Predicting Code Efficiency Automatically on the Google Code Jam Dataset

Jijiang Yan (jjyan@uw.edu) and Daniel Graf (grafdan@ethz.ch)
Department of Computer Science & Engineering, University of Washington

CSE515 Statistical Methods in Computer Science – Spring 2013

Background

- Largest online coding competition with over *10,000* participants every year from over 150 countries
- Algorithmic task sets, each having:
  - small input: brute force is enough
  - large input: needs clever algorithm
- Popularity of automatic grading systems for programming classes on large scale (like coursera)

Motivation

- Organizers can not review all submissions in a timely manner
- Interest in automatic detection of new types of solutions, attacks etc.
- Compiling and running submitted code requires a lot of resources and a trusted environment
- Hope for deeper insights from automatic classification of efficiency

Goals

- Collect and prepare all the submissions from several tasks
- Extract static features of the submitted code source files
- Train and evaluate classifiers
  - Naive Bayes and logistic regression for single tasks
  - Multi-task logistic regression for application on new tasks

Data Collection and Feature Extraction

- Collected correct C++ submissions for 6 different tasks
  - *18,606* programs with *1.275* million lines of code
- Extracted *35* features using static string searches, like: counts of keywords (defines, includes, loops, conditionals, STL-classes), lengths of comments, depth of branching, number of functions, biggest integer constant and so on
- Converted to binary features by comparing with quantiles (i.e. median only or 3-quantiles) and others

Multi-Task Logistic Regression

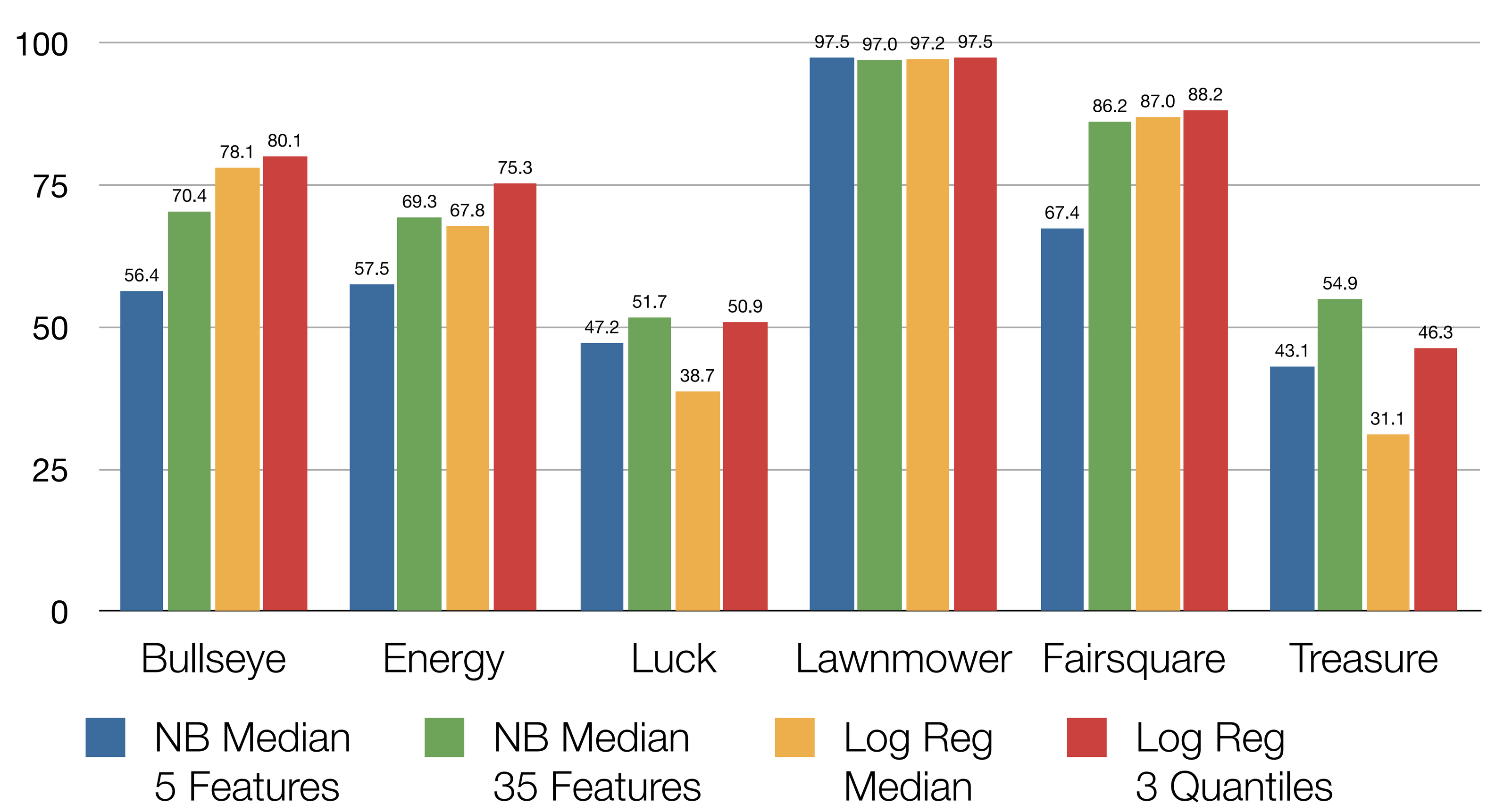
by À. Lapedriza, D. Masip, and J. Vitrià, 2007
Pattern Recognition and Image Analysis, Springer

- Train models for multiple tasks $T_1, \ldots, T_M$ with weight matrix $W = (\mathbf{w}^{(1)}, \ldots, \mathbf{w}^{(M)})$ simultaneously
- Regularize feature weights by penalizing deviations from the mean weight vector $\bar{\mathbf{w}}$, resulting in the loss function $G(W) = L(D, W) + \frac{\lambda_1}{2}\|\bar{\mathbf{w}}\|_2 + \frac{\lambda_2}{2}\sum_{i=1}^{M}\|\mathbf{w}^{(i)} - \bar{\mathbf{w}}\|_2$ where $L(D, W)$ is the negated log-likelihood estimator

Single-Task Classifier Results

- For Naive Bayes using median as threshold is best
- Logistic regression outperforms Naive Bayes in 4 of 6 tasks
- Logistic regression regularization parameter has only a significant effect in the 2 hard tasks where NB is better

F1-measure for separate classifiers on each of the 6 tasks



Multi-Task Classifier Results

- Multi-task log. reg. outperforms the standard log. reg.
- However: Naive Bayes is as good as multi-task log. reg.
- Lawnmower task in training set introduces a strong bias
- Log. reg. classifier on the same task is still a lot better

F1-measure when training on 3 tasks and testing on 3 other tasks