Ensemble Classifiers

Ensemble Learning

An ensemble of classifiers is a set of classifiers whose individual decisions are combined in some way (typically by weighted or unweighted voting) to classify new examples

Aka committees

Why Ensemble Learning

- Combining predictions of an ensemble is often more accurate than the individual classifiers that make them up.
- The classifiers should be accurate and diverse.
- An accurate classifier is one that has an error rate of better than random guessing (known as weak learners).
- Two classifiers are diverse if they make different errors on new data points.

Uncorrelated errors of individual classifiers can be eliminated by averaging.

Assume: 21 base classifiers, each with the same error rate $p = 0.3$.

Probability of getting $r$ incorrect votes is a binomial distribution

$$P(r) = \frac{n!}{r!(n-r)!} p^r (1-p)^{n-r}$$
Why Ensemble Learning

**Majority voting,**

\[
P(\text{ensemble wrong}) = P(r \geq 11) = .026
\]

If individual error rate > 0.5, the error rate of voted ensemble will increase

---

**Why Ensemble Learning**

Three fundamental reasons why an ensemble may work better than a single classifier

- Statistical
- Computational
- Representational
A learning algorithm can be viewed as searching a space $H$ of hypotheses to identify the best hypothesis.

Given a finite amount of data, many hypothesis are typically equally good.

Averaging these accurate classifiers may be a better approximation to $f$.

Since interesting hypothesis spaces are huge/infinite, heuristic search is essential.

So the learner might get stuck in a local minimum.

One strategy for avoiding local minima: repeat the search many times with random restarts -> construct an ensemble.
The desired target function may not be realizable using individual classifiers, but may be approximated by ensemble averaging.

Consider a binary learning task over $[0, 1] \times [0, 1]$, and the hypothesis space $H$ of “discs”.

$h_1, h_2, h_3 \in H$
Why Ensemble Learning

The three fundamental issues are the most important ways in which existing learning algorithms fail.

Ensemble methods may reduce these three key shortcomings of standard learning algorithms.
How to construct ensemble?

Two basic questions in designing ensembles:

- How to generate the base classifiers?
  
  \[ h_1, h_2, \ldots \]

- How to combine them?

How to combine classifiers

Usually take a weighted vote, e.g.:

\[
ensemble(x) = \text{sign}(\sum_i w_i h_i(x))
\]

- \( w_i > 0 \) is the weight of hypothesis \( h_i \)
- \( w_i > w_j \) means \( h_i \) is more reliable than \( h_j \)
How to construct ensemble?

A variety of approaches

- Bagging (Bootstrap aggregation)
- Boosting (Adaboost)
- Error correcting output codes (ECOC)
- ...

Bagging = Bootstrap AGGregation

Given a training set $D$ of size $n$, generate a new training set $D_1$ of size $n$ by sampling examples from $D$ uniformly and with replacement, known as a bootstrap sample.

Repeat this sampling procedure, getting a sequence of $k$ training sets.

Run the learning algorithm $k$ times, each time with a different training set.

The Bagged Classifier then combines the predictions of the individual classifiers to generate the final outcome (voting).
Bagging CART

Misclassification rates

<table>
<thead>
<tr>
<th>Data Set</th>
<th>$\bar{e}_S$</th>
<th>$\bar{e}_B$</th>
<th>Decrease</th>
</tr>
</thead>
<tbody>
<tr>
<td>waveform</td>
<td>29.0</td>
<td>19.4</td>
<td>33%</td>
</tr>
<tr>
<td>heart</td>
<td>10.0</td>
<td>5.3</td>
<td>47%</td>
</tr>
<tr>
<td>breast cancer</td>
<td>6.0</td>
<td>4.2</td>
<td>30%</td>
</tr>
<tr>
<td>ionosphere</td>
<td>11.2</td>
<td>8.6</td>
<td>23%</td>
</tr>
<tr>
<td>diabetes</td>
<td>23.4</td>
<td>18.8</td>
<td>20%</td>
</tr>
<tr>
<td>glass</td>
<td>32.0</td>
<td>24.9</td>
<td>22%</td>
</tr>
<tr>
<td>soybean</td>
<td>14.5</td>
<td>10.6</td>
<td>27%</td>
</tr>
</tbody>
</table>

-50 bootstrap samples

Breiman, L. (1996), Bagging Predictors, Machine learning

Bagging

A critical factor in whether bagging will improve accuracy is the stability of the learning algorithm

A learning algorithm is unstable if a small change in the training data can result in large changes in the output classifier

Unstable: neural networks, decision trees;
Stable: linear regression, $k$-nearest neighbor

Bagging works well for unstable procedures

It is a relatively easy way to improve an existing method
Boosting

- Boosting also manipulate the training set
- The base classifiers are trained in sequence
- Training Examples may have unequal weights
- Each base classifier is trained using a weighted data set
- Update weights by placing more weight on training examples that were misclassified and less weight on examples that were correctly classified
  - the learner is forced to focus on portions of data space not previously well predicted
- The predictions of base classifiers are combined through a weighted majority vote

---

Suppose base classifier $h$ is a weak learner - one that can learn a hypothesis that is better than rolling a dice - but perhaps only a tiny bit better

Learning from weighted examples

- It is easy to modify most learning algorithms to deal with weighted instances:
- Decision tree: modify entropy, information gain equations, when counting, increase count by weight rather than 1
- Training examples can be sampled according to weights (distribution), or replicate examples proportional to their weights
**Boosting**

A number of variants: Adaboost (Adaptive Boosting), ...


---

**AdaBoost Algorithm**

Given \( \{(x_1, t_1), \ldots, (x_N, t_N)\} \) where \( t_i \in \{-1, +1\} \)

- Initially assign uniform weights \( w_0(i) = 1 / N \).
- At each iteration \( k \)
  1. Train a base classifier \( h_k(x) \) using weights \( w_i \)
  2. Evaluate the error rate \( \varepsilon_k = \sum_i w_k(i)I(h_k(x_i) \neq t_i) \)
  3. Evaluate the weight of the classifier \( h_k(x) \): \( \alpha_k = \ln \frac{1-\varepsilon_k}{\varepsilon_k} \) \( (\alpha_k > 0 \text{ if } \varepsilon_k < 1/2; \text{ larger as } \varepsilon_k \text{ gets smaller}) \)
  4. Update weights \( w_{k+1}(i) = w_k(i) \exp[\alpha_k I(h_k(x_i) \neq t_i)] \)
     then normalize weights
     (equivalent to \( \alpha'_k = \alpha_k / 2 \), \( w_{k+1}(i) = w_k(i) \exp[-\alpha'_k h_k(x_i) t_i] \))
- The final classifier \( H(x) = \text{sign}(\sum_k \alpha_k h_k(x)) \)
Boosting

- Boosting was originally motivated using statistical learning theory.
- Suppose $h$ is a weak learner.
- Boosting $h$ yields an ensemble with arbitrarily low error on the training data.
- Theoretical upper bound on the generalization error – very loose.

AdaBoost Example

*Decision stumps* as base classifiers: classify an instance according to a single feature value – a decision tree with a single node.
AdaBoost Example

$m = 3$

AdaBoost Example

$m = 10$

AdaBoost Example

$m = 6$

AdaBoost Example

$m = 150$
AdaBoost Experiments

27 data sets from UCI Repository

Text categorization: Reuters newswire article
Weak learners: test on the presence or absence of a word or phrase
AdaBoost Experiments

Text categorization: AP newswire headlines
Weak learners: test on the presence or absence of a word or phrase

Ensemble Experimental Results

- Experimental studies to compare ensemble methods

- AdaBoost often gives the best results
- AdaBoost is subject to over-fitting when there is significant classification noise (incorrect class labels) in the training data
Boosting - Summary

- Basic motivation: creating a committee of experts is typically more effective than trying to derive a single super-genius
- Boosting provides a simple and powerful method for turning weak learners into strong learners
- The simple algorithm described here has been extended to:
  - multi-class classification problems
  - classifiers that produce confidences associated with class predictions (e.g., posterior probabilities as opposed to class assignments)
  - regression

ECOC

- So far, we have been building the ensemble by tweaking the distribution of training instances
- Error correcting output coding (ECOC) involves tweaking the output targets (class) to be learned
- Solving multiclass learning problems
  - using an ensemble of binary classifiers

Thomas G. Dietterich and G. Bakiri (1995), Solving Multiclass Learning Problems via Error-Correcting Output Codes, JAIR.
Handwritten number recognition

We decompose the learning task into six subproblems:

- Learn classifiers specialized to each of the 6 subproblems.
- To classify a new scribble, predict the class whose codeword is closest (Hamming distance) to the predicted code.

<table>
<thead>
<tr>
<th>Code Word</th>
<th>vl</th>
<th>hl</th>
<th>dl</th>
<th>cc</th>
<th>ol</th>
<th>or</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class 0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Class 1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Class 2</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Class 3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Class 4</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Class 5</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Class 6</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Class 7</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Class 8</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Class 9</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

### Abbreviation | Meaning
--- | ---
vl | contains vertical line
hl | contains horizontal line
dl | contains diagonal line
cc | contains closed curve
ol | contains curve open to left
or | contains curve open to right

Employ output code

Multiclass classifier employing an ensemble of binary classifiers:

- Each class is assigned a unique binary string of length $L$, a **codeword**.
- $L$ binary functions are learned, one for each bit position.
- New examples are classified by evaluating each of the $L$ binary classifiers to generate an $L$-bit string $s$, and the example is assigned to the class whose codeword is closest (in Hamming distance) to $s$. 

---

- p. 33
- p. 34
Employ output code

How to design good codes?

- Meaningful output representations
  - The minimum Hamming distance between pairs of codewords tends to be very low

- Robust to errors?
  - A measure of the quality of a code is the minimum Hamming distance between any pair of code words.

- Error-correcting codes

Error-correcting codes

E.g., the International Standard Book Number (ISBN) system

- ISBN system identifies every book with a ten-digit number, such as 0-226-53420-0. (13 digits since 1 January 2007)

- The first nine digits are the actual number, the tenth (a check digit) is added according to a mathematical formula based on the first nine.

- If a single one of the digits is changed, as in a misprint when ordering a book, a simple check verifies that something is wrong.
Suppose we want to send $n$-bit (binary) messages through a noisy channel.

To ensure robustness to noise, we can map each $n$-bit message into an $m$-bit code word ($m>n$). When receive an $m$-bit string, translate it to message corresponding to the nearest (Hamming distance) code word.

Key to robustness: assign the codes so that all clean message code words are at distances of $d \geq 3$ Hamming units from each other – can correct at least one bit error.

A measure of the quality of an error-correcting code is the minimum Hamming distance between any pair of code words.

If the minimum Hamming distance is $d$, then the code can correct at least $(d - 1)/2$ single bit errors.
Error-correcting codes

The code in the table has minimum Hamming distance seven. This code can correct up to three errors out of the 15 bits.

Code Design for ECOC

A good error-correcting output code for a $k$-class problem should satisfy two properties:

- **Row separation:** Each codeword should be well-separated in Hamming distance from each of the other codewords.

- **Column separation:** Each bit-position function $f_i$ should be uncorrelated with the functions to be learned for the other bit positions $f_j$. – Intuition: If two columns are similar, then a learning algorithm may make similar (correlated) mistakes. Error-correcting codes only succeed if the errors made in the individual bit positions are relatively uncorrelated, so that the number of simultaneous errors in many bit positions is small.
If there are \( k \) classes, there will be at most \( 2^{k-1} - 1 \) usable columns after removing complements and the all-ones column. \( k = 4 \) we get a 7-column code with minimum inter-row Hamming distance 4.

Unless the number of classes is at least five, it is difficult to satisfy both of these properties.

Construct good error-correcting output codes is difficult for large \( k \)

Use heuristic search algorithms

---

Performance of ECOC

An asterisk indicates that the difference is statistically significant at the 0.05 level
Multiclass learning

- AdaBoost can be combined with ECOC
  Schapire (1997) Using output codes to boost multiclass learning problems

Summary

- Ensembles: basic motivation - creating a committee of experts is typically more effective than trying to derive a single super-genius
- Popular ensemble techniques
  - manipulate training data: bagging and boosting
  - manipulate output values: error-correcting output coding
  - Manipulate input feature space
    - train classifiers using different subset of features
    - work when the input features are highly redundant