook} = \text{Rain} \land \text{Wind} = \text{Weak})
    \]

**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**

- 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:

**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.
CART - Impurity Measures

- **Misclassification impurity**
  \[ i(N) = 1 - \max_j p(C_j|N) \]
  
  "Fraction of the training patterns in category \( C_j \) that end up in node \( N \)."

- **Gini impurity (variance impurity)**
  \[ i(N) = \sum_{i \neq j} p(C_i|N)p(C_j|N) \]
  
  \[ = \frac{1}{2} - \sum_j p^2(C_j|N) \]
  
  "Expected error rate at node \( N \) if the category label is selected randomly."

CART - Impurity Measures

- **Entropy impurity**
  \[ i(N) = -\sum_j p(C_j|N) \log_2 p(C_j|N) \]
  
  "Reduction in entropy = gain in information."

CART - 2. Picking a Good Splitting Feature

- **Application**
  - Select the query that decreases impurity the most
    \[ \Delta i(N) = i(N) - P_k i(N_k) - (1 - P_k) i(N_{\bar{k}}) \]

- **Multiway generalization (gain ratio impurity):**
  - Maximize
    \[ \Delta i(s) = \frac{1}{Z} \left( i(N) - \sum_{k=1}^K P_k i(N_k) \right) \]
  - where the normalization factor ensures that large \( K \) are not inherently favored:
    \[ Z = -\sum_{k=1}^K P_k \log_2 P_k \]

CART - Picking a Good Splitting Feature

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

- **Evaluating candidate splits**
  - Nominal attributes: exhaustive search over all possibilities.
  - Real-valued attributes: only need to consider changes in label.
    - Only need to test candidate splits where \( \text{label}(x_i) \neq \text{label}(x_{i+1}) \).
**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**
  - Accuracy on training data is high,
  - Accuracy on test data is lower.

### Accuracy vs. Hypothesis Complexity

**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.**

\[ C_N = \arg\max_k p(C_k | N) \]

**CART - 4. (Post-)Pruning**

- **Stopped splitting often suffers from “horizon effect”**
  - Decision for optimal split at node \( N \) is independent of decisions at descendant nodes.
  - Might stop splitting too early.
  - Stopped splitting biases learning algorithm towards trees in which the greatest impurity reduction is near the root node.
- **Often better strategy**
  - Grow tree fully (until leaf nodes have minimum impurity).
  - Then prune away subtrees whose elimination results only in a small increase in impurity.
- **Benefits**
  - Avoids the horizon effect.
  - Better use of training data (no hold-out set for cross-validation).

### (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, or 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 \( x_a \)
    - Compute possible splits on \( x_a \) using all \( N \) points.
    - Compute possible splits on \( x_a \) 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 be incorporated into Gini impurity
    \[ i(N) = \sum_{ij} \lambda_{ij} p(C_i)p(C_j) \]

Historical Development

- ID3 (Quinlan 1986)
  - One of the first widely used decision tree algorithms.
  - Intended to be used with nominal (unordered) variables
    - Real variables are first binned into discrete intervals.
  - General branching factor
    - Use gain ratio impurity based on entropy (information gain) criterion.
  - Algorithm
    - Select attribute $a$ that best classifies examples, assign it to root.
    - For each possible value $v_i$ of $a$:
      - Add new tree branch corresponding to test $a = v_i$.
      - If example_list($v_i$) is empty, add leaf node with most common label in example_list($a$).
      - Else, recursively call ID3 for the subtree with attributes $A \setminus a$.

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.
      - $O(DN \log N)$
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.

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