Ensemble methods

Machine learning competition with a $1 million prize
Bias/Variance Tradeoff

Hastie, Tibshirani, Friedman “Elements of Statistical Learning” 2001
Reduce Variance Without Increasing Bias

• **Averaging** reduces variance:

\[
\frac{Var(\bar{X})}{N} \quad \frac{Var(X)}{N}
\]

(when predictions are independent)

Average models to reduce model variance

One problem:
  
  only one training set
  
  where do multiple models come from?
Bagging: Bootstrap Aggregation

- Leo Breiman (1994)
- Take repeated bootstrap samples from training set $D$.
- **Bootstrap sampling**: Given set $D$ containing $N$ training examples, create $D'$ by drawing $N$ examples at random with replacement from $D$.

- Bagging:
  - Create $k$ bootstrap samples $D_1 \ldots D_k$.
  - Train distinct classifier on each $D_i$.
  - Classify new instance by majority vote / average.
Bagging

• Best case:
  \[ \text{Var}(\text{Bagging}(L(x, D))) \frac{\text{Variance}(L(x, D))}{N} \]

In practice:
  models are correlated, so reduction is smaller than 1/N
  variance of models trained on fewer training cases
  usually somewhat larger
Bagging Example
decision tree learning algorithm; very similar to ID3

CART decision boundary
100 bagged trees

shades of blue/red indicate strength of vote for particular classification
Reduce Bias$^2$ and Decrease Variance?

- Bagging reduces variance by averaging
- Bagging has little effect on bias
- Can we average \textit{and} reduce bias?
- Yes:

  - \textbf{Boosting}
Theory and Applications of Boosting

Rob Schapire
Example: “How May I Help You?”

[Gorin et al.]

• **goal**: automatically categorize type of call requested by phone customer (*Collect*, *CallingCard*, *PersonToPerson*, etc.)
  
  • yes I’d like to place a collect call long distance please (*Collect*)
  
  • operator I need to make a call but I need to bill it to my office (*ThirdNumber*)
  
  • yes I’d like to place a call on my master card please (*CallingCard*)
  
  • I just called a number in sioux city and I musta rang the wrong number because I got the wrong party and I would like to have that taken off of my bill (*BillingCredit*)

• **observation**:
  
  • easy to find “rules of thumb” that are “often” correct
    
    • e.g.: “IF ‘card’ occurs in utterance THEN predict ‘CallingCard’ ”
  
  • hard to find single highly accurate prediction rule
The Boosting Approach

- devise computer program for deriving rough rules of thumb
- apply procedure to subset of examples
- obtain rule of thumb
- apply to 2nd subset of examples
- obtain 2nd rule of thumb
- repeat $T$ times
Key Details

- **how to choose examples on each round?**
  - concentrate on “hardest” examples (those most often misclassified by previous rules of thumb)

- **how to combine rules of thumb into single prediction rule?**
  - take (weighted) majority vote of rules of thumb
Boosting

- **boosting** = general method of converting rough rules of thumb into highly accurate prediction rule

- technically:
  - assume given “weak” learning algorithm that can consistently find classifiers (“rules of thumb”) at least slightly better than random, say, accuracy $\geq 55\%$ (in two-class setting) [“weak learning assumption”]
  - given sufficient data, a boosting algorithm can provably construct single classifier with very high accuracy, say, 99%
Preamble: Early History
Strong and Weak Learnability

- boosting’s roots are in “PAC” learning model [Valiant ’84]
- get random examples from unknown, arbitrary distribution
- strong PAC learning algorithm:
  - for any distribution with high probability given polynomially many examples (and polynomial time) can find classifier with arbitrarily small generalization error
- weak PAC learning algorithm
  - same, but generalization error only needs to be slightly better than random guessing \((\frac{1}{2} - \gamma)\)
- [Kearns & Valiant ’88]:
  - does weak learnability imply strong learnability?
If Boosting Possible, Then...

• can use (fairly) wild guesses to produce highly accurate predictions
• if can learn “part way” then can learn “all the way”
• should be able to improve any learning algorithm
• for any learning problem:
  • either can always learn with nearly perfect accuracy
  • or there exist cases where cannot learn even slightly better than random guessing
First Boosting Algorithms

- [Schapire ’89]:
  - first provable boosting algorithm
- [Freund ’90]:
  - “optimal” algorithm that “boosts by majority”
- [Drucker, Schapire & Simard ’92]:
  - first experiments using boosting
  - limited by practical drawbacks
- [Freund & Schapire ’95]:
  - introduced “AdaBoost” algorithm
  - strong practical advantages over previous boosting algorithms
Application: Detecting Faces

- problem: find faces in photograph or movie
- weak classifiers: detect light/dark rectangles in image

- many clever tricks to make extremely fast and accurate
Basic Algorithm and Core Theory

• introduction to AdaBoost
• analysis of training error
• analysis of test error and the margins theory
• experiments and applications
Basic Algorithm and Core Theory

- introduction to AdaBoost
- analysis of training error
- analysis of test error and the margins theory
- experiments and applications
A Formal Description of Boosting

- given training set \((x_1, y_1), \ldots, (x_m, y_m)\)
- \(y_i \in \{-1, +1\}\) correct label of instance \(x_i \in X\)
- for \(t = 1, \ldots, T\):
  - construct distribution \(D_t\) on \(\{1, \ldots, m\}\)
  - find weak classifier ("rule of thumb") \(h_t : X \rightarrow \{-1, +1\}\)
    
    \[\epsilon_t = \Pr_{i \sim D_t}[h_t(x_i) \neq y_i]\]
    
    with error \(\epsilon_t\) on \(D_t\):
  - output final/combined classifier \(H_{\text{final}}\)
AdaBoost

- constructing $D_t$:
  - $D_1(i) = 1/m$
  - given $D_t$ and $h_t$:

$$D_{t+1}(i) = \frac{D_t(i)}{Z_t} \times \left\{ \begin{array}{ll}
e^{-\alpha_t} & \text{if } y_i = h_t(x_i) \\
e^{\alpha_t} & \text{if } y_i \neq h_t(x_i) \end{array} \right.$$ 

$$= \frac{D_t(i)}{Z_t} \exp(-\alpha_t y_i h_t(x_i))$$

where $Z_t =$ normalization factor

$$\alpha_t = \frac{1}{2} \ln \left( \frac{1 - \epsilon_t}{\epsilon_t} \right) > 0$$

- final classifier:

$$H_{\text{final}}(x) = \text{sign} \left( \sum_t \alpha_t h_t(x) \right)$$
Toy Example

$D_1$

weak classifiers $= \text{vertical or horizontal half-planes}$
Round 1

\[ h_1 \]

\[ \varepsilon_1 = 0.30 \]
\[ \alpha_1 = 0.42 \]
Round 2

\[ \alpha_2 = 0.65 \]

\[ \varepsilon_2 = 0.21 \]
Final Classifier

\[
H_{\text{final}} = \text{sign} \left( \begin{array}{c}
0.42 \\
+ 0.65 \\
+ 0.92
\end{array} \right)
\]
Voted combination of classifiers

- The general problem here is to try to combine many simple “weak” classifiers into a single “strong” classifier.

- We consider voted combinations of simple binary $\pm 1$ component classifiers:

$$h_m(x) = \alpha_1 h(x; \theta_1) + \ldots + \alpha_m h(x; \theta_m)$$

where the (non-negative) votes $\alpha_i$ can be used to emphasize component classifiers that are more reliable than others.
Components: decision stumps

- Consider the following simple family of component classifiers generating ±1 labels:

\[ h(x; \theta) = \text{sign}(w_1 x_k - w_0) \]

where \( \theta = \{k, w_1, w_0\} \). These are called decision stumps.

- Each decision stump pays attention to only a single component of the input vector.
Voted combination cont’d

• We need to define a loss function for the combination so we can determine which new component $h(x; \theta)$ to add and how many votes it should receive

$$h_m(x) = \alpha_1 h(x; \theta_1) + \ldots + \alpha_m h(x; \theta_m)$$

• While there are many options for the loss function we consider here only a simple exponential loss

$$\exp\{-y \cdot h_m(x)\}$$
Modularity, errors, and loss

- Consider adding the $m^{th}$ component:

\[
\sum_{i=1}^{n} \exp\left\{ -y_i \left[ h_{m-1}(x_i) + \alpha_m h(x_i; \theta_m) \right] \right\}
\]

\[
= \sum_{i=1}^{n} \exp\left\{ -y_i h_{m-1}(x_i) - y_i \alpha_m h(x_i; \theta_m) \right\}
\]
Modularity, errors, and loss

- Consider adding the $m^{th}$ component:

$$\sum_{i=1}^{n} \exp\left\{ -y_i [h_{m-1}(x_i) + \alpha_m h(x_i; \theta_m)] \right\}$$

$$= \sum_{i=1}^{n} \exp\left\{ -y_i h_{m-1}(x_i) - y_i \alpha_m h(x_i; \theta_m) \right\}$$

$$= \sum_{i=1}^{n} \underbrace{\exp\left\{ -y_i h_{m-1}(x_i) \right\}}_{\text{fixed at stage } m} \exp\left\{ -y_i \alpha_m h(x_i; \theta_m) \right\}$$
Modularity, errors, and loss

Consider adding the $m^{th}$ component:

$$\sum_{i=1}^{n} \exp\{-y_i[h_{m-1}(x_i) + \alpha_m h(x_i; \theta_m)]\}$$

$$= \sum_{i=1}^{n} \exp\{-y_i h_{m-1}(x_i) - y_i \alpha_m h(x_i; \theta_m)\}$$

$$= \sum_{i=1}^{n} \underbrace{\exp\{-y_i h_{m-1}(x_i)\}}_{\text{fixed at stage } m} \exp\{-y_i \alpha_m h(x_i; \theta_m)\}$$

$$= \sum_{i=1}^{n} W_i^{(m-1)} \exp\{-y_i \alpha_m h(x_i; \theta_m)\}$$

So at the $m^{th}$ iteration the new component (and the votes) should optimize a weighted loss (weighted towards mistakes).

Tommi Jaakkola, MIT CSAIL
Empirical exponential loss cont’d

- To increase modularity we’d like to further decouple the optimization of $h(x; \theta_m)$ from the associated votes $\alpha_m$.

- To this end we select $h(x; \theta_m)$ that optimizes the rate at which the loss would decrease as a function of $\alpha_m$.

\[
\frac{\partial}{\partial \alpha_m} \left. \sum_{i=1}^{n} W_i^{(m-1)} \exp\{-y_i \alpha_m h(x_i; \theta_m)\} \right|_{\alpha_m=0} = 
\]

\[
\left[ \sum_{i=1}^{n} W_i^{(m-1)} \exp\{-y_i \alpha_m h(x_i; \theta_m)\} \right] \cdot \left( - y_i h(x_i; \theta_m) \right) \left. \right|_{\alpha_m=0} 
\]

\[
= \left[ \sum_{i=1}^{n} W_i^{(m-1)} (- y_i h(x_i; \theta_m)) \right]
\]
Empirical exponential loss cont’d

• We find \( h(x; \hat{\theta}_m) \) that minimizes

\[
- \sum_{i=1}^{n} W_i^{(m-1)} y_i h(x_i; \theta_m)
\]

We can also normalize the weights:

\[
- \sum_{i=1}^{n} \frac{W_i^{(m-1)}}{\sum_{j=1}^{n} W_j^{(m-1)}} y_i h(x_i; \theta_m)
\]

\[
= - \sum_{i=1}^{n} \tilde{W}_i^{(m-1)} y_i h(x_i; \theta_m)
\]

so that \( \sum_{i=1}^{n} \tilde{W}_i^{(m-1)} = 1 \).
Selecting a new component: summary

• We find \( h(x; \hat{\theta}_m) \) that minimizes

\[
- \sum_{i=1}^{n} \tilde{W}_i^{(m-1)} y_i h(x_i; \theta_m)
\]

where \( \sum_{i=1}^{n} \tilde{W}_i^{(m-1)} = 1 \).

• \( \alpha_m \) is subsequently chosen to minimize

\[
\sum_{i=1}^{n} \tilde{W}_i^{(m-1)} \exp\{-y_i \alpha_m h(x_i; \hat{\theta}_m)\}
\]
The AdaBoost algorithm

0) Set $\tilde{W}_i^{(0)} = 1/n$ for $i = 1, \ldots, n$

1) At the $m^{th}$ iteration we find (any) classifier $h(x; \hat{\theta}_m)$ for which the weighted classification error $\epsilon_m$

$$
\epsilon_m = 0.5 - \frac{1}{2} \left( \sum_{i=1}^{n} \tilde{W}_i^{(m-1)} y_i h(x_i; \hat{\theta}_m) \right)
$$

is better than chance.

2) The new component is assigned votes based on its error:

$$
\hat{\alpha}_m = 0.5 \log \left( \frac{(1 - \epsilon_m)}{\epsilon_m} \right)
$$

3) The weights are updated according to ($Z_m$ is chosen so that the new weights $\tilde{W}_i^{(m)}$ sum to one):

$$
\tilde{W}_i^{(m)} = \frac{1}{Z_m} \cdot \tilde{W}_i^{(m-1)} \cdot \exp \{ -y_i \hat{\alpha}_m h(x_i; \hat{\theta}_m) \}
$$