Multi-Class Boosting

Mario Rodriguez
UC Santa Cruz
CMPS 242 '09
Outline

• Wine Quality Dataset

• Classifiers
  • SoftMax Regression
  • Decision Tree
  • Neural Network

• Boosting
  • AdaBoost.M1
  • SAMME

• Conclusions
Wine Quality Dataset

- Found at UCI Machine Learning Repository
- Goal: predict wine quality based on physicochemical data
- White Wine Dataset
  - 4898 data points, 11 real attributes, 1 discrete target
  - Target is ranking from 1-10 (no zeros in data)

“Regarding the preferences, each sample was evaluated by a minimum of three sensory assessors (using blind tastes), which graded the wine in a scale that ranges from 0 (very bad) to 10 (excellent). The final sensory score is given by the median of these evaluations.” (http://www3.dsi.uminho.pt/pcordova/winequality09.pdf)
Classifiers

• Treated problem as a classification task (not regression)

• Used the following comparators:
  • SoftMax Regression, an example of a linear classifier (my implementation)
  • Decision Tree, sort of in-between linear and non-linear, with jagged decision boundaries (my implementation)
  • Neural Network, an example of a non-linear classifier (R's nnet package)
Classifiers (SoftMax)

- Normalized data to 0 mean and 1 standard deviation and added unit vector covariate (bias)

- Modeled the target variable as multinomial distribution conditional on the regressors:
  \[ Y = \frac{\exp(X \cdot W)}{\text{repmat}\left(\sum(\exp(X \cdot W), 2), 1, \text{size}(W, 2)\right)} \]

- Used Gradient Ascent to find the parameter matrix \( W \) that maximized the log-likelihood:
  \[ \text{log\_likelihood} = \text{sum}(\log(\text{softmax}(X, W))) \]
SoftMax Regression Parameter Optimization

Number of Epochs

Log-Likelihood

Student Version of MATLAB
Classifiers (Tree)

- Greedily built a binary decision tree by splitting on the covariate that minimized the entropy impurity (until leaf node support, 0.01% of training data, was broken):

  \[
  \text{entropy} = -\sum(P_x \times \log(P_x))
  \]

- Not pruning yielded better test set performance
- Did not do any feature pre-processing (z-score transformation, etc)
Node 1 (branch)  
Rule: $x_{11} < 9.75$

Node 3 (branch)  
Rule: $x_{6} < 13.5$

Node 7 (branch)  
Rule: $x_{11} < 11.05$

Node 416 (leaf)  
Class: 6

*Student Version of MATLAB*
Classifiers (NNet)

- Trained 20 neural nets with a varying number of hidden nodes in a single hidden layer (same feature pre-processing as softmax regression)
- Best network architecture was found to be one with 7 nodes in the hidden layer (11-7-10), with a total of 164 weights to be fitted:

  \# weights = 11*Nh \quad \text{edges from input to single hidden layer} \\
  + 1*Nh \quad \text{edges from bias to single hidden layer} \\
  + Nh*10 \quad \text{edges from single hidden layer to output} \\
  + 1*10 \quad \text{edges from bias to output}
Boosting

- Goal is to improve the accuracy of a given learning algorithm
- Decision Trees were chosen to be the component classifiers in the boosted ensemble:
  - Relatively fast to learn
  - Handle discrete and continuous covariates easily
  - No feature pre-processing required
  - “Built-in” feature selection
  - Boosting “tames” a single tree's high-variance without demanding high-bias
Boosting (AdaBoost.M1)

- Adaboost.M1 is the original AdaBoost applied to the multi-class classification problem:

```plaintext
for t=1:T
    [classifier, error, diff_vec] = train_classifier_on_weighted_data(X, y, w);
    if (error<=0 || error>=.5) return;
    alpha = log((1-error)/error);
    ensemble{t,1} = alpha; ensemble{t,2} = classifier;
    w = normalize(w.*exp(alpha*(diff_vec*2-1)));
```

- At prediction time, each component classifier contributes its alpha to the class it predicts

- The highest class wins (normalizing the alphas gives a distribution as prediction)
Boosting (AdaBoost.M1)

- AdaBoost.M1 requires the components classifiers to have an accuracy better than 50%
- One obvious way to remedy this is to break the original problem into several binary classification problems:
  - One-against-one for all pairs of classes
  - One-against-all for each class
- There is a cleaner solution when the component classifiers are able to do multi-class prediction...
Boosting (SAMME)

• SAMME stands for Sequential Additive Modeling with a Multi-class Exponential loss function (2 small changes to AdaBoost.M1):

  for t=1:T
  
  \[ [\text{classifier}, \text{error}, \text{diff_vec}] = \text{train_classifier_on_weighted_data}(X, y, w); \]
  
  if (error<=0 || error>= (1-(1/k))) return;

  alpha = log((1-error)/error) + log(k-1);

  ensemble{t,1}=alpha; ensemble{t,2}=classifier;

  w=normalize(w.*exp(alpha*(diff_vec*2-1)));

• Now component classifiers just need to be better than average

• AdaBoost.M1 is SAMME for k=2
Boosting (SAMME)

- The changes to AdaBoost.M1 are not arbitrary, they come from a statistical perspective of AdaBoost in which it fits an additive expansion in a set of elementary “basis” functions which seeks to optimize an exponential loss function:
  \[ L(y, f(x)) = \exp(-y \ast f(x)) \]

- This exponential loss function generalizes easily to the multi-class case as:
  \[ L(y, f(x)) = \exp\left(-\frac{1}{k} y^\top \ast f(x)\right) \]

(http://www-stat.stanford.edu/~hastie/Papers/samme.pdf)
Conclusions

• Boosting decision trees an excellent “off-the-shelf” algorithm
  • Fairly automatic, fast, salable, and great results
  • Over-fitting?!?
• Boosting can turn any classifier into a probabilistic classifier (connections to optimal Bayes classifier)
• Perceived wine quality tends to increase with the level of alcohol content :)