
- **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 \( w_n^{(1)} = \frac{1}{N} \) 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
      \[
      J_m = \sum_{i=1}^{N} w_{i}^{(m)} I(h_m(x_i) \neq t_i) \quad I(A) = \begin{cases} 1, & \text{if } A \text{ is true} \\ 0, & \text{else} \end{cases}
      \]
   b) Estimate the weighted error of this classifier on \( X \): \( \epsilon_m = \sum_{i=1}^{N} w_{i}^{(m)} I(h_m(x_i) \neq t_i) \sum_{i=1}^{N} w_{i}^{(m)} \)
   c) Calculate a weighting coefficient for \( h_m(x) \): \( \alpha_m = \ln \left( \frac{1 - \epsilon_m}{\epsilon_m} \right) \)
   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.
- Disadvantage: exponential penalty for large negative values!
- Less robust to outliers or misclassified data points!

Decision Trees

- Very old technique
  - Origin in the 60s, might seem outdated.
  - 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...

Example:
- "Classify Saturday mornings according to whether they're suitable for playing tennis."
  - Sunny
  - Overcast
  - Rain
  - High/Medium/Low
  - Strong/Weak
  - Yes/No
Decision Trees

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

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 - 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 \( i(N) \).
    - Several possible definitions explored.

CART Framework

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

- **Application**
  - Select the query that decreases impurity the most
  \[ \Delta i(N) = i(N) - P_L i(N_L) - (1 - P_L) i(N_R) \]
  - 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:
  \[ Z = - \sum_{k=1}^{K} P_k \log_2 P_k \]

- **Which impurity measure should we choose?**
  - Some problems with misclassification impurity.
    - Discontinuous derivative.
    - Problems when searching over continuous parameter space.
    - Sometimes misclassification impurity does not decrease when Gini impurity would.
  - Both entropy impurity and Gini impurity perform well.
    - No big difference in terms of classifier performance.
    - In practice, stopping criterion and pruning method are often more important.

- **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.
  - Order all data points based on attribute \( x_j \).
  - Only need to test candidate splits where \( \text{label}(x_j) = \text{label}(x_{j+1}) \).
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 vs Hypothesis Complexity](chart)

- 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.

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

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 dismisses 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)
    - \[ \chi^2 = \sum_{i=1}^{N} \frac{(n_{i,left} - \hat{n}_{i,left})^2}{\hat{n}_{i,left}} \]
    - “expected number from random split”
    - 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.

CART - 4. (Post-)Pruning

- Stopped splitting often suffers from “horizon effect”
  - Decision for optimal split at node \( N \) is independent of decisions at descendent 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).

Decision Trees - Feature Choice

- Best results if proper features are used

![Decision Tree Examples](chart)

- Bad tree
- Good tree

- 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
  - \( i(N) = \sum_{ij} \lambda_{ij} p(C_i)p(C_j) \)

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 \([x_1, ... , x_N]\)
  - 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.
    - \( O(DN \log N) \)

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 \( \text{example_list}(a) \) is empty, add leaf node with most common label in \( \text{example_list}(a) \).
      - Else, recursively call ID3 for the subtree with attributes \( A \setminus a \).

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.
    - 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.
Topics of This Lecture

• Decision Trees
  • CART
  • Impurity measures
  • Stopping criteria
  • Pruning
  • 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 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.
    • 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 $T$ 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 on ~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???
A Graphical Interpretation

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.

- The original paper for Random Forests: