Ensembles of Classifiers

Mark Craven
Computer Sciences 760
Fall 2016

www.biostat.wisc.edu/~craven/cs760/

Some of the slides in these lectures have been adapted/borrowed from materials developed by Tom Dietterich, Pedro Domingos, Tom Mitchell, David Page, and Jude Shavlik

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.
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 \( y \leftarrow \text{plurality\_vote}( h_1(x) \ldots h_T(x) ) \)

regression:
given: test instance \( x \)
predict \( y \leftarrow \text{mean}( h_1(x) \ldots h_T(x) ) \)
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 $\epsilon$ 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 $C$?
  - 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 $i$: $w_1(i) \leftarrow 1/m$  \hspace{1cm} // initialize instance weights
for $t \leftarrow 1$ to $T$ do
  for all $i$: $p_t(i) \leftarrow w_t(i) / \left( \sum_j w_t(j) \right)$ \hspace{1cm} // normalize weights
  $h_t \leftarrow$ model learned using $L$ on $D$ and $p_t$
  $\epsilon_t \leftarrow \sum_i p_t(i)(1 - \delta(h_t(x^{(i)}), y^{(i)}))$  \hspace{1cm} // calculate weighted error
  if $\epsilon_t > 0.5$ then
    $T \leftarrow t - 1$
    break
  $\beta_t \leftarrow \epsilon_t / (1 - \epsilon_t)$
  for all $i$ where $h_t(x^{(i)}) = y^{(i)}$ \hspace{1cm} // 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 $p_i$ 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 $p_i$; 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 $f << |F|$)
  choose the best feature split in $R$
do not prune trees

classification/regression:
as in bagging
Recent large-scale comparison
[ 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>$\kappa$</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 — we’ll discuss this when we cover SVMs.

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>Code Word</th>
<th>$f_0$</th>
<th>$f_3$</th>
<th>$f_2$</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></td>
<td>1</td>
<td>1</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>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>1</td>
<td>0</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>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td></td>
<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>
</tr>
<tr>
<td>3</td>
<td></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>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>0</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>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td></td>
<td>1</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>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td></td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</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>0</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td></td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</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>9</td>
<td></td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</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 $\left\lfloor \frac{d-1}{2} \right\rfloor$ 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.

The figure illustrates examples of row and column separation with the distances between codewords.

$7$ bits apart

$6$ bits apart

$d = 7$ so this code can correct \( \left\lfloor \frac{7 - 1}{2} \right\rfloor = 3 \) errors

ECOC evaluation with C4.5

![Evaluation Chart]

Figure from Bakiri & Dietterich, JAIR, 1995
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]