Machine Learning Techniques for Data Mining

Eibe Frank
University of Waikato
New Zealand
PART VII

Moving on: Engineering the input and output
Applying a learner is not all

- Already discussed: scheme/parameter selection
  - Important: selection process should be treated as part of the learning process
- Modifying the input: attribute selection, discretization, data cleansing, transformations
- Modifying the output: combining classification models to improve performance
  - Bagging, boosting, stacking, error-correcting output codes (and Bayesian model averaging)
Attribute selection

- Adding a random (i.e. irrelevant) attribute can significantly degrade C4.5’s performance
  - Problem: attribute selection based on smaller and smaller amounts of data
- IBL is also very susceptible to irrelevant attributes
  - Number of training instances required increases exponentially with number of irrelevant attributes
- Naïve Bayes doesn’t have this problem
- Relevant attributes can also be harmful
Scheme-independent selection

- *Filter* approach: assessment based on general characteristics of the data
- One method: find subset of attributes that is enough to separate all the instances
- Another method: use different learning scheme (e.g. C4.5, 1R) to select attributes
- IBL-based attribute weighting techniques can also be used (but can’t find redundant attributes)
- CFS: uses correlation-based evaluation of subsets
Attribute subsets for weather data
Searching the attribute space

- Number of possible attribute subsets is exponential in the number of attributes
- Common greedy approaches: forward selection and backward elimination
- More sophisticated strategies:
  - Bidirectional search
  - Best-first search: can find the optimum solution
  - Beam search: approximation to best-first search
  - Genetic algorithms
Scheme-specific selection

- *Wrapper* approach: attribute selection implemented as wrapper around learning scheme
  - Evaluation criterion: cross-validation performance
- Time consuming: adds factor $k^2$ even for greedy approaches with $k$ attributes
  - Linearity in $k$ requires prior ranking of attributes
- Scheme-specific attribute selection essential for learning decision tables
- Can be done efficiently for DTs and Naïve Bayes
Discretizing numeric attributes

- Can be used to avoid making normality assumption in Naïve Bayes and Clustering
- Simple discretization scheme is used in 1R
- C4.5 performs *local* discretization
- *Global* discretization can be advantageous because it’s based on more data
  - Learner can be applied to discretized attribute *or*
  - It can be applied to binary attributes coding the cut points in the discretized attribute
Unsupervised discretization

- *Unsupervised* discretization generates intervals without looking at class labels
  - Only possible way when clustering
- Two main strategies:
  - *Equal-interval binning*
  - *Equal-frequency binning* (also called *histogram equalization*)
- Inferior to supervised schemes in classification tasks
Entropy-based discretization

- *Supervised* method that builds a decision tree with pre-pruning on the attribute being discretized
  - Entropy used as splitting criterion
  - MDLP used as stopping criterion
- State-of-the-art discretization method
- Application of MDLP:
  - “Theory” is the splitting point (\(\log_2[N-1]\) bits) plus class distribution in each subset
  - DL before/after adding splitting point is compared
Example: temperature attribute
Formula for MDLP

- $N$ instances and
  - $k$ classes and entropy $E$ in original set
  - $k_1$ classes and entropy $E_1$ in first subset
  - $k_2$ classes and entropy $E_2$ in first subset

$$\text{gain} > \frac{\log_2 (N - 1)}{N} + \frac{\log_2 (3^k - 2) - kE + k_1 E_1 + k_2 E_2}{N}$$

- Doesn’t result in any discretization intervals for the temperature attribute
Other discretization methods

- Top-down procedure can be replaced by bottom-up method
- MDLP can be replaced by chi-squared test
- Dynamic programming can be used to find optimum $k$-way split for given additive criterion
  - Requires time quadratic in number of instances if entropy is used as criterion
  - Can be done in linear time if error rate is used as evaluation criterion
Error-based vs. entropy-based
The converse of discretization

- Scheme used by IB1: indicator attributes
- Doesn’t make use of potential ordering information
- M5’ generates ordering of nominal values and codes ordering using binary attributes
- This strategy can be used for any attribute for which values are ordered
  - Avoids problem of using integer attribute to code ordering: would imply a metric
- In general: subsets of attributes coded as binary attributes
Automatic data cleansing

- Improving decision trees: relearn tree with misclassified instances removed
- Better strategy (of course): let human expert check misclassified instances
- When systematic noise is present it’s better not to modify the data
- Also: attribute noise should be left in training set
- (Unsystematic) class noise in training set should be eliminated if possible
Robust regression

- Statistical methods that address problem of outliers are called robust
- Possible way of making regression more robust:
  - Minimize absolute error instead of squared error
  - Remove outliers (i.e. 10% of points farthest from the regression plane)
  - Minimize median instead of mean of squares (copes with outliers in $x$ and $y$ direction)
    - Finds narrowest strip covering half the observations
Example: least median of squares
Detecting anomalies

- Visualization best way of detecting anomalies (but often can’t be done)
- Automatic approach: committee of different learning schemes
  - E.g. decision tree, nearest-neighbor learner, and a linear discriminant function
  - Conservative approach: only delete instances which are incorrectly classified by all of them
  - Problem: might sacrifice instances of small classes
Combining multiple models

- Basic idea of “meta” learning schemes: build different “experts” and let them vote
- Advantage: often improves predictive performance
- Disadvantage: produces output that is very hard to analyze
- Schemes we will discuss: bagging, boosting, stacking, and error-correcting output codes
  - The first three can be applied to both classification and numeric prediction problems
Bagging

- Employs simplest way of combining predictions: voting/averaging
- Each model receives equal weight
- “Idealized” version of bagging:
  - Sample several training sets of size $n$ (instead of just having one training set of size $n$)
  - Build a classifier for each training set
  - Combine the classifier’s predictions
- This improves performance in almost all cases if learning scheme is *unstable* (i.e. decision trees)
Bias-variance decomposition

- Theoretical tool for analyzing how much specific training set affects performance of classifier
- Assume we have an infinite number of classifiers built from different training sets of size $n$
  - The bias of a learning scheme is the expected error of the combined classifier on new data
  - The variance of a learning scheme is the expected error due to the particular training set used
  - Total expected error: bias + variance
More on bagging

- Bagging reduces variance by voting/averaging, thus reducing the overall expected error
  - In the case of classification there are pathological situations where the overall error might increase
  - Usually, the more classifiers the better
- Problem: we only have one dataset!
- Solution: generate new datasets of size $n$ by sampling with replacement from original dataset
- Can help a lot if data is noisy
Bagging classifiers

model generation
Let n be the number of instances in the training data.
For each of t iterations:
  Sample n instances with replacement from training set.
  Apply the learning algorithm to the sample.
  Store the resulting model.

classification
For each of the t models:
  Predict class of instance using model.
Return class that has been predicted most often.
Boosting

- Also uses voting/averaging but models are weighted according to their performance
- Iterative procedure: new models are influenced by performance of previously built ones
  - New model is encouraged to become expert for instances classified incorrectly by earlier models
  - Intuitive justification: models should be experts that complement each other
- There are several variants of this algorithm
AdaBoost.M1

model generation
Assign equal weight to each training instance.
For each of t iterations:
   Apply learning algorithm to weighted dataset and store resulting model.
   Compute error e of model on weighted dataset and store error.
   If e equal to zero, or e greater or equal to 0.5:
      Terminate model generation.
   For each instance in dataset:
      If instance classified correctly by model:
         Multiply weight of instance by e / (1 - e).
   Normalize weight of all instances.

classification
Assign weight of zero to all classes.
For each of the t (or less) models:
   Add -\log(e / (1 - e)) to weight of class predicted by model.
Return class with highest weight.
More on boosting

- Can be applied without weights using resampling with probability determined by weights
  - Disadvantage: not all instances are used
  - Advantage: resampling can be repeated if error exceeds 0.5
- Stems from *computational learning theory*
- Theoretical result: training error decreases exponentially
- Also: works if base classifiers not too complex and their error doesn’t become too large too quickly
A bit more on boosting

- Puzzling fact: generalization error can decrease long after training error has reached zero
  - Seems to contradict Occam’s Razor!
  - However, problem disappears if margin (confidence) is considered instead of error
    - Margin: difference between estimated probability for true class and most likely other class (between $-1, 1$)
- Boosting works with weak learners: only condition is that error doesn’t exceed 0.5
- LogitBoost: more sophisticated boosting scheme
Stacking

- Hard to analyze theoretically: “black magic”
- Uses meta learner instead of voting to combine predictions of base learners
  - Predictions of base learners (level-0 models) are used as input for meta learner (level-1 model)
- Base learners usually different learning schemes
- Predictions on training data can’t be used to generate data for level-1 model!
  - Cross-validation-like scheme is employed
More on stacking

- If base learners can output probabilities it’s better to use those as input to meta learner
- Which algorithm to use to generate meta learner?
  - In principle, any learning scheme can be applied
  - David Wolpert: “relatively global, smooth” model
    - Base learners do most of the work
    - Reduces risk of overfitting
- Stacking can also be applied to numeric prediction (and density estimation)
**Error-correcting output codes**

- Very elegant method of transforming multiclass problem into two-class problem
  - Simple scheme: as many binary class attributes as original classes using one-per-class coding

<table>
<thead>
<tr>
<th>class</th>
<th>class vector</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>1000</td>
</tr>
<tr>
<td>b</td>
<td>0100</td>
</tr>
<tr>
<td>c</td>
<td>0010</td>
</tr>
<tr>
<td>d</td>
<td>0001</td>
</tr>
</tbody>
</table>

- Idea: use *error-correcting codes* instead
More on ECOCs

- Example:

<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>

- What’s the true class if base classifiers predict 1011111?

- We want code words for which minimum *hamming distance* between any pair of words $d$ is large
  - Up to $(d-1)/2$ single-bit errors can be corrected
A bit more on ECOCs

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

- Only works for problems with more than 3 classes: for 3 classes there are only $2^3$ possible columns
Exhaustive ECOCs

- With few classes exhaustive codes can be build (like the one on an earlier slide)
- Exhaustive code for $k$ classes:
  - The columns comprise every possible $k$-string
  - Except for complements and all-zero/one strings
  - Each code word contains $2^{k-1}-1$ bits
- Code word for 1st class: all ones
- 2nd class: $2^{k-2}$ zeroes followed by $2^{k-2}-1$ ones
- i\textsuperscript{th} class: alternating runs of $2^{k-i}$ zeroes and ones, the last run being one short
One last slide on ECOCs

- With more classes, exhaustive codes are infeasible
  - Number of columns increases exponentially
- Random code words have good error-correcting properties on average!
- More sophisticated methods exist for generating ECOCs using a small number of columns
- ECOCs don’t work with NN classifier
  - But: works if different attribute subsets are used to predict each output bit