CPSC 406: In class activity 1 — Due March 19

Akshdeep Sandhu SN(20665148)

In homework 3, you used CVX to perform portfolio optimization. Ready-built solvers are great tools for prototyping, because it helps limit bugs to modeling errors. However, often there is a tradeoff, whether in scalability, or in generality. In this homework, you will program your own optimization methods on several machine learning problems

1. **Preprocessing data** In this homework you will implement several binary classification models in order to disambiguate 4 from 9. In order to do so, we need a matrix $A \in \mathbf{R}^{m \times n}$ where $m = ?$ is the number of training samples, and $n = 28 \times 27 = 728$ is the feature length. Additionally, we need a vector $b \in \mathbf{R}^m$ where $b_i = +1$ if the ith sample is a 4, and -1 if it is a 9. As in many data science projects, the first step is in processing the data in order to find these constants.

   (a) First, filter out all the datapoints in the test and train set corresponding to any sample that is not a 4 or 9. With the train data, this can be done using

   ```
   idx = trainY == 4 | trainY == 9;
   A = double(trainX(idx,:));
   b = double(trainY(idx));
   ```

   The extra `double` is for type casting. Often, MATLAB operations are not defined for types other than double, although for data storage using a more efficient type (int, boolean) may often be preferable.

   Do something similar to generate `Atest` and `btest`.

   Answer:

   ```
   idx = testY == 4 | testY == 9;
   Atest = double(testX(idx,:));
   btest = double(testY(idx));
   ```

   (b) Now, transform b so that instead of taking labels 4 and 9, it takes labels +1 and -1.

   Answer:

   ```
   b(b == 4) = [1];
   b(b== 9) = [-1];
   ```

   (c) **De-biasing and normalizing** Two tricks that often make models work a lot better is removing the bias and making the variance as close to 0 as possible. To remove the bias, perform

   ```
   Amean = mean(A,1);
   A = A - ones(m,1)*Amean;
   ```

   To remove the variance, perform

   ```
   Astd = std(A,1);
   A = A./ max(ones(m,1)*Astd,1);
   ```

   The extra normalization of the denominator is to avoid dividing by 0.

   Note that *order matters*; we cannot remove the variance before removing the mean. (Why?)

   Note also we actually divide by the standard deviation of X, which is the square root of the variance.

2. **Linear regression** We now have our data matrix A and label matrix b corresponding to the training set. Now, we want

$$x_{LS} = \underset{x}{\operatorname{argmin}} \|Ax - b\|_2^2$$

(a) Solve this using a direct method (e.g. `A \ b`). Evaluate the *loss* of the model

$$\text{loss}(x_{LS}) = \|Ax - b\|_2^2.$$

Answer:

```
xls = A\b';\\
loss = norm(A*xls - b');\\
loss =  46.2200
```

(b) Discuss how to now construct a *classifier*, e.g. some function $C_x(a)$ that takes some feature vector $a \in \mathbf{R}^m$ and returns $+1$ if the image is a 4, and -1 otherwise. Evaluate the *training misclassification rate*

$$\texttt{train misclass rate}(x_{LS}) = \frac{1}{m}\sum_{i=1}^{m} I(C_{x_{LS}}(A_i), b_i)$$

where $I(x, y) = 1$ if $x \neq y$ and $I(x, y) = 0$ if $x = y$.

Answer:

```
%xls is solution to ls problem

%b_hat is predicted class
b_hat = zeros(m,1);

for i=1:m
    %loop over rows and classifiy as 1 or -1
    if A(i,:)*xls > 0
        b_hat(i)= 1;
    else
        b_hat(i)= -1;
    end
end


%compute training missclassfication rate
train_miss_class_rate = miss_class(b',b_hat);
```

Training miss classification rate = 0.0308

The miss-classification function used is:

```
function miss_class_rate = miss_class(b,b_hat)
%function that returns the missclassification rate
%param b: vector of observed class
%param b_hat: vector of estimated class
%return miss_class_rate: missclassifcation error rate

miss_class_rate = 0;
m = size(b,1);

for j = 1:m
    if b(j) ~= b_hat(j)
        %if actual and estimated classes are not equal, increase missclassification
        %rate
        miss_class_rate = miss_class_rate + 1;
    end
end

miss_class_rate = miss_class_rate/m;

end
```

(c) Both the loss and training error are good ways of evaluating how well we did in both fitting the model and solving the optimization problem. However, it can only tell us about data we already observed. We don't know what will happen with data we haven't yet seen. To do this, we need to evaluate the *testing error*.

However, before using the testing data, we must first preprocess it in *exactly* the same way we preprocessed the training data. That means using the *same* vector for both the mean and the standard deviation, computed from the *trainig data alone*.

To preprocess the training data, run

```
Atest = Atest - ones(mtest,1)*Amean;
Atest = Atest./ max(ones(mtest,1)*Astd,1);
```

without recomputing `Amean` or `Astd`.

Discuss briefly why not recomputing these constants is so important.

Since we transformed the training data, in our pre-prepossessing step we must do the same to our test data. The training data was normalized the data and tried to reduce variance. We did this by getting estimates of the mean and variance from the training data. Therefore we assumed that some population mean must exist. This same transformation therefore must be applied to our test data as we are assuming that they come from the same population.

(d) Evaluate the *testing misclassification rate*

$$\texttt{test misclass rate}(x_{LS}) = \frac{1}{\hat{m}} \sum_{i=1}^{\hat{m}} I(C_{x_{LS}}(\hat{A}_i), \hat{b}_i)$$

Answer:
Using a general function that was coded to compute the predicted classes (in matlab file and appendix).
The testing error was: 0.036163

Linear regression models are a great "first pass" algorithm to classify data, because they are so easy to implement, and can be used to double check if the data preprocessing was done correctly. However, there are many reasons why they may not be the best classification tool. In the second half, we will investigate a slightly more sophisticated model, more suited for classification.

3. **Logistic regression**

(a) Conceptual questions

i. Explain why it is equivalent to maximize the log likelihood and likelihood.
The transforming the likelihood function by taking the log makes it easier for use to compute the max. This is fine since log transform is monotonic. Therefore the derivative of the function wont really change.

ii. Simplify the loss function, and derive the gradient and Hessian. (Hint: check homework 1.) Is the function convex, concave, or neither

This was also done in lecture notes:
LL = $f(x) = \Sigma_{k=1}^{m} - b_k log(\sigma(a_k^T x)) - (1 - b_k) log(1 - sigma(a_k^T x))$
$\nabla f(x) = A^T z - b; z_k = \sigma(a_k^T x)$
$\nabla^2 f(x) = A^T diag(z) diag(e - z) A$

(b) Coding questions

i. In logistic regression, we are dealing with 0,1 labels rather than -1,1 labels. To fix this, take the b vector and readjust: for example, `b = (b+1)/2.`

ii. Using an initial value $x^{(0)} = 0$, code up gradient descent to minimize this new loss function[1]. Run for 1000 iterations with fixed step size $1/m$ (m = total number of training samples.) Plot the model loss as a function of iteration, and report the final train and test misclassification rate.

```
%generate weights vectors
x_logreg = zeros(size(testX,2),1);

%fixed step size
alpha = 1/m;

sigmoid = @(x)(1./(1+exp(-x)));
cost = @(s)(( -b * log(sigmoid(s)+eps) - (1-b')' * log(1-sigmoid(s)+eps)));

%preform gradient desenct
for i=1:1000

    z = sigmoid(A*x_logreg);
    grad = A'*(z - b');
    x_logreg = x_logreg - alpha*grad;

    c(i) = cost(A*x_logreg);%( -b * log(sigmoid(A*x_logreg)+eps) - (1-b')' * log(1-sig
end
```

[1] If you worry that $\log(\sigma(a_i^T x)) = -\infty$ or $\log(1 - \sigma(a_i^T x)) = -\infty$ because the sigmoid function returns 0 or 1, you may replace $\sigma(a_i^T x)$ with $\max(\min(\sigma(a_i^T x), \epsilon), \epsilon)$ where ϵ is some small number.

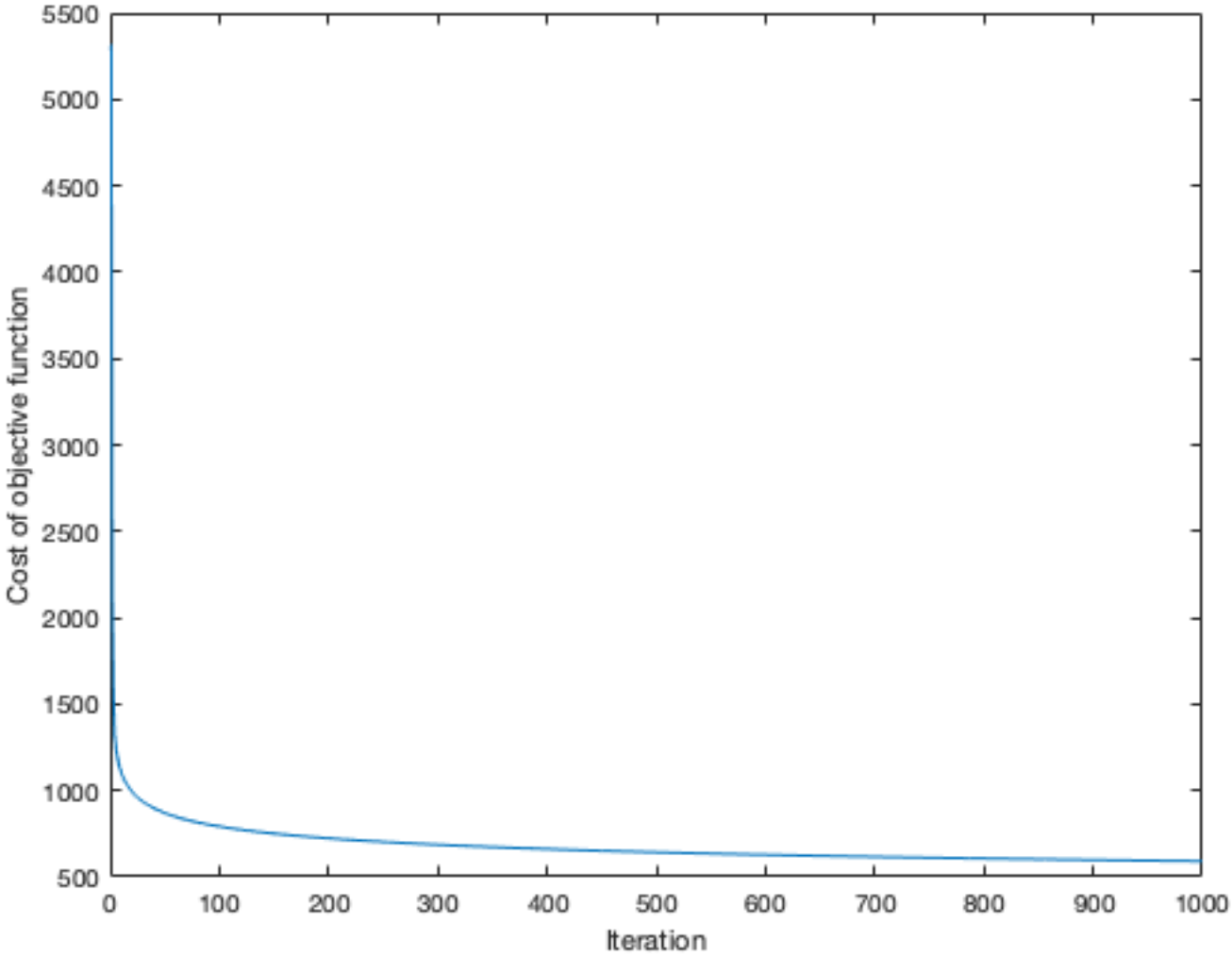

Figure 1: Gradient Descent iteration vs Cost

Logistic regression traing error: 0.01993
Logistic regression test error is: 0.028629

iii. Using an initial value $x^{(0)} = 0$, code up gradient descent to minimize this new loss function. Run for 1000 iterations with a backtracking line search with $s = 1$, $\alpha = \beta = 0.5$. Plot the model loss as a function of iteration, and report the final train and test misclassification rate.

```
%% Gradient descent wtih backtracking search

x_logreg = zeros(size(testX,2),1);

%line search parameters
alpha = 0.5;
beta = 0.5;
s =1;

%cost and sigmoid function
sigmoid = @(x)(1./(1+exp(-x)));
cost = @(s)(( -b * log(sigmoid(s)+eps) - (1-b')' * log(1-sigmoid(s)+eps)));


%preform gradient desenct with backtracking line search
for i=1:1000

    z = sigmoid(A*x_logreg);
    grad = A'*(z - b'); %gradient (computing descent direction)

    %starting line search to select best step size
    fx = cost(A*x_logreg);

    while fx - cost(A*(x_logreg -alpha*grad)) <  -alpha*s*norm(grad)^2
        alpha = beta*alpha;
    end

    %computed best step size
    x_logreg = x_logreg - alpha*grad;

    %update cost for plotting
    c(i) = cost(A*x_logreg);
end
```

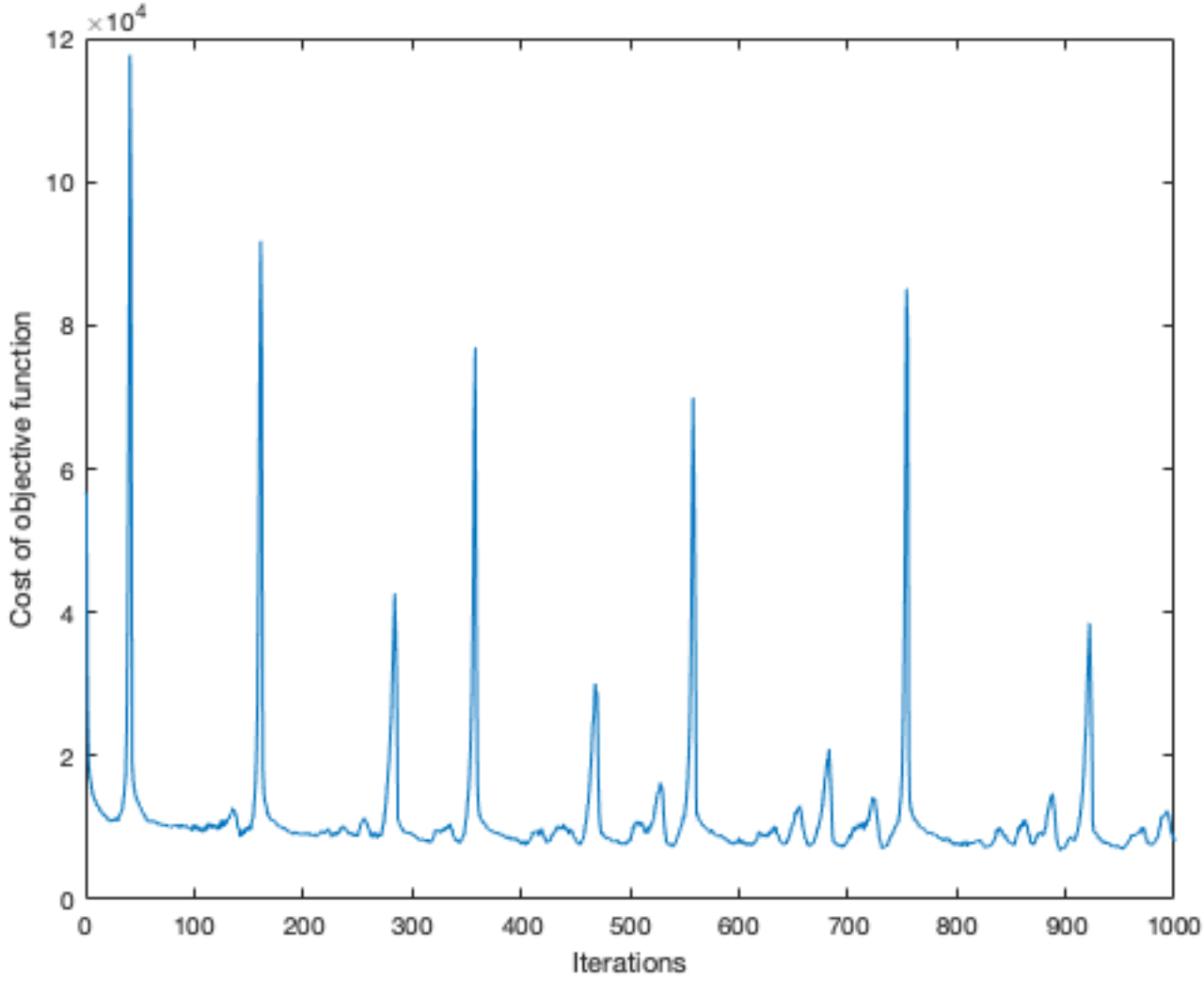

Figure 2: Caption

Logistic regression (with line search) traing error : 0.019167
Logistic regression(with line search) test error: 0.032145

iv. Compare in 1-2 sentences gradient descent with line search vs constant step size. Which is better?

Gradient descent with a fixed step step size had a much lower training and test error compared with line search. The chosen step size however was very small.

v. Compare in 1-2 sentences the linear and logistic model. Which is better? Is it better by much? Was the gain worth it?

Logistic regression seemed to be a much better fit to both the training (0.02 vs 0.03) and test data (0.036 vs 0.03). Therefore it was better by a few percentages, which may be a lot in practise. However, deriving the gradient and computing gradient descent was time consuming, when a simple regression worked fairly well.

4. **Debugging**

   Pick three images that fit the leasts quares model the worst, and plot them. Do the same with the logistic model.

(a) 1a

(b) 1b

(c) 1c

Figure 3: Plots of images that fit least sqaures model poorly

(a) 2a

(b) 2b

(c) 2c

Figure 4: Plots of images that fit logistic model poorly