**Logistic Regression (MATLAB)**

**Domingo Martinez**

**PROBLEM**

## Prediction

For a given weight w and a new feature vector x, predict the class applying multiclassification logistic regression. Write down formulas and pseudocode. Implement the solution in either MATLAB or python. Using matrix operation without loop.

## Training

Learning the weights by minimizing the cross-entropy loss function.

1. Consider both the original loss and the one with a L2 regularization (of course this involves a weight of the regularizer).
2. Derive the gradient formula.
3. Implement functions to evaluate the gradient and the loss using matrix operation.
4. Implement the following methods for training:

Steepest descent (using gradient with a step size)

* Hint: start with an initial step size (parameter to tune), increase the step size by a factor of 1.01 each iteration where the loss goes down, and decrease it by a factor 0.5 if the loss went up. If you are smart you may also undo the last update in that case to make sure the loss decreases every iteration.

2nd order optimization (Quasi-Newton method): use optimization package:

* in MATLAB, you can use fminunc from the optimization toolbox.
* In Python, you can try fmin\_l\_bfgs\_b, fmin\_bfgs, or fmin\_ncg from scipy.optimize. fmin\_l\_bfgs\_b is for constrained optimization. Set the bounds to None since we do not have constraints on our weights.
* Set the initial values of all weights to zero. Also try random (and reasonable small) initial weights.
* See attached optimize\_main.m for a template to start with. This is in matlab. Use it as an inspiration if you prefer python. Try to understand the parameters in opt and try out different combinations.

1. To make sure the gradient is calculated correctly, validate by comparing with finite-difference approximation. See the following link for a detailed explanation.

<http://ufldl.stanford.edu/tutorial/supervised/DebuggingGradientChecking/>

In Python, use scipy.optimize.check\_grad. In MATLAB, use checkgrad.m

<https://www.mathworks.com/matlabcentral/fileexchange/49437-insidde-thz-toolbox-v0-2/content/INSIDDE_thz_toolbox_v0.2/learning/autoencoder/checkgrad.m>

Assuming a correct evaluation of the loss function, if your gradient is implemented correctly, the output should be very small. Of course this function is very expensive, thus you should try it on a small scale data (say try it on Iris dataset, which only has 4 features).

1. Optimize parameters by cross-validation. Explain details of the cross-validation.

## Evaluation Metrics

Evaluate by comparing different baselines, different parameter settings. The point is to see both the accuracy and the efficiency.

1. Accuracy: number of correctly classified test data over the whole dataset.
2. Convergence rate: plot the loss function as a function of the number of iterations. You should see a converging curve. Try this for both steepest descent and quasi-newton (with an optimized parameters).
3. Measure the average time for each iteration, and for prediction.

## Data

Evaluate the methods on three different datasets.

1. Iris dataset.
2. Handwritten digit dataset:

<http://scikit-learn.org/stable/auto_examples/classification/plot_digits_classification.html>

* Want the full experience: try MNIST <http://yann.lecun.com/exdb/mnist/>

1. Cifar-10 dataset (color images):

<https://www.kaggle.com/c/cifar-10>

1. Learn to use PCA to reduce the dimensions of the data before feeding to the classifier. Of course, the number of reduced dimensions is a parameter to tune. For cifar-10, start with 100, 200, and 300. Also compare the accuracy/time difference when using or not using PCA.

<http://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html>

<https://www.mathworks.com/help/stats/pca.html>

**SOLUTION**

## Prediction

For binary logistic regression, we have two class where our . A vector of data features **X** and one vector **W** of weights to parametrize.

|  |  |
| --- | --- |
|  |  |

Using the approach for logistic regression we have:

Decision boundary

“Negative”

hyperplane

“Positive”

hyperplane

Since we only need values from cero to one to measure the probability of being in one of the class, we introduce the logistic or sigmoid function, and plugging in the former formula we have.

|  |
| --- |
|  |

On the other hand, the probability of y being cero is given by:

|  |
| --- |
|  |

Given a training data , and where is the total number of samples in the training data and is the original feature dimension, the goal is find an optimal value for the vector .

To learn the parameter vector we will use the likelihood estimation given by:

|  |  |  |
| --- | --- | --- |
|  |  | |
|  |  |  |

Where each sample is given by:

|  |  |  |
| --- | --- | --- |
|  |  | |
|  |  |  |

Taking the gradient of with respect to , we have:

|  |  |  |
| --- | --- | --- |
|  |  | |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

Considering all training samples, we have the formula for gradient decent:

|  |  |
| --- | --- |
|  |  |

Where the update rule or cross-entropy loss for this gradient decent method whit learn rate , would be:

|  |  |
| --- | --- |
|  |  |

For multi-classification, we have a training set for this case we assume:

|  |
| --- |
|  |

We can write down the likelihood function using 1-of-K condign scheme, when the probability of y being one class is independent of the others. We can create a target vector where, if will have one for the -th element and zero for the other elements. Therefore, the likelihood function for the ith sample is given by:

|  |  |  |
| --- | --- | --- |
|  |  | |
|  |  |  |

Taking the gradian with respect we have:

|  |  |
| --- | --- |
|  |  |

Where the update rule or cross-entropy loss for the multi-class gradient descent method whit learn rate , would be:

|  |  |
| --- | --- |
|  |  |

Now we can write down the modifications of the formulas to add regularization and control the overfitting on training our model. For the likelihood cost function, we will use:

|  |  |
| --- | --- |
|  |  |

Where, is the number of features and is the bias term.

Correspondingly, the partial derivatives for regularized logistic regression likelihood function is defined as:

|  |  |  |
| --- | --- | --- |
|  |  | For |
|  |  | For |

Now we are ready to implement this two function in MATLAB. The function for logistic regression is implemented as follow:

function g = logistic(z)

g = 1.0./(1.0 + exp(-z));

end

To implement the two last functions for the cost and the gradient with regularization parameter lambda we use the following code. Note that the gradient result is a matrix whit columns of gradient for each class, also since we are using **X** where each row is a sample data and W is a column vector. We don’t have to perform the transpose in the call function logistic(**X**\***w**).

function [J, grad] = regCostFunction(w, X, y, lambda)

% Compute the cost of using

% w as the parameter to learn for regularized logistic regression and the

% gradient of the cost (partial derivate for the given weight),

% this function receives a lambda parameter of regularization when

% is cero that means without regularization.

% Initialize some useful values

n = length(y); % number of training examples

% variables to return cost and gradient matrix

J = 0;

grad = zeros(size(w));

% Add ones to the X data matrix

X = [ones(n, 1) X];

% compute the cost of a given weight

h = logistic(X\*w);

J = (1/n)\*sum(-y.\*log(h)-(1-y).\*log(1-h))+ lambda/(2\*n)\*sum(w(2:end).^2);

%compute the partial derivate for the given weight

p = length(w)-1;

grad(1) = (1/n)\*sum(h-y.\*X(:,1));

grad(2:end) = (1/n)\*sum(repmat(h-y,[1,p]).\*X(:,2:end))'+(lambda/n)\*w(2:end);

end

## Training

Having implemented our model and getting gradient and cost function, the next step is the training it. To do that we use the following implementation. For each class, we perform the target vector to training each model, identifying each one of the given class, and keeping track of the cost function in each iteration. Then we plot it to know what is the optimal value for the number of iterations, and to evaluate the adequate value of alpha step size to use.

function [all\_w, J\_history] = steepestGradientDescent(X, y, alpha, num\_iters, num\_labels, lambda)

% Function that implement gradient descent steep by steep

% History of the cost function in each iteration

J\_history = zeros(num\_iters, num\_labels);

% Some useful variables

n = size(X, 2);

% Variable of all optimal weight found for each class or label

all\_w = zeros(num\_labels, n + 1);

for c = 1:num\_labels

fprintf('\nTrainning k: %f', c);

w = all\_w(c, :)';

for iter = 1:num\_iters

[J, grad] = regCostFunction(w, X, (y == c), lambda);

w = (w - (alpha\*grad));

J\_history(iter, c) = J;

end

all\_w(c, :) = w';

end

end

The implementation of second order of optimization Quasi-Newton method was based on the MATLAB function named “fminuc” from the optimization toolbox. The initial values of all weights were set to zero as show the following code.

function [all\_w] = quasiNewton(X, y, num\_labels, lambda, num\_iters)

% implementation of quasiNewton 2 order optimization

% variables

n = size(X, 2);

% Variable to return

all\_w = zeros(num\_labels, n + 1);

J\_history = zeros(num\_iters, num\_labels);

%options = optimset('GradObj', 'on', 'MaxIter', num\_iters, 'Display','iter', 'PlotFcns', @optimplotfval);

options = optimset('GradObj', 'on', 'MaxIter', num\_iters, 'Display','iter');

for c = 1:num\_labels

fprintf('\nTrainning k: %f\n', c);

initial\_w = all\_w(c, :)';

[all\_w(c,:)] = fminunc (@(t)(regCostFunction(t, X, (y == c), lambda)), initial\_w,

options);

end

end

Also, there is an additional implementation for minimize a multivariable function using conjugate gradian algorithm to perform a better cost training method. The code for implement this function is the following.

function [all\_w, J\_history] = fmincgFunction(X, y, num\_labels, lambda, num\_iters)

% Trains multiple logistic regression classifiers and returns all

% the classifiers in a matrix all\_w, where the i-th row of all\_w

% corresponds to the classifier for label i,

% fmincg works similarly to fminunc, but is more efficient when we

% are dealing with large number of parameters.

% variables

n = size(X, 2);

% return variables

all\_w = zeros(num\_labels, n + 1);

J\_history = zeros(num\_iters, num\_labels);

options = optimset('GradObj', 'on', 'MaxIter', num\_iters);

for c = 1:num\_labels

fprintf('\nTrainning k: %f\n', c);

initial\_w = all\_w(c, :)';

[all\_w(c,:), J] = fmincg (@(t)(regCostFunction(t, X, (y == c), lambda)), initial\_w, options);

% To capture the cost history on each iteration if the optimization

% function fmincg don't can not optimze any more complete the history

% cost with the last value cost evaluated, else record the complete

% history for the iteration.

if (length(J)<num\_iters)

aux = zeros(num\_iters - length(J),1);

aux(:) = J(length(J));

J = [J;aux];

J\_history(:, c) = J;

else

J\_history(:, c) = J;

end

end

end

Once we have the optimal calculated weight, we can use it to evaluate what is the probability of each sample of being of each class, the probability whit the biggest participation will be the class assigned to classify the data. The code to implement this functionality is given by:

function p = predict (all\_w, X)

% Using all the parameters weights all\_w, evaluate and return the

% maximun prediction found.

m = size(X, 1);

num\_labels = size(all\_w, 1);

% return value of class of predictions for each sample row.

p = zeros(size(X, 1), 1);

% Add ones to the X data matrix

X = [ones(m, 1) X];

hw = X \* all\_w';

[temp, p] = max(hw, [], 2);

end

To make sure that the gradient is calculated correctly we validate it by comparing with finite difference approximation using the function in MATLAB *checkgra.m****,*** as we will see on the iris dataset experiment.

## Metrics Evaluation

There are different evaluation metrics applied in this project: Plots of the data to use, plots of the gradient descent and convergence rate in each iteration, decision boundaries of classifiers, plots of bias vs variance, time of training each method and a complete parametrizable setup part in each of the different dataset used as long with a detail log and the times required to run each experiment.

The cross-validation method used was hold one out, this means, generally, 70% of the data were used to train the model and 30% to test it, for this propose was left a parameter setup to configure the split amount between training and test data. Then was measured the accuracy with respect the number of correct predictions for training and test data.

**EXPERIMENTS**

## Iris Simple Dataset

For this experiment was used Iris dataset for three kinds of flowers (150 samples) but only two features (Sepal and petal length) with the intension of evaluate all the implementation and the correctness work of all the parts. The implementation for this dataset experiment is developed on the file *iris\_simple.m*, with parameters as following:

%% Setup of parameters

samples=150; % number of samples to use (max 150 for iris)

holdOut=1/3; % parameter to split training data and test

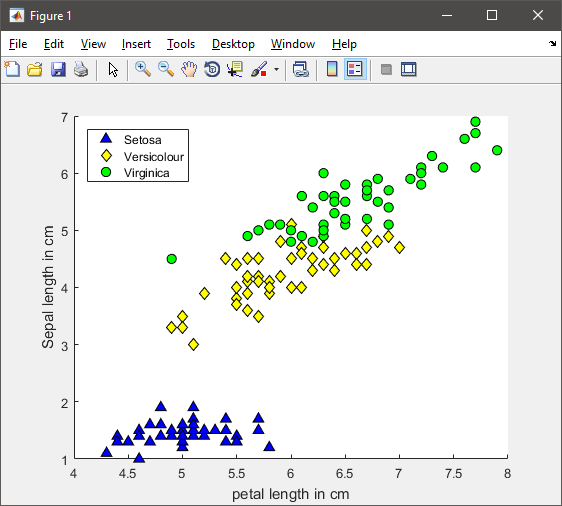
num\_labels = 3; % 3 labels for iris data set

lambda = 0; % Regularization parameter

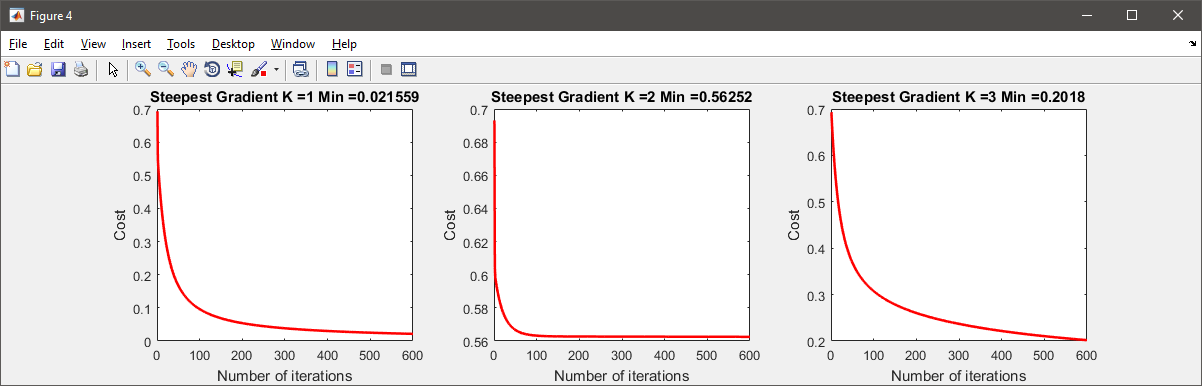
iterations = 600; % Number of iterations gradient descent

alpha = 0.1; % Steep size for gradient

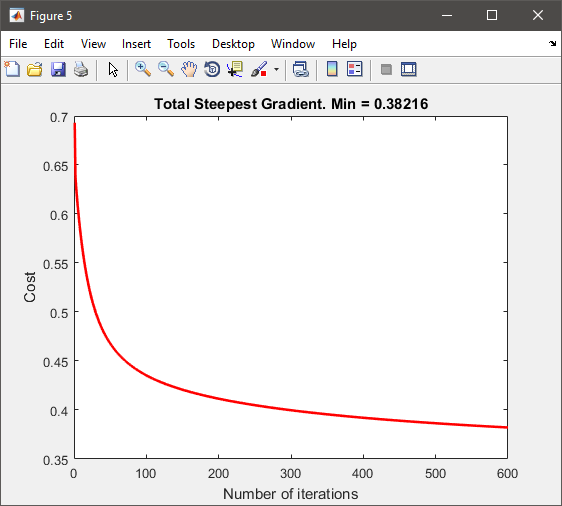
The first step was plot the data in order to visualize it and have a beter comprenhantion of the model to train. The model to use in all the experimients was a linear one, that means, the maximun polinomial degree is one.



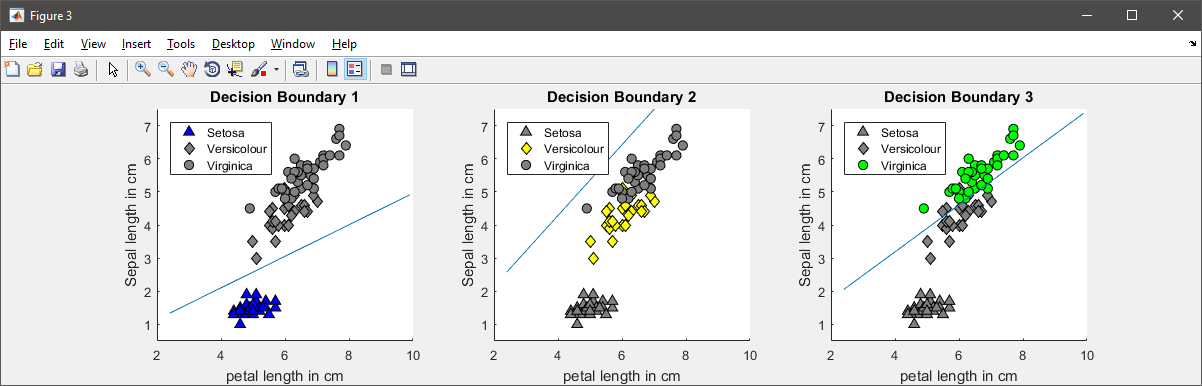
Then to choose the correct number of iteration was showed a plot for the gradient decent for each of the classes present in this experiment. The plot shows that, with 600 iterations the gradient converge, and do not reach a significant better result if we increase the number of iterations.



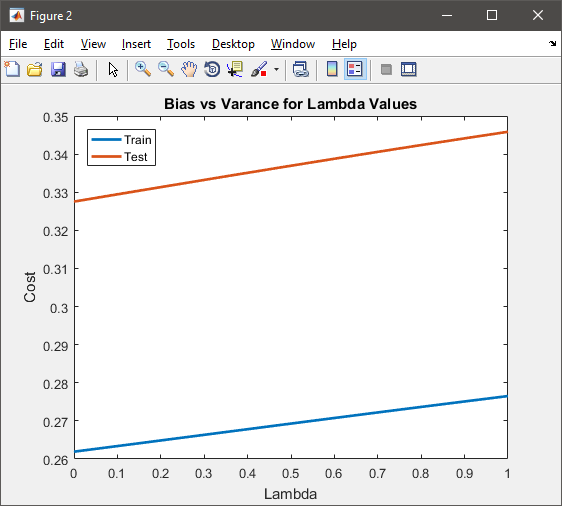
We can see a summary plot of the gradient decent for all three classes.



Following, we can see a plot of the decision boundaries for each of the models trained to have a better understanding of what is going on in the process of making decision on the data. The first plot corresponds to the trained model that decide what data is a Setoza flower (triangles in blue) based on the two-parameters given sepal and petal length. We can see that all the points below of the decision boundary in blue line are data with probability greater than 0.5. The other two kind of flowers are considered for this trained model as no Setoza and are showed in gray color. That data has a probability less of 0.5 of being a Setoza flower. In the same way, we can see the decision boundaries for the classifier models Versicolor and Virginica.



To evaluate the correct parameter lambda regularization to use, we calculate the cost function with different values of lambda, the following plot shows the result for lambda values from zero to one. This means that our model is too simple and the number of features is not enough to overfeed our model, choosing a lambda parameter of cero is the best option for this experiment.



## Iris Dataset

In this experiment was used full Iris dataset for three kinds of flowers (150 samples) and all the features. This experiment, introduce the use of second order of optimization quasi-newton method. The implementation for this dataset experiment is developed on the file *iris.m*, with parameters as following:

%% Setup of parameters

samples=150; % number of samples to use (max 150 for iris)

holdOut=1/3; % parameter to split training data and test

num\_labels = 3; % 3 labels for iris data set

lambda = 0; % Regularization parameter

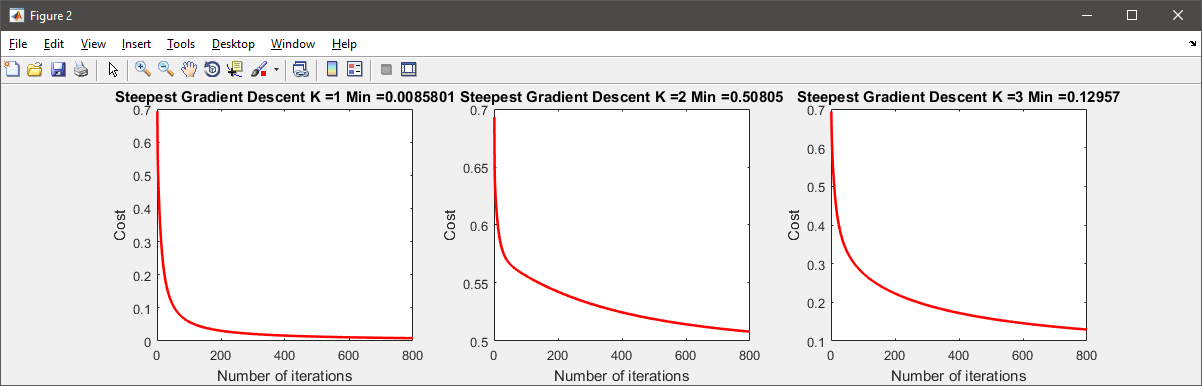
iterations = 800; % Number of iterations gradient descent

iterations\_QN = 50; % Number for Quasi-Newton

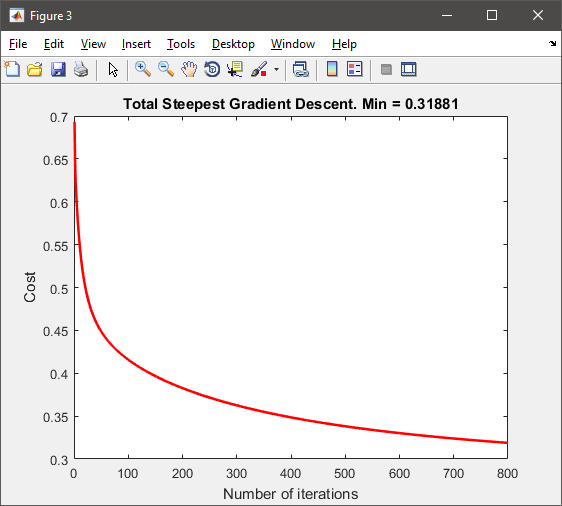
alpha = 0.1; % Steep size for gradient

The plot for the data on this experimient was not possible since the data have four features that means four dimmentios to plot. To plot that it's necessary a procedure to reduce the number of features witout loss much information like PCA which will being aplying it in the last experimient with Cifar dataset.

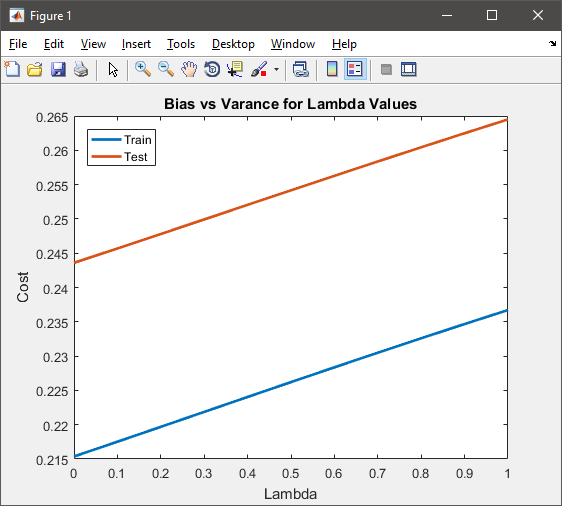
To choose the correct number of iteration, was showed a plot for the gradient decent for each of the classes present in this experiment. The plot shows that, for 800 iterations the gradient converge and do not reach a significant better result if we increase the number of iterations.



We can see a summary plot of the gradian decent for all three classes.



The plot for the decision boundary was not possible for the four dimensions where the data are. To evaluate the correct parameter lambda regularization to use, we calculated the cost function with different values of lambda, the following plot shows the result for lambda values from zero to one. This means that our model is too simple and the number of features is not enough to overfeed our model, choosing a lambda parameter of cero is the best option for this experiment.



For the number of iterations to use in the second order optimization Quasi-Newton method we evaluated the log output of the iterations of the function who has the biggest cost. For the last iteration, we can see that training the classifier for k = 3 (Virginica) reached 50 iteration with 0.038 so we choose 50 iterations as optimal value for Quasi-Newton method.

Training k: 3.000000

Norm of First-order

Iteration f(x) step optimality CG-iterations

0 0.693147 0.908

1 0.360914 1.14551 0.342 2

…

50 0.0383688 4.93745 0.000226 2

51 0.038339 0.344723 7.87e-05 2

Checking the gradient, we have a number very small that tell us that the gradient is implemented correctly.

ans =

3.9549e-10

To choose the value of the steep size for the gradient decent was tested different values and was check it with the plot of iterations vs cost, given a good result for values of alpha of 0.1

## Digit Dataset

In this experiment was used the digit dataset for 10 distinct types of handwritten digits, 5000 samples and all 400 features images in gray scale of 20x20 pixels. In this experiment, we introduce the use of second order of optimization with conjugate gradian algorithm method to deal with biggest number of features. The implementation for this dataset was developed on the file *digit.m*, with parameters as following:

%% Setup of parameters

samples=5000; % number of samples to use (max 5000 for digits)

holdOut=1/3; % parameter to split training data and test

num\_labels = 10; % 10 different labels for digits data set

lambda = 2; % Regularization parameter

iterations = 300; % Number of iterations steepest gradient descent

iterations\_QN = 25; % Number for Quasi-Newton

alpha = 0.1; % Steep size for gradient

%% Optional parameters (y/n)

test\_QNewton\_opt='y'; % Test Quasi-Newton (conjugate gradient algorithm)(faster)

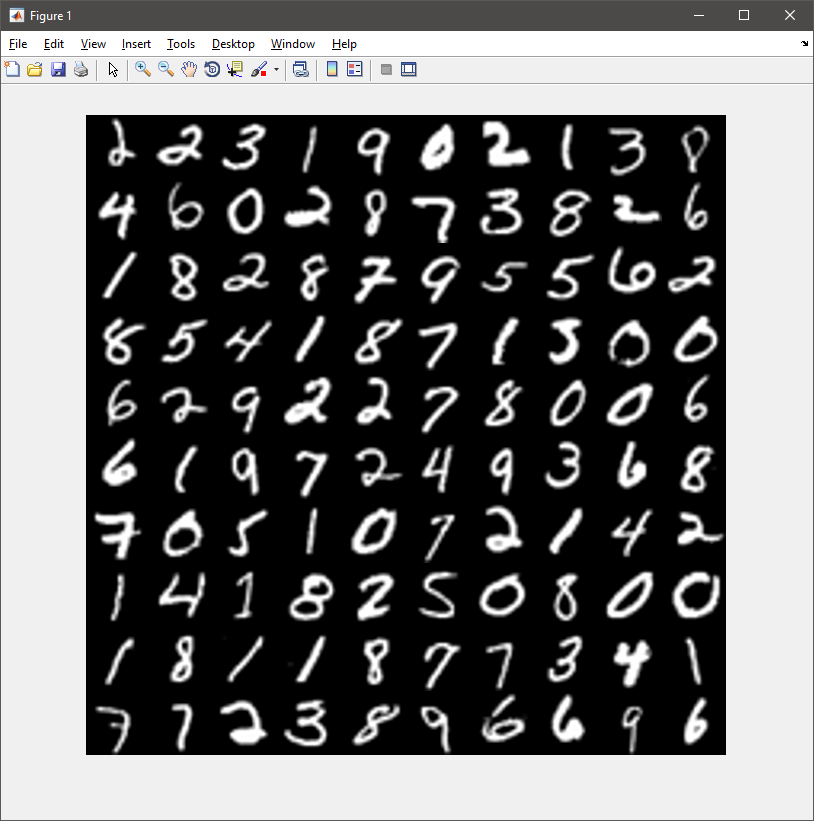
test\_steepst\_descent='n'; % Test gradian steepest descent (time consuming)

test\_QNewton\_descent='n'; % Test Quasi-Newton descent (time consuming)

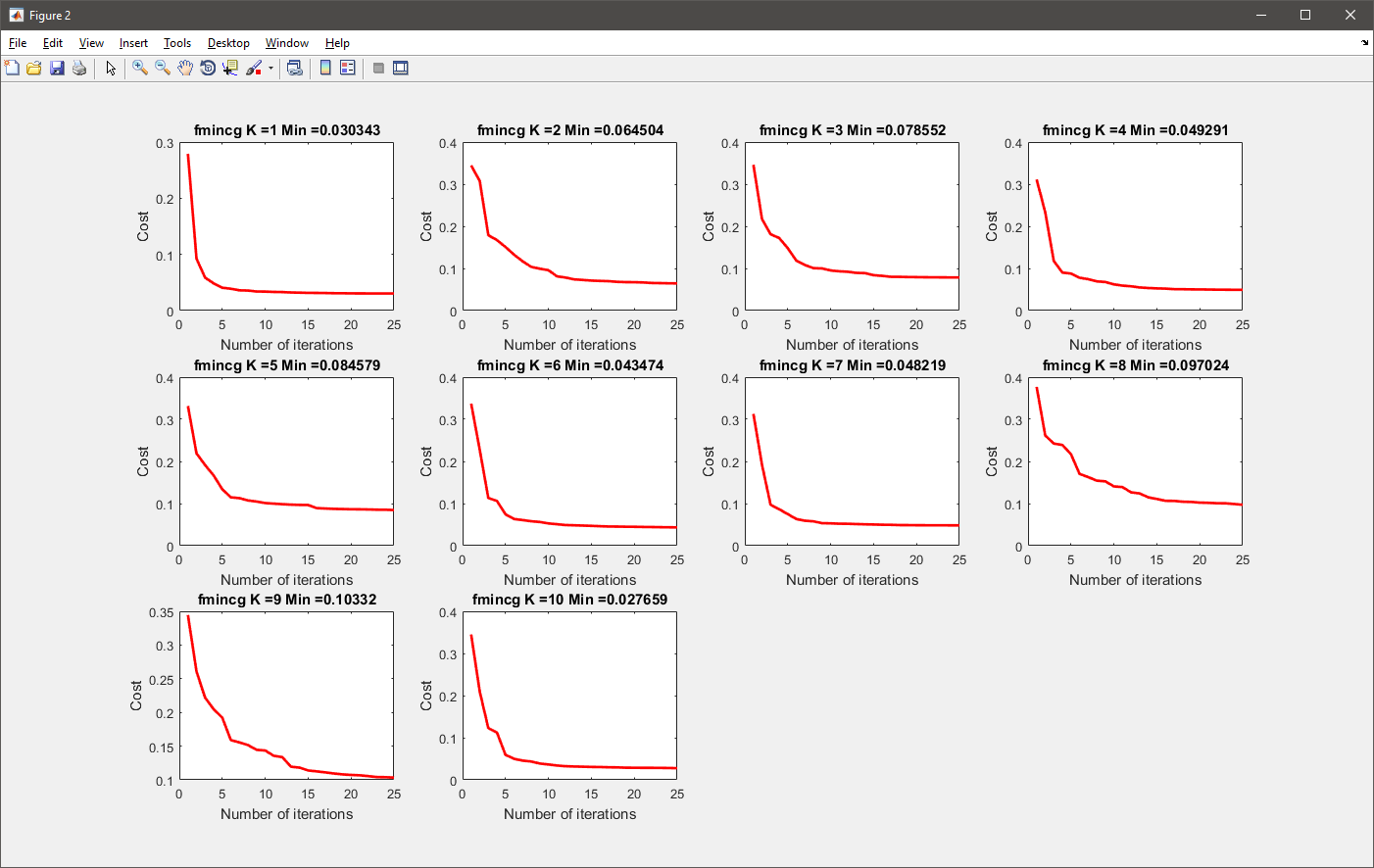
test\_optimal\_lambda='n'; % Test different lambda parameters (time consuming)

The second part, optional parameters, allows testing different optimization methods with the same data as well test the optimal value of regularization.

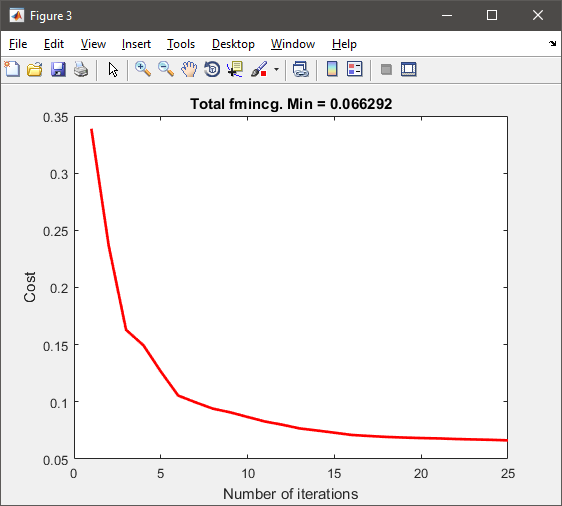
As well as the other experiments, the first step was shows the kind of imagen to work, this is a random representation of 100 different images for the handwritten digits dataset.



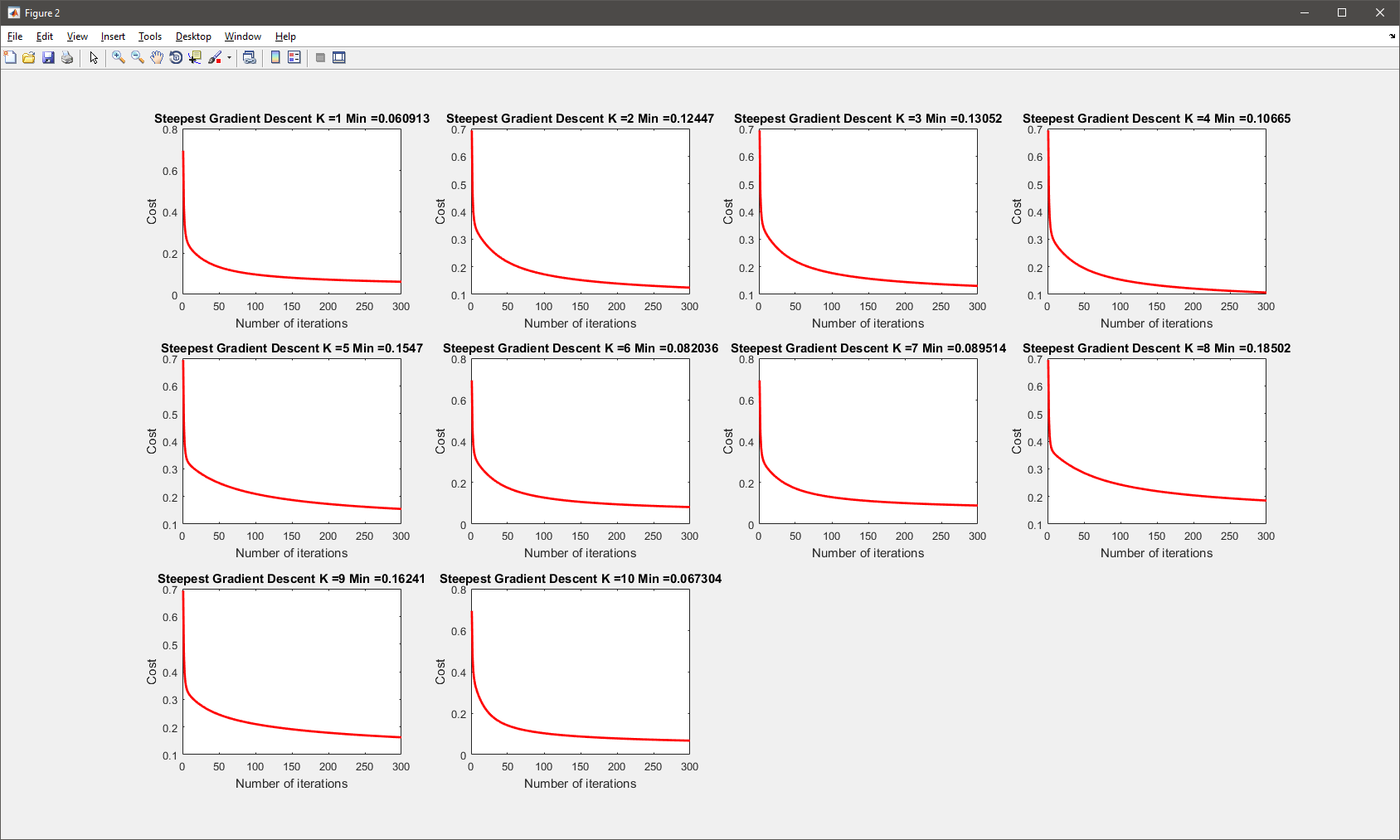
To choose the correct number of iteration was showed a plot for the gradient decent for each of the 10 classes present in this experiment. The plot shows, for 25 iterations using the second order of optimization with conjugate gradian algorithm method, the gradient converge and do not reach a significant better result if we increase the number of iterations.



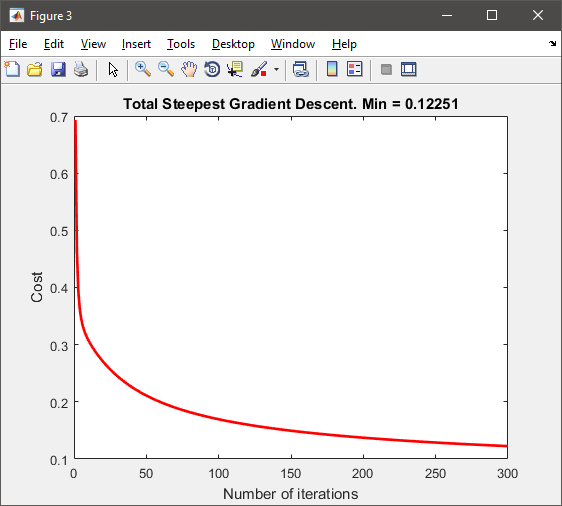
We can see a summary plot of the gradian decent for all 10 classes.



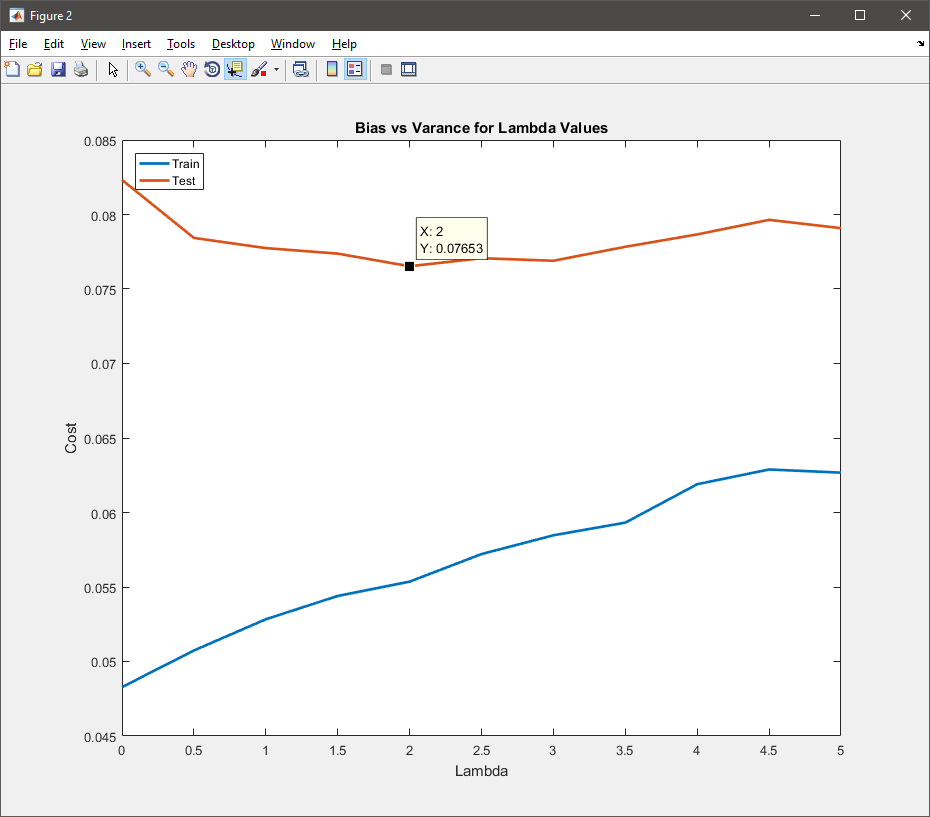
On the other hand, the training using steepest descent take a major number of 300 iterations as shows the following figure:



We can see a summary plot of the gradian decent for all 10 classes. This shows that steeps gradient decent is less efficient since takes a major number of iterations and the minimum value reached is bigger than the value of 0.066 reached using the former method.



To evaluate the correct parameter lambda regularization to use, we calculated the cost function with different values of lambda, the following plot shows the result for lambda values from zero to five. This time we can see an optimal value for lambda equal 2 for our test cost error, because we are dealing with a bigger number of features that complement the simplicity of our model, is necessary to increase the regularization value to not overfit the model at the training phase.



The number of iterations to use in the second order optimization Quasi-Newton method will be the same 25 iterations. The result will show up at the results part of this document.

## Cifar-10 Dataset

For this experiment was used only 10,000 samples of the available data. The data have ten different classes than appears as following:

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **airplane** | https://www.cs.toronto.edu/~kriz/cifar-10-sample/airplane1.png | https://www.cs.toronto.edu/~kriz/cifar-10-sample/airplane2.png | https://www.cs.toronto.edu/~kriz/cifar-10-sample/airplane3.png | https://www.cs.toronto.edu/~kriz/cifar-10-sample/airplane4.png | https://www.cs.toronto.edu/~kriz/cifar-10-sample/airplane5.png | https://www.cs.toronto.edu/~kriz/cifar-10-sample/airplane6.png | https://www.cs.toronto.edu/~kriz/cifar-10-sample/airplane7.png | https://www.cs.toronto.edu/~kriz/cifar-10-sample/airplane8.png | https://www.cs.toronto.edu/~kriz/cifar-10-sample/airplane9.png | https://www.cs.toronto.edu/~kriz/cifar-10-sample/airplane10.png |
| **automobile** | https://www.cs.toronto.edu/~kriz/cifar-10-sample/automobile1.png | https://www.cs.toronto.edu/~kriz/cifar-10-sample/automobile2.png | https://www.cs.toronto.edu/~kriz/cifar-10-sample/automobile3.png | https://www.cs.toronto.edu/~kriz/cifar-10-sample/automobile4.png | https://www.cs.toronto.edu/~kriz/cifar-10-sample/automobile5.png | https://www.cs.toronto.edu/~kriz/cifar-10-sample/automobile6.png | https://www.cs.toronto.edu/~kriz/cifar-10-sample/automobile7.png | https://www.cs.toronto.edu/~kriz/cifar-10-sample/automobile8.png | https://www.cs.toronto.edu/~kriz/cifar-10-sample/automobile9.png | https://www.cs.toronto.edu/~kriz/cifar-10-sample/automobile10.png |
| **bird** | https://www.cs.toronto.edu/~kriz/cifar-10-sample/bird1.png | https://www.cs.toronto.edu/~kriz/cifar-10-sample/bird2.png | https://www.cs.toronto.edu/~kriz/cifar-10-sample/bird3.png | https://www.cs.toronto.edu/~kriz/cifar-10-sample/bird4.png | https://www.cs.toronto.edu/~kriz/cifar-10-sample/bird5.png | https://www.cs.toronto.edu/~kriz/cifar-10-sample/bird6.png | https://www.cs.toronto.edu/~kriz/cifar-10-sample/bird7.png | https://www.cs.toronto.edu/~kriz/cifar-10-sample/bird8.png | https://www.cs.toronto.edu/~kriz/cifar-10-sample/bird9.png | https://www.cs.toronto.edu/~kriz/cifar-10-sample/bird10.png |
| **cat** | https://www.cs.toronto.edu/~kriz/cifar-10-sample/cat1.png | https://www.cs.toronto.edu/~kriz/cifar-10-sample/cat2.png | https://www.cs.toronto.edu/~kriz/cifar-10-sample/cat3.png | https://www.cs.toronto.edu/~kriz/cifar-10-sample/cat4.png | https://www.cs.toronto.edu/~kriz/cifar-10-sample/cat5.png | https://www.cs.toronto.edu/~kriz/cifar-10-sample/cat6.png | https://www.cs.toronto.edu/~kriz/cifar-10-sample/cat7.png | https://www.cs.toronto.edu/~kriz/cifar-10-sample/cat8.png | https://www.cs.toronto.edu/~kriz/cifar-10-sample/cat9.png | https://www.cs.toronto.edu/~kriz/cifar-10-sample/cat10.png |
| **deer** | https://www.cs.toronto.edu/~kriz/cifar-10-sample/deer1.png | https://www.cs.toronto.edu/~kriz/cifar-10-sample/deer2.png | https://www.cs.toronto.edu/~kriz/cifar-10-sample/deer3.png | https://www.cs.toronto.edu/~kriz/cifar-10-sample/deer4.png | https://www.cs.toronto.edu/~kriz/cifar-10-sample/deer5.png | https://www.cs.toronto.edu/~kriz/cifar-10-sample/deer6.png | https://www.cs.toronto.edu/~kriz/cifar-10-sample/deer7.png | https://www.cs.toronto.edu/~kriz/cifar-10-sample/deer8.png | https://www.cs.toronto.edu/~kriz/cifar-10-sample/deer9.png | https://www.cs.toronto.edu/~kriz/cifar-10-sample/deer10.png |
| **dog** | https://www.cs.toronto.edu/~kriz/cifar-10-sample/dog1.png | https://www.cs.toronto.edu/~kriz/cifar-10-sample/dog2.png | https://www.cs.toronto.edu/~kriz/cifar-10-sample/dog3.png | https://www.cs.toronto.edu/~kriz/cifar-10-sample/dog4.png | https://www.cs.toronto.edu/~kriz/cifar-10-sample/dog5.png | https://www.cs.toronto.edu/~kriz/cifar-10-sample/dog6.png | https://www.cs.toronto.edu/~kriz/cifar-10-sample/dog7.png | https://www.cs.toronto.edu/~kriz/cifar-10-sample/dog8.png | https://www.cs.toronto.edu/~kriz/cifar-10-sample/dog9.png | https://www.cs.toronto.edu/~kriz/cifar-10-sample/dog10.png |
| **frog** | https://www.cs.toronto.edu/~kriz/cifar-10-sample/frog1.png | https://www.cs.toronto.edu/~kriz/cifar-10-sample/frog2.png | https://www.cs.toronto.edu/~kriz/cifar-10-sample/frog3.png | https://www.cs.toronto.edu/~kriz/cifar-10-sample/frog4.png | https://www.cs.toronto.edu/~kriz/cifar-10-sample/frog5.png | https://www.cs.toronto.edu/~kriz/cifar-10-sample/frog6.png | https://www.cs.toronto.edu/~kriz/cifar-10-sample/frog7.png | https://www.cs.toronto.edu/~kriz/cifar-10-sample/frog8.png | https://www.cs.toronto.edu/~kriz/cifar-10-sample/frog9.png | https://www.cs.toronto.edu/~kriz/cifar-10-sample/frog10.png |
| **horse** | https://www.cs.toronto.edu/~kriz/cifar-10-sample/horse1.png | https://www.cs.toronto.edu/~kriz/cifar-10-sample/horse2.png | https://www.cs.toronto.edu/~kriz/cifar-10-sample/horse3.png | https://www.cs.toronto.edu/~kriz/cifar-10-sample/horse4.png | https://www.cs.toronto.edu/~kriz/cifar-10-sample/horse5.png | https://www.cs.toronto.edu/~kriz/cifar-10-sample/horse6.png | https://www.cs.toronto.edu/~kriz/cifar-10-sample/horse7.png | https://www.cs.toronto.edu/~kriz/cifar-10-sample/horse8.png | https://www.cs.toronto.edu/~kriz/cifar-10-sample/horse9.png | https://www.cs.toronto.edu/~kriz/cifar-10-sample/horse10.png |
| **ship** | https://www.cs.toronto.edu/~kriz/cifar-10-sample/ship1.png | https://www.cs.toronto.edu/~kriz/cifar-10-sample/ship2.png | https://www.cs.toronto.edu/~kriz/cifar-10-sample/ship3.png | https://www.cs.toronto.edu/~kriz/cifar-10-sample/ship4.png | https://www.cs.toronto.edu/~kriz/cifar-10-sample/ship5.png | https://www.cs.toronto.edu/~kriz/cifar-10-sample/ship6.png | https://www.cs.toronto.edu/~kriz/cifar-10-sample/ship7.png | https://www.cs.toronto.edu/~kriz/cifar-10-sample/ship8.png | https://www.cs.toronto.edu/~kriz/cifar-10-sample/ship9.png | https://www.cs.toronto.edu/~kriz/cifar-10-sample/ship10.png |
| **truck** | https://www.cs.toronto.edu/~kriz/cifar-10-sample/truck1.png | https://www.cs.toronto.edu/~kriz/cifar-10-sample/truck2.png | https://www.cs.toronto.edu/~kriz/cifar-10-sample/truck3.png | https://www.cs.toronto.edu/~kriz/cifar-10-sample/truck4.png | https://www.cs.toronto.edu/~kriz/cifar-10-sample/truck5.png | https://www.cs.toronto.edu/~kriz/cifar-10-sample/truck6.png | https://www.cs.toronto.edu/~kriz/cifar-10-sample/truck7.png | https://www.cs.toronto.edu/~kriz/cifar-10-sample/truck8.png | https://www.cs.toronto.edu/~kriz/cifar-10-sample/truck9.png | https://www.cs.toronto.edu/~kriz/cifar-10-sample/truck10.png |

Each image has three different RBG channels and a resolution of 32x32 pixel, that means for this experiment the total number of features is 3,072. It’s a clever idea not convert the images to gray scale, since there is valuable information to use for the classifier, for instance we can see that all deer animals are majority brown. We can choose to use only 10,000 samples or try for the complete 60,000 samples in the data set, only is needed uncomment a few lines in code that are well described. The implementation for this dataset experiment is developed on the file *cifar.m*, with parameter as following:

%% Setup of parameters

samples=10000; % number of samples to use

num\_labels = 10; % 10 labels, from 1 to 10

K = 95; % Number of principal components to use

lambda = 160; % Regularization parameter

iterations = 1000; % Number of iterations gradient descent

iterations\_QN = 125; % Number for Quasi-Newton

alpha = 0.012; % Steep size for gradient

%% Optional parameters (y/n)

test\_QNewton\_opt='y'; % Test Quasi-Newton (conjugate gradient algorithm)(faster)

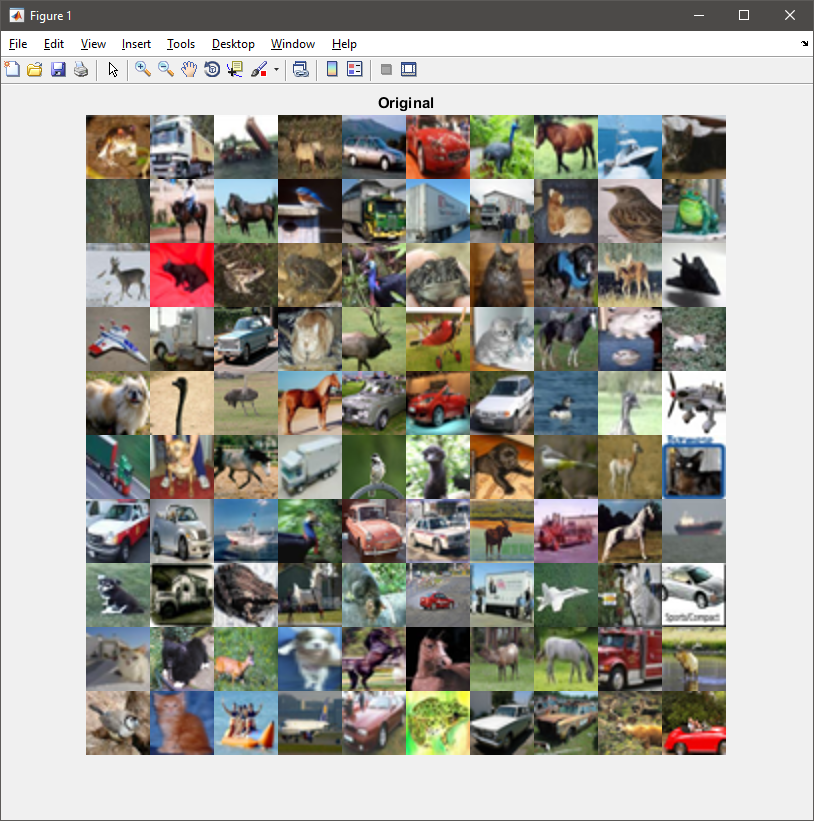
test\_steepst\_descent='n'; % Test gradian steepest descent (time consuming)

test\_QNewton\_descent='n'; % Test Quasi-Newton descent (time consuming)

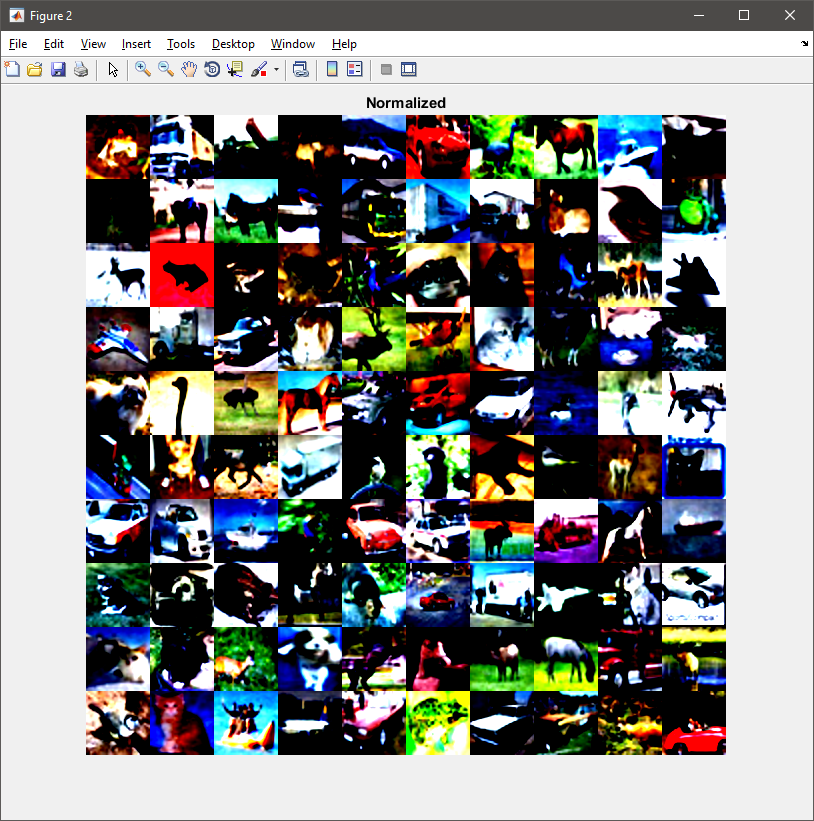
test\_optimal\_lambda='n'; % Test different lambda parameters (time consuming)

The second part optional parameter, allows testing different optimization methods with the same data as well test the optimal value of regularization.

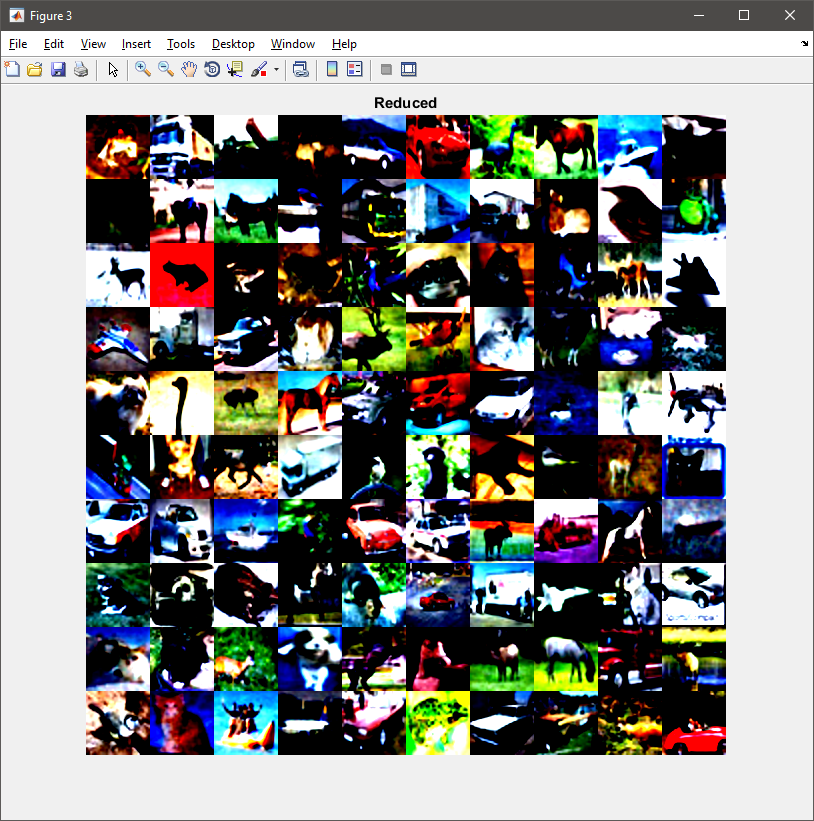
As well as the other experiments, the first step was shows the kind of image to work, these are the 100 first images representation for the Cifar-10 dataset.



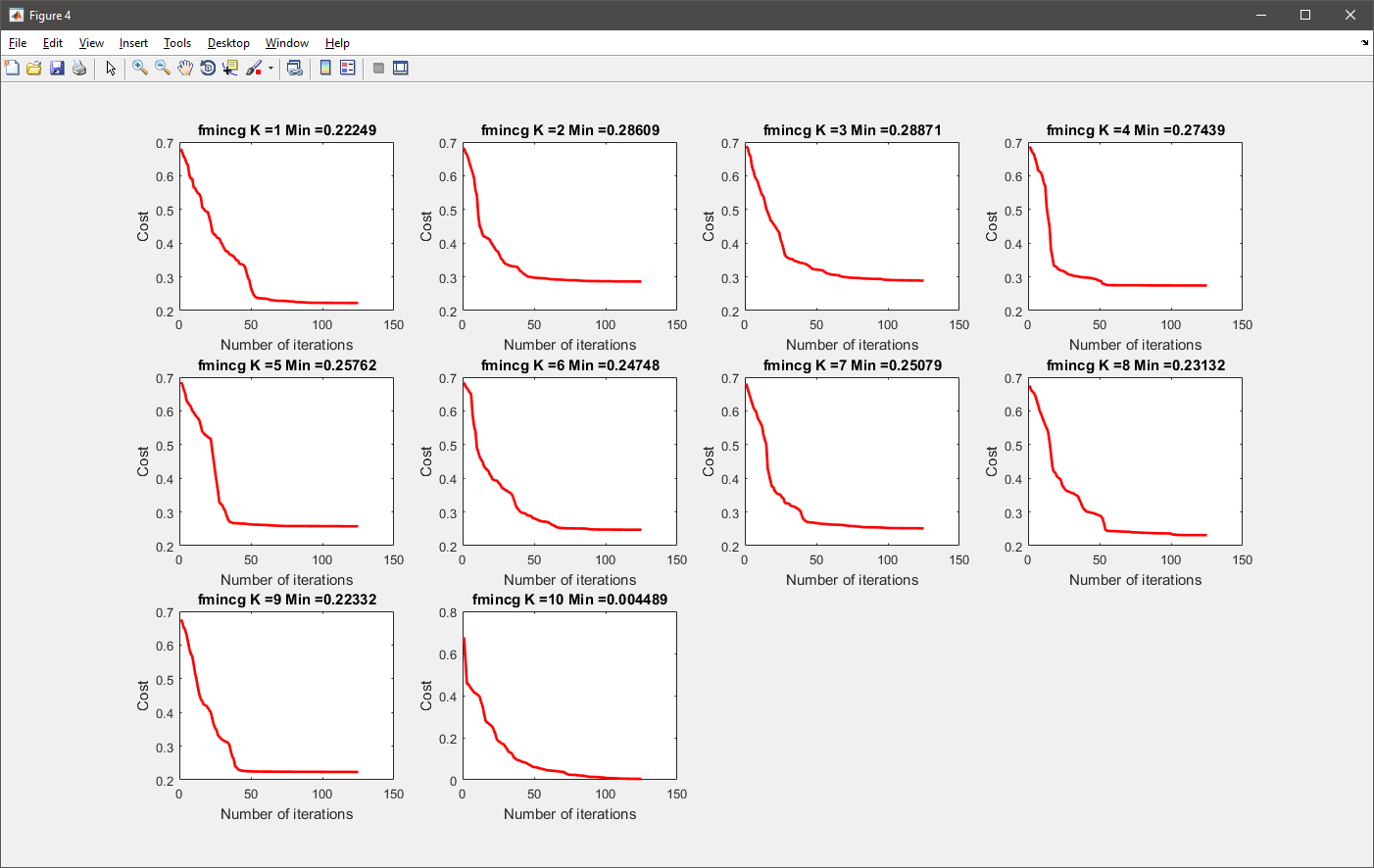
Then we need normalize each imagen as pre-processing step to optimize the converging of the gradient decent. Using the function feature normalize for each of the channels of each imagen. The result is showing in the next imagen.



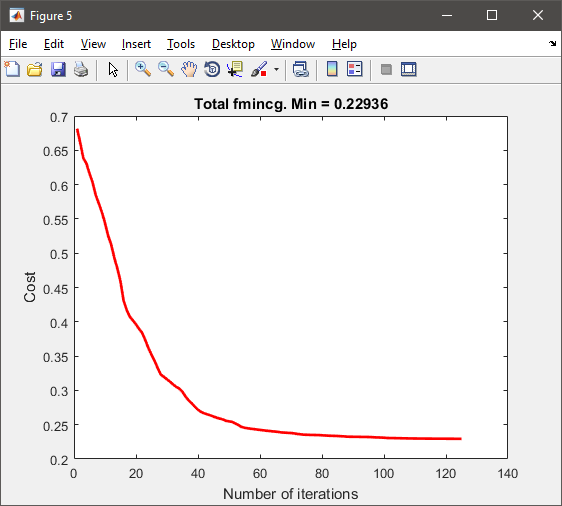
Although we have the images whit intensity pixel values normalized, still there are information that is redundant and to speed up the training of our model we need apply PCA. For this exercise, the number of features was reduced to be 95 primary components for each channel RBG of the images. The variance retained for this value is 99.88 that means, we are losing only 0.12 of the information. The result can be seen on the next image that is nearly identical to the former one but with only a total of 285 features this is inclusive less features than the handwriting dataset.



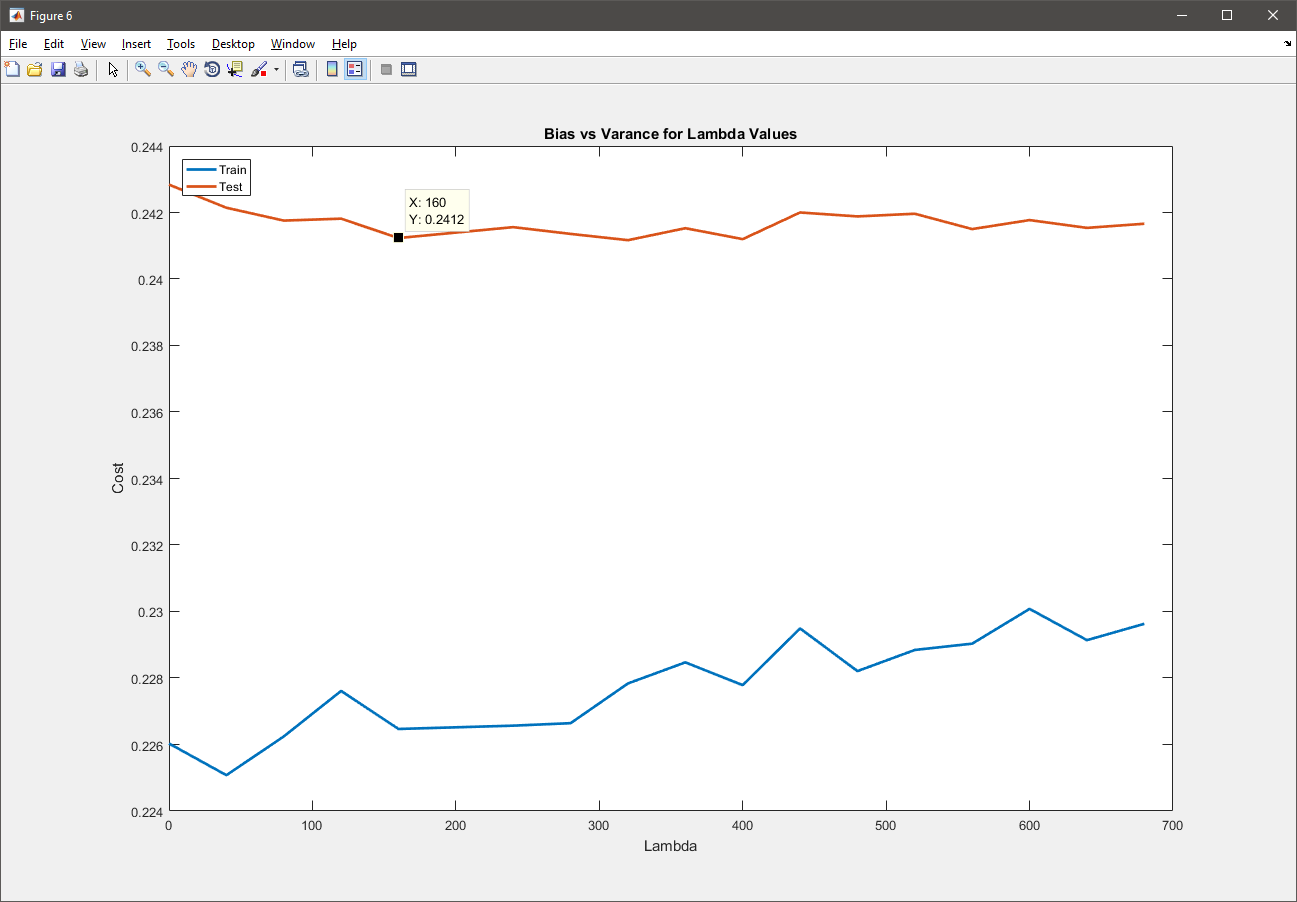
To choose the correct number of iterations. Was showed a plot of the gradient decent for each of the 10 classes present in this experiment. The plot shows that for 125 iterations using the second order of optimization with conjugate gradian algorithm method, the gradient converge and do not reach a significant better result if we increase the number of iterations.



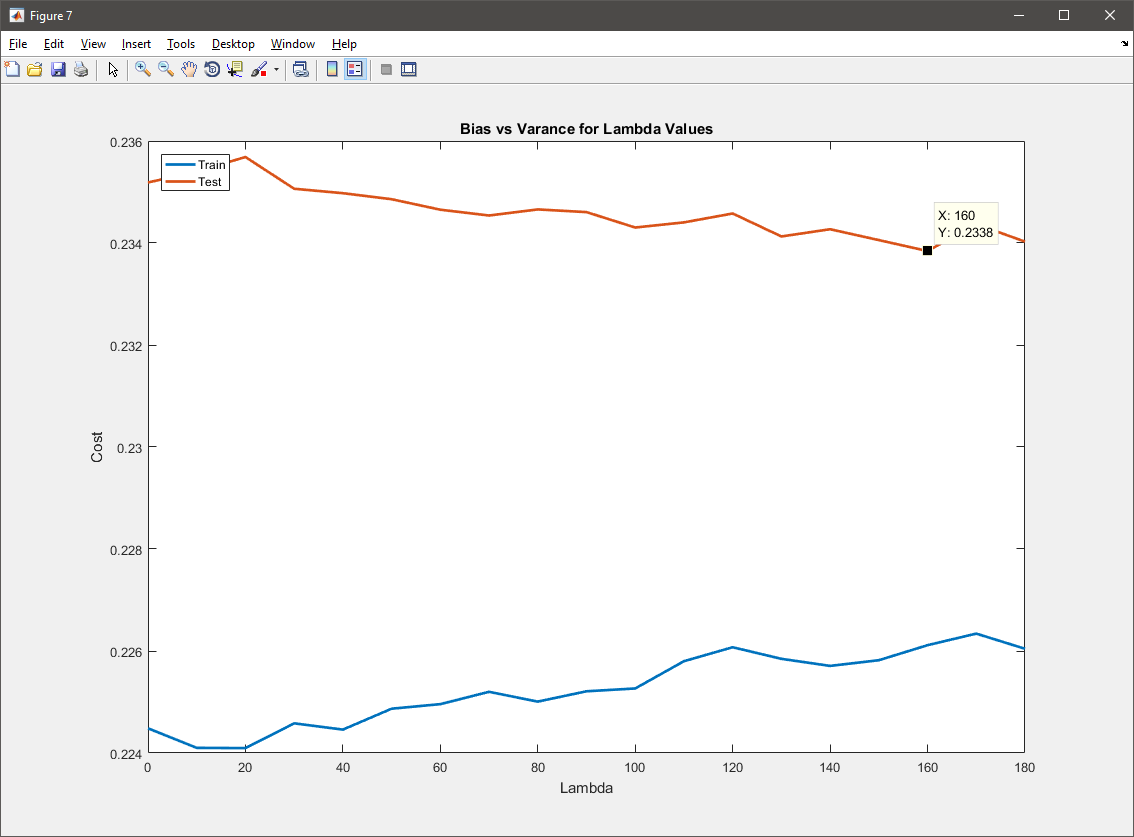
We can see a resume plot of the gradian decent for all 10 classes.



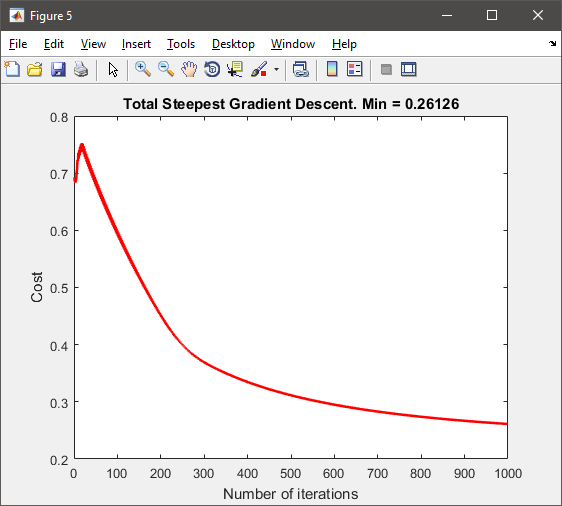
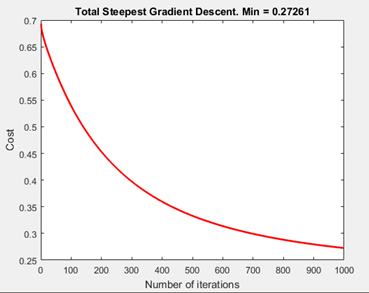
To evaluate the correctness parameter lambda regularization to use, we calculated the cost function with different values of lambda, the following plot shows the result for lambda values from zero to 700. This time, we can see an optimal value for lambda equal 160 in our test cost error. Because we are dealing with an even bigger number of features that complement the simplicity of our model, is necessary to increase the regularization to a higher value to not overfit the model at the training phase.



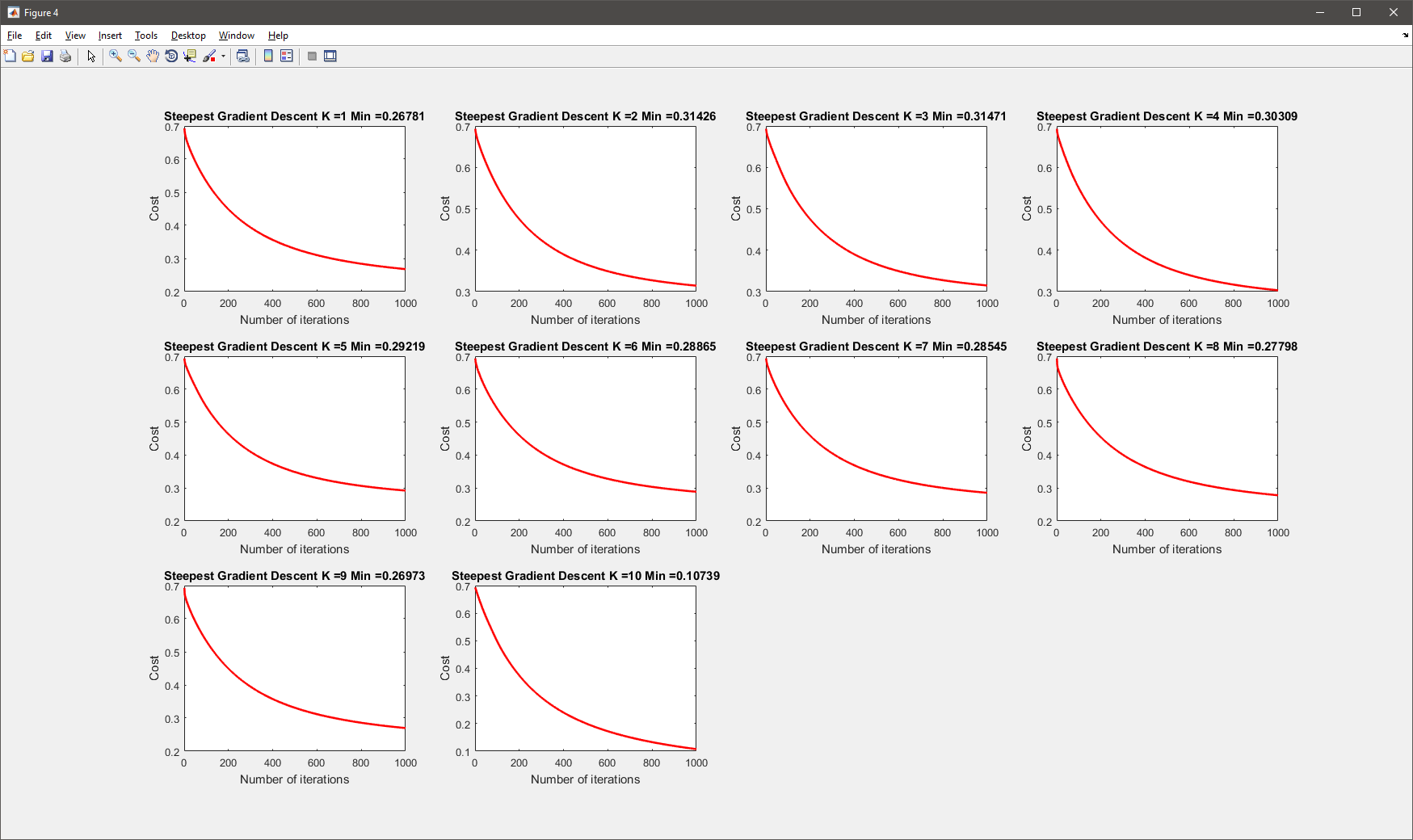
To try to improve even more, and make sure of the correctness value for the parameter lambda, was plotted a second graph but this time with lambda from cero to 180. Certifying the correctness minimum value on the test error cost.



The value for the steepsize alpha parameter of gradient decent in this occasion shows that is too high 0.012 and was necessary to set lower in 0.01 to get a better result in the steepest descent as show in the second figure.

Here a detailed plot of the cost value vs iterations for each classifier.



**RESULTS**

## Iris Simple Dataset

For the simple iris dataset, were reached over 90% on the test accuracy, with a time for training steepest gradient descent of 0.14 seconds and a prediction time of 524 microseconds.

Part 4: Training Logistic Regression Steepest Gradient Descent...

Training k: 1.000000

Training k: 2.000000

Training k: 3.000000 (Done 0.146801)

Checking gradian... ans = 4.5249e-11

Part 5: Predict values cross-validation... (Done 0.000524)

Training Set Accuracy: 92.000000

Test Set Accuracy: 96.000000

From the plot of decision boundaries and bias vs variance, we can conclude that our lineal model is too simple. For this data and the given two features if we want to make a good classification of Versicolor flowers, one improve could be, make greater the polynomial degree of the model to better fit on the data. Since the available training data is small, this accuracy can be improving even more if we train our model using more than one-fold for our cross-validation method.

## Iris Dataset

The iris dataset with all the features was tested with two methods, the first one, steepest gradient descent reached a test accuracy of 94%, whit a training time of 0.23 seconds that correspond of an increase the size of features from two to four. The prediction time was 1.4 milliseconds.

Part 4: Training Logistic Regression steepest Gradient Descent...

Training k: 1.000000

Training k: 2.000000

Training k: 3.000000 (0.232104)

Checking gradian... ans =7.1736e-11

Part 5: Predict values cross-validation steepestGradientDescent...(Done 0.001459)

Training Set Accuracy: 98.000000

Test Set Accuracy: 94.000000

The second method used was, Quasi Newton gradient decent. Test accuracy for this training reach 98%, whit a training time of 0.5 seconds, this probably for the option to show the information of all the iterations. The prediction time was 2.9 milliseconds.

Part 6: Training Logistic Regression Quasi-Newton...

Trainning k: 1.000000

Norm of First-order

Iteration f(x) step optimality CG-iterations

0 0.693147 1.34

…

36 0.0461305 0.0746291 2.57e-05 2

Local minimum possible. (Done 0.598724)

Checking gradian...ans =5.7154e-10

Part 7: Predict values cross-validation Quasi-Newton...(Done 0.002940)

Training Set Accuracy: 96.000000

Test Set Accuracy: 98.000000

From the plot of decision boundaries and bias vs variance, we can conclude that our lineal model is too simple even with the two additional features of our data if we want to make a good classification one aspect to improve could be make greater the polynomial degree of the model to better fit on the data. And, since the available training data is small, this accuracy can be improving even more if we train our model using more than one-fold for our cross-validation method.

## Digit Dataset

The first experiment training logistic regression with conjugate gradient algorithm on the handwriting digit dataset got us an accuracy of 90.33% on our test data, the training time was 17 seconds and the prediction time was 29 milliseconds.

Part 4: Trainning Logistic Regression Quasi-Newton (conjugate gradient algorithm)...

Trainning k: 1.000000

Iteration 1 | Cost: 2.793639e-01

…

Iteration 25 | Cost: 2.517202e-02 (Done 17.337676)

Predict values Quasi-Newton (conjugate gradient algorithm)... (Done 0.029389)

Training Set Accuracy: 93.491302

Test Set Accuracy: 90.336134

Second experiment was using steepest gradient descent, and the test accuracy obtained was 86.49%, with a training time of 94.45 seconds and prediction time of 29 milliseconds.

Part 5: Training Logistic Regression steepestGradientDescent...

Training k: 1.000000

…

Training k: 10.000000 (Done 94.455374)

Predict values cross-validation steepestGradientDescent...(Done 0.029469)

Training Set Accuracy: 87.252549

Test Set Accuracy: 86.494598

The last experiment consisted in training our model with second order of optimization Quasi-Newton method, and the test accuracy was 90.75%, with a training time of 22 minutes and prediction time of 55 milliseconds.

Part 6: Training Logistic Regression Quasi-Newton...

Trainning k: 1.000000

Norm of First-order

Iteration f(x) step optimality CG-iterations

0 0.693147 0.395

…

9 0.0282596 0.00892505 8.04e-07 14

(Done 1345.674158)

Predict values cross-validation Quasi-Newton...(Done 0.055128)

Training Set Accuracy: 93.641272

Test Set Accuracy: 90.756303

From this result we can conclude that, the method with conjugate gradient algorithm performs very well dealing with a considerable number of features. The training did with steepest gradient descent algorithm obtained less accuracy but in a reasonable time, this could be improved with a dynamic change of the steep size parameter alpha, for example: when the difference between the actual and former cost in an iteration is large, make the steep size bigger and small otherwise until converge, this could improve the time and accuracy for this method.

For the training with Quasi-Newton method we can conclude has an excellent accuracy on our test data but with a huge time necessary to optimize the values for our weigh vector W.

## Cifar-10 Dataset

For the Cifar-10 dataset was performed four different experiments, on the three firsts was used 10,000 samples of data and the last one was performed using all the available data for the dataset, to measure how much accuracy can be reach.

The first experiment, was training logistic regression with conjugate gradient algorithm got us an accuracy of 38.40% on our test data, the training time was 126.36 seconds and the prediction time was 42 milliseconds.

Loading and Visualizing Data ...(Done 9.966755)

Calculing variance...(Done 0.012042)

Variance ratained for K= 95.000000 is: 99.885175 percent

Appliying PCA to all the data...(Done 9.934233)

Splitting the data...(Done 0.036361)

Number of features to use: 285.000000

Number of training: 9000.000000, and test samples: 1000.000000

Part 4: Trainning Logistic Regression Quasi-Newton (conjugate gradient algorithm)...

Trainning k: 1.000000

…

Iteration 124 | Cost: 2.034913e-04

Iteration 125 | Cost: 2.009761e-04 (Done 126.315769)

Predict values Quasi-Newton (conjugate gradient algorithm)...(Done 0.042285)

Training Set Accuracy: 42.988889

Test Set Accuracy: 38.400000

Second experiment was using steepest gradient descent, and the test accuracy obtained was 38.40%, with a training time of 591.80 seconds (10 minutes approx.) and prediction time of 46 milliseconds.

Loading and Visualizing Data ...(Done 9.207913)

Calculing variance...(Done 0.013326)

Variance ratained for K= 95.000000 is: 99.885175 percent

Appliying PCA to all the data...(Done 9.705132)

Splitting the data...(Done 0.034437)

Number of features to use: 285.000000

Number of training: 9000.000000, and test samples: 1000.000000

Part 4: Trainning Logistic Regression Quasi-Newton (conjugate gradient algorithm)...(Desactived)

Part 5: Trainning Logistic Regression steepestGradientDescent...

Training k: 1.000000

…

Training k: 10.000000 (Done 591.809123)

Predict values cross-validation steepestGradientDescent...(Done 0.046211)

Training Set Accuracy: 41.911111

Test Set Accuracy: 38.400000

The third experiment consisted in training our model with second order of optimization Quasi-Newton method, and the resulting test accuracy was 37.00%, with a training time of 24 minutes and prediction time of 42 milliseconds.

Loading and Visualizing Data ...(Done 8.815589)

Calculing variance...(Done 0.003040)

Variance ratained for K= 95.000000 is: 99.885175 percent

Appliying PCA to all the data...(Done 9.591372)

Splitting the data...(Done 0.033986)

Number of features to use: 285.000000

Number of training: 9000.000000, and test samples: 1000.000000

Part 4: Trainning Logistic Regression Quasi-Newton (conjugate gradient algorithm)...(Desactived)

Part 5: Trainning Logistic Regression steepestGradientDescent...(Desactived)

Part 6: Trainning Logistic Regression Quasi-Newton...

Trainning k: 1.000000

…

(Done 1440.169182)

Predict values cross-validation Quasi-Newton...(Done 0.042328)

Training Set Accuracy: 43.711111

Test Set Accuracy: 37.000000

The fourth and last experiment was using all the data available to train the model with logistic regression with conjugate gradient algorithm this give us an accuracy of 39.67% on our test data, the training time was 14 minutes and the prediction time was 22 milliseconds.

Loading and Visualizing Data ...(Done 13.597875)

Calculing variance...(Done 0.005165)

Variance ratained for K= 95.000000 is: 99.885175 percent

Appliying PCA to all the data...(Done 39.189962)

Splitting the data...(Done 1.063548)

Number of features to use: 285.000000

Number of training: 54000.000000, and test samples: 6000.000000

Part 4: Training Logistic Regression Quasi-Newton (conjugate gradient algorithm)...

Trainning k: 1.000000

Iteration 1 | Cost: 6.812722e-01

…

Iteration 125 | Cost: 1.388514e-04 (Done 853.327016)

Predict values Quasi-Newton (conjugate gradient algorithm)...(Done 0.225311)

Training Set Accuracy: 40.042593

Test Set Accuracy: 39.666667

As conclusion, we can see that the accuracy for the test data can reach 39.66%, which is acceptable compared with 10.00% that can be reach if we perform a randomly choose.