Goals for the lecture

you should understand the following concepts

- ensemble
- bootstrap sample
- bagging
- boosting
- random forests
- error correcting output codes
What is an ensemble?

A set of learned models whose individual decisions are combined in some way to make predictions for new instances.
When can an ensemble be more accurate?

- when the errors made by the individual predictors are (somewhat) uncorrelated, and the predictors’ error rates are better than guessing (< 0.5 for 2-class problem)
- consider an idealized case…

![Figure 1. The Probability That Exactly \( \ell \) (of 21) Hypotheses Will Make an Error, Assuming Each Hypothesis Has an Error Rate of 0.3 and Makes Its Errors Independently of the Other Hypotheses.](image)

Figure from Dietterich, *AI Magazine*, 1997
How can we get diverse classifiers?

- In practice, we can’t get classifiers whose errors are completely uncorrelated, but we can encourage diversity in their errors by
  - choosing a variety of learning algorithms
  - choosing a variety of settings (e.g. # hidden units in neural nets) for the learning algorithm
  - choosing different subsamples of the training set (*bagging*)
  - using different probability distributions over the training instances (*boosting*)
  - choosing different features and subsamples (*random forests*)
Bagging (Bootstrap Aggregation)

learning:
given: learner $L$, training set $D = \{ \langle x^{(1)}, y^{(1)} \rangle \ldots \langle x^{(m)}, y^{(m)} \rangle \}$
for $i \leftarrow 1$ to $T$ do
  $D_i \leftarrow m$ instances randomly drawn with replacement from $D$
  $h_i \leftarrow$ model learned using $L$ on $D_i$

classification:
given: test instance $x$
predict $y \leftarrow$ plurality_vote($h_1(x) \ldots h_T(x)$)

regression:
given: test instance $x$
predict $y \leftarrow$ mean($h_1(x) \ldots h_T(x)$)
Bagging

- each sampled training set is a *bootstrap replicate*
  - contains \( m \) instances (the same as the original training set)
  - on average it includes 63.2% of the original training set
  - some instances appear multiple times

- can be used with any base learner

- works best with *unstable* learning methods: those for which small changes in \( D \) result in relatively large changes in learned models
Empirical evaluation of bagging with C4.5

Bagging reduced error of C4.5 on most data sets; wasn’t harmful on any
Boosting

• Boosting came out of PAC learning analysis

• A weak PAC learning algorithm is one that cannot PAC learn for arbitrary $\varepsilon$ and $\delta$, although its hypotheses are slightly better than random guessing

• Suppose we have a weak PAC learning algorithm $L$ for a concept class $C$. Can we use $L$ as a subroutine to create a strong PAC learner for $C$?
  • Yes, by boosting! [Schapire, *Machine Learning* 1990]
  • The original boosting algorithm was of theoretical interest, but assumed an unbounded source of training instances

• A later boosting algorithm, AdaBoost, has had notable practical success
AdaBoost


given: learner \( L \), # stages \( T \), training set \( D = \{ \langle x^{(1)}, y^{(1)} \rangle \ldots \langle x^{(m)}, y^{(m)} \rangle \} \)

for all \( i \): \( w_1(i) \leftarrow 1/m \)  // initialize instance weights
for \( t \leftarrow 1 \) to \( T \) do
    for all \( i \): \( p_t(i) \leftarrow w_t(i) / (\Sigma_j w_t(j)) \)  // normalize weights
    \( h_t \leftarrow \) model learned using \( L \) on \( D \) and \( p_t \)
    \( \varepsilon_t \leftarrow \Sigma_i p_t(i)(1 - \delta(h_t(x^{(i)}), y^{(i)})) \)  // calculate weighted error
    if \( \varepsilon_t > 0.5 \) then
        \( T \leftarrow t - 1 \)
        break
    \( \beta_t \leftarrow \varepsilon_t / (1 - \varepsilon_t) \)
    for all \( i \) where \( h_t(x^{(i)}) = y^{(i)} \)  // down-weight correct examples
        \( w_{t+1}(i) \leftarrow w_t(i) \beta_t \)

return:

\[
h(x) = \arg\max_y \sum_{t=1}^{T} \left( \log \frac{1}{\beta_t} \right) \delta(h_t(x), y)
\]
Implementing weighted instances with AdaBoost

- AdaBoost calls the base learner $L$ with probability distribution $p_t$ specified by weights on the instances.

- There are two ways to handle this:
  1. Adapt $L$ to learn from weighted instances; straightforward for decision trees and naïve Bayes, among others.
  2. Make a large ($>> m$) unweighted set of instances by replicating each instance many times; sample this set according to $p_t$; run $L$ in the ordinary manner.
AdaBoost variants

• AdaBoost.M1: 1-of-n multiclass tasks

• AdaBoost.M2: arbitrary multiclass tasks

• AdaBoost.R: regression

• confidence-rated predictions (learners output their confidence in predicted class for each instance)

• etc.
Empirical evaluation of boosting with C4.5

Figure from Dietterich, *AI Magazine*, 1997
Bagging and boosting with C4.5

Figure from Dietterich, AI Magazine, 1997
Empirical study of bagging vs. boosting [Opitz & Maclin, JAIR 1999]

- 23 data sets
- C4.5 and neural nets as base learners
- bagging almost always better than single decision tree or neural net
- boosting can be much better than bagging
- however, boosting can sometimes reduce accuracy (too much emphasis on outliers?)
Random forests
[Breiman, Machine Learning 2001]

given: candidate feature splits $F$, training set $D = \{ \langle x^{(1)}, y^{(1)} \rangle \ldots \langle x^{(m)}, y^{(m)} \rangle \}$
for $i \leftarrow 1$ to $T$ do
    $D_i \leftarrow m$ instances randomly drawn with replacement from $D$
    $h_i \leftarrow$ randomized decision tree learned with $F$, $D_i$

randomized decision tree learning:
to select a split at a node
    $R \leftarrow$ randomly select (without replacement) $f$ feature splits from $F$
    (where $f \ll |F|$ )
    choose the best feature split in $R$
do not prune trees

classification/regression:
as in bagging
One large-scale empirical study
[Fernández-Delgado *JMLR* 2014]

- compared 179 classifiers on 121 data sets
- random forest was the best family of classifiers (3 classifiers in the top 5)

<table>
<thead>
<tr>
<th>Rank</th>
<th>Acc.</th>
<th>κ</th>
<th>Classifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>32.9</td>
<td>82.0</td>
<td>63.5</td>
<td>parRF.t (RF)</td>
</tr>
<tr>
<td>33.1</td>
<td>82.3</td>
<td>63.6</td>
<td>rf.t (RF)</td>
</tr>
<tr>
<td>36.8</td>
<td>81.8</td>
<td>62.2</td>
<td>svm.C (SVM)</td>
</tr>
<tr>
<td>38.0</td>
<td>81.2</td>
<td>60.1</td>
<td>svmPoly.t (SVM)</td>
</tr>
<tr>
<td>39.4</td>
<td>81.9</td>
<td>62.5</td>
<td>rforest.R (RF)</td>
</tr>
<tr>
<td>39.6</td>
<td>82.0</td>
<td>62.0</td>
<td>elm_kernel.m (NNET)</td>
</tr>
<tr>
<td>40.3</td>
<td>81.4</td>
<td>61.1</td>
<td>svmRadialCost.t (SVM)</td>
</tr>
<tr>
<td>42.5</td>
<td>81.0</td>
<td>60.0</td>
<td>svmRadial.t (SVM)</td>
</tr>
<tr>
<td>42.9</td>
<td>80.6</td>
<td>61.0</td>
<td>C5.0.t (BST)</td>
</tr>
<tr>
<td>44.1</td>
<td>79.4</td>
<td>60.5</td>
<td>avNNet.t (NNET)</td>
</tr>
<tr>
<td>45.5</td>
<td>79.5</td>
<td>61.0</td>
<td>nnet.t (NNET)</td>
</tr>
<tr>
<td>47.0</td>
<td>78.7</td>
<td>59.4</td>
<td>pcaNNet.t (NNET)</td>
</tr>
<tr>
<td>47.1</td>
<td>80.8</td>
<td>53.0</td>
<td>BG_LibSVM.w (BAG)</td>
</tr>
<tr>
<td>47.3</td>
<td>80.3</td>
<td>62.0</td>
<td>mlp.t (NNET)</td>
</tr>
<tr>
<td>47.6</td>
<td>80.6</td>
<td>60.0</td>
<td>RotationForest.w (RF)</td>
</tr>
<tr>
<td>50.1</td>
<td>80.9</td>
<td>61.6</td>
<td>RRF.t (RF)</td>
</tr>
<tr>
<td>51.6</td>
<td>80.7</td>
<td>61.4</td>
<td>RRFglobal.t (RF)</td>
</tr>
<tr>
<td>52.5</td>
<td>80.6</td>
<td>58.0</td>
<td>MAB_LibSVM.w (BST)</td>
</tr>
<tr>
<td>52.6</td>
<td>79.9</td>
<td>56.9</td>
<td>LibSVM.w (SVM)</td>
</tr>
<tr>
<td>57.6</td>
<td>79.1</td>
<td>59.3</td>
<td>adaboost.R (BST)</td>
</tr>
</tbody>
</table>
One application of random forests: human pose recognition in the Xbox Kinect

[Shotton et al., CVPR 2011]

Classification task

- Given: a depth image
- Do: classify each pixel into one of 31 body parts
Bias/variance and ensembles

- bagging & random forests work mainly by reducing variance

- boosting works by
  - primarily reducing bias in the early stages
  - primarily reducing variance in latter stages

- there is also a margin-maximization interpretation for why boosting works
Learning models for multi-class problems

- consider a learning task with \( k > 2 \) classes
- with some learning methods, we can learn one model to predict the \( k \) classes

- an alternative approach is to learn \( k \) models; each represents one class vs. the rest

- but we could learn models to represent other encodings as well
Error correcting output codes
[Dietterich & Bakiri, *JAIR* 1995]

- ensemble method devised specifically for problems with many classes
  - represent each class by a multi-bit code word
  - learn a classifier to represent each bit function

```
<table>
<thead>
<tr>
<th>Class</th>
<th>$f_0$</th>
<th>$f_1$</th>
<th>$f_2$</th>
<th>$f_3$</th>
<th>$f_4$</th>
<th>$f_5$</th>
<th>$f_6$</th>
<th>$f_7$</th>
<th>$f_8$</th>
<th>$f_9$</th>
<th>$f_{10}$</th>
<th>$f_{11}$</th>
<th>$f_{12}$</th>
<th>$f_{13}$</th>
<th>$f_{14}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>9</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>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
```
Classification with ECOC

- to classify a test instance $x$ using an ECOC ensemble with $T$ classifiers
  1. form a vector $h(x) = \langle h_1(x), \ldots, h_T(x) \rangle$ where $h_i(x)$ is the prediction of the model for the $i^{th}$ bit
  2. find the codeword $c$ with the smallest Hamming distance to $h(x)$
  3. predict the class associated with $c$

- if the minimum Hamming distance between any pair of codewords is $d$, we can still get the right classification with $\left\lceil \frac{d-1}{2} \right\rceil$ single-bit errors

recall, $\lfloor x \rfloor$ is the largest integer not greater than $x$
Error correcting code design

a good ECOC should satisfy two properties

1. *row separation*: each codeword should be well separated in Hamming distance from every other codeword

2. *column separation*: each bit position should be uncorrelated with the other bit positions

\[
d = 7 \quad \text{so this code can correct} \quad \left\lfloor \frac{7 - 1}{2} \right\rfloor = 3 \text{ errors}
\]
ECOC evaluation with C4.5

Figure from Bakiri & Dietterich, JAIR, 1995
ECOC evaluation with neural nets

Figure from Bakiri & Dietterich, JAIR, 1995
(Functional) Gradient Boosting

• Consider learning a regression tree to minimize squared error.

• Boosting adds a new tree (or model of any base learner type) to fix current errors, by reweighting wrongly-predicted examples; Breiman realized could just fit next tree to current residuals.

  • Current model: \( F(x) = w_1F_1(x) + \ldots + w_nF_n(x) \)

  • Each example \((x_i, y_i)\) now becomes \((x_i, r_i)\), where \(r_i = y_i - F(x_i)\).

• Friedman, Bartlett, others saw residual \( y_i - F(x_i) \) is just gradient of squared error loss \( \frac{1}{2}(y_i-F(x_i))^2 \) with respect to \( F(x_i) \); in general, can fit next model to negative gradient of any loss function if can efficiently find a model aligned with negative gradient of that loss.
Gradient Boosting with Squared Error (from Friedman, 1999)

\[ F_0(x) = \bar{y} \]

For \( m = 1 \) to \( M \) do:

\[ \tilde{y}_i = y_i - F_{m-1}(x_i), \quad i = 1, N \]

\[ (\rho_m, a_m) = \arg \min_{a, \rho} \sum_{i=1}^{N} [\tilde{y}_i - \rho h(x_i; a)]^2 \]

\[ F_m(x) = F_{m-1}(x) + \rho_m h(x; a_m) \]

endFor
• Friedman, Hastie, collaborators realized that once you learned the next tree, instead of fitting one best coefficient (weight) to the tree, why not re-fit a whole vector of coefficients, one per leaf

• This is tree boost, algorithm on next slide (slide from Hastie, 1999)
1. Initialize $f_0(x) = \arg \min_\gamma \sum_{i=1}^N L(y_i, \gamma)$.

2. For $m = 1$ to $M$:
   (a) For $i = 1, 2, \ldots, N$ compute
   \[
   r_{im} = - \left[ \frac{\partial L(y_i, f(x_i))}{\partial f(x_i)} \right]_{f = f_{m-1}}.
   \]
   (b) Fit a regression tree to the targets $r_{im}$ giving terminal regions $R_{jm}$, $j = 1, 2, \ldots, J_m$.
   (c) For $j = 1, 2, \ldots, J_m$ compute
   \[
   \gamma_{jm} = \arg \min_\gamma \sum_{x_i \in R_{jm}} L(y_i, f_{m-1}(x_i) + \gamma).
   \]
   (d) Update
   \[
   f_m(x) = f_{m-1}(x) + \sum_{j=1}^{J_m} \gamma_{jm} I(x \in R_{jm}).
   \]

3. Output $\hat{f}(x) = f_M(x)$. 

29
Other Ensemble Methods

- Use different parameter settings with same algorithm
- Use different learning algorithms
- Instead of voting or weighted voting, learn the combining function itself
  - Called “Stacking”
  - Higher risk of overfitting
  - Ideally, train arbitrator function on different subset of data than used for input models
- Naïve Bayes is weighted vote of stumps
Comments on ensembles

• They very often provide a boost in accuracy over base learner
• It’s a good idea to evaluate an ensemble approach for almost any practical learning problem
• They increase runtime over base learner, but compute cycles are usually much cheaper than training instances
• Some ensemble approaches (e.g. bagging, random forests) are easily parallelized
• Prediction contests (e.g. Kaggle, Netflix Prize) usually won by ensemble solutions
• Ensemble models are usually low on the comprehensibility scale, although see work by
  [Craven & Shavlik, NIPS 1996]
  [Domingos, Intelligent Data Analysis 1998]
  [Van Assche & Blockeel, ECML 2007]