Accuracy Boosting Using Ensemble Classifiers

- Outline
- Ensemble methods
- Bagging
- Boosting
- Error-correcting output coding
- Why does ensemble learning work?
Readings


What is ensemble learning?

Ensemble learning refers to a collection of methods that learn a target function by training a number of individual learners and combining their predictions

A gambler, frustrated by persistent horse-racing losses and envious of his friends’ winnings, decides to allow a group of his fellow gamblers to make bets on his behalf. He decides he will wager a fixed sum of money in every race, but that he will apportion his money among his friends based on how well they are doing. Certainly, if he knew psychically ahead of time which of his friends would win the most, he would naturally have that friend handle all his wagers. Lacking such clairvoyance, however, he attempts to allocate each race’s wager in such a way that his total winnings for the season will be reasonably close to what he would have won had he bet everything with the luckiest of his friends.

[Freund & Schapire, 1995]
Ensemble Learning

- **Intuition:** Combining Predictions of an ensemble is more accurate than a single classifier
- **Justification:**
  – It is easy to find quite correct “rules of thumb”
  – It is hard to find single highly accurate prediction rule
  – If the training examples are few and the hypothesis space is large then there are several equally accurate classifiers
  – Hypothesis space does not contain the true function, but it has several good approximations
  – Exhaustive global search in the hypothesis space is expensive so we can combine the predictions of several locally accurate classifiers

---

**Ensemble learning**

- **Learning phase**
  - \( T \)
  - \( T_1 \rightarrow h_1 \)
  - \( T_2 \rightarrow h_2 \)
  - \( \ldots \)
  - \( T_S \rightarrow h_S \)
  - \( \{ \) different training sets and/or learning algorithms

- **Classification phase**
  - \((x, ?)\)
  - \( h^* = F(h_1, h_2, \ldots, h_S) \)
  - \((x, y^*)\)
How to make an effective ensemble?

Two basic questions in designing ensembles:

- How to generate the base classifiers?
  \( h_1, h_2, \ldots \)
- How to combine them?
  \( F(h_1(x), h_2(x), \ldots) \)

How to combine classifiers

Usually take a weighted vote:

\[
\text{ensemble}(x) = \text{sign}(\sum_i w_i h_i(x))
\]

- \( w_i \) is the weight of hypothesis \( h_i \)
- \( w_i > w_j \) means \( h_i \) is more reliable than \( h_j \)
- typically \( w_i > 0 \) (though could have \( w_i < 0 \) meaning \( h_i \) is more often wrong than right)
- Bayesian averaging is an example
- (Fancier schemes are possible but uncommon)
How to generate base classifiers

• A variety of approaches
• Bagging (Bootstrap aggregation)
• Boosting (Specifically, Adaboost – Adaptive Boosting algorithm)
• …

Bagging

• Generate a random sample from training set by selecting elements with replacement
• Repeat this sampling procedure, getting a sequence of \( k \) independent training sets
• A corresponding sequence of classifiers \( C_1, C_2, \ldots, C_k \) is constructed from these training sets, by using the same classification algorithm
• To classify an unknown sample \( X \), let each classifier predict
• The Bagged Classifier \( C^* \) then combines the predictions of the individual classifiers to generate the final outcome. (sometimes combination is simple voting)
BAGGing = Bootstrap AGGregation (Breiman, 1996)

- for i = 1, 2, ..., K:
  - $T_i \leftarrow$ randomly select M training instances with replacement
  - $h_i \leftarrow$ learn ($T_i$)
- Combine the $T_i$ using uniform voting ($w_i = 1/K$ for all $i$)
CART decision boundary

100 bagged trees

shades of blue/red indicate strength of vote for particular classification
Classification results
Misclassification rates

Bagging References

- Leo Breiman’s homepage
  www.stat.berkeley.edu/users/breiman/
  *Machine Learning*, 26:2, 123-140.
  Bagging and Nonlinear Estimation”
  www.stat.stanford.edu/~jhf
Boosting

• Boosting, like bagging, is an ensemble method.
• The prediction generated by the classifier is a combination of the prediction of several predictors.
• What is different?
  – It is iterative
  – Boosting: Successive classifiers depends upon its predecessors
  – Previous methods e.g., bagging: Individual classifiers were independent
  – Training Examples may have unequal weights
  – Look at errors from previous classifier step to decide how to focus on next iteration over data
  – Set weights to focus more on 'hard' examples. (the ones on which we committed mistakes in the previous iterations)

Boosting Algorithm

- $W(x)$ is the distribution of weights over the $N$ training points $\sum W(x_j)=1$
- Initially assign uniform weights $W_0(x) = 1/N$ for all $x$, step $k=0$
- At each iteration $k$:
  - Find best weak classifier $C_k(x)$ using weights $W_k(x)$ with
    - error rate $\varepsilon_k$
    - $\alpha_k$ is the weight of the classifier $C_k$
    - For each $x_i$, update weights based on $\varepsilon_k$ to get $W_{k+1}(x_i)$
  - $C_{FINAL}(x) = \text{sign} \left[ \sum \alpha_i C_i(x) \right]$
Boosting (Algorithm)

\[ C(x) = \sum_{j=1}^{M} \alpha_j C_j(x) \]

Basic Idea:
- assign a weight to every training set instance
- initially, all instances have the same weight
- as boosting proceeds, it adjusts weights based on how well we have predicted data points so far
  - data points correctly predicted \( \rightarrow \) low weight
  - data points mispredicted \( \rightarrow \) high weight
- Results: as learning proceeds, the learner is forced to focus on portions of data space not previously well predicted
Boosting Demo

http://www.cse.ucsd.edu/~yfreund/adaboost/index.html

AdaBoost Algorithm

- $W(x)$ is the distribution of weights over the $N$ training points $\sum W(x_i) = 1$
- Initially assign uniform weights $W_0(x) = 1/N$ for all $x$.
- At each iteration $k$:
  - Find best weak classifier $C_k(x)$ using weights $W_k(x)$
  - Compute $\varepsilon_k$ the error rate as $\varepsilon_k = [\sum W(x_i) \cdot I(y_i \neq C_k(x_i))] / [\sum W(x_i)]$
  - weight the classifier $C_k$ by $\alpha_k$
    - $\alpha_k = \log ((1 - \varepsilon_k)/\varepsilon_k)$
  - For each $x_i$, $W_{k+1}(x_i) = W_k(x_i) \cdot \exp[\alpha_k \cdot I(y_i \neq C_k(x_i))]$
- $C_{FINAL}(x) = \text{sign} \left[ \sum \alpha_i C_i(x) \right]$
AdaBoost Example

Original Training set: Equal Weights to all training samples

ROUND 1

Copyright Vasant Honavar, 2006.
AdaBoost Example

ROUND 2

\[ \epsilon_2 = 0.21 \]
\[ \alpha_2 = 0.65 \]

\( h_2 \)

\( D_3 \)

AdaBoost Example

ROUND 3

\[ \epsilon_3 = 0.14 \]
\[ \alpha_3 = 0.92 \]
Boosting

• Suppose \( L \) is a weak learner - one that can learn a hypothesis that is better than rolling a dice – but perhaps only a tiny bit better
  – Theorem: Boosting \( L \) yields an ensemble with arbitrarily low error on the training data!
Boosting performance

Decision stumps are very simple classifiers that test condition on a single attribute.

Suppose we use decision stumps as individual classifiers whose predictions were combined to generate the final prediction.

Suppose we plot the misclassification rate of the Boosting algorithm against the number of iterations performed.

```
<table>
<thead>
<tr>
<th>eye color = brown?</th>
<th>height &gt; 5 feet?</th>
</tr>
</thead>
<tbody>
<tr>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>predict +1</td>
<td>predict +1</td>
</tr>
<tr>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>predict -1</td>
<td>predict -1</td>
</tr>
</tbody>
</table>
```

Misclassification rates

Friedman, Hastie, Tibshirani [1998]

<table>
<thead>
<tr>
<th></th>
<th>Breast</th>
<th>Ionosphere</th>
<th>Glass</th>
<th>Sonar</th>
<th>Waveform</th>
</tr>
</thead>
<tbody>
<tr>
<td>CART</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AdaBoost CART</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LogitBoost CART</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Copyright Vasant Honavar, 2006.
Boosting performance

• Observations
  – First few (about 50) iterations increase the accuracy substantially. Seen by the steep decrease in misclassification rate.
  – As iterations increase training error decreases
  – As iterations increase, generalization error decreases?
Can Boosting do well if?

- Individual classifiers are not very accurate and have high variance (e.g., decision stumps)?
  - It can if the individual classifiers have considerable mutual disagreement.
- Individual classifier is very accurate and has low variance (e.g., SVM with a good kernel function)?
  - No.

Boosting as an Additive Model

- The final prediction in boosting $f(x)$ can be expressed as an additive expansion of individual classifiers
  $$f(x) = \sum_{m=1}^{M} \beta_m b(x; \gamma_m)$$

- The process is iterative and can be expressed as follows.
  $$f_m(x) = f_{m-1}(x) + \beta_m b(x; \gamma_m)$$

- Typically we would try to minimize a loss function on the training examples
  $$\min_{\beta_m, \gamma_m} \sum_{i=1}^{N} L(y_i, \sum_{m=1}^{M} \beta_m b(x_i; \gamma_m))$$
Boosting as an Additive Model

• Simple case: Squared-error loss

\[
L(y, f(x)) = \frac{1}{2} (y - f(x))^2
\]

• Forward stage-wise modeling amounts to just fitting the residuals from previous iteration.

\[
L(y_i, f_{m-1}(x_i) + \beta b(x_i; \gamma)) \\
= (y_i - f_{m-1}(x_i) - \beta b(x_i; \gamma))^2 \\
= (r_{im} - \beta b(x_i; \gamma))^2
\]

• Squared-error loss not robust for classification

Boosting as an Additive Model

• AdaBoost for Classification uses the exponential loss function:

\[- L(y, f(x)) = \exp(-y \cdot f(x))\]

\[
\arg\min_{f} \sum_{i=1}^{N} L(y_i, f(x_i)) \\
= \arg\min_{\beta, G_m} \sum_{i=1}^{N} \exp(-y_i \cdot [f_{m-1}(x_i) + \beta \cdot G_m(x_i)]) \\
= \arg\min_{\beta, G_m} \sum_{i=1}^{N} \exp(-y_i \cdot f_{m-1}(x_i)) \cdot \exp(-y_i \cdot \beta \cdot G_m(x_i))
\]
Boosting As Additive Model

First assume that $\beta$ is constant, and minimize w.r.t. $G$:

$$\arg\min_{\beta,G} \sum_{i=1}^{N} \exp(-y_i \cdot f_{m-1}(x_i)) \exp(-y_i \cdot \beta \cdot G_m(x_i))$$

$$= \arg\min_{\beta,G} \sum_{i=1}^{N} w_{i}^{(m)} \cdot \exp(-y_i \cdot \beta \cdot G_m(x_i)), \text{ where } w_{i}^{(m)} = \exp(-y_i \cdot f_{m-1}(x_i))$$

$$= \arg\min_{G} \sum_{y_i=G(x_i)}^{N} w_{i}^{(m)} \cdot e^{-\beta} + \sum_{y_i \neq G(x_i)}^{N} w_{i}^{(m)} \cdot e^{\beta}$$

$$= \arg\min_{G} (e^{\beta} - e^{-\beta}) \sum_{i=1}^{N} [w_{i}^{(m)} \cdot I(y_i \neq G(x_i))] + e^{-\beta} \sum_{i=1}^{N} w_{i}^{(m)}$$

$$= \arg\min_{G} (e^{\beta} - e^{-\beta}) \frac{\sum_{i=1}^{N} [w_{i}^{(m)} \cdot I(y_i \neq G(x_i))]}{\sum_{i=1}^{N} w_{i}^{(m)}} + e^{-\beta}$$

**err**$_m$: the training error on the weighted samples

On each iteration we must find a classifier that minimizes the training error on the weighted samples!
Boosting as an Additive Model

Now that we have found $G$, we minimize w.r.t. $\beta$:

$$H(\beta) = \text{err}_m \cdot (e^{\beta} - e^{-\beta}) + e^{-\beta}$$

$$\frac{\partial H}{\partial \beta} = \text{err}_m \cdot (e^{\beta} + e^{-\beta}) - e^{-\beta} = 0$$

$$1 - e^{\beta} \cdot \text{err}_m (e^{\beta} + e^{-\beta}) = 0$$

$$1 - e^{2\beta} \cdot \text{err}_m - \text{err}_m = 0$$

$$\frac{1 - \text{err}_m}{\text{err}_m} = e^{2\beta}$$

$$\beta = \frac{1}{2} \log \left( \frac{1 - \text{err}_m}{\text{err}_m} \right)$$

---

**Input**: sequence of $N$ labeled examples $\{(x_1, y_1), \ldots, (x_N, y_N)\}$

- distribution $D$ over the $N$ examples
- weak learning algorithm $\text{WeakLearn}$
- integer $T$ specifying number of iterations

**Initialize** the weight vector: $w_i = D(i)$ for $i = 1, \ldots, N$.

Do for $t = 1, 2, \ldots, T$:

1. Set
   $$p'_t = \frac{w'_t}{\sum_{i=1}^N w'_t}$$

2. Call $\text{WeakLearn}$, providing it with the distribution $p'_t$; get back a hypothesis $h_t : X \rightarrow \{0, 1\}$.

3. Calculate the error of $h_t$: $\epsilon_t = \sum_{i=1}^N p'_t |h_t(x_i) - y_i|$.

4. Set $\beta_t = \epsilon_t / (1 - \epsilon_t)$.

5. Set the new weights vector to be
   $$w_i^{t+1} = w_i^t \beta_t^{1 - |h_t(x_i) - y_i|}$$

**Output** the hypothesis

$$h_f(x) = \begin{cases} 1 & \text{if } \sum_{t=1}^T \left( \frac{1}{\beta_t} \right) h_t(x) \geq \frac{1}{2} \sum_{t=1}^T \log \frac{1}{\beta_t} \\ 0 & \text{otherwise} \end{cases}$$
Learning from weighted instances?

2. Call WeakLearn, providing it with the distribution \( p^j \); get back a hypothesis \( h_i : X \rightarrow [0,1] \)

- So far, learning algorithms have taken as input a set of examples, all assumed to be equally important
- What if we also get a weight vector saying how important each instance is?
- It is easy to modify most learning algorithms to deal with weighted instances:
  - Decision tree: Easy to modify entropy, information-gain equations to look at weight of a set of instances, rather than the count (which simply assumes all weights=1)

AdaBoost (Characteristics)

- Why exponential loss function?
  - Computational
    - Simple modular re-weighting
    - Determining optimal parameters is relatively easy
  - Statistical
    - In a two label case it determines one half the log odds of \( P(Y=1|x) \)
    - We can use the sign as the classification rule
- Accuracy depends upon number of iterations
Why do ensembles work?

- Because uncorrelated errors of individual classifiers can be eliminated by averaging.
- Assume: 40 base classifiers, majority voting, each error rate 0.3
- Probability of getting $r$ incorrect votes from 40

\[
P(r) = \frac{n!}{r!(n-r)!} \text{error}_b(h)^r (1-\text{error}_b(h))^{n-r}
\]

$p\text{(Ensemble is wrong)} = p(>20 \text{ incorrect votes}) = 0.01$

This analysis makes lots of assumptions; can we say something “deeper”?

Deeper explanations?

- Statistical
- Computational
- Representational
Statistical

- Given a finite amount of data, many hypotheses are typically equally good.
- How can the learning algorithm select among them?

\[
h_{all} = \text{hypothesis from all data}
\]

\[
h_1, h_2, \ldots \text{ may be averaged better approximation to } f \text{ than } h_{all}
\]

Representational

The desired target function may not be realizable using individual classifiers, but may be approximated by ensemble averaging.

Suppose you want to build a decision boundary with decision trees. The decision boundaries of decision trees are hyper-planes parallel to the coordinate axes. By averaging a large number of such "staircases", the diagonal decision boundary can be approximated with arbitrarily good accuracy.
Representational (another example)

• Consider a binary learning task over \([0,1] \times [0,1]\), and the hypothesis space \(H\) of “discs”

\[
\begin{align*}
&h_1, h_2, h_3 \in H \\
\end{align*}
\]

• \(H\) ensemble = vote together \(h_1, h_2, h_3\)

• Even if target concept \(\not\in H\), a mixture of hypothesis \(\in H\) might be highly accurate
Computational

- All learning algorithms do some sort of search through some space of hypotheses to find one that is “good enough” for the given training data
- Since interesting hypothesis spaces are huge/infinite, heuristic search is essential (e.g. decision tree learner does a greedy search in space of possible decision trees)
- So the learner might get stuck in a local minimum
- One strategy for avoiding local minima: repeat the search many times with random restarts ➔ bagging

Boosting - Summary

- Basic motivation – creating a committee of experts is typically more effective than trying to derive a single super-genius
- Boosting provides a simple and powerful method for turning weak learners into strong learners
- The simple algorithm described here has been extended to:
  - multi-class classification problems
  - classifiers that produce confidences associated with class predictions (e.g., posterior probabilities as opposed to class assignments)
  - Weak classifiers trained on subsets of attributes
  - Recent theoretical results have shown deep connections between boosting and maximizing margin of separation (similar to SVM)
Error correcting output codes (ECOC)

- So far, we've been building the ensemble by tweaking the distribution of training instances.
- ECOC involves tweaking the output (class) to be learned.

*Example: Handwritten number recognition*

```
7  4  3  5  2  ➡️  7, 4, 3, 5, 2
```

"obvious" approach: learn function: Scribble → {0,1,2,…,9} doesn’t work very well (too hard!)

What if we “decompose” the learning task into six “subproblems”?

```
<table>
<thead>
<tr>
<th>Class</th>
<th>Code Word</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0 0 0 1 0</td>
</tr>
<tr>
<td>1</td>
<td>1 0 0 0 0</td>
</tr>
<tr>
<td>2</td>
<td>0 1 1 0 1</td>
</tr>
<tr>
<td>3</td>
<td>0 0 0 1 0</td>
</tr>
<tr>
<td>4</td>
<td>1 0 0 0 0</td>
</tr>
<tr>
<td>5</td>
<td>0 0 0 0 0</td>
</tr>
<tr>
<td>6</td>
<td>0 0 0 0 1</td>
</tr>
<tr>
<td>7</td>
<td>0 0 0 0 1</td>
</tr>
<tr>
<td>8</td>
<td>0 0 0 1 0</td>
</tr>
<tr>
<td>9</td>
<td>0 0 1 1 0</td>
</tr>
</tbody>
</table>
```

**Abbreviation**
- vl: contains vertical line
- hl: contains horizontal line
- dl: contains diagonal line
- cc: contains closed curve
- ol: contains curve open to left
- or: contains curve open to right

```
1. learn an ensemble of classifiers, one specialized to each of the 6 “sub-problems”
2. to classify a new scribble, invoke each ensemble member, then predict the class whose code-word is closest (Hamming distance) to the predicted code.
```
Error-correcting codes

Suppose we want to send \( n \)-bit messages through a noisy channel. To ensure robustness to noise, we can map each \( n \)-bit message into an \( m \)-bit code (\( m > n \)) – note \(|\text{codes}| >> |\text{messages}|\)

When receive a code, translate it to message corresponding to the “nearest” (Hamming distance) code

Key to robustness: assign the codes so that each \( n \)-bit “clean” message is surrounded by a “buffer zone” of similar \( m \)-bit codes to which no other \( n \)-bit message is mapped.

\[ \text{blue} = \text{message (} n \text{ bits)} \]
\[ \text{yellow} = \text{code (} m \text{ bits)} \]
\[ \text{white} = \text{intended message} \]
\[ \text{red} = \text{received code} \]

ISBN

- The International Standard Book Number (ISBN) system identifies every book with a ten-digit number, such as 0-226-53420-0.
- The first nine digits are the actual number but the tenth is added according to a mathematical formula based on the first nine.
- If a single one of the digits is changed, as in a misprint when ordering a book, a simple check verifies that something is wrong.
Designing code-words for ECOC learning

Coding: $k$ labels $\rightarrow$ $m$ bit codewords

Good coding:

- row separation: want “assigned” codes to be well-separated by lots of “unassigned” codes

- column separation: each bit $i$ of the codes should be uncorrelated with all other bits $j$

Selecting good codes is hard!

<table>
<thead>
<tr>
<th>class</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Monday</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Tuesday</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Wednesday</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Thursday</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Friday</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Saturday</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Sunday</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Copyright Vasant Honavar, 2006.
Summary...

- **Ensembles**: basic motivation – creating a committee of experts is typically more effective than trying to derive a single super-genius
- **Key issues**:
  - Generation of base models
  - Integration of base models
- **Popular ensemble techniques**
  - manipulate training data: bagging and boosting (ensemble of “experts”, each specializing on different portions of the instance space)
  - Manipulate input feature space
  - manipulate output values: error-correcting output coding (ensemble of “experts”, each predicting 1 bit of the multibit full class label)