Data Mining
Practical Machine Learning Tools and Techniques

Slides for Chapter 7 of Data Mining by I. H. Witten and E. Frank

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
  - Combining models to improve performance

1. 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
  - But, 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):
    \[ \sum_j \sqrt{\sum_i U(A_i, A_j)} \]

Attribute subsets for weather data
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

Scheme-specific selection

- Wrapper approach to attribute selection
- Implement “wrapper” around learning scheme
- Evaluation criterion: cross-validation performance
- Time consuming
  - greedy approach, \( k \) attributes \( \Rightarrow k^2 \times \) 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 schemas search
- Learning decision tables: scheme-specific attribute selection essential
- Efficient for decision tables and Naive Bayes

WEKA’s Visualize panel ...

In WEKA ...

- Attribute subset evaluators (see pp. 421-422)
  - Correlation-based feature selection (CFS)
  - `weka.attributeSelection.CfsSubsetEval`
- Classifier subset evaluator
  - `weka.attributeSelection.ClassifierSubsetEval`
- Consistency attribute subset evaluator
  - `weka.attributeSelection.ConsistencySubsetEval`
- Use a classifier plus cross-validation
  - `weka.attributeSelectionWRAPPERSubsetEval`

In WEKA ...

- Single attribute evaluators (see pp. 421-423)
  - Chi-squared statistic
    - `weka.attributeSelection.ChiSquaredAttributeEval`
  - Gain ratio
    - `weka.attributeSelection.GainRatioAttributeEval`
  - Information gain
    - `weka.attributeSelection.InfoGainAttributeEval`
  - 1R methodology
    - `weka.attributeSelection.OneRAttributeEval`
In WEKA ...

- Single attribute evaluators continued ...
  - Principal components analysis (PCA) and transformation
    - weka.attributeSelection.PrincipalComponents
  - Instance-based (Recursive Elimination of Features)
    - weka.attributeSelection.ReliefFAtributeEval
  - Linear support vector machine (SVM)
    - weka.attributeSelection.SVMAtributeEval
  - Symmetric uncertainty
    - weka.attributeSelection.SymmetricalUncertAttributeEval

In WEKA ...

- Search methods for attribute selection (see pp. 421-425)
  - Greedy hill-climbing with / without backtracking
    - weka.attributeSelection.BestFirst / GreedyStepwise
  - Search exhaustively
    - weka.attributeSelection.ExhaustiveSearch
  - Search using a simple genetic algorithm (GA)
    - weka.attributeSelection.GeneticSearch
  - Race search methodology
    - weka.attributeSelection.RaceSearch
  - Random
    - weka.attributeSelection.RandomSearch
  - Sort attributes and rank via attribute subset evaluator
    - weka.attributeSelection.RankSearch

In WEKA ... another way ...

- 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)

2. 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 is based on more data
  - Apply learner to
    - k-valued discretized attribute or to
    - k–1 binary attributes that code the cut points

Discretization: unsupervised
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 MDL principle:
  - The “theory” is
  - the splitting point \( \log_2(N-1) \) bits
  - plus class distribution in each subset
  - Compare description lengths before/after adding splitting point

Example: temperature attribute

- **Formula for MDLP stopping criterion**
  - \( N \) instances
  - Original set: \( k \) classes, entropy \( E \)
  - First subset: \( k_1 \) classes, entropy \( E_1 \)
  - Second subset: \( k_2 \) classes, entropy \( E_2 \)

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

- Results in no discretization intervals for temperature attribute

Supervised discretization: other methods

- Can replace top-down (splitting) procedure by bottom-up (merging) method
- Can replace MDLP (entropy-based) stopping criterion by (statistical) chi-squared test
- Can use dynamic programming (versus exponential brute-force algorithm) 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. 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)*
The converse of discretization

- Make nominal values into “numeric” ones
  1. Indicator attributes (used by IB1)
     - Makes no use of potential ordering information
  2. Code an ordered nominal 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 attributes as binary

In WEKA ...

- Unsupervised attribute filters (see pp. 395-396)
  - Convert numeric to nominal (allows equal-interval vs. equal-frequency binning, plus numerous other parameters)
    - weka.filters.unsupervised.attribute.Discretize
  - Replace nominal with Boolean (1 within range, else 0)
    - weka.filters.unsupervised.attribute.MakeIndicator
  - Change nominal to several binary attributes (1 per value)
    - weka.filters.unsupervised.attribute.NominalToBinary
  - Convert all numeric attributes into binary ones
    - weka.filters.unsupervised.attribute.NumericToBinary

More ...

3. Data transformations

- 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 (probabilistic) cluster membership
  - Adding noise to data (for robustness tests)
  - Removing data randomly or selectively
  - Obfuscating the data (for anonymity)
Data Transformations in WEKA...

In WEKA...

- Some unsupervised attribute filters (see pp. 395-401)
  - Insert attribute at given position
    - `weka.filters.unsupervised.attribute.Add`
  - Copies attributes to preserve them when filters overwrite attribute values
    - `weka.filters.unsupervised.attribute.Copy`
  - Delete all attributes of a given type (nominal, numeric, ...)
    - `weka.filters.unsupervised.attribute.RemoveType`
  - Apply clustering algorithm to data before filtering
    - `weka.filters.unsupervised.attribute.AddCluster`

More...

In WEKA...

- More unsupervised attribute filters (see pp. 395-401)
  - Create new attribute by applying a mathematical function to numeric attributes
    - `weka.filters.unsupervised.attribute.AddExpression`
  - Create new attribute by applying a given Java function to numeric attributes
    - `weka.filters.unsupervised.attribute.AddJavaExpression`
  - Scale numeric values to range [0,1]
    - `weka.filters.unsupervised.attribute.Normalize`
  - Transform numeric values to have zero mean and unit variance
    - `weka.filters.unsupervised.attribute.Standardize`

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>

*Note: Component = axis, since each axis accounts for its share of the variance*

- Can transform data into space given by principal components
- Common to standardize attributes prior to applying PCA
- Could also apply this recursively in decision tree learner

In WEKA...

- PCA unsupervised attribute filter (see pp. 395-401)
  - Perform principal components analysis and transformation of the data (default 95% variance in original data)
    - `weka.filters.unsupervised.attribute.PCComponent`
  - code based on WEKA’s PCA attribute selection scheme
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
  - Much cheaper than PCA!

In WEKA ...

- Randomizing attribute filters (see pp. 396 & 400)
  - Change a given percentage of values of a nominal attribute
  - weka.filters.unsupervised.attribute.AddNoise
  - Rename relation, attributes, and nominal and string attribute values
  - weka.filters.unsupervised.attribute.Obfuscate
  - Use random matrix to project dataset to a lower-dimensional subspace
  - weka.filters.unsupervised.attribute.RandomProjection

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, term frequencies ($f_j$), $\log(1+f_j)$, or TF × IDF: $f_j \log \frac{\# \text{documents}}{\text{documents that include word } j}$
  - Only retain alphabetic sequences?
  - What should be used as delimiters?
  - Should words be converted to lowercase?
  - Should stopwords (e.g., the, and, but) be ignored?
  - Should hapax legomena be included? Or even just the $k$ most frequent words?

In WEKA ...

- String conversion attribute filters (see pp. 396 & 399)
  - Convert to set number of nominal values
  - weka.filters.unsupervised.attribute.StringToNominal
  - Produce attributes representing word frequency in a string
  - weka.filters.unsupervised.attribute.StringToWordVector

Time series

- In time series data, each instance represents a different time step (e.g., weather or stock market prediction)
- Some simple transformations:
  - Shift values from the past/future
  - Compute difference (delta) between instances (i.e. “derivative”)
- 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

In WEKA ...

- Time series attribute filters (see pp. 396 & 399-400)
  - Replace attribute values in current instance with equivalent value in some other (previous or future) instance
  - weka.filters.unsupervised.attribute.TimeSeriesTranslate
  - Replace attribute values in current instance with distance between current value and the value in some other instance
  - weka.filters.unsupervised.attribute.TimeSeriesDelta
4. Automatic data cleansing

- What for? Poor quality of available data ...
- 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

Data cleansing in WEKA ...

- Unsupervised instance filters (see pp. 400-401)
  - Remove instances incorrectly classified according to a
    specified classifier – useful for removing outliers
    - weka.filters.unsupervised.instance.RemoveMisclassified
  - Remove a given percentage of a dataset
    - weka.filters.unsupervised.instance.RemovePercentage
  - Remove a given range of instances
    - weka.filters.unsupervised.instance.RemoveRange
  - Filter out instances with certain attribute values
    - weka.filters.unsupervised.instance.RemoveWithValues
  - Produce random subsample of a dataset, sampling with
    replacement
    - weka.filters.unsupervised.instance.Resample

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

- Visualization can help to detect anomalies
  - “Automatic” approach: committee of different
    learning schemes to filter the data
    - E.g. Committee consisting of a
      - decision tree
      - nearest-neighbor learner
      - linear discriminant function
    - Conservative approach: delete instances
      incorrectly classified by them all
    - Problem: might sacrifice instances of small classes
    - Better: Human inspection of instances identified
      by the filter as suspect.

5. Combining multiple models

- Basic idea:
  build different “experts”, let them vote
- Advantage:
  - often improves predictive performance
- Disadvantage:
  - usually produces output that is very hard to
    analyze
  - but: there are approaches that aim to produce
    a single comprehensible structure
Bagging

- Combining predictions by voting/averaging
  - Simplest way
  - Each model receives equal weight
- “Idealized” version of bootstrap aggregating:
  - Bootstrap: Sample (with replacement) several training sets of size $n$ (instead of just having one training set of size $n$)
  - Build a classifier for each training set
  - Aggregation: Combine the classifiers’ predictions
- Learning scheme is unstable $\Rightarrow$ almost always improves performance
- Small change in training data can make big change in model (e.g. decision trees)

Bias-variance decomposition

- Used to analyze how much selection of any specific training set affects performance
- Assume infinitely many classifiers, built from different training sets of size $n$
- For any learning scheme,
  - $Bias = \text{expected error of the combined classifier on new data}$
  - $Variance = \text{expected error due to the particular training set used}$
- Total expected error $= bias + variance$

More on bagging

- Bagging works because it reduces variance by voting/averaging
  - Note: in some pathological hypothetical situations the overall error might increase
  - Usually, the more classifiers the better
  - Problem: we only have one dataset!
  - Solution: generate new ones of size $n$ by sampling from it with replacement
  - Can help a lot if data is noisy
  - Can also be applied to numeric prediction
  - Aside: bias-variance decomposition originally only known for numeric prediction

Bagging classifiers

Model generation

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

Classification

For each of the $t$ models:
  - Predict class of instance using model
  - Return class that is predicted most often

Bagging with costs

- Bagging unpruned decision trees known to produce good probability estimates
  - Where, instead of voting, the individual classifiers’ probability estimates are averaged
  - Note: this can also improve the success rate
  - Can use this with minimum-expected cost approach for learning problems with costs
  - Problem: not interpretable
    - MetaCost re-labels training data using bagging with costs and then builds single tree

Bagging in WEKA...
Bagging in WEKA...

- Metalearning algorithms for Bagging (see pp. 414-415)
  - Bag a classifier; works for regression, too
    - \texttt{weka.classifiers.meta.Bagging}
  - Make base classifier cost-sensitive
    - \texttt{weka.classifiers.meta.CostSensitiveClassifier}
  - Make a classifier cost-sensitive via Domingos (1999)
    - \texttt{weka.classifiers.meta.MetaCost}

Randomization I

- Can randomize learning algorithm instead of input
- Some algorithms already have a random component: e.g. initial weights in neural net/MLP
- Most algorithms can be randomized, e.g. greedy algorithms:
  - Pick from the $N$ best options at random instead of always picking the best options
  - \textit{E.g.}: attribute selection in decision trees

Randomization II

- Randomization can be combined with bagging
  - \textit{E.g.} learning random forests by building a randomized decision tree in each iteration of the bagging algorithm
- Unlike bagging, randomization can be applied to stable learners
  - randomize to make classifiers diverse
  - \textit{E.g.} Nearest-neighbor classifiers can be randomized by using different, randomly chosen subsets of attributes

Randomization in WEKA...

- Metalearning algs for randomization (see pp. 414-415)
  - Build an ensemble of randomizable base classifiers
    - \texttt{weka.classifiers.meta.RandomCommittee}

Boosting

- Also uses voting (classification) or averaging (numeric prediction)
- But: weights models according to performance
- Also: Iterative - new models are influenced by performance of previously built ones
  - Encourage new model to become an “expert” for instances misclassified by earlier models
  - Intuitive justification: models should be experts that complement each other
AdaBoost.M1 (for classification)

Model generation
Assign equal weight to each training instance
For $t$ iterations:
  Apply learning algorithm to weighted dataset, store resulting model
  Compute model's error $e$ on weighted dataset
  If $t < T^*$, error $e > 0.5$:
    Terminate model generation
    For each instance in dataset:
      If classified correctly by model:
        Multiply instance's weight by $\frac{1}{1-e}$
    Normalize weight of all instances

Classification
Assign weight $= 0$ to all classes
For each of the $t$ models (or fewer):
  For the class this model predicts
  $\text{add} -\log \frac{e}{1-e}$ to this class's weight
Return class with highest weight

More on boosting I
- Boosting needs weights ... but
  - can adapt learning algorithm ... or
  - can apply boosting without weights
    - resample with probability determined by weights
    - disadvantage: not all instances are used
    - advantage: if error > 0.5, can resample again
- Idea of boosting stems from computational learning theory
  - Theoretical result:
    - training error decreases exponentially
  - Also:
    - works if base classifiers are not too complex, and
    - their error does not become too large too quickly

Boosting in WEKA ...

More on boosting II
- Continue boosting after training error = 0?
- Puzzling fact:
  generalization error continues to decrease!
  - Seems to contradict Occam's Razor
- Explanation:
  consider margin (confidence), not error
  - Difference between estimated probability for true class and nearest other class (between -1 and 1)
- Boosting works with weak (simple) learners only condition: error does not exceed 0.5
- In practice, boosting sometimes overfits (in contrast to bagging) = less accurate than single classifier

Boosting in WEKA ...

Additive regression I
- Turns out that boosting is a greedy algorithm for fitting additive models
- More specifically, boosting implements forward stagewise additive modeling
- Same kind of algorithm for numeric prediction:
  1. Build standard regression model (e.g. regression tree)
  2. Gather residuals, learn model predicting residuals (e.g. regression tree), and repeat
- To predict, simply sum up individual predictions from all models

Metalearning algs for boosting (see pp. 415-416)
- Boost using the AdaBoost.M1 method
  - `weka.classifiers.meta.AdaBoostM1`
- Combine boosting with a variant of bagging to prevent overfitting
  - `weka.classifiers.meta.MultiBoostAB`
- Build ensembles of diverse classifiers by using specially constructed artificial training examples
  - `weka.classifiers.meta.Decorate`
Additive regression II

- Minimizes squared error of ensemble if base learner minimizes squared error of predictions
  - Note: Does not make sense to use standard linear regression as a base learner
  - Can use it with simple (single attribute) linear regression to build multiple linear regression model
- Stopping criteria: Use cross-validation to avoid overfitting ...

Additive logistic regression

- Can use the logit transformation to get algorithm for classification
  - More precisely, class probability estimation
  - Probability estimation problem is transformed into regression problem
  - Regression scheme is used as base learner (e.g. regression tree learner)
- Can use forward stagewise algorithm: at each stage, add model that maximizes probability of data
  - If \( f_j \) is the \( j \)th regression model, the ensemble predicts probability
    \[
    p(y | \tilde{a}) = \frac{1}{1 + \exp(- \sum f_j / a)}
    \] for the first class

LogitBoost (for 2-class problems)

Model generation

For \( j = 1 \) to \( t \) iterations:
- For each instance \( a[i] \):
  - Set the target value for the regression to
    \[
    z[i] = (y[i] - p(1|a[i])) / \{(p(1|a[i]) \times (1 - p(1|a[i]))
    \]
  - Set the weight of instance \( a[i] \) to \( p(1|a[i]) \times (1 - p(1|a[i]))
- Fit a regression model \( f[j] \) to the data with class values \( z[i] \) and weights \( w[i] \)

Classification

Predict 1" class if \( p(1 | a) > 0.5 \), otherwise predict 2" class

- Maximizes probability if base learner minimizes squared error
- Difference to AdaBoost: optimizes probability/likelihood instead of exponential loss; uses regression method as base
- Can be adapted to multi-class problems
- Shrinking and cross-validation-based selection apply

Option trees

- Issue: Ensembles are not interpretable
- Can we generate a single model?
  - One possibility: "cloning" the ensemble by using lots of artificial data that is labeled by ensemble
  - Another possibility: generating a single structure that represents ensemble in compact fashion
- Option tree: decision tree with option nodes
  - Idea: follow all possible branches at option node
  - Predictions from different branches are merged using voting or by averaging probability estimates

Example

- Can be learned by modifying tree learner:
  - Create option node if there are several equally promising splits (within user-specified interval)
  - When pruning, error at option node is average error of options

Alternating decision trees

- Can also grow option tree by incrementally adding nodes to it via a boosting algorithm
- Structure called alternating decision tree, with splitter nodes and prediction nodes
  - Prediction nodes are leaves if no splitter nodes have been added to them yet
  - Standard alternating tree applies to 2-class problems
- To obtain prediction, filter instance down all applicable branches and sum predictions
  - Predict one class or the other depending on whether the sum is positive or negative
Growing alternating trees

- Tree is grown using a boosting algorithm
  - E.g. LogitBoost described earlier
  - Assume that base learner produces single conjunctive rule in each boosting iteration (note: rule for regression)
  - Each rule could simply be added into the tree, including the numeric prediction obtained from the rule
  - Problem: tree would grow very large very quickly
  - Solution: base learner should only consider candidate rules that extend existing branches
    - Extension adds splitter node and two prediction nodes (assuming binary split)
  - Standard algorithm chooses best extension among all possible extensions applicable to tree
  - More efficient heuristics can be employed instead

More boosting in WEKA ...

- Add'l. Metalearning algos for boosting (see pp. 415-416)
  - Enhance performance of regression method by iteratively fitting the residuals
    - weka.classifiers.meta.AdditiveRegression
  - Perform additive logistic regression
    - weka.classifiers.meta.LogitBoost
  - Batch-based incremental learning by racing logit-boosted committees
    - weka.classifiers.meta.RacedIncrementalLogitBoost

Stacking

- To combine predictions of base learners, don’t vote, use meta learner
  - Base learners: level-0 models
  - Meta learner: level-1 model
  - Predictions of base learners are input to meta learner
  - Base learners are usually different schemes
  - Can’t use predictions on training data to generate data for level-1 model!
    - Instead use cross-validation-like scheme
  - Hard to analyze theoretically: “black magic”
More on stacking

- If base learners can output probabilities, use those as input to meta learner instead
- Which algorithm to use for meta learner?
  - In principle, any learning scheme
  - Prefer "relatively global, smooth" model
  - Base learners do most of the work
  - Reduces risk of overfitting
- Stacking can be applied to numeric prediction too

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

ECOC: Error-correcting output codes

- Multiclass problem \( \Rightarrow \) binary problems
  - Simple 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

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

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

Stacking in WEKA ...

- Classifiers: Support Vector Machine, K Nearest Neighbours, Multilayer Perceptron, Naive Bayes, Decision Table
- Stacking method: "Randomized Stacking"
- Stack size: 30
- Stacking attributes: Bayesian, C4.5, Naive Bayes, C5.0, KNearest Neighbours
- Output: 3,000 runs
Stacking in WEKA ...

- Metalearning algos for stacking (see pp. 415-416)
  - Combine several classifiers via stacking
    - `weka.classifiers.meta.Stacking`
  - More efficient version of stacking
    - `weka.classifiers.meta.StackingC`
  - Metalearners whose inputs are base-level predictions marked as correct or incorrect
    - `weka.classifiers.meta.Grading`

6. Using unlabeled data

- Semisupervised learning: attempts to use unlabeled data as well as labeled data
  - The aim is to improve classification performance
  - Why try to do this? Unlabeled data is often plentiful and labeling data can be expensive
    - Web mining: classifying web pages
    - Text mining: identifying names in text
    - Video mining: classifying people in the news
  - Leveraging the large pool of unlabeled examples would be very attractive

Clustering for classification

- Idea: use naive Bayes on labeled examples and then apply EM
  - First, build naive Bayes model on labeled data
  - Second, label unlabeled data based on class probabilities (“expectation” step)
  - Third, train new naive Bayes model based on all the data (“maximization” step)
  - Fourth, repeat 2\textsuperscript{nd} and 3\textsuperscript{rd} step until convergence
  - Essentially the same as EM for clustering with fixed cluster membership probabilities for labeled data and \#clusters = \#classes

Comments

- Has been applied successfully to document classification
  - Certain phrases are indicative of classes
  - Some of these phrases occur only in the unlabeled data, some in both sets
  - EM can generalize the model by taking advantage of co-occurrence of these phrases
  - Refinement 1: reduce weight of unlabeled data
  - Refinement 2: allow multiple clusters per class

Co-training

- Method for learning from multiple views (multiple sets of attributes), e.g.:
  - First set of attributes describes content of web page
  - Second set of attributes describes links that link to the web page
  - Step 1: build model from each view
  - Step 2: use models to assign labels to unlabeled data
  - Step 3: select those unlabeled examples that were most confidently predicted (ideally, preserving ratio of classes)
  - Step 4: add those examples to the training set
  - Step 5: go to Step 1 until data exhausted
  - Assumption: views are independent

EM and co-training

- Like EM for semisupervised learning, but view is switched in each iteration of EM
  - Uses all the unlabeled data (probabilistically labeled) for training
  - Has also been used successfully with support vector machines
    - Using logistic models fit to output of SVMs
  - Co-training also seems to work when views are chosen randomly!
    - Why? Possibly because co-trained classifier is more robust