Ensembles; bagging and boosting; decision trees

Prof. Adriana Kovashka
University of Pittsburgh
April 11, 2016
Plan for Today

• Ensemble methods
  – Bagging
  – Boosting
• Boosting application: Face detection
• Decision trees
Learning Ensembles

- Learn multiple alternative definitions of a concept using different training data or different learning algorithms
- Train several classifiers: SVM, KNN, logistic regression, decision tree, neural network etc.
- Call these classifiers $f_1(x), f_2(x), \ldots, f_M(x)$
- Take majority of predictions:
  $$y = \text{majority}( f_1(x), f_2(x), \ldots, f_M(x) )$$
- For regression use mean or median of the predictions
- Averaging is a form of regularization: each model can individually overfit but the average is able to overcome the overfitting

Slide credit: Ray Mooney / Subhransu Maji
Learning Ensembles

- Learn multiple alternative definitions of a concept using different training data or different learning algorithms
- Combine decisions of multiple definitions
Value of Ensembles

• When combing multiple *independent* and *diverse* decisions each of which is at least more accurate than random guessing, random errors cancel each other out, correct decisions are reinforced.

• Human ensembles are demonstrably better
  – How many jelly beans in the jar?: Individual estimates vs. group average.
  – Who Wants to be a Millionaire: Expert friend vs. audience vote.
Homogenous Ensembles

• Use a single, arbitrary learning algorithm but manipulate training data to make it learn multiple models.
  – Data1 ≠ Data2 ≠ … ≠ Data m
  – Learner1 = Learner2 = … = Learner m

• Different methods for changing training data:
  – Bagging: Resample training data
  – Boosting: Reweight training data
Bagging

- Create ensembles by repeatedly randomly resampling the training data (Brieman, 1996).
- Given a training set of size $n$, create $m$ samples of size $n$ by drawing $n$ examples from the original data, \textit{with replacement}.
- Combine the $m$ resulting models using simple majority vote.
- Decreases error by decreasing the variance in the results due to \textit{unstable learners}, algorithms (like decision trees) whose output can change dramatically when the training data is slightly changed.
- However, often the errors of the different models are correlated, which defies the purpose of bagging.
Boosting

• Originally developed by computational learning theorists to guarantee performance improvements on fitting training data for a weak learner that only needs to generate a hypothesis with a training accuracy greater than 0.5 (Schapire, 1990).

• Revised to be a practical algorithm, AdaBoost, for building ensembles that empirically improves generalization performance (Freund & Shapire, 1996).

• Examples are given weights. At each iteration, a new hypothesis is learned and the examples are reweighted to focus the system on examples that the most recently learned classifier got wrong.
Boosting: Basic Algorithm

• General Loop:
  Set all examples to have equal uniform weights.
  For $m$ from 1 to $M$ do:
    Find the weak learner $h_m$ that achieves lowest weighted training error
    Increase the weights of examples that $h_m$ classifies incorrectly

• During testing: Each of the $M$ classifiers gets a weighted vote proportional to its accuracy on the training data; final classifier is a linear combination of all weak learners.

• Base (weak) learner must focus on correctly classifying the most highly weighted examples while strongly avoiding over-fitting.

• Weak learners must perform better than chance.

Slide credit: Ray Mooney, Lana Lazebnik, Kristen Grauman
Boosting Illustration

Weak Classifier 1
Boosting Illustration

Weights Increased

Paul Viola
Boosting Illustration

Weak Classifier 2
Boosting Illustration

Weights Increased
Boosting Illustration

Weak Classifier 3
Final classifier is a combination of weak classifiers
AdaBoost

1. Initialize the data weighting coefficients \( \{w_n\} \) by setting \( w_n^{(1)} = 1/N \) for \( n = 1, \ldots, N \).
2. For \( m = 1, \ldots, M \):
   (a) Fit a classifier \( y_m(x) \) to the training data by minimizing the weighted error function

   \[
   J_m = \sum_{n=1}^{N} w_n^{(m)} I(y_m(x_n) \neq t_n) \tag{14.15}
   \]

   where \( I(y_m(x_n) \neq t_n) \) is the indicator function and equals 1 when \( y_m(x_n) \neq t_n \) and 0 otherwise.
   (b) Evaluate the quantities

   \[
   \epsilon_m = \frac{\sum_{n=1}^{N} w_n^{(m)} I(y_m(x_n) \neq t_n)}{\sum_{n=1}^{N} w_n^{(m)}} \tag{14.16}
   \]

   and then use these to evaluate

   \[
   \alpha_m = \ln \left( \frac{1 - \epsilon_m}{\epsilon_m} \right). \tag{14.17}
   \]
   (c) Update the data weighting coefficients

   \[
   w_n^{(m+1)} = w_n^{(m)} \exp \{\alpha_m I(y_m(x_n) \neq t_n)\} \tag{14.18}
   \]
   (d) Normalize the weights so they sum to 1

3. Make predictions using the final model, which is given by

   \[
   Y_M(x) = \text{sign} \left( \sum_{m=1}^{M} \alpha_m y_m(x) \right). \tag{14.19}
   \]
Learning with Weighted Examples

• Generic approach is to replicate examples in the training set proportional to their weights (e.g. 10 replicas of an example with a weight of 0.01 and 100 for one with weight 0.1).

• Most algorithms can be enhanced to efficiently incorporate weights directly in the learning algorithm so that the effect is the same.

• For decision trees, for calculating information gain, when counting example $i$, simply increment the corresponding count by $w_i$ rather than by 1.
Experimental Results on Ensembles
(Freund & Schapire, 1996; Quinlan, 1996)

• Ensembles have been used to improve
generalization accuracy on a wide variety of
problems.
• On average, Boosting provides a larger increase in
accuracy than Bagging.
• Boosting on rare occasions can degrade accuracy.
• Bagging more consistently provides a modest
improvement.
• Boosting is particularly subject to over-fitting
when there is significant noise in the training data.
Issues in Ensembles

- Parallelism in Ensembles: Bagging is easily parallelized, Boosting is not.
- Variants of Boosting to handle noisy data.
- How “weak” should a base-learner for Boosting be?
- What is the theoretical explanation of Boosting’s ability to improve generalization?
- Exactly how does the diversity of ensembles affect their generalization performance?
- Combining Boosting and Bagging?
Boosting application: Face detection
Challenges of face detection

- Sliding window detector must evaluate tens of thousands of location/scale combinations.
- Faces are rare: 0–10 per image.
- A megapixel image has \(~10^6\) pixels and a comparable number of candidate face locations.
- For computational efficiency, we should try to spend as little time as possible on the non-face windows.
Viola-Jones face detector

Main idea:

– Represent local texture with efficiently computable “rectangular” features within window of interest
– Select discriminative features to be weak classifiers
– Use boosted combination of them as final classifier
– Form a cascade of such classifiers, rejecting clear negatives quickly (not discussed, see hidden slides)
Viola-Jones detector: features

“Rectangular” filters
Feature output is difference between adjacent regions

\[ \text{Value} = \sum \text{(pixels in white area)} - \sum \text{(pixels in black area)} \]

Efficiently computable with integral image: any sum can be computed in constant time.
Example
Considering all possible filter parameters: position, scale, and type:

180,000+ possible features associated with each 24 x 24 window

Which subset of these features should we use to determine if a window has a face?

Use AdaBoost both to select the informative features and to form the classifier
Viola-Jones detector: AdaBoost

- 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.

Resulting weak classifier:

$$h_t(x) = \begin{cases} +1 & \text{if } f_t(x) > \theta_t \\ -1 & \text{otherwise} \end{cases}$$

For next round, reweight the examples according to errors, choose another filter/threshold combo.
First two features selected by boosting:

This feature combination can yield 100% detection rate and 50% false positive rate
Viola-Jones face detector: Results
Viola-Jones face detector: Results
Decision Stumps

- Like the thresholded features=classifiers in face detection
- A single-level decision tree (discussed next)

![Decision Stump Diagram](Figure from Wikipedia)

- For HW4, use a total of 10-30 of these, and don’t worry about them having better-than-chance performance
Plan for Today

• Ensemble methods
  – Bagging
  – Boosting

• Boosting application: Face detection

• Decision trees
Decision Trees

- Tree-based classifiers for instances represented as feature-vectors. Nodes test features, there is one branch for each value of the feature, and leaves specify the category.

- Can represent arbitrary conjunction and disjunction. Can represent any classification function over discrete feature vectors.

- Can be rewritten as a set of rules:
  - red \land circle \rightarrow pos
  - red \land circle \rightarrow A
  - blue \rightarrow B
  - red \land square \rightarrow B
  - green \rightarrow C
  - red \land triangle \rightarrow C

Slide credit: Ray Mooney
What about continuous features?

- Continuous (real-valued) features can be handled by allowing nodes to split a real valued feature into two ranges based on a threshold (e.g. length < 3 and length ≥ 3).
- Classification trees have discrete class labels at the leaves.

Adapted from Ray Mooney
Top-Down Decision Tree Induction

- Recursively build a tree top-down by divide and conquer.

```
<big, red, circle>: +    <small, red, circle>: +
<small, red, square>: -  <big, blue, circle>: -
```

![Decision Tree Diagram]

Slide credit: Ray Mooney
Top-Down Decision Tree Induction

- Recursively build a tree top-down by divide and conquer.

<big, red, circle>: +  <small, red, circle>: +
<small, red, square>: −  <big, blue, circle>: −

Slide credit: Ray Mooney
Decision Tree Induction Pseudocode

DTree\( (examples, features) \) returns a tree

If all \( examples \) are in one category, return a leaf node with that category label.
Else if the set of \( features \) is empty, return a leaf node with the category label that is the most common in \( examples \).
Else pick a feature \( F \) and create a node \( R \) for it

For each possible value \( v_i \) of \( F \):

Let \( examples_i \) be the subset of examples that have value \( v_i \) for \( F \)
Add an out-going edge \( E \) to node \( R \) labeled with the value \( v_i \).
If \( examples_i \) is empty
then attach a leaf node to edge \( E \) labeled with the category that is the most common in \( examples \).
else call DTree\( (examples_i, features - \{F\}) \) and attach the resulting tree as the subtree under edge \( E \).
Return the subtree rooted at \( R \).
Picking a Good Split Feature

- Goal is to have the resulting tree be as small as possible, per Occam’s razor.
- Finding a minimal decision tree (nodes, leaves, or depth) is an NP-hard optimization problem.
- Top-down divide-and-conquer method does a greedy search for a simple tree but does not guarantee to find the smallest; that’s ok.
- Want to pick a feature that creates subsets of examples that are relatively “pure” in a single class so they are “closer” to being leaf nodes.
- There are a variety of heuristics for picking a good test, a popular one is based on information gain that originated with the ID3 system of Quinlan (1979).
Entropy

- Entropy (disorder, impurity) of a set of examples, S, relative to a binary classification is:

\[
Entropy(S) = -p_1 \log_2(p_1) - p_0 \log_2(p_0)
\]

where \(p_1\) is the fraction of positive examples in S and \(p_0\) is the fraction of negatives.

- If all examples are in one category, entropy is zero (we define \(0 \cdot \log(0) = 0\)).

- If examples are equally mixed (\(p_1 = p_0 = 0.5\)), entropy is a maximum of 1.

- For multi-class problems with \(c\) categories, entropy generalizes to:

\[
Entropy(S) = \sum_{i=1}^{c} - p_i \log_2(p_i)
\]

Adapted from Ray Mooney
Entropy Plot for Binary Classification
Information Gain

- The information gain of a feature $F$ is the expected reduction in entropy resulting from splitting on this feature.

$$Gain(S, F) = Entropy(S) - \sum_{v \in Values(F)} \frac{|S_v|}{|S|} Entropy(S_v)$$

where $S_v$ is the subset of $S$ having value $v$ for feature $F$.

- Entropy of each resulting subset weighted by its relative size.

- Example:
  - $<\text{big, red, circle}>$: +
  - $<\text{small, red, circle}>$: +
  - $<\text{small, red, square}>$: -
  - $<\text{big, blue, circle}>$: -

  2+, 2 -: $E=1$

  size
  \[
  \begin{array}{c|c|c}
  \text{size} & \text{big} & \text{small} \\
  \hline
  1+ & 1+ & 1- \\
  \text{E=1} & \text{E=1} & \text{E=1} \\
  \end{array}
  \]

  Gain=$1-(0.5 \cdot 1 + 0.5 \cdot 1) = 0$

  color
  \[
  \begin{array}{c|c|c}
  \text{color} & \text{red} & \text{blue} \\
  \hline
  2+ & 1- & 0+ \\
  \text{E=1} & \text{E=0} & \text{E=1} \\
  \end{array}
  \]

  Gain=$1-(0.75 \cdot 0.918 + 0.25 \cdot 0) = 0.311$

  shape
  \[
  \begin{array}{c|c|c}
  \text{shape} & \text{circle} & \text{square} \\
  \hline
  2+ & 1- & 0+ \\
  \text{E=1} & \text{E=0} & \text{E=1} \\
  \end{array}
  \]

  Gain=$1-(0.75 \cdot 0.918 + 0.25 \cdot 0) = 0.311$

Slide credit: Ray Mooney
Another Example Decision Tree Classifier

- **Example problem:** decide whether to wait for a table at a restaurant, based on the following attributes:
  1. **Alternate:** is there an alternative restaurant nearby?
  2. **Bar:** is there a comfortable bar area to wait in?
  3. **Fri/Sat:** is today Friday or Saturday?
  4. **Hungry:** are we hungry?
  5. **Patrons:** number of people in the restaurant (None, Some, Full)
  6. **Price:** price range ($, $$, $$$)
  7. **Raining:** is it raining outside?
  8. **Reservation:** have we made a reservation?
  9. **Type:** kind of restaurant (French, Italian, Thai, Burger)
  10. **WaitEstimate:** estimated waiting time (0-10, 10-30, 30-60, >60)
### Another Example Decision Tree Classifier

<table>
<thead>
<tr>
<th>Example</th>
<th>Alt</th>
<th>Bar</th>
<th>Fri</th>
<th>Hun</th>
<th>Pat</th>
<th>Price</th>
<th>Rain</th>
<th>Res</th>
<th>Type</th>
<th>Est</th>
<th>Target</th>
<th>Wait</th>
</tr>
</thead>
<tbody>
<tr>
<td>$X_1$</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>T</td>
<td>Some</td>
<td>$$$$</td>
<td>F</td>
<td>T</td>
<td>French</td>
<td>0–10</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>$X_2$</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>T</td>
<td>Full</td>
<td>$</td>
<td>F</td>
<td>F</td>
<td>Thai</td>
<td>30–60</td>
<td>F</td>
<td></td>
</tr>
<tr>
<td>$X_3$</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>Some</td>
<td>$</td>
<td>F</td>
<td>F</td>
<td>Burger</td>
<td>0–10</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>$X_4$</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>T</td>
<td>Full</td>
<td>$</td>
<td>F</td>
<td>F</td>
<td>Thai</td>
<td>10–30</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>$X_5$</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>Full</td>
<td>$$$$</td>
<td>F</td>
<td>T</td>
<td>French</td>
<td>&gt;60</td>
<td>F</td>
<td></td>
</tr>
<tr>
<td>$X_6$</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>Some</td>
<td>$$</td>
<td>T</td>
<td>T</td>
<td>Italian</td>
<td>0–10</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>$X_7$</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>None</td>
<td>$</td>
<td>T</td>
<td>F</td>
<td>Burger</td>
<td>0–10</td>
<td>F</td>
<td></td>
</tr>
<tr>
<td>$X_8$</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>T</td>
<td>Some</td>
<td>$$</td>
<td>T</td>
<td>T</td>
<td>Thai</td>
<td>0–10</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>$X_9$</td>
<td>F</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>Full</td>
<td>$</td>
<td>T</td>
<td>F</td>
<td>Burger</td>
<td>&gt;60</td>
<td>F</td>
<td></td>
</tr>
<tr>
<td>$X_{10}$</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>Full</td>
<td>$$$$</td>
<td>F</td>
<td>T</td>
<td>Italian</td>
<td>10–30</td>
<td>F</td>
<td></td>
</tr>
<tr>
<td>$X_{11}$</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>None</td>
<td>$</td>
<td>F</td>
<td>F</td>
<td>Thai</td>
<td>0–10</td>
<td>F</td>
<td></td>
</tr>
<tr>
<td>$X_{12}$</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>Full</td>
<td>$</td>
<td>F</td>
<td>F</td>
<td>Burger</td>
<td>30–60</td>
<td>T</td>
<td></td>
</tr>
</tbody>
</table>
Another Example Decision Tree Classifier

Patrons?  
- None  
- Some  
- Full

Wait Estimate?  
- >60  
- 30-60  
- 10-30  
- 0-10

Alternate?  
- No  
- Yes

Reservation?  
- No  
- Yes

Fri/Sat?  
- No  
- Yes

Hungry?  
- No  
- Yes

Alternate?  
- No  
- Yes

Raining?  
- No  
- Yes

Lana Lazebnik
Another Example Decision Tree Classifier

<table>
<thead>
<tr>
<th>Example</th>
<th>Alt</th>
<th>Bar</th>
<th>Fri</th>
<th>Hun</th>
<th>Pat</th>
<th>Price</th>
<th>Rain</th>
<th>Res</th>
<th>Type</th>
<th>Est</th>
<th>Target</th>
<th>Wait</th>
</tr>
</thead>
<tbody>
<tr>
<td>X₁</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>T</td>
<td>Some</td>
<td>$$$</td>
<td>F</td>
<td>T</td>
<td>French</td>
<td>0–10</td>
<td>T</td>
<td></td>
</tr>
<tr>
<td>X₂</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>T</td>
<td>Full</td>
<td>$</td>
<td>F</td>
<td>F</td>
<td>Thai</td>
<td>30–60</td>
<td>F</td>
<td></td>
</tr>
<tr>
<td>X₃</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>Some</td>
<td>$</td>
<td>F</td>
<td>F</td>
<td>Burger</td>
<td>0–10</td>
<td>T</td>
<td></td>
</tr>
<tr>
<td>X₄</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>T</td>
<td>Full</td>
<td>$</td>
<td>F</td>
<td>F</td>
<td>Thai</td>
<td>10–30</td>
<td>T</td>
<td></td>
</tr>
<tr>
<td>X₅</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>Full</td>
<td>$$$</td>
<td>F</td>
<td>T</td>
<td>French</td>
<td>&gt;60</td>
<td>F</td>
<td></td>
</tr>
<tr>
<td>X₆</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>Some</td>
<td>$$</td>
<td>T</td>
<td>T</td>
<td>Italian</td>
<td>0–10</td>
<td>T</td>
<td></td>
</tr>
<tr>
<td>X₇</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>None</td>
<td>$</td>
<td>T</td>
<td>F</td>
<td>Burger</td>
<td>0–10</td>
<td>F</td>
<td></td>
</tr>
<tr>
<td>X₈</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>T</td>
<td>Some</td>
<td>$$</td>
<td>T</td>
<td>T</td>
<td>Thai</td>
<td>0–10</td>
<td>T</td>
<td></td>
</tr>
<tr>
<td>X₉</td>
<td>F</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>Full</td>
<td>$</td>
<td>T</td>
<td>F</td>
<td>Burger</td>
<td>&gt;60</td>
<td>F</td>
<td></td>
</tr>
<tr>
<td>X₁₀</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>Full</td>
<td>$$$</td>
<td>F</td>
<td>T</td>
<td>Italian</td>
<td>10–30</td>
<td>F</td>
<td></td>
</tr>
<tr>
<td>X₁₁</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>None</td>
<td>$</td>
<td>F</td>
<td>F</td>
<td>Thai</td>
<td>0–10</td>
<td>F</td>
<td></td>
</tr>
<tr>
<td>X₁₂</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>Full</td>
<td>$</td>
<td>F</td>
<td>F</td>
<td>Burger</td>
<td>30–60</td>
<td>T</td>
<td></td>
</tr>
</tbody>
</table>
Another Example Decision Tree Classifier

Patrons?
- None
- Some
- Full

Wait Estimate?
- >60
- 30-60
- 10-30
- 0-10

Alternate?
- No
- Yes

Reservation?
- No
- Yes

Fri/Sat?
- No
- Yes

Hungry?
- No
- Yes

Alternate?
- No
- Yes

Bar?
- No
- Yes

Raining?
- No
- Yes
## Another Example Decision Tree Classifier

<table>
<thead>
<tr>
<th>Example</th>
<th>Attrs</th>
<th>Target</th>
</tr>
</thead>
</table>
Overfitting

• Learning a tree that classifies the training data perfectly may not lead to the tree with the best generalization to unseen data.
  – There may be noise in the training data that the tree is erroneously fitting.
  – The algorithm may be making poor decisions towards the leaves of the tree that are based on very little data and may not reflect reliable trends.

Slide credit: Ray Mooney
Overfitting Noise in Decision Trees

- Category or feature noise can easily cause overfitting.
  - Add noisy instance <medium, blue, circle>: pos (but really neg)
Overfitting Noise in Decision Trees

- Category or feature noise can easily cause overfitting.
  - Add noisy instance <medium, blue, circle>: pos (but really neg)

- Noise can also cause different instances of the same feature vector to have different classes. Impossible to fit this data and must label leaf with the majority class.
  - <big, red, circle>: neg (but really pos)

- Conflicting examples can also arise if the features are incomplete and inadequate to determine the class or if the target concept is non-deterministic.

Slide credit: Ray Mooney
Overfitting Prevention (Pruning) Methods

- 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.

- Method for determining which subtrees to prune:
  - Cross-validation: Reserve some training data as a hold-out set (validation set) to evaluate utility of subtrees.
  - Statistical test: Use a statistical test on the training data to determine if any observed regularity can be dismisses as likely due to random chance.
  - Minimum description length (MDL): Determine if the additional complexity of the hypothesis is less complex than just explicitly remembering any exceptions resulting from pruning.