John Nonnenmacher

CPE 593 Final Project

May 14, 2021

Automated Exploratory Data Analysis

Introduction:

For this project I worked on my own to come up with an automated exploratory data analysis algorithm. I did this by creating multiple different algorithms to fit a variety of different curves to a given dataset. Root mean squared error was utilized inorder to judge the overall fitness of a curve. The algorithm attempts to fit a constant value, a linear sequence, a quadratic sequence, a 2D linear sequence, a 2D quadratic sequence, an exponential curve, and a sine wave to a given dataset. It then calculates the root mean squared error for each of the fits and it iterates through them in order to determine the best algorithm.

Curve Fitting:

The algorithm for constant value is unique compared to the other algorithms. The algorithm for constant value works by iterating through all of the output values and finding the mean value of the output in order to determine the constant value that fits the data the best. It then iterates through the output values again in order to calculate the total squared error. It then returns the root of that squared error along with the constant value. This algorithm is unique because it only has one parameter and it did not require gradient descent. The reason for this is because the output was not dependent on the input. All the other algorithms made use of batch gradient descent. How the other

algorithms worked is that first the loss function for the algorithm was calculated. From the loss function the partial derivatives of each parameter were calculated. (This work can be found in the documents folder in the repository under the file calculations). Once the partial derivatives were calculated the algorithm would utilize batch gradient descent to come up with the ideal values of the parameters. How gradient descent works is that initially the values of the parameters are set to 0. From there the error of the function is calculated given those parameters. Once the error is calculated the error associated to each parameter is calculated by utilizing the partial derivative of the loss function and multiplying that by the total error. That number is then multiplied by the learning rate (a very small number to make sure the parameters do not over step) and is subtracted from the parameters current value inorder to get a new value for the parameter. This process is repeated for a large number of iterations or until the difference between the parameters and the new parameters is deemed negligible. There are three types of gradient descent: batch, stochastic, and mini batch. Batch utilizes the entire dataset to determine the error, while stochastic utilizes a single data point to determine the error. And mini batch gradient descent is a combination of the two and utilizes a small number of datapoints to determine the error. Batch was utilized because while it is slower it is also more representative of the dataset. This whole process was able to determine the parameters that best fit the data for the given function type.

Algorithm Comparison:

Once the parameters for a curve were determined. The algorithm would iterate through the data once more to determine the root mean square error of the curve. The root mean square error would then be compared to the root mean squared error of

every other curve to determine which curve fits the data the best. The algorithm returns the root mean squared error for all of the algorithms and prints out a statement about which curve fits the data the best, and the parameters calculated for that curve.

Running the Program:

There are two different ways to run the project. The first is by creating an array of data and utilizing the DataAnalysis object on your array. I have already done an example of this for each of the possible functions and I added normally distributed noise to each of the functions so that the data is more realistic. To run my sample data go to CPE593DataAnalysis in your terminal. Than do the following commands:

cd demo

mvn clean install

mvn compile exec:java -Dexec.mainClass="jack.App"

The second way to run the project is by creating a binary file for both your input and output variable. If you do not wish to create your own I have provided 1 binary input file "input.dat" and 3 binary output files "lin.dat", "quad.dat", and "sin.dat" When running the project this way the project will give you some instructions and than ask you for the name of your input file and your output file. Make sure that both of these files are in the repository and exist/spelled correctly otherwise it will cause an error. It will ask you some other questions like do you want to add a second input and is the binary file a file of double[] or Integer[]. Answering incorrectly can result in an error. To run the project this way go to CPE593DataAnalysis in your terminal. Than do the following commands:

cd demo

mvn clean install

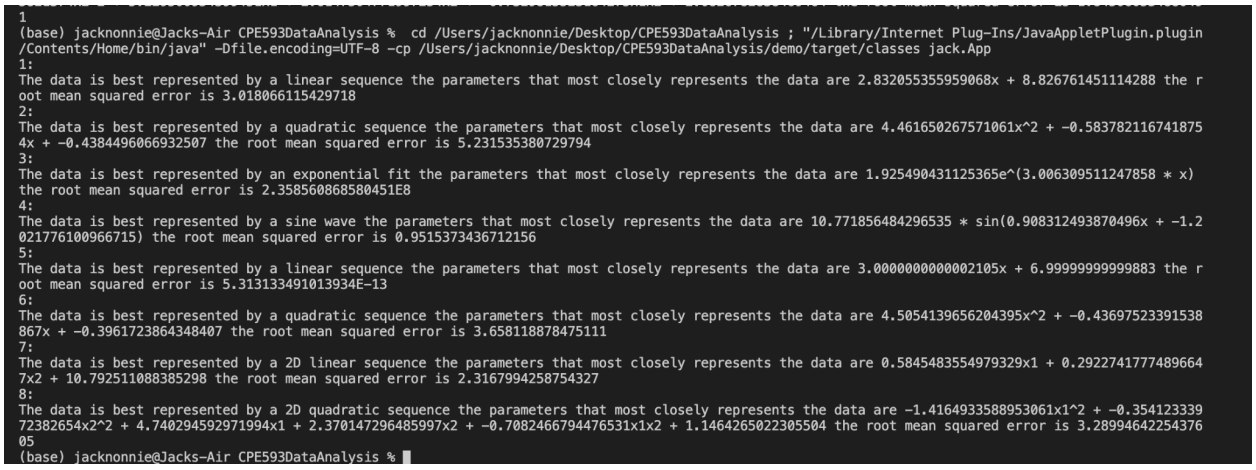
mvn compile exec:java -Dexec.mainClass="jack.Input"

Testing:

The first round of testing came from just testing the algorithms for each individual curve fit to make sure they were working correctly. This was continuous throughout the entire project, and each curve was tested before moving onto the next. The second round of testing came at integrating the algorithms into the DataAnalysis object. The DataAnalysis class is what was utilized to compare the different curve fits. In this round of testing the algorithm was being tested to make sure that it returned the correct curve. The third round of testing was when noise was added to the data. And the fourth round of testing was testing reading and writing of binary files and if that still returned accurate results.

Output:

There are two outputs for this file the first being the output of the list of data and the second being the output from the reading of binary files. The outputs are shown respectively:



```
(base) jacknonnie@Jacks-Air CPE593DataAnalysis %  cd /Users/jacknonnie/Desktop/CPE593DataAnalysis ; "/Library/Internet Plug-Ins/JavaAppletPlugin.plugin
/Contents/Home/bin/java" -Dfile.encoding=UTF-8 -cp /Users/jacknonnie/Desktop/CPE593DataAnalysis/demo/target/classes jack.Input
The program is going to ask you for the name of the binary file that is the input variable, the output variable, if its a double and if there are two i
nput variables.
From your answer the data will automatically be analysed and tell you the result that most closely matches your data
Make sure the binary file was already added to the repository prior to running the program and that you spell the name of the file correctly. Failure t
o do so will cause an error.
Enter the input variable binary file name:
input.dat
If you would like to add a second input file type 'Y' type anything else for 1 input variable
n
Enter the output variable binary file name:
lin.dat
If the binary data you uploaded is a int[] type 'Y' if it is a double[] type in anything else
n
The data is best represented by a linear sequence the parameters that most closely represents the data are 2.489074473517958x + 7.401761779879935 the r
oot mean squared error is 1.2514483576655608
(base) jacknonnie@Jacks-Air CPE593DataAnalysis %  cd /Users/jacknonnie/Desktop/CPE593DataAnalysis ; "/Library/Internet Plug-Ins/JavaAppletPlugin.plugin
/Contents/Home/bin/java" -Dfile.encoding=UTF-8 -cp /Users/jacknonnie/Desktop/CPE593DataAnalysis/demo/target/classes jack.Input
The program is going to ask you for the name of the binary file that is the input variable, the output variable, if its a double and if there are two i
nput variables.
From your answer the data will automatically be analysed and tell you the result that most closely matches your data
Make sure the binary file was already added to the repository prior to running the program and that you spell the name of the file correctly. Failure t
o do so will cause an error.
Enter the input variable binary file name:
input.dat
If you would like to add a second input file type 'Y' type anything else for 1 input variable
n
Enter the output variable binary file name:
quad.dat
If the binary data you uploaded is a int[] type 'Y' if it is a double[] type in anything else
n
The data is best represented by a quadratic sequence the parameters that most closely represents the data are 0.603236397281216x^2 + 1.5026422147352354
x + 0.07436716692955041 the root mean squared error is 3.742138176157308
(base) jacknonnie@Jacks-Air CPE593DataAnalysis %  cd /Users/jacknonnie/Desktop/CPE593DataAnalysis ; "/Library/Internet Plug-Ins/JavaAppletPlugin.plugin
/Contents/Home/bin/java" -Dfile.encoding=UTF-8 -cp /Users/jacknonnie/Desktop/CPE593DataAnalysis/demo/target/classes jack.Input
The program is going to ask you for the name of the binary file that is the input variable, the output variable, if its a double and if there are two i
nput variables.
From your answer the data will automatically be analysed and tell you the result that most closely matches your data
Make sure the binary file was already added to the repository prior to running the program and that you spell the name of the file correctly. Failure t
o do so will cause an error.
Enter the input variable binary file name:
input.dat
If you would like to add a second input file type 'Y' type anything else for 1 input variable
n
Enter the output variable binary file name:
sin.dat
If the binary data you uploaded is a int[] type 'Y' if it is a double[] type in anything else
n
The data is best represented by a sine wave the parameters that most closely represents the data are 11.48413048784303 * sin(-0.3414635603715179x + 2.6
376793620953416) the root mean squared error is 2.843564249942474
(base) jacknonnie@Jacks-Air CPE593DataAnalysis %
```

Each output correctly classifies the data into the correct curve. Each output also remarkably has an incredibly small root mean square error. The one exception being the exponential data (The exponential data modeled the data incredibly well and the log error was incredibly small, but due to the nature of exponential growth a small percentage wise error results in a massive actual error. The error for the exponential was still significantly smaller than the error for all other curves on the same data).

Special Cases:

The first special case is in the collecting of data for the exponential function. Since exponentials grow at an incredible rate the only way to model the data is by taking the log of both sides of the loss equation. This does not affect the results of the curve fitting but was unique enough to be noted. The second special case does affect the results of the algorithm. This special case is in reference to curve fitting a sine wave.

When doing so if there are not enough inputs for the sine wave to cycle and/or the b parameter ($y = asin(bx + c)$) is of a size that with the current inputs prevents any cycling the sine wave will have a chance to be misinterpreted as another function. The final special case is that if the inputs get too large especially for the quadratic function and the exponential function the squared error may get too large early on in the gradient descent process causing a miscalculation.

Conclusion:

Overall the whole project was a massive success. I definitely learned a lot from the whole project. I know that coding this project in python would have been easier but it was really cool coding it in Java to learn the background behind everything and it definitely made the project faster. Moving forward with the project I would like to add a cubic splice to the curve fitting algorithm. The reason why I did not include it in this project is because I did not know how to adjust the break points of a splice in order to get it to fit the data and I also did not know how to set the first and second derivative equal to the new curve at the break points. I would also like to add an adaptive learning rate to the project because I think that that would be very useful.