Machine Learning In JavaScript

David Frankl
dfrankl@stanford.edu

December 10, 2014

1 Introduction

The web is ubiquitous, yet many machine learning algoriths cannot be readily found and applied in JavaScript. An inherent downside to machine learning in JavaScript is lack of speed. However, as the language becomes increasingly more popular, the need for machine learning algorithms steadily rises. First and foremost, JavaScript is the language of the web browser. Having machine learning available in the web browser allows for delivery of machine learning tools to users in the most convenient way possible. This opens up opportunities for non-Software Developers to gain access to the power of machine learning. In addition, browser-based machine learning allows for effective visualization of algorithms, which can help with education, as well as quick visualization of data. Second, with the rise of Node.js, JavaScript is being increasingly relied upon to do non-Browser Based work. The growth of the Node.js ecosystem will rely on the availability of easy to use libraries. A final reason for implementing machine learning in JavaScript is Atwood's law: "any application that can be written in JavaScript, will eventually be written in JavaScript" [1].

2 Implementation

My project consists of two relatively isolated pieces of implementation. First is the algorithm implementation, and second is visualization of some aspect of the algorithm, typically convergence. The result is a library which I have called MachineLearningJS, and a website which hosts the visualizations, MachineLearningJS.com.

2.1 Preliminaries

To minimize dependencies and library size, I have implemented the basic mathematical operations myself. Most important are operations on vectors and matrices, which I have represented as one-dimensional and two-dimensional arrays respectively. Required matrix operations include matrix multiplication, transpose, and inverse. To improve the stability of the matrix operations (at the cost of decreased efficiency), the library has optional exact arithmetic by using rational representation. To allow for this usage, we convert from floats to fractions based on the theory of continued fractions. Suppose the continued fraction representation of real number r is

$$r = a_0 + \cfrac{1}{a_1 + \cfrac{1}{a_2 + \ddots}} = [a_0; a_1, a_2, \dots]$$

Then,

$$r = a_0 + [a_1; a_2, a_3, \dots]$$

where $[a_1; a_2, a_3, \dots]$ is the continued fraction representation of $\frac{1}{r-\lfloor r \rfloor}$. Thus we have an iterative procedure for finding the best possible rational representation of r.

2.2 Algorithms

I implemented Linear Regression, Logistic Regression, Least Squares Approximation of Polynomial Fit, and K-Means Clustering. Both Linear Regression and Logistic Regression utilize Stochastic Gradient Descent. The step parameter α is tuned dynamically with the following procedure on each iteration:

```
If error(current iteration) > error(previous iteration):
    alpha = alpha/2
Else with probability .1:
    alpha = alpha * 1.1
```

This allows α to converge to the maximum step size possible, without the need for manual tuning. Least Squares is implemented using the normal equation, with exact arithmetic on by default.

Algorithm usage follows an object oriented approach, whereby the user creates an object representing the model, then calls train() and test() manually. In total, the library consists of 9KB of compressed JavaScript.

3 Visualization

Visualizations of Linear and Logistic regression make use of a trendline which shows the progress of gradient descent on converging to an optimal solution. With Linear Regression, the adjacent figure gives a plot of error rate over time. The adjacent figure to Logistic Regression gives a plot of percent of correct classifications of the training set over time. For Polynomial Regression, I allow the user to manipulate the degree of the polynomial, which gives an intuitive understanding of the relationship between polynomial degree, approximation error, and under/over-fitting. For K-Means Clustering, the clusters are represented by concentric circles, and their center point is updated on each iteration. All of the algorithms can be tested on multiple datasets. I get many of the test datasets from an archive of real-world data hosted by Larry Winner [2].

4 Future Work

In the future, two improvements can be made. First, the breadth of the library can be increased. In addition, the implementations can be tuned for efficiency. For example, the matrix operations are for now implemented naively. Both matrix inverse and matrix multiplication would benefit from relatively simple improvements. Finally, I plan to adapt the codebase to function as a module on the Node.js platform, and provide convenience functions to allow the use of input files rather than input arrays.

References

[1] Jeff Atwood. The principle of least power. http://blog.codinghorror.com/the-principle-of-least-power.

[2] Larry Winner. Miscellaneous datasets. http://www.stat.ufl.edu/~winner/datasets.html. Accessed: 2014-12-10.