Data Mining
Practical Machine Learning Tools and Techniques

Slides for Chapter 8 of *Data Mining* by I. H. Witten, E. Frank and M. A. Hall
Ensemble learning

- Combining multiple models
  - The basic idea

- Bagging
  - Bias-variance decomposition, bagging with costs

- Randomization
  - Rotation forests

- Boosting
  - AdaBoost, the power of boosting

- Additive regression
  - Numeric prediction, additive logistic regression

- Interpretable ensembles
  - Option trees, alternating decision trees, logistic model trees

- Stacking
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:
  - 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 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 $\approx bias + variance$
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
Randomization

- Can randomize learning algorithm instead of input
- Some algorithms already have a random component: e.g. initial weights in neural net
- Most algorithms can be randomized, e.g. greedy algorithms:
  - Pick from the $N$ best options at random instead of always picking the best options
  - Eg.: attribute selection in decision trees
- More generally applicable than bagging: e.g. random subsets in nearest-neighbor scheme
- Can be combined with bagging
Rotation forests

- Bagging creates ensembles of accurate classifiers with relatively low diversity
  - Bootstrap sampling creates training sets with a distribution that resembles the original data
- Randomness in the learning algorithm increases diversity but sacrifices accuracy of individual ensemble members
- Rotation forests have the goal of creating accurate and diverse ensemble members
Rotation forests

- Combine random attribute sets, bagging and principal components to generate an ensemble of decision trees
- An iteration involves
  - Randomly dividing the input attributes into $k$ disjoint subsets
  - Applying PCA to each of the $k$ subsets in turn
  - Learning a decision tree from the $k$ sets of PCA directions
- Further increases in diversity can be achieved by creating a bootstrap sample in each iteration before applying PCA
Boosting

- Also uses voting/averaging
- Weights models according to performance
- 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
- Several variants
**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 $e = 0$ or $e \geq 0.5$:
  - Terminate model generation
- For each instance in dataset:
  - If classified correctly by model:
    - Multiply instance’s weight by $e / (1-e)$
- Normalize weight of all instances

**Classification**

Assign weight = 0 to all classes

For each of the $t$ (or less) models:
- For the class this model predicts
  - add $-\log e / (1-e)$ 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
• Stems from *computational learning theory*
• Theoretical result:
  • training error decreases exponentially
• Also:
  • works if base classifiers are not too complex, and
  • their error doesn’t become too large too quickly
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 learners
  - only condition: error doesn’t exceed 0.5
- In practice, boosting sometimes overfits (in contrast to bagging)
Additive regression I

- 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 (e.g., tree).
  2. Gather residuals, learn model predicting residuals (e.g., tree), and repeat.
- To predict, simply sum up individual predictions from all models.
Additive regression II

- Minimizes squared error of ensemble if base learner minimizes squared error.
- Doesn't make sense to use it with standard multiple linear regression, why?
- Can use it with *simple* linear regression to build multiple linear regression model.
- Use cross-validation to decide when to stop.
- Another trick: shrink predictions of the base models by multiplying with pos. constant < 1.
  - Caveat: need to start with model 0 that predicts the mean.
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
  \[
  p(1 | \tilde{a}) = \frac{1}{1 + \exp(-\sum f_j(\tilde{a}))}
  \]
  for the first class
LogitBoost

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\textsuperscript{st} class if \( p(1 | a) > 0.5 \), otherwise predict 2\textsuperscript{nd} class

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

- 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
- 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
Example
Growing alternating trees

- Tree is grown using a boosting algorithm
  - Eg. 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 splits)
  - Standard algorithm chooses best extension among all possible extensions applicable to tree
  - More efficient heuristics can be employed instead
Logistic model trees

- Option trees may still be difficult to interpret
- Can also use boosting to build decision trees with linear models at the leaves (i.e. trees without options)
- Algorithm for building logistic model trees:
  - Run LogitBoost with simple linear regression as base learner (choosing the best attribute in each iteration)
  - Interrupt boosting when cross-validated performance of additive model no longer increases
  - Split data (e.g. as in C4.5) and resume boosting in subsets of data
  - Prune tree using cross-validation-based pruning strategy (from CART tree learner)
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