Machine Learning: Symbolische Ansätze

Ensemble Methods

- Bias-Variance Trade-off
- Basic Idea of Ensembles
- Bagging
  - Basic Algorithm
  - Bagging with Costs

- Randomization
  - Random Forests
- Boosting
- Stacking
- Error-Correcting Output Codes (ECOC)
Bias and Variance Decomposition

- **Bias:**
  - the part of the error that is caused by bad model

- **Variance:**
  - the part of the error that is caused by the data sample

- **Bias-Variance Trade-off:**
  - algorithms that can easily adapt to any given decision boundary are very sensitive to small variations in the data
    - and vice versa
  - Models with a low bias often have a high variance
    - e.g., nearest neighbor, unpruned decision trees
  - Models with a low variance often have a high bias
    - e.g., decision stump, linear model
Ensemble Classifiers

- **IDEA:**
  - do not learn a *single* classifier but learn a *set of classifiers*
  - combine the predictions of multiple classifiers

- **MOTIVATION:**
  - reduce variance: results are less dependent on peculiarities of a single training set
  - reduce bias: a combination of multiple classifiers may learn a more expressive concept class than a single classifier

- **KEY STEP:**
  - formation of an ensemble of *diverse* classifiers from a single training set
Why do ensembles work?

- Suppose there are 25 base classifiers
  - Each classifier has error rate, $\varepsilon = 0.35$
  - Assume classifiers are independent
    - i.e., probability that a classifier makes a mistake does not depend on whether other classifiers made a mistake
  - Note: in practice they are not independent!

- Probability that the ensemble classifier makes a wrong prediction
  - The ensemble makes a wrong prediction if the majority of the classifiers makes a wrong prediction
  - The probability that 13 or more classifiers err is
    \[
    \sum_{i=13}^{25} \binom{25}{i} \varepsilon^i (1-\varepsilon)^{25-i} \approx 0.06 \ll \varepsilon
    \]
Bagging: General Idea

<table>
<thead>
<tr>
<th>Step 1: Create Multiple Data Sets</th>
</tr>
</thead>
<tbody>
<tr>
<td>Step 2: Build Multiple Classifiers</td>
</tr>
<tr>
<td>Step 3: Combine Classifiers</td>
</tr>
</tbody>
</table>

Original Training data

\[ D \]

\[ D_1 \]
\[ D_2 \]
\[ \ldots \]
\[ D_{t-1} \]
\[ D_t \]

\[ C_1 \]
\[ C_2 \]
\[ C_{t-1} \]
\[ C_t \]

\[ C^* \]
Generate Bootstrap Samples

- Generate new training sets using sampling with replacement (bootstrap samples)

<table>
<thead>
<tr>
<th>Original Data</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bagging (Round 1)</td>
<td>7</td>
<td>8</td>
<td>10</td>
<td>8</td>
<td>2</td>
<td>5</td>
<td>10</td>
<td>10</td>
<td>5</td>
<td>9</td>
</tr>
<tr>
<td>Bagging (Round 2)</td>
<td>1</td>
<td>4</td>
<td>9</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>7</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>Bagging (Round 3)</td>
<td>1</td>
<td>8</td>
<td>5</td>
<td>10</td>
<td>5</td>
<td>5</td>
<td>9</td>
<td>6</td>
<td>3</td>
<td>7</td>
</tr>
</tbody>
</table>

- some examples may appear in more than one set
- some examples will appear more than once in a set
- for each set of size \( n \), the probability that a given example appears in it is
  \[
  \Pr(x \in D_i) = 1 - \left(1 - \frac{1}{n}\right)^n \rightarrow 0.6322
  \]
  i.e., on average, less than 2/3 of the examples appear in any single bootstrap sample
Bagging Algorithm

1. for $m = 1$ to $t$  // $t$ ... number of iterations
   a) draw (with replacement) a bootstrap sample $D_m$ of the data
   b) learn a classifier $C_m$ from $D_m$

2. for each test example
   a) try all classifiers $C_m$
   b) predict the class that receives the highest number of votes

- variations are possible
  - e.g., size of subset, sampling w/o replacement, etc.
- many related variants
  - sampling of features, not instances
  - learn a set of classifiers with different algorithms
Bagged Decision Trees

Original Tree

Bootstrap Tree 1

Bootstrap Tree 2

Bootstrap Tree 3

from Hastie, Tibshirani, Friedman: The Elements of Statistical Learning, Springer Verlag 2001
Bagged Trees

8.7 Bagging

Test Error

0.35

0.30

0.25

0.20

Number of Bootstrap Samples

Original Tree

Bagged Trees

Bayes

weighted voting

voting

from Hastie, Tibshirani, Friedman: The Elements of Statistical Learning, Springer Verlag 2001
Bagging with costs

- Bagging unpruned decision trees is known to produce **good probability estimates**
  - Where, instead of voting, the individual classifiers' probability estimates $\Pr_n(j|x)$ are averaged
    \[
    \Pr(j|x) = \frac{1}{t} \sum_{n=1}^{t} \Pr_n(j|x)
    \]
  - Note: this can also improve the error rate

- We can use this with minimum-expected cost approach for learning problems with costs
  - predict class $c$ with
    \[
    c = \arg\min_i \sum_j C(i|j) \Pr(j|x)
    \]

- Problem: not interpretable
  - *MetaCost* re-labels training data using bagging with costs and then builds single tree (Domingos, 1997)
Randomization

- Randomize the learning algorithm instead of the input data
- Some algorithms already have a random component
  - eg. initial weights in neural net
- Most algorithms can be randomized, eg. greedy algorithms:
  - Pick from the $N$ best options at random instead of always picking the best options
  - Eg.: test selection in decision trees or rule learning
- Can be combined with bagging
Random Forests

- Combines bagging and random attribute subset selection:
  - Build the tree from a bootstrap sample
  - Instead of choosing the best split among all attributes, select the best split among a random subset of \( k \) attributes
    - is equal to bagging when \( k \) equals the number of attributes)
- There is a bias/variance tradeoff with \( k \):
  - The smaller \( k \), the greater the reduction of variance but also the higher the increase of bias
Boosting

- **Basic Idea:**
  - later classifiers focus on examples that were misclassified by earlier classifiers
  - weight the predictions of the classifiers with their error

- **Realization**
  - perform multiple iterations
    - each time using different example weights
  - weight update between iterations
    - increase the weight of incorrectly classified examples
    - this ensures that they will become more important in the next iterations (misclassification errors for these examples count more heavily)
  - combine results of all iterations
    - weighted by their respective error measures
Dealing with Weighted Examples

Two possibilities (→ cost-sensitive learning)

- **directly**
  - example $e_i$ has weight $w_i$
  - number of examples $n \Rightarrow$ total example weight $\sum_{i=1}^{n} w_i$

- **via sampling**
  - interpret the weights as probabilities
  - examples with larger weights are more likely to be sampled
  - assumptions
    - sampling with replacement
    - weights are well distributed in [0,1]
    - learning algorithm sensible to varying numbers of identical examples in training data
1. initialize example weights \( w_i = \frac{1}{N} \) \((i = 1..N)\)

2. for \( m = 1 \) to \( t \) // \( t \ldots \) number of iterations
   a) learn a classifier \( C_m \) using the current example weights
   b) compute a weighted error estimate
      \[
      err_m = \frac{1}{\sum_{i=1}^{N} w_i} \sum_{i=1}^{N} w_i \text{ of all incorrectly classified } e_i
      \]
   c) compute a classifier weight
      \[
      \alpha_m = \frac{1}{2} \ln \left( \frac{1 - err_m}{err_m} \right)
      \]
   d) for all correctly classified examples \( e_i : w_i \leftarrow w_i e^{-\alpha_m} \)
   e) for all incorrectly classified examples \( e_i : w_i \leftarrow w_i e^{\alpha_m} \)
   f) normalize the weights \( w_i \) so that they sum to 1

3. for each test example
   a) try all classifiers \( C_m \)
   b) predict the class that receives the highest sum of weights \( \alpha_m \)
Illustration of the Weights

- **Classifier Weights** $\alpha_m$
  - differences near 0 or 1 are emphasized

- **Example Weights**
  - multiplier for correct and incorrect examples, depending on error
Boosting – Error rate example

- boosting of decision stumps on simulated data

![Graph showing error rate over boosting iterations for a single stump and a 400 node tree.](image-url)
Toy Example

- An Applet demonstrating AdaBoost
  - [http://www.cse.ucsd.edu/~yfreund/adaboost/](http://www.cse.ucsd.edu/~yfreund/adaboost/)

(taken from Verma & Thrun, Slides to CALD Course CMU 15-781, Machine Learning, Fall 2000)
Round 1

\[ h_1 \]

\[ \varepsilon_1 = 0.30 \]
\[ \alpha_1 = 0.42 \]
Round 2

\begin{align*}
\epsilon_2 &= 0.21 \\
\alpha_2 &= 0.65
\end{align*}
Round 3

\[ \varepsilon_3 = 0.14 \]
\[ \alpha_3 = 0.92 \]
Final Hypothesis

\[ H_{\text{final}} = \text{sign} \left( 0.42 + 0.65 + 0.92 \right) \]
Example

**FIGURE 8.11.** Data with two features and two classes, separated by a linear boundary. Left panel: decision boundary estimated from bagging the decision rule from a single split, axis-oriented classifier. Right panel: decision boundary from boosting the decision rule of the same classifier. The test error rates are 0.166, and 0.065 respectively. Boosting is described in Chapter 10.
Comparison Bagging/Boosting

- **Bagging**
  - noise-tolerant
  - produces better class probability estimates
  - not so accurate
  - statistical basis
  - related to random sampling

- **Boosting**
  - very susceptible to noise in the data
  - produces rather bad class probability estimates
  - if it works, it works really well
  - based on learning theory (statistical interpretations are possible)
  - related to windowing
Additive regression

- It turns out that boosting is a greedy algorithm for fitting additive models
- More specifically, implements forward stagewise additive modeling
- Same kind of algorithm for numeric prediction:

  1. Build standard regression model (eg. tree)
  2. Gather residuals
  3. Learn model predicting residuals (eg. tree)

- To predict, simply sum up individual predictions from all models
Combining Predictions

- **voting**
  - each ensemble member votes for one of the classes
  - predict the class with the highest number of votes (e.g., bagging)

- **weighted voting**
  - make a *weighted* sum of the votes of the ensemble members
  - weights typically depend
    - on the classifiers confidence in its prediction (e.g., the estimated probability of the predicted class)
    - on error estimates of the classifier (e.g., boosting)

- **stacking**
  - Why not use a classifier for making the final decision?
  - training material are the class labels of the training data and the (cross-validated) predictions of the ensemble members
Stacking

- **Basic Idea:**
  - learn a function that combines the predictions of the individual classifiers

- **Algorithm:**
  - train $n$ different classifiers $C_1...C_n$ (the base classifiers)
  - obtain predictions of the classifiers for the training examples
  - form a new data set (the meta data)
    - **classes**
      - the same as the original dataset
    - **attributes**
      - one attribute for each base classifier
      - value is the prediction of this classifier on the example
  - train a separate classifier $M$ (the meta classifier)

This is better done with cross-validation!
Stacking (2)

- **Example:**

<table>
<thead>
<tr>
<th>Attributes</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_{11}$</td>
<td>$t$</td>
</tr>
<tr>
<td>$x_{21}$</td>
<td>$f$</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>$x_{n_{c1}}$</td>
<td>$t$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Attributes</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_{1n_a}$</td>
<td></td>
</tr>
<tr>
<td>$x_{2n_a}$</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>$x_{n_{c}n_a}$</td>
<td></td>
</tr>
</tbody>
</table>

Using a stacked classifier:
- try each of the classifiers $C_1...C_n$
- form a feature vector consisting of their predictions
- submit these feature vectors to the meta classifier $M$
Error-correcting output codes
(Dietterich & Bakiri, 1995)

- Class Binarization technique
  - Multiclass problem → binary problems
  - Simple scheme: One-vs-all coding
- Idea: use error-correcting codes instead
  - one code vector per class
- Prediction:
  - base classifiers predict 101111, true class = ??
- Use code words that have large pairwise Hamming distance $d$
  - Can correct up to $(d - 1)/2$ single-bit errors

<table>
<thead>
<tr>
<th>Class</th>
<th>Class Vector</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>1 0 0 0</td>
</tr>
<tr>
<td>b</td>
<td>0 1 0 0</td>
</tr>
<tr>
<td>c</td>
<td>0 0 1 0</td>
</tr>
<tr>
<td>d</td>
<td>0 0 0 1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Class</th>
<th>Class Vector</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>1 1 1 1 1 1 1</td>
</tr>
<tr>
<td>b</td>
<td>0 0 0 0 1 1 1</td>
</tr>
<tr>
<td>c</td>
<td>0 0 1 1 0 0 1</td>
</tr>
<tr>
<td>d</td>
<td>0 1 0 1 0 1 0</td>
</tr>
</tbody>
</table>

7 binary classifiers
More on ECOCs

- Two criteria:
  - **Row separation:**
    - minimum distance between rows
  - **Column separation:**
    - minimum distance between columns
    - (and columns’ complements)
    - Why? Because if columns are identical, base classifiers will likely make the same errors
    - Error-correction is weakened if errors are correlated

- 3 classes $\implies$ only $2^3$ possible columns
  - (and 4 out of the 8 are complements)
  - Cannot achieve row and column separation

- Only works for problems with $> 3$ classes
Exhaustive ECOCs

- Exhaustive code for k classes:
  - Columns comprise every possible k-string …
  - … except for complements and all-zero/one strings
  - Each code word contains \(2^{k-1} - 1\) bits
- Class 1: code word is all ones
- Class 2: \(2^{k-2}\) zeroes followed by \(2^{k-2} - 1\) ones
- Class \(i\): alternating runs of \(2^{k-i}\) 0s and 1s
  - last run is one short

Exhaustive code, \(k = 4\)

<table>
<thead>
<tr>
<th>class</th>
<th>class vector</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>1111111</td>
</tr>
<tr>
<td>b</td>
<td>0000111</td>
</tr>
<tr>
<td>c</td>
<td>0011001</td>
</tr>
<tr>
<td>d</td>
<td>0101010</td>
</tr>
</tbody>
</table>
Extensions of ECOCs

- Many different coding strategies have been proposed
  - exhaustive codes infeasible for large numbers of classes
    - Number of columns increases exponentially
  - Random code words have good error-correcting properties on average!
- Ternary ECOCs (Allwein et al., 2000)
  - use three-valued codes -1/0/1, i.e., positive / ignore / negative
  - this can, e.g., also model pairwise classification
- ECOCs don’t work with NN classifier
  - because the same neighbor(s) are used in all binary classifiers for making the prediction
  - But: works if different attribute subsets are used to predict each output bit
Forming an Ensemble

- Modifying the data
  - Subsampling
    - bagging
    - boosting
  - feature subsets
    - randomly feature samples

- Modifying the learning task
  - pairwise classification / round robin learning
  - error-correcting output codes

- Exploiting the algorithm characteristics
  - algorithms with random components
    - neural networks
  - randomizing algorithms
    - randomized decision trees
  - use multiple algorithms with different characteristics

- Exploiting problem characteristics
  - e.g., hyperlink ensembles