CS229/CS229A Final Project Writeup: Supervised Learning - Stock Trend Classifier

Submitted: 12/16/2011

ChihChi Kao
ckao@stanford.edu

0. Note for teaching staff

Unfortunately my project partner, Brain Von Osdol, has decided to withdraw from CS229 this quarter after the week of CS229 midterm exam. This rather late notice limit my options completing the final project on my own effort.

This said, I am very passionate about what we were trying to achieve and am willing to complete the work by myself. Because it was rather late into the quarter when my partner withdrew from class, therefore I don't think it is a good idea to join a team. It would not be a fair game for both the team members from other CS229 project team and myself.

I assume that I have the explicit approval form CS229/CS229A staff that it is okay to complete this final project on my own since I did not hear back after submitted project milestone. Also, I am hoping when it comes to grading, the grade can be evaluated at the quality of my work, instead of the quantity. I may not be able to implement all the features/classifiers that we originally wanted to build, but every step into the project will still demonstrate my solid understanding to supervised machine learning concepts and model diagnostic skills.

1. Project Setup

In this project I am going to train a handful of classifiers to predict the stock trend on the next day in terms of the closing price. In other words, the class label of example at day d is 1 if the close price (d+1) > close price (d) and 0 otherwise.

I will design a numbers of features that reflect the embedded information that ultimately affect the stock closing price of the next trading day. Will analyze non-linear features along side with linear features in this project.

At each step, I will perform error analysis and diagnose the learning algorithm, feature, and learning objective to improve the overall prediction accuracy. I will use the analysis mentioned in class to address the biggest error-contributing factor in the next step.

Finally a performance metric will be designed to measure the performance of my classifier. I will also provide a plan for future improvement.

2. Stock Data Preprocess

In this step I transform the raw stock data that is stored in csv format file to the conventional data format adopted in CS229/CS229A.

2.1

The first thing I do in this section is to generate the class label ($y^{(i)}$) for a given example. Since $x^{(i)}$'s are examples each representing a feature vector corresponds to date i. I define the class label $y^{(i)} = 1$, if the stock closing price at day i is lower than it at day (i+1). In other word, class label is 1 if the stock closing price is going up the next trading day, and 0 otherwise.

2.2

In this section, I introduce new features that are likely to contain information about the stock-pricing trend.

The starter feature set I used is rather naïve

and will be improved between now and the project deadline.

Concretely, the feature set is :{open price, closing price, day high, day low, adjusted closing price, volume}

2.3

Here I split the data into three sets: training, cross-validation, and test set. The weight factor used is: training: 60%, cv: 20%, and test: 20%.

3. Training with Linear Regression classifier

In this step a handful of popular classifiers are implemented and their performance examined in multiple steps. Features are first normalized according to their mean and variance.

3.1 the "quick-n-dirty" first attempt

The logistic regression with regularization classifier is used as the first quick-n-dirty classifier using only the "naïve" feature set in hope to obtain insight about the underlying problem.

The cost function J here is the standard cost function used in logistic regression and has the following form:

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m}\left[-y^{(i)}\log(h_\theta(x^{(i)})) - (1-y^{(i)})\log(1-h_\theta(x^{(i)}))\right] + \frac{\lambda}{2m}\sum_{j=1}^{n}\theta_j^2$$

I use the matlab function fmincg.m given from CS229A programming exercise to search the theta vector that minimizes the above cost function.

A learning curve is generated with 10 different sized training examples drawn from the training set pool to help decide the next step of project. The matlab output as well as the learning curve plot are listed below:

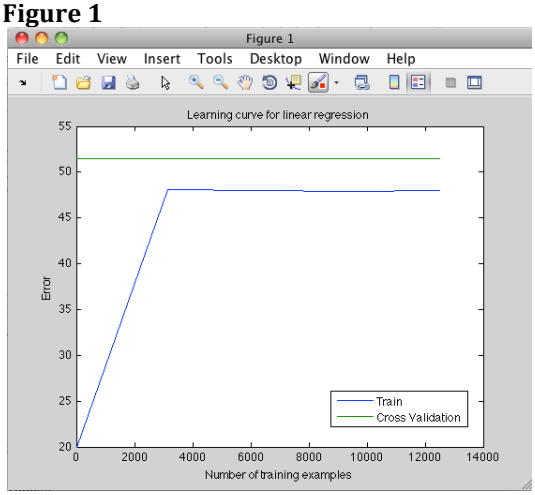
```
# Training Examples     Train Error Cross Validation Error
            10          20.000000   51.499880
            3133        48.132780   51.499880
            6256        47.985934   51.499880
            9379        47.862245   51.499880
            12502       47.984323   51.499880
```

It is clear that more samples are not beneficial with respect to the stock pricing prediction. Especially from the cv set error, it suggests that the parameter, theta, learned from training data is worth no more than a random guess and therefore has no value in reality.

As a result, I plan to significantly improve the feature set so more information is captured in the feature vector. This leads to the next step of this project: Introduce a set of stock technical indices into the feature set, hopefully including all the tech indices that can be derived from the known stock pricing data.

Secondly, it is still undetermined whether the learning algorithm, i.e. logistic regression with regularization, or the objective function of the maximization problem introduces more classification error. In order to answer this question, I will, after improving the naïve feature set, implement a different classifier, eg. SVM, and compare the classification error/accuracy with both classifiers to decide the next step of this project.

Figure 1



Training Curve of quick-n-dirty classifier

4. Improving Feature Set

4.1 Rich feature set

The basic feature set contains only the price and volume information. Now, I add 1) the derivative of both price and volume and 2) the moving average to the feature set. Concretely, I use finite difference equation to approximate the first and second order derivatives and the different statistic moving average formulas, such as simple, weighted, and exponential moving average.

Now, train the logistic regression classifier (LR in this report) w/o regularization. The matlab output as well as the learning curve plot are listed below:

```
Logistic Regression Classifier
# Training Examples     Train Error Cross Validation Error
            10          0.000000    22.222222
            3133        6.415576    6.791457
            6256        14.402174   15.118790
            9379        15.353449   16.318695
            12502       18.589026   19.246460
```

Note that the performance is dramatically improved with this updated feature set, suggesting that it captures the time dependent behavior of the stock price.

Diagnostic:

From figure 2, it is clear that as number of training example increases, the LR classifier still underfits the problem and results in increasing train/cv set error. In other word, the model has a high bias as number of training sample increases and therefore we need more features to better describe the intrinsic stock pricing information.

4.2 Adding more features

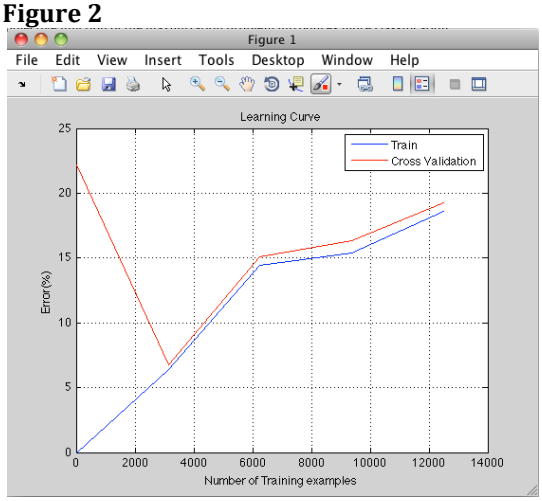
- Adding 3rd order derivatives to various price features

```
Logistic Regression Classifier
# Training Examples     Train Error Cross Validation Error
            10          0.000000    35.013199
            3133        13.437600   12.503000
            6256        16.687980   15.790737
            9379        10.598145   10.343173
            12502       12.765957   12.119030
```

From figure 3, we observe the improvements of adding higher order derivative feature with respect to time because the train and test error converges at around 12-13% when training example increases, comparing to ~20% obtained by 4.1 feature set.

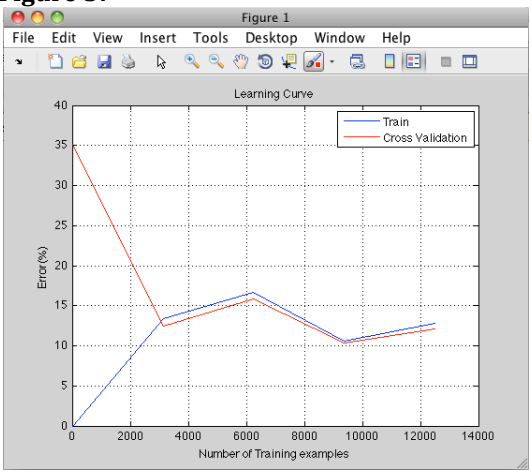
However, when taking a closer look at the value of cost function at each iteration, I found out the cost with this training set the underlying minimizer (fminunc) often can't improve the cost after a short amount of iterations (usually < 10 iterations). This observation leads to the next round of the model improvement, concretely, the optimization objective improvement.

Figure 2



Training Curve of LR classifier with rich feature set

Figure 3:



Training Curve of LR classifier with (improved) rich feature set

5. Improving Optimization Objective

Figure 4 shows the cost at each iteration when using the feature set obtained in 4.2 and fminunc as the function minimizer.

It is obvious that iterations after the third are not improving much on finding the minimum of cost.
I then tried to use a different function minimizer, fmincg, but obtained a set of result doesn't not make sense.

```
Iteration     1 | Cost: 6.270883e-01
Iteration     2 | Cost: 5.626661e-01
Iteration     3 | Cost:  NaN
Iteration     4 | Cost:  NaN
Iteration     5 | Cost:  NaN
Iteration     6 | Cost:  NaN
Iteration     7 | Cost:  NaN
Iteration     8 | Cost:  NaN
Iteration     9 | Cost:  NaN
Iteration    10 | Cost:  NaN
```

Clearly something went wrong when updating the cost function after the first couple iterations. I then examined the matlab code to compute the logistic regression cost function. Below is the snippet of function lrCostFunction I have:

```
function [J, grad] =
lrCostFunction(theta, X, y, lambda)
m = length(y); % number of training
examples
hx = sigmoid(X*theta);
theta_nointercept = theta;
theta_nointercept(1) = 0;

% compute the regularized cost function
(use innerProduct to avoid 0*log(0)
J = ((lambda *
(theta_nointercept'*theta_nointercept) /
2) - (y' * log(hx) + (1-y)' * log(1-hx)))
/ m;
% compute the gradient of the regularized
cost function
grad = X'*(hx - y);
grad = (grad + lambda .*
theta_nointercept) ./ m;
end
```

Note that the underlined statement. I found that although it computes the cost leveraging the vectorized computation, it failed to calculation 0 * log(0) as we need to defined this term to be 0. Without special handling of above case, matlab generates a NaN object and it eventually propagates out to convolute the cost.

This below statement, combined with a helper function, successfully address the above NaN issue.

```
J = ((lambda *
(theta_nointercept'*theta_nointercept) /
2) - (innerProduct(y,log(hx)) +
innerProduct(1-y,log(1-hx)))) / m;

function value = innerProduct(x,y)
% calculate the vector inner product and
check for 0*log(0)
% x need to be a indicator vector ie x(i)
== 0 or 1 only
d1 = length(x); d2 = length(y);
if d1 ~= d2
    fprintf('Vector dimension does not
match.\n');
    error;
else
    idx = (x==1);
    value = sum(y(idx));
end
end
```

Note that function innerProduct makes the assumption that input vector x is a vector where its elements can only be 0 or 1. We then calculates the inner product of x and y by summing over y indexed by x. Implemented the above 0*log(0) handling, the classifier performance dramatically improves to 99% accuracy evaluated on random drawn test set (using either function minimizers).

```
Logistic Regression Classifier
# Training Examples     Train Error Cross Validation Error
          10          0.000000    16.774658
          3133        2.042771    2.663787
          6256        1.007033    1.487881
          9379        0.799659    1.055916
          12502       0.487922    0.527958
```

Note that the performance is again dramatically improved with this fix, suggesting that now the lrCostFunction correctly handles the case where y*log(hx) = 0 * log(0) case. i.e the prediction is identical to the class label.

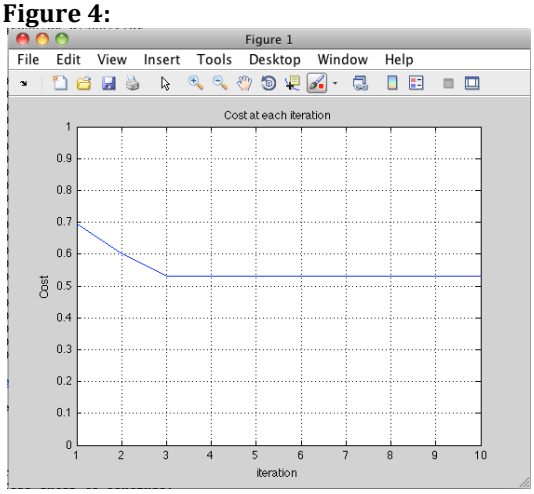
Diagnostic:

From figure 5, it is clear that as number of training example increases, the LR classifier successfully converges to a very high level of accuracy. From the learning curve, we can conclude that this model suffers from either high-bias or high-variance issue.

Since we have achieved the desired level of accuracy, it justifies that it is not needed to future investigate those "twitter related" features because if they do help on represent the intrinsic information that alters the stock trend, the benefit of researching on it is minimum. (at least when we try to predict the trend of Dow Jones Index)

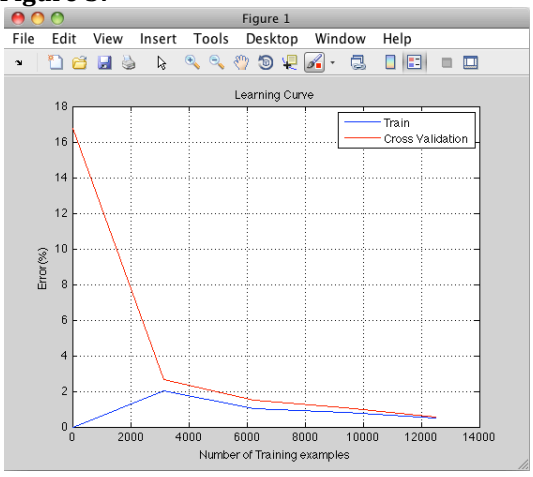
Also, implementing/using a SVM classifier at this stage is also removed from the to do list, same as LR with normalization because the current Logistic Regression Classifier works just fine.

Figure 4:



Optimization Objective (J) v.s. Number of iterations

Figure 5:



Training Curve of LR classifier with rich feature set after fixing NaN issue (Figure 5)

6. Summary & Future Work

A quick-n-dirty logistic regression classifier is first implemented to predict the stock close price(trend) of the next day and a multi-round error analysis and model diagnostic were performed at each phase. Finally, with a rich feature set and a correct implemented optimization algorithm the classifier is able to achieve the desire level of accuracy.

The work completed in this project, in my honest opinion, has successfully demonstrated my thorough understanding on designing and optimizing/diagnosing an sophisticated supervised machine learning model from end to end.

My current plan for future work steers toward to learning a policy, possibly using the reinforcement-learning paradigm, to maximize reward (capital gain) on investing solely on Dow Jones Index having now a model that predicts the trend of next trading day. Eventually I'd like to build an autonomous machine learning trader.