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

- 8.1 Combining multiple models
  - The basic idea
- 8.2 Bagging
  - Bias-variance decomposition, bagging with costs
- 8.4 Boosting
  - AdaBoost, the power of boosting
- 8.7 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* ⇒ 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* = expected error of the combined classifier on new data
  - *Variance* = 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

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

AdaBoost.M1

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 ≥ 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)

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