Homework

• Project: CoNLL 2004 shared task

• Homework for next week:
  ★ Download code and data
  ★ Modify Erik’s baseline script to handle double object constructions (or write your own)
  ★ Read Manning and Schütze, pp. 597–604
Committee machines

- Committee machines (or ensemble machines) combine the predictions of more than one classifier.

- Committee machines depend on the members being reasonably accurate (better than guessing) and diverse (errors are uncorrelated).

- Majority vote and weighted majority are simple ensemble methods.

- Committee machines reduce statistical error, computational error, and representational error.
Bagging

- Given training data and a learning method, how to we generate a diverse committee of classifiers?

- We can inject randomness into the learning procedure by varying initial conditions or hyperparameters

- **Bagging** (bootstrap aggregation): randomly generate lots (25–200) of training sets by sampling the original training data with replacement majority vote

- Reduces variance, so most effective for high variance, low bias classifiers
Error-correcting output coding

- Error-correcting output codes are redundant encodings which allow reliable data transmission in the presence of noise.
- They let us generate diverse classifiers by manipulating the target function.
- Exhaustive codes work for a small number of class (3–10), otherwise random codes perform well on average.
- ECOC classification reduces bias and variance.
- Can be combined with other methods (e.g., bagging) to reduce error rate even further.
Boosting

• Boosting iteratively increases the performance of a weak base learner

• AdaBoost (Adaptive Boosting) constructs a series of classifiers by re-weighting the training data

• Weight for misclassified items are increased and for correctly classified items are decreased

• The training distribution is shifted to emphasize the ‘hard’ cases

• Final result is a majority vote, weighted by accuracy
AdaBoost

- Initialize weights $w_i$ to $1/N$, $i = 1, \ldots, N$

- For $m = 1$ to $M$
  - fit a classifier $G_m(X)$ to $X$ using weights $w_i^{(m)}$
  - compute training error: $\epsilon_m = \sum w_i^{(m)} I(y_i \neq G_m(x_i))$
  - compute $\alpha_m = \frac{1}{2} \log \frac{1-\epsilon_m}{\epsilon_m}$
  - set new weights:
    $$ w_i^{(m+1)} = \frac{w_i^{(m)} \exp[-\alpha_m y_i G_m(x_i)]}{Z_m}, \quad i = 1, \ldots, N $$

- Return
  $$ G(x) = \text{sign} \left[ \sum_m \alpha_m G_m(x) \right] $$
AdaBoost

- The training error of $G$ is bounded by:

$$\frac{1}{N} \sum I(y_i \neq G(x_i)) \leq \frac{1}{N} \sum \exp[-y_i \sum \alpha_m G_m(x_i)]$$

$$\leq \exp[-2 \sum \gamma_m^2]$$

where

$$\gamma_m = \frac{1}{2} - \epsilon_m$$

- As long as $\gamma_m \leq \frac{1}{2}$, the overall error rate will drop exponentially
AdaBoost

- Weak learners can be very simple (decision stumps)
- AdaBoost performs extremely well, sometimes considered the best black-box learning method
- Why it works so well has been a bit of a mystery, spurring some important theoretical advances
- Extensions to multiclass classifiers and regression
- Demo applet
AdaBoost

- AdaBoost continues to decrease generalization error even after training error is zero.

- The *margin* is a measure of the confidence in a classification:

\[
\text{margin}_G(x, y) = \frac{y \sum \alpha_m G_m(x)}{\sum |\alpha_m|}
\]

- Shapire, et al. (1998) show that for any \( \theta > 0 \) the generalization error is at most:

\[
R[G] \leq P[\text{margin}_G(x, y) \leq \theta] + \tilde{O} \left( \sqrt{\frac{d}{N\theta^2}} \right)
\]

- This bound doesn’t depend on \( M \), so extra iterations need not increase the generalization error.
What is AdaBoost really doing?

Consider an alternative algorithm: forward stagewise additive modeling

- Initialize $f_0(x) = 0$
- For $m = 1$ to $M$
  - Compute
    \[(\alpha_m, \gamma_m) = \underset{\alpha, \gamma}{\text{argmin}} \sum_i L(y_i, f_{m-1}(x_i) + \alpha b(x_i; \gamma_i))\]
  
  - Set $f_m = f_{m-1}(x) + \alpha_m b(x; \gamma_m)$

This will construct a function $f$ which is a sum of basis functions $b(x; \gamma)$ and minimizes the loss function $L$
Exponential loss

- Suppose the basis functions are our weak classifiers $G_i$, and the loss function is:

$$L(y, f(x)) = \exp(-y f(x))$$

- Then, the update becomes:

$$(\alpha_m, G_m) = \arg\min_{\alpha, G} \sum_i \exp[-y_i(f_{m-1}(x_i) + \alpha G(x_i))]$$

$$= \arg\min_{\alpha, G} \sum_i w_i \exp(-\alpha y_i G(x_i))$$

where

$$w_i = \exp(-y_i f_{m-1}(x_i))$$
Exponential loss

- To solve this, note that:
  \[
  \sum_i w_i \exp(-\alpha y_i G(x_i)) = e^{-\alpha} \sum_{y_i=G(x_i)} w_i + e^\alpha \sum_{y_i \neq G(x_i)} w_i
  \]
  \[
  = (e^\alpha - e^{-\alpha}) \sum_i w_i I(y_i \neq G(x_i)) + e^{-\alpha} \sum_i w_i
  \]

- So, minimizing with respect to $G_m$ gives us:
  \[
  G_m = \arg\min_G \sum_i w_i I(y_i \neq G(x_i))
  \]

- Now if we solve for $\alpha$, we get (where $e_m$ is the weighted training error rate of $G_m$):
  \[
  \alpha_m = \frac{1}{2} \log \frac{1 - e_m}{e_m}
  \]
Exponential loss

• Now we update the $f$:

$$f_m(x) = f_{m-1}(x) + \alpha_m G_m(x)$$

• This means the weights on the next round will be:

$$w_i^{(m+1)} = w_i^{(m)} \exp[-\alpha_m y_i G_m(x_i)]$$

• ...which is AdaBoost!

• So, AdaBoost is a greedy algorithm to construct an additive combination of $G$’s which minimizes the exponential loss:

$$L(y, f(x)) = \exp(-y f(x))$$
Exponential loss

- Why exponential loss? Mainly, because it’s an upper bound on training error which is easy to work with.

- Also, it can be shown that:

\[
\frac{1}{2} \log \frac{P(Y = 1 | x)}{P(Y = -1 | x)} = \arg\min_{f(x)} E_{Y|x}[\exp(-Yf(x))]
\]

- By minimizing the exponential loss, we’re estimating (one half of) the log odds of \(P(Y = 1 | x)\).

- Also, note that:

\[
P(Y = 1 | x) = \frac{\exp f(x)}{\exp -f(x) + \exp f(x)} = \frac{1}{1 + \exp -2f(x)}
\]
Exponential loss

- This equivalence shows how boosting relates to other machine learning methods

- It also points to ways AdaBoost could be improved

- Minimizing the exponential loss is the same as maximizing the (population) log likelihood:

\[ L(Y, f(x)) = -\log(1 + \exp(-2Yf(x))) \]

- A new algorithm LogitBoost maximizes this log likelihood using Newton-style updates (Friedman, Hastie, and Tibshirani 2000)
Interpreting models

- One strength of tree-based classifiers is their simple structure, which makes them easy to interpret.

- In data mining applications, we are often more interested in the structure that a classifier discovers in the training data than in the accuracy of the rule itself.

- Committee machines are virtually impossible to interpret, even if the base learners are simple.

- We can measure the relative importance of a variable by summing the estimated error reduction at each node, averaged over all trees in an ensemble.
Boosting

- Boosting is a very powerful method, which reduces bias and variance, produces very lower error rates, and is highly resistant to overtraining.

- Boosting finds an additive collection of basis functions which minimizes the exponential loss (and therefore training error).

- Boosting increase the margin, so generalization improves even after the training error reaches zero.

- Boosting is more closely related to logistic regression and maximum entropy models than to other committee methods.

- There are other things that boosting turns out to be the same as (game theoretic interpretations, etc.).
Committee machines

- Bagging and boosting are the best known committee machine methods, but there are others

- *Twicing* (Tukey 1977) fits a model to the data, then fits a model to the errors

- *Stacking* (Wolpert 1992) uses leave-one-out cross validation to construct and weight models (*jackknife*)

- *Bumping* (Tibshirani and Knight 1999) is like bagging, but uses the best single model rather than averaging all of them

- *Arcing* (Breiman 1998) is a hybrid of bagging and boosting

- *Random forests* (Breiman 1999) are a collection of trees built using randomly selected subsets of features
A variant of the committee machine model can be used to take advantage of unannotated or semi-annotated data.

Simple self-training uses the output of a classifier as the input to the next round of training (helps a little, but not much).

If we have more than one classifier, we can improve this:

- Construct classifiers using training data
- Classify unannotated examples
- If enough learners agree on the classification of an example, add it to the training set
- Repeat.
Co-training (Blum and Mitchell 1998) splits features into independent subsets:

- Train two classifiers using these independent feature sets, to get two independent classifiers $A$ and $B$
- Use $A$ to classify unlabeled data, and collect positive and negative examples with the highest classification confidence
- Do the same with $B$, and add the newly labeled data to the training sample
- Repeat

- Use a combination of $A$ and $B$ as final classifier
Co-training

- When the independent feature sets satisfy some basic assumptions, co-training can improve a weak initial hypothesis to arbitrary accuracy.

- Even when feature sets aren’t independent (e.g., randomly selected words in a bag-of-words model), co-training works well.

- All these methods can be improved significantly by *active learning*.

- These semi-supervised learning methods allow more efficient use of (expensive, slow) human annotators.
Committee machines

● Combinations of classifiers can perform better than any one classifier, so long as:
  ✶ each classifier is more accurate than randomly guessing
  ✶ the errors made by each classifier are uncorrelated

● Various strategies for producing useful committees

● Committee machines reduce variance, and sometimes reduce bias as well

● Some committee methods are related to other classifiers in interesting ways

● Committee methods can also be used to take advantage of unannotated or partially annotated data
Generalization

- What happened to the curse of dimensionality?
- Dimensionality (as number of features) has no clear interpretation for complex modeling procedures
- And what about simplicity? Didn’t we say simpler = lower variance?
- How is an ensemble of trees simpler than a single tree?
- How is MaxEnt simpler than Naive Bayes?
- How is MaxEnt+prior simpler than MaxEnt?
Generalization

- Notions like dimensionality and simplicity aren’t really what we’re interested in.

- Dimensionality reduction and simplification are meant to improve generalization – the ability to abstract away from accidental details of a training sample.

- A better way to get at that is by restricting capacity.

- Deleting features or simplifying models may (or may not) reduce the representational capacity of a model.
Generalization

- Take MaxEnt models with a Gaussian prior:

$$\lambda^* = \arg\max_{\lambda} L(\lambda) - \frac{1}{2\sigma^2} \sum_i \lambda_i^2$$

- This is equivalent to:

$$\lambda^* = \arg\max_{\lambda} L(\lambda)$$

where

$$\sum_i \lambda_i^2 \leq s^2$$

- The solution is the point which maximizes $L$ and falls inside the hypersphere with radius $s$

- The prior reduces the capacity of the model, which (empirically) improves generalization