Data Mining
Practical Machine Learning Tools and Techniques

Slides for Chapter 7 of Data Mining by I. H. Witten, E. Frank and M. A. Hall

Data transformations
- Attribute selection
  - Scheme-independent, scheme-specific
- Attribute discretization
  - Unsupervised, supervised, error-vs-entropy-based, converse of discretization
- Projections
  - Principal component analysis, random projections, partial least-squares, text, time series
- Sampling
  - Reservoir sampling
- Dirty data
  - Data cleansing, robust regression, anomaly detection
- Transforming multiple classes to binary ones
  - Simple approaches, error-correcting codes, ensembles of nested dichotomies
- Calibrating class probabilities

Just apply a learner? NO!
- Scheme/parameter selection
  treat selection process as part of the learning process
- Modifying the input:
  - Data engineering to make learning possible or easier
- Modifying the output
  - Re-calibrating probability estimates

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 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 attribute selection

- **Filter** approach: assess based on general characteristics of the data
- One method: find smallest subset of attributes that separates data
- Another method: use different learning scheme
  - e.g. use attributes selected by C4.5 and 1R, or coefficients of linear model, possibly applied recursively (**recursive feature elimination**)
- IBL-based attribute weighting techniques:
  - can’t find redundant attributes (but fix has been suggested)
- Correlation-based Feature Selection (CFS):
  - correlation between attributes measured by symmetric uncertainty:
    \[
    U(A, B) = 2^{\frac{H(A) + H(B) - H(A, B)}{H(A) + H(B)}} \in [0, 1]
    \]
    goodness of subset of attributes measured by (breaking ties in favor of smaller subsets):
    \[
    \sum_j U(A_j, C) / \sqrt{\sum_i \sum_j U(A_i, A_j)}
    \]

Searching attribute space

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

Attribute subsets for weather data

- **Wrapper** approach to attribute selection
- Implement “wrapper” around learning scheme
  - Evaluation criterion: cross-validation performance
- Time consuming
  - greedy approach, \(k\) attributes \(\Rightarrow\) \(k^2 \times \text{time}\)
  - prior ranking of attributes \(\Rightarrow\) linear in \(k\)
- Can use significance test to stop cross-validation for subset early if it is unlikely to “win” (**race search**)
  - can be used with forward, backward selection, prior ranking, or special-purpose **schemata search**
- Learning decision tables: scheme-specific attribute selection essential
  - Efficient for decision tables and Naïve Bayes
Attribute discretization

- Avoids normality assumption in Naïve Bayes and clustering
- 1R: uses simple discretization scheme
- C4.5 performs local discretization
- Global discretization can be advantageous because it's based on more data
- Apply learner to
  - $k$-valued discretized attribute or to
  - $k - 1$ binary attributes that code the cut points

Discretization: unsupervised

- Determine intervals without knowing class labels
  - When clustering, the only possible way!
- Two strategies:
  - Equal-interval binning
  - Equal-frequency binning (also called histogram equalization)
- Normally inferior to supervised schemes in classification tasks
  - But equal-frequency binning works well with naïve Bayes if number of intervals is set to square root of size of dataset (proportional $k$-interval discretization)

Discretization: supervised

- Entropy-based method
- Build a decision tree with pre-pruning on the attribute being discretized
  - Use entropy as splitting criterion
  - Use minimum description length principle as stopping criterion
- Works well: the state of the art
- To apply min description length principle:
  - The “theory” is
    - the splitting point ($\log_2(N - 1)$ bits)
    - plus class distribution in each subset
  - Compare description lengths before/after adding split

Example: temperature attribute
Formula for MDLP

- \( N \) instances
  - Original set: \( k \) classes, entropy \( E \)
  - First subset: \( k_1 \) classes, entropy \( E_1 \)
  - Second subset: \( k_2 \) classes, entropy \( E_2 \)

\[
gain > \frac{\log_2(N-1)}{N} + \frac{\log_2(3^k-2)-kE+k_1E_1+k_2E_2}{N}
\]

- Results in no discretization intervals for temperature attribute

Supervised discretization: other methods

- Can replace top-down procedure by bottom-up method
- Can replace MDLP by chi-squared test
- Can use dynamic programming to find optimum \( k \)-way split for given additive criterion
  - Requires time quadratic in the number of instances
  - But can be done in linear time if error rate is used instead of entropy

Error-based vs. entropy-based

- Question: could the best discretization ever have two adjacent intervals with the same class?
- Wrong answer: No. For if so,
  - Collapse the two
  - Free up an interval
  - Use it somewhere else
  - *(This is what error-based discretization will do)*
- Right answer: Surprisingly, yes.
  - *(and entropy-based discretization can do it)*

A 2-class, 2-attribute problem

Entropy-based discretization can detect change of class distribution
The converse of discretization

- Make ordinal values into “numeric” ones

1. Indicator attributes (used by IB1)
   - Makes no use of potential ordering information
2. Code an ordinal attribute into binary ones (used by M5’)
   - Can be used for any ordered attribute
   - Better than coding ordering into an integer (which implies a metric)
- In general: code subset of attribute values as binary

Projections

- Simple transformations can often make a large difference in performance
- Example transformations (not necessarily for performance improvement):
  - Difference of two date attributes
  - Ratio of two numeric (ratio-scale) attributes
  - Concatenating the values of nominal attributes
  - Encoding cluster membership
  - Adding noise to data
  - Removing data randomly or selectively
  - Obfuscating the data

Principal component analysis

- Method for identifying the important “directions” in the data
- Can rotate data into (reduced) coordinate system that is given by those directions
- Algorithm:
  1. Find direction (axis) of greatest variance
  2. Find direction of greatest variance that is perpendicular to previous direction and repeat
- Implementation: find eigenvectors of covariance matrix by diagonalization
  - Eigenvectors (sorted by eigenvalues) are the directions
Example: 10-dimensional data

<table>
<thead>
<tr>
<th>Axis</th>
<th>Variance</th>
<th>Cumulative</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>61.2%</td>
<td>61.2%</td>
</tr>
<tr>
<td>2</td>
<td>18.0%</td>
<td>79.2%</td>
</tr>
<tr>
<td>3</td>
<td>4.7%</td>
<td>83.9%</td>
</tr>
<tr>
<td>4</td>
<td>4.0%</td>
<td>87.9%</td>
</tr>
<tr>
<td>5</td>
<td>3.2%</td>
<td>91.1%</td>
</tr>
<tr>
<td>6</td>
<td>2.9%</td>
<td>94.0%</td>
</tr>
<tr>
<td>7</td>
<td>2.0%</td>
<td>96.0%</td>
</tr>
<tr>
<td>8</td>
<td>1.7%</td>
<td>97.7%</td>
</tr>
<tr>
<td>9</td>
<td>1.4%</td>
<td>99.1%</td>
</tr>
<tr>
<td>10</td>
<td>0.9%</td>
<td>100.0%</td>
</tr>
</tbody>
</table>

- Can transform data into space given by components
- Data is normally standardized for PCA
- Could also apply this recursively in tree learner

Partial least-squares regression

- PCA is often a pre-processing step before applying a learning algorithm
  - When linear regression is applied the resulting model is known as principal components regression
  - Output can be re-expressed in terms of the original attributes
- Partial least-squares differs from PCA in that it takes the class attribute into account
  - Finds directions that have high variance and are strongly correlated with the class

Random projections

- PCA is nice but expensive: cubic in number of attributes
- Alternative: use random directions (projections) instead of principle components
- Surprising: random projections preserve distance relationships quite well (on average)
  - Can use them to apply kD-trees to high-dimensional data
  - Can improve stability by using ensemble of models based on different projections

Algorithm

1. Start with standardized input attributes
2. Attribute coefficients of the first PLS direction:
   - Compute the dot product between each attribute vector and the class vector in turn
3. Coefficients for next PLS direction:
   - Original attribute values are first replaced by difference (residual) between the attribute’s value and the prediction from a simple univariate regression that uses the previous PLS direction as a predictor of that attribute
   - Compute the dot product between each attribute’s residual vector and the class vector in turn
4. Repeat from 3
**PLS Example (2 attributes only)**

<table>
<thead>
<tr>
<th>Table 1.5 CPU Performance Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>Min</td>
</tr>
<tr>
<td>-----</td>
</tr>
<tr>
<td>MYCT</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>5</td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td>207</td>
</tr>
<tr>
<td>208</td>
</tr>
<tr>
<td>209</td>
</tr>
</tbody>
</table>

Table 7.1 First Five Instances from the CPU Performance Data

<table>
<thead>
<tr>
<th>(a)</th>
<th>(b)</th>
<th>(c)</th>
</tr>
</thead>
<tbody>
<tr>
<td>chmin</td>
<td>chmax</td>
<td>prp</td>
</tr>
<tr>
<td>1.7889</td>
<td>1.7678</td>
<td>198</td>
</tr>
<tr>
<td>-0.4472</td>
<td>-0.3536</td>
<td>269</td>
</tr>
<tr>
<td>-0.4472</td>
<td>-0.3536</td>
<td>220</td>
</tr>
<tr>
<td>-0.4472</td>
<td>-0.3536</td>
<td>172</td>
</tr>
<tr>
<td>-0.4472</td>
<td>-0.7071</td>
<td>132</td>
</tr>
</tbody>
</table>

(a) original values, (b) first partial least-squares direction, and (c) residuals from the first direction.

**Text to attribute vectors**

- Many data mining applications involve textual data (e.g., string attributes in ARFF).
- Standard transformation: convert string into bag of words by *tokenization*.
  - Attribute values are binary, word frequencies ($f_{ij}$), log(1+$f_{ij}$), or TF x IDF: $f_{ij} \log \frac{\# \text{documents}}{\# \text{documents that include word } i}$
  - Only retain alphabetic sequences?
  - What should be used as delimiters?
  - Should words be converted to lowercase?
  - Should *stopwords* be ignored?
  - Should *hapax legomena* be included? Or even just the $k$ most frequent words?

**PLS Example (cont’d)**

- PRP*CHMIN= -0.4472, PRP*CHMAX=22.981
  - PLS1= -0.4472 CHMIN + 22.981 CHMAX
- Univariate Regression:
  - CHMIN= 0.0438 PLS1
  - CHMAX=0.0444 PLS1
- PLS2= -23.6002 CHMIN -0.4593 CHMAX
- All attribute residuals are zero now
- Use PLS directions as input for linear regression: *partial least squares regression model*
- If all directions are used, result is the same as with original attributes

**Time series**

- In time series data, each instance represents a different time step.
- Some simple transformations:
  - Shift values from the past/future
  - Compute difference (delta) between instances (i.e. “derivative”) when transforming
- In some datasets, samples are not regular but time is given by *timestamp* attribute
  - Need to normalize by step size when transforming
- Transformations need to be adapted if attributes represent different time steps
Sampling

- Sampling is typically a simple procedure
- What if training instances arrive one by one but we don't know the total number in advance?
  - Or perhaps there are so many that it is impractical to store them all before sampling?
- Is it possible to produce a uniformly random sample of a fixed size? Yes.
- Reservoir sampling
  - Fill the reservoir, of size \( r \), with the first \( r \) instances to arrive
  - Subsequent instances replace a randomly selected reservoir element with probability \( r/i \), where \( i \) is the number of instances seen so far

Automatic data cleansing

- To improve a decision tree:
  - Remove misclassified instances, then re-learn!
- Better (of course!):
  - Human expert checks misclassified instances
- Attribute noise vs class noise
  - Attribute noise should be left in training set (don't train on clean set and test on dirty one)
  - Systematic class noise (e.g. one class substituted for another): leave in training set
  - Unsystematic class noise: eliminate from training set, if possible

Robust regression

- “Robust” statistical method ⇒ one that addresses problem of outliers
- To make regression more robust:
  - Minimize absolute error, not squared error
  - Remove outliers (e.g. 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

Number of international phone calls from Belgium, 1950–1973
Detecting anomalies

- Visualization can help to detect anomalies
- Automatic approach: committee of different learning schemes
  - E.g.
    - decision tree
    - nearest-neighbor learner
    - linear discriminant function
  - Conservative approach: delete instances incorrectly classified by all of them
  - Problem: might sacrifice instances of small classes

One-Class Learning

- Usually training data is available for all classes
- Some problems exhibit only a single class at training time
  - Test instances may belong to this class or a new class not present at training time
- One-class classification
  - Predict either target or unknown
- Some problems can be re-formulated into two-class ones
- Other applications truly don't have negative data
  - E.g. password hardening

Outlier detection

- One-class classification is often called outlier/novelty detection
- Generic approach: identify outliers as instances that lie beyond distance \( d \) from percentage \( p \) of the training data
- Alternatively, estimate density of the target class and mark low probability test instances as outliers
  - Threshold can be adjusted to obtain a suitable rate of outliers

Generating artificial data

- Another possibility is to generate artificial data for the outlier class
  - Can then apply any off-the-shelf classifier
  - Can tune rejection rate threshold if classifier produces probability estimates
- Generate uniformly random data
  - Too much will overwhelm the target class!
    - Can be avoided if learning accurate probabilities rather than minimizing classification error
  - Curse of dimensionality – as # attributes increase it becomes infeasible to generate enough data to get good coverage of the space
Generating artificial data

- Generate data that is close to the target class
  - No longer uniformly distributed and must take this distribution into account when computing membership scores for the one-class model
- $T$ – target class, $A$ – artificial class. Want $Pr[X | T]$, for any instance $X$; we know $Pr[X | A]$
- Combine some amount of $A$ with instances of $T$ and use a class probability estimator to estimate $Pr[T | X]$; then by Bayes’ rule:
  $$Pr[X | T] = \frac{(1 - Pr[T])Pr[X | T]}{Pr[T]Pr[T | X]} Pr[X | A]$$
- For classification, choose a threshold to tune rejection rate
- How to choose $Pr[X | A]$? Apply a density estimator to the target class and use resulting function to model the artificial class

Transforming multiple classes to binary ones

- Some learning algorithms only work with two class problems
  - Sophisticated multi-class variants exist in many cases but can be very slow or difficult to implement
- A common alternative is to transform multi-class problems into multiple two-class ones
- Simple methods
  - Discriminate each class against the union of the others – one-vs.-rest
  - Build a classifier for every pair of classes – pairwise classification

Error-correcting output codes

- Multiclass problem $\Rightarrow$ binary problems
- Simple one-vs.rest scheme: One-per-class coding
- Idea: use error-correcting codes instead
- base classifiers predict 1011111, true class = ??
- Use code words that have large Hamming distance between any pair
- 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>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>

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 $\Rightarrow$ 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

<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>

More on ECOCs

- More classes $\Rightarrow$ exhaustive codes infeasible
  - Number of columns increases exponentially
  - Random code words have good error-correcting properties on average!
  - There are sophisticated methods for generating ECOCs with just a few columns
  - ECOCs don’t work with NN classifier
    - But: works if different attribute subsets are used to predict each output bit

Ensembles of nested dichotomies

- ECOCs produce classifications, but what if we want class probability estimates as well?
  - e.g. for cost-sensitive classification via minimum expected cost
- **Nested dichotomies**
  - Decomposes multi-class to binary
  - Works with two-class classifiers that can produce class probability estimates
  - Recursively split the full set of classes into smaller and smaller subsets, while splitting the full dataset of instances into subsets corresponding to these subsets of classes
    - Yields a binary tree of classes called a nested dichotomy

**Example**

Full set of classes: $[a, b, c, d]$

Two disjoint subsets: $[a, c] \quad [b, d]$

<table>
<thead>
<tr>
<th>Class vector</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
</tr>
<tr>
<td>b</td>
</tr>
<tr>
<td>c</td>
</tr>
<tr>
<td>d</td>
</tr>
</tbody>
</table>
Probability estimation

- Suppose we want to compute \( \Pr[a \mid x] \)?
  - Learn two class models for each of the three internal nodes
  - From the two-class model at the root: \( \Pr[a, c \mid x] \)
  - From the left-hand child of the root: \( \Pr[a \mid x, \{a, c\}] \)
  - Using the chain rule:
    \[
    \Pr[a \mid x] = \Pr[a \mid \{a, c\}, x] \times \Pr[\{a, c\} \mid x]
    \]

- Issues
  - Estimation errors for deep hierarchies
  - How to decide on hierarchical decomposition of classes?

Ensembles of nested dichotomies

- If there is no reason a priori to prefer any particular decomposition then use them all
  - Impractical for any non-trivial number of classes
- Consider a subset by taking a random sample of possible tree structures
  - Caching of models (since a given two class problem may occur in multiple trees)
  - Average probability estimates over the trees
  - Experiments show that this approach yields accurate multiclass classifiers
  - Can even improve the performance of methods that can already handle multiclass problems!

Calibrating class probabilities

- Class probability estimation is harder than classification
  - Classification error is minimized as long as the correct class is predicted with max probability
  - Estimates that yield correct classification may be quite poor with respect to quadratic or informational loss
- Often important to have accurate class probabilities
  - e.g. cost-sensitive prediction using the minimum expected cost method

Calibrating class probabilities

- Consider a two class problem. Probabilities that are correct for classification may be:
  - Too optimistic – too close to either 0 or 1
  - Too pessimistic – not close enough to 0 or 1

Reliability diagram showing overoptimistic probability estimation for a two-class problem
Calibrating class probabilities

- Reliability diagram generated by collecting predicted probabilities and relative frequencies from a 10-fold cross-validation
  - Predicted probabilities discretized into 20 ranges via equal-frequency discretization
  - Correct bias by using post-hoc calibration to map observed curve to the diagonal
  - A rough approach can use the data from the reliability diagram directly
- Discretization-based calibration is fast...
  - But determining the appropriate number of discretization intervals is not easy

View as a function estimation problem

- One input – estimated class probability – and one output – the calibrated probability
- Assuming the function is piecewise constant and monotonically increasing
  - *Isotonic regression* minimizes the squared error between observed class “probabilities (0/1) and resulting calibrated class probabilities
  - Alternatively, use *logistic regression* to estimate the calibration function
    - Must use the *log-odds* of the estimated class probabilities as input
    - Multiclass logistic regression can be used for calibration in the multiclass case