Experiments on spam detection with Boosting, SVM and Naive Bayes
CMPS 242 final project, winter 2008

Carl Liu$^1$

$^1$University of California, Santa Cruz

Last update: March 20, 2008
Introduction

Algorithms

Experiments

Results

Two more algorithms

Conclusion
Goal: evaluate performance of 3 popular classifiers
Application: spam detection
Experiments: error rate, learning curve, speed
Environment: Matlab
Outline

1. Introduction
2. Algorithms
3. Experiments
4. Results
5. Two more algorithms
6. Conclusion
AdaBoost

Initialize weights $D_1(i) = \frac{1}{N}, i = 1, 2, \ldots, N$

FOR $t = 1$ to $T$

- Call weak learner and get a weak hypothesis $h_t$
- Calculate error rate $\epsilon_t = \sum_{i: h_t(d_i) \neq y_i} D_t(i)$
- Set $\alpha_t = \frac{1}{2} \ln\left(\frac{1-\epsilon_t}{\epsilon_t}\right)$
- Update $D_{t+1}(i) = \frac{D_t(i) \exp(-\alpha_t y_i h_t(x_i))}{Z_t}$
  
  $Z_t$ is a normalization factor that makes $D_t + 1$ a distribution

ENDFOR

Output the final hypothesis: $H(x) = \text{sign}\left(\sum_{t=1}^T \alpha_t h_t(x)\right)$
Nice properties about AdaBoost

- fast
- simple and easy to program
- no parameters to tune (except T)
- flexible can combine with any learning algorithm
- no prior knowledge needed about weak learner
- effective
Support Vector Machines (SVM)
Support Vector Machines (SVM)

Decision boundary: $\mathbf{w} \cdot \mathbf{x} + b = 0$

Optimization problem:

$$\min_{\mathbf{w}} \frac{||\mathbf{w}||^2}{2} \text{subject to } y_i(\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1, \ i = 1, 2, \ldots, N.$$ 

Find support vectors:

$$W(\alpha) = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j (\mathbf{x}_i \cdot \mathbf{x}_j) y_i y_j$$

Final hypothesis:

$$F(\mathbf{x}_j) = \text{sign}(\mathbf{w}^* \cdot \mathbf{x}_j - b)$$

$$\mathbf{w}^* = \sum_{i=1}^{r} \alpha_i y_i \mathbf{x}_i$$
Kernel tricks

SVM target function:

\[ F(x_j) = \text{sign} \left( \sum_{i=1}^{r} \alpha_i y_i < x_i, x_j > - b \right) \]

A kernel function is in the form:

\[ K(x, y) = \phi(x)^T \phi(y) \]

\( \phi \) maps features from original space to an inner-product space. By replacing inner-products \( < x, y > \) with \( K(x, y) \) (i.e. \( < \phi(x), \phi(y) > \)), the algorithm can learn using \( \phi \). It means we do not even have to find \( \phi \), which is very expensive to calculate in high dimensional space.
AdaBoost and SVM

Similar decision boundary

$$\max_{\alpha} \min_i \frac{(\alpha \cdot h(x_i))y_i}{\|\alpha\| \|h(x_i)\|}$$

Differences:

**Table:** Difference between AdaBoost and SVM

<table>
<thead>
<tr>
<th></th>
<th>norms</th>
<th>optimization</th>
<th>computation model</th>
</tr>
</thead>
<tbody>
<tr>
<td>AdaBoost</td>
<td>$l_\infty$, $l_1$</td>
<td>linear programming</td>
<td>greedy search weights</td>
</tr>
<tr>
<td>SVM</td>
<td>$l_2$</td>
<td>quadratic programming</td>
<td>kernel tricks</td>
</tr>
</tbody>
</table>
Naive Bayes

The basic assumption of Naive Bayes is each word in the email is drawn independently from a distribution. To classify a message, we use:

$$p(c_s|x) = \frac{p(c_s) \cdot p(x|c_s)}{p(x)} = \frac{p(c_s) \cdot p(x|c_s)}{p(c_s) \cdot p(x|c_s) + p(c_h) \cdot p(x|c_h)} > T$$

where $x$ is the email message, $c_h$ and $c_s$ are category nonspam(ham) and spam.
Two Naive Bayes models

- Multi-variate Bernoulli model: message generated by Bernoulli trials.
  \[ p(\mathbf{x}|c_s) = \prod_{i=1}^{m} p(t_i|c_s)^{x_i} \cdot (1 - p(t_i|c_s))^{(1-x_i)} \]

- Multinomial model: capture the word frequency
  \[ p(\mathbf{x}|c_s) = p(|d|) \cdot |d|! \cdot \prod_{i=1}^{m} \frac{p(t_i|c_s)^{x_i}}{x_i!} \]
Outline

1. Introduction
2. Algorithms
3. Experiments
4. Results
5. Two more algorithms
6. Conclusion
Dataset

- provided by D. Sculley
- 2000 messages, 628 spams (31.4%)
- Matlab data file, first value 0: non-spam, 1: spam. other values 1: has word, 0: does not have word
- dictionary size: 2000. i.e. 2000 dimensional feature space.
10-fold cross validation

K-fold cross validation:
- 1 subsets for test, k-1 for train
- Every data got test once, training k-1 times
- Compute average error
- Good: Reduce variance as k is increased
- Bad: k times as much computation to make an evaluation
Experiments on spam detection with Boosting

Measurement

- error rate = \( \frac{\text{misclassified messages}}{\text{total messages}} \)
- miss rate = \( \frac{\text{misclassified nonspam messages}}{\text{total nonspam messages}} \)
- false alarm rate = \( \frac{\text{misclassified spam messages}}{\text{total spam messages}} \)
- Learning curves
- Speed
- Over-fitting resistance
## Error rate

Table: Error rate, miss rate and false alarm rate

<table>
<thead>
<tr>
<th>Method</th>
<th>Train error</th>
<th>Test error</th>
<th>Train miss</th>
<th>Test miss</th>
<th>Train f_alarm</th>
<th>Test f_alarm</th>
</tr>
</thead>
<tbody>
<tr>
<td>AdaBoost</td>
<td>2.25%</td>
<td>7.81%</td>
<td>1.38%</td>
<td>5.58%</td>
<td>4.07%</td>
<td>13.01%</td>
</tr>
<tr>
<td>SVM</td>
<td>6.03%</td>
<td>9.86%</td>
<td>4.31%</td>
<td>6.82%</td>
<td>8.76%</td>
<td>15.45%</td>
</tr>
<tr>
<td>Naive Bayes</td>
<td>n/a</td>
<td>7.95%</td>
<td>n/a</td>
<td>6.03%</td>
<td>n/a</td>
<td>14.45%</td>
</tr>
</tbody>
</table>
Learning curves

![Learning curve - by samples](image1)

![Learning curve - by features](image2)
Speed

![Testing Speed - by samples](image1)

![Testing Speed - by features](image2)
Over-fit test on AdaBoost

![AdaBoost over-fitting test graph]
Outline

1. Introduction
2. Algorithms
3. Experiments
4. Results
5. Two more algorithms
6. Conclusion
K Nearest Neighbours and Perceptron

Learning curve - by samples

- KNN
- Perceptron

Testing error(%)

Samples

Testing error(%) vs Samples graph showing the performance of KNN and Perceptron algorithms.

0 200 400 600 800 1000 1200 1400 1600 1800 2000

Samples
Conclusion

- AdaBoost has the lowest error rate
- Naive Bayes is the fastest
- SVM is acceptable
- Overall: Naive Bayes performs best
Q&A

Questions?