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

Generating diversity

- 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

Generating diversity

- Another strategy is to select disjoint subsets of features
- Reduces dimensionality for each classifier (important for classifiers like neural nets)
- Each resulting classifier must still be fairly accurate, so only works if features can be broken down into independent subsets
- Volcano identification (Cherkauer 1996) – 32 neural nets using 8 subsets of 119 features × 4 network sizes
- Not especially useful for NLP (but: cotraining)

Generating diversity

- We can also generate multiple classifiers by manipulating their target functions
- Error-correcting output codes are redundant encodings which allow reliable data transmission in the presence of noise
- Classifications are a kind of data transmission, and classification errors are a kind of noise
- We can use error-correcting strategies to reduce classification error (Dietterich and Bakari 1995)
Suppose we have a four-class problem. We could construct a binary classifier for each class (one-per-class method):

\[
\begin{align*}
c_1 & : 1000 \\
c_2 & : 0100 \\
c_3 & : 0010 \\
c_4 & : 0001 \\
\end{align*}
\]

This can, of course, be reduced (distributed coding):

\[
\begin{align*}
c_1 & : 00 \\
c_2 & : 10 \\
c_3 & : 01 \\
c_4 & : 11 \\
\end{align*}
\]

We can also produce a redundant code, using 10 binary classifiers:

\[
\begin{align*}
c_1 & : 1010011100 \\
c_2 & : 0100011101 \\
c_3 & : 1001100011 \\
c_4 & : 1011111010 \\
\end{align*}
\]

If the bitstring produced by a test example doesn’t match one of these codes exactly, choose the one with the fewest number of different bits (Hamming distance).

For example, if our 10 binary classifiers produced the bitstring:

\[1010011101\]

we would predict class \(c_4\).

If the minimum Hamming distance between any two classes is \(d\), then the encoding can correct at least \(\left\lfloor \frac{d-1}{2} \right\rfloor\) single-bit errors.

In the one-per-class case \(d = 2\), and for the minimal encoding \(d = 1\) (no error correction!)

In the redundant code \(d = 4\), so we can survive at least one error and still get the answer right.

To apply ECOC classification, we need to produce a suitable encoding.

\(\text{Row separation}\) increases the minimum Hamming distance between codes, and so increases the number of errors which can be corrected.

\(\text{Column separation}\) increases the independence of the individual classifiers, reducing the probability of simultaneous errors.

Exhaustive code for \(k\) classes uses \(2^{k-1} - 1\) bits.

First class gets a code of all 1’s.

The \(i\)th class gets alternating runs of \(2^{k-i}\) zeros and ones.
Error-correcting output coding

- Exhaustive code for five classes:
  
  \[ \begin{align*}
  c_1 & : 1111111111111111 \\
  c_2 & : 000000011111111 \\
  c_3 & : 000111100001111 \\
  c_4 & : 001100110011001 \\
  c_5 & : 010101010101010 
  \end{align*} \]

- Minimum Hamming distance is 8, and no two columns are identical or complementary

- Only really practical for \( 3 < k \leq 7 \)

- For medium-sized problems (\( 8 < k \leq 11 \)), a subset of columns from an exhaustive code will work well

- For longer codes, randomly generated bitstrings of length \( L \) will have an average Hamming distance of \( \frac{L}{2} \) (which can be heuristically improved)

- Sensible code lengths for \( k \) classes range from \( \log_2 k \) to \( \frac{2^k - 1}{2} \)

- Assignment of classes to codes doesn’t matter

Error-correcting output coding

- Each classifier learns a subset of the interclass boundaries

- Each boundary is learned by many classifiers

- Decoding is a kind of voting (and so reduces variance)

- ECOC decoding also apparently reduces bias

- Can be combined with other methods (e.g., bagging) to reduce error rate even further

- Applied to text classification using naive Bayes (Berger 1999):
The committee methods we’ve seen so far take reasonably accurate single classifiers and improve them.

How accurate do the base members need to be?

Weak vs. strong (“probably approximately correct”) learners

Hypothesis Boosting Problem (Valiant 1989) – does the existence of a weak learner imply the existence of a stronger learner?

Shapire (1990) offered an algorithm for boosting weak learners to strong learners.

Shapire’s proto-boosting algorithm forms the basis for a series of ensemble methods.

First, train a weak learner $W_1$ on $N_1$ examples.

Next, use that learner to filter new training examples:

$\checkmark$ Flip a coin

$\checkmark$ If heads, use $W_1$ to classify new examples until one is misclassified, and add that to a new training set

$\checkmark$ If tails, use $W_1$ to classify new examples until one is correctly classified, and add that

$\checkmark$ Continue until $N_1$ examples have been collected.

Now we’ve got a second training set (on which $W_1$ will have an error rate of 50%) that we use to train a second weak learner $W_2$.

Finally, collect $N_1$ examples on which $W_1$ and $W_2$ disagree, and use them to train $W_3$.

To classify a new example, send it to $W_1$ and $W_2$. If they don’t agree, use $W_3$.

If the error rate of the weak learners is $\epsilon$, the error rate of the ensemble is bounded by:

$$g(\epsilon) = 3\epsilon^2 - 2\epsilon^3$$

By applying the procedure recursively, the error rate can be made arbitrarily small, thus converting a weak learner into a strong learner.

In practice, this will require vast quantities of training data and so isn’t very practical.
Boosting

- A family of boosting algorithms improved on Shapire’s (1990) filtering algorithm
- What boosting methods have in common is that they train weak learners on different training distributions
- Unlike bagging, boosting lowers both the bias and the variance
- So, boosting may be helpful with a wide range of base learners,
- Most popular is decision stumps (high bias, low variance)

AdaBoost

- Freund and Schapire (1997) present AdaBoost (what most people mean when they refer to “boosting”)
- Suppose we have a two-class problem, with features $X$ and classes $Y \in \{-1, 1\}$, and a classifier $G : X \rightarrow \{-1, 1\}$
- The training error of the classifier is:

$$\epsilon = \frac{1}{N} \sum I(y_i \neq G(x_i))$$

**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:

$$w_i^{(m+1)} = \frac{w_i^{(m)} \exp\left[-\alpha_m y_i G_m(x_i)\right]}{Z_m}, \ i = 1, \ldots, N$$
- 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

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

Looking at training error is interesting, but doesn’t tell us everything

What we really want to know is how well the model generalizes to new data

An intuition: simple hypotheses generalize better from training data than complex hypotheses

How do we measure the complexity of a hypothesis space?

The key notion is the Vapnik-Chernovenkis (VC) dimension a measure of how twisty the decision boundaries in a hypothesis space can be
The number of model parameters is a rough measure of complexity: more parameters = more complex

But compare a linear boundary \( a_0 + a_1 x \geq 0 \), with two parameters, to a boundary like \( \sin(\alpha x) \geq 0 \), with one

The VC dimension of a class of functions \( \{f(x, \alpha)\} \) is the largest number of points which can be shattered by members of \( \{f(x, \alpha)\} \)

A set of points is shattered be a class of functions if all possible binary class assignments can be perfectly separated by a member of the class

In general, a hyperplane in \( r \) dimensions has a VC dimension of \( r + 1 \)

The class \( \{\sin(\alpha x)\} \) can shatter any number of points, and has an infinite VC dimension

Given a classifier with a VC dimension of \( h \) and \( N \) training examples, then is \( h < N \) we have with probability \( 1 - \delta \):

\[
R[G] \leq R_{emp}[G] + \sqrt{\frac{1}{N} \left( h \left( \log \frac{2N}{h} + 1 \right) + \log \frac{4}{\delta} \right)}
\]

Note that this increases with \( h \) and decreases with \( N \)
AdaBoost

- Back to AdaBoost, we have:

\[ R[G] \leq R_{emp}[G] + \hat{O}\left(\sqrt{\frac{Md}{N}}\right) \]

where \( M \) is the number of iterations, \( d \) is the VC dimension of the base learner, and \( N \) is the number of training examples.

- So, to limit generalization error, we should limit \( M \) and \( d \).

- But, a puzzle: sometimes with AdaBoost, generalization error continues to fall even when training error has reached zero!

Extra iterations can increase margins even after training error is zero.

This bound doesn’t depend on \( M \), so extra iterations need not increase the generalization error.

What is AdaBoost really doing?

- It constructs a linear combination of base learners \( G_m \) which minimizes the **exponential loss**:

\[
L(y, G(x)) = \exp[-yG(x)]
\]

\[ = \exp\left[-y \sum \alpha_m G_m(x)\right] \]

So, AdaBoost is a species of **forward stagewise additive modeling**.

More to come on this…

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:
  - Bagging is especially useful for reducing variance in high variance/low bias classifiers (like decision trees)
  - Error-correcting output coding extends binary classifiers to multi-class problems, and reduces both bias and variance
  - Boosting is a very powerful method, which reduces bias and variance, produces very lower error rates, and is highly resistant to overtraining