

Linear Regression

# Table of Contents

[Table of Contents 1](#_Toc462925955)

[Overview 1](#_Toc462925956)

[Using the project and testing your results 2](#_Toc462925957)

[Python package requirements 2](#_Toc462925958)

[Implementation 2](#_Toc462925959)

[Basic Linear Regression Implementation 3](#_Toc462925960)

[linearHeuristic - 3](#_Toc462925961)

[linearCost - 3](#_Toc462925962)

[Gradient Descent 4](#_Toc462925963)

[Linear Regression with Polynomial features support implementation 5](#_Toc462925964)

[Linear Regression with Regularization Implementation 5](#_Toc462925965)

[linearCost 6](#_Toc462925966)

[linearGradientDescent 6](#_Toc462925967)

[Tuning 6](#_Toc462925968)

[Extra Credit 7](#_Toc462925969)

[Grading Breakdown 8](#_Toc462925970)

[Submissions 8](#_Toc462925971)

# Overview

In this lab you will be implementing methods needed to perform the linear regression process. To implement linear regression, you will need to define methods for a heuristic function, a cost function and gradient descent. The lab assignment will require that you have support for basic linear regression, as well as supporting multiple and/or polynomial features and the regularization process.

You will also be required to tune learning rates, regularization rates and gradient descent iterations so as to find the best results possible

# Using the project and testing your results

The lab project was created using Visual Studio and uses Python, C and C++ languages. You are not required to use Visual Studio for development, but other options may not be supported by the lab specialists.

The project contains Course Director created solutions and should work before you add your solutions. You will replace the instructor’s solutions with your own.

There is a visualization and a basic test that can be run for basic linear regression, polynomial support and regularization. To run the visualizations, in the LRPython folder you should find “linearBasic.bat”, “LinearPolynomail.bat” and “LinearPolyReg.bat” files. Running these should open a console window with test output as well as a web page with graphs of the training and test data as well as the current heuristic results. These visualization methods do not call gradient descent until one of the buttons at the top of the web page is pressed.

For basic testing of your methods, the LRPython project contains “LRBasicTest.py”, “LRPollyTest.py” and “LRPolyRegTest.py” files. These files can be run from visual studio or command prompt and will present results of the heuristic, cost and gradient descent methods as well as display expected results.

# Python package requirements

This lab project is dependent on you having the “pandas” and “bokeh” Python packages installed on your machine. It is believed you will have these packages from you Data Visualization course, as well as know how to use “PIP” to install any missing packages.

# Implementation

The support for basic linear regression, polynomial features and regularization can be created and tested iteratively. It is recommended that you implement and test these features one at a time, getting one to work before moving on to the next.

The Regression project is a C++ project containing one file called Regression.cpp. The Regression source file contains definitions for the three Linear Regression methods needed. These methods are each calling a solution function that you should remove before implementing your own solution. You are welcome to create additional methods and files as needed.

If you would rather create solutions to these methods using Python you may do so in the LRPython project file called “Regression.py”. If using Python, please make sure your solution does not take too long to run. We will be adding timing requirements to this lab in the future, but we need to see some students results before defining those requirements. If your results take a long time to process it will slow down your work and make the person grading your results grumpy. ☺

# Basic Linear Regression Implementation

You should start by creating solutions to the heuristic, cost and gradient descent methods that support the basic Linear Regression system, such that we attempt to map a straight line to our training data.

Your results for these features can be tested by running the “LinearBasic.bat” file to visualize your iterative results, or run the “LRBasicTest.py” for simple text based results.

## linearHeuristic -

For this implementation we will define the function of a line, based on input data contained in our features. The formula to implement is defined as:

All the data needed to implement this function is passed as arguments already. The inputWeights argument will be a size one array containing the value to be used for . The bias argument represents the value. The features argument is also a size one array containing the value to be used for .

## linearCost -

The linear cost formula to be used can be defined as:

The arguments to this method should include all the data needed to implement this formula, as well as comments to help you decide what the parameters represent. The portion of the formula can be implemented with a for loop that starts an iterator at 1 (i=1) and runs until the iterator equals m (numFeatures). In the body of the for loop keep a running sum of the results of

## Gradient Descent

The formula for our basic gradient descent is:

repeat until convergence:

{

update =

update =

= update

= update

}

Where:

/// This is for the “bias” weight

Instead of testing for convergence, you will run your gradient descent algorithm for a number of iterations defined by the argument “numIterations”.

# Linear Regression with Polynomial features support implementation

Adding support for polynomial features will also provide support for multiple input features. We will only be testing polynomial features, and not multiple features as multiple features are difficult to visualize beyond 2 features.

Your results for these features can be tested by running the “LinearPolynomial.bat” file to visualize your iterative results, or run the “LRPolyTest.py” for simple text based results.

Having already completed the Basic Linear Regression implementation, may mean that only your heuristic method needs to be adjusted. The new linearHeuristic method should be defined as:

Where equals 1, and equals . The through are already set correctly for you in the linearHeuristic argument “features”. The n in the formula is defined by argument “numInputs”.

You will also want to make sure that your linearCost and linearGradientDescent methods do not assume there is only one feature (). These methods both take in the argument “numInputs” which define the number of inputs and the number of features in each unique data point (*numInputs == length(features[0]*).

# Linear Regression with Regularization Implementation

We implement support for regularization to provide a means to reduce overfitting. To implement this support we will adjust our cost function, and gradient descent methods to favor smaller values for our input weights, which will reduce the variance of our heuristic results.

Your results for these features can be tested by running the “LinearPolyReg.bat” file to visualize your iterative results, or run the “LRPolyRegTest.py” for simple text based results.

## linearCost

Adjust your linearCost formula to match:

Only the “” part of the formula should be new. The is the penalty amount called lambda. The n is the number of input features in each data set, defined by argument “numInputs”.

## linearGradientDescent

Since we changed our cost function, we will have to adjust the derivative of it used in the gradient descent formula. Your new cost function derivative should be:

# Tuning

You will be required to tune your learning rate, regularization rate and number of iterations so as to get the best results possible, in the fewest iterations. You are only required to do this for the visualization of the final system using polynomial features and regularization.

In the LRPython project you should find the file “LinearRegressionPolyReg.py” which is the entry point when you run the “LinearPolyReg.bat” file. In LinearRegressionPolyReg.py you will find variables being assigned for the learning rate (alpha), regularization penalty (lambdaVal) and the number of times gradient descent should be run (numIterations). Your task is to find values for the three variables that cause gradient descent to converge in the fewest number of iterations, and have the lowest cost results for our **test** data.

The alpha value is the learning rate. Increasing this value may cause gradient descent to converge in fewer iterations. If too high of a value is used, gradient descent will never converge and your cost will iteratively increase. If Python complains about a “NAN” value in the command window this is likely a sign that your combined values for “alpha” and “lambdaVal” are too high.

The lambdaVal represents the regularization value. Increasing this value will reduce the variance of your heuristic. This will likely result in increasing your training data cost and reducing your test data cost. Set this to a value that gives the lowest test cost after gradient descent converges.

Once you have the alpha and lambdaVal values set, adjust the numIterations value to be close to just enough iterations to cause gradient descent to converge with the data used. The numIterations value defines the number of gradient descent iterations performed when the first button (top left) of the visualization is pressed.

As an alternative to manually adjusting the numIterations, you may adjust the functionality of the system so as to dynamically test if gradient descent has converged and stop iterating once it does. Make sure to print out the number of iterations needed to converge if you take this route. This route will require more Python adjustments.

# Extra Credit

Implementing a vector based solution for the three formulas will be worth up to 5% extra credit on this lab. To do this you will need a solution for multiplying and transposing arbitrarily sized matrices and vectors. You are encouraged to find and use a third party solution to provide needed vector and matrix math support, as opposed to creating this yourself.

# Grading Breakdown

Each component will be graded on an all-or-nothing scale, partial credit is not guaranteed. Each component must work before points can be earned on the next feature.

|  |  |
| --- | --- |
| 50% | Running “LRBasicTest.py” gives expected results |
| +25% | Running “LRPolyTest.py” gives expected results |
| +15% | Running “LRPolyRegTest.py” gives expected results |
| +10% | Tuning provides optimal results when “LinearPolyReg.bat” is run |
|  |  |
|  |  |
| -10% | Program Crash, -10 is a minimum penalty. |
| -2% | Per compiler warnings |
| -5% | Incorrect Turn-In, don’t make me take off points for this. It makes me feel like a jerk. |

# Submissions

Please clean your project (VS menu, Build->Clean Solution), then zip the project. Next, rename the archive file to match this format *LastName.FirstName.LR.zip* and submit it on FSO.