Justin Reusnow

Professor Chakraborty

CMP SCI 4340

5 December 2017

Project 5

The way my implementation of linear regression using regularization works in terms of data generation, is that each run a different set of 12 points (in the given range) are generated. I wasn’t sure if this is what was desired based on the project specification, but it is what I went with to further test my code (after all, a differing set of data each run means a different way to test the code each time). I have read the book quite thoroughly and done my best to implement it correctly, and I think I have done the majority of it correctly, however when I look at the outputs something still seems wrong but I have spent a considerable amount of time investigating the issue and cannot seem to determine what could be the issue, thus I will report my findings here despite my gut feeling about the data. To do this, I will start with the resulting output from a standard run of the program, and then explain each step:

========================= RESULTS =========================

(a) Twelve (X, Y) coordinate pairs:

• (8.85719975924918, 88.44998757524374)

• (10.670845770879657, 123.86694946590026)

• (-0.3194028062982732, 10.102018152671212)

• (0.16648234562598274, 10.027716371405129)

• (3.557571234683021, 22.656313089844073)

• (-0.5910155937678914, 10.349299432076814)

• (3.7690005194872063, 24.205364915894833)

• (-1.6417070910387201, 12.695202172766816)

• (0.09419656420569433, 10.008872992708158)

• (3.7026699224592603, 23.709764554684465)

• (-0.3951330377375355, 10.156130117511692)

• (-1.534301563650935, 12.354081288221703)

(b) Original Regression Line:

• "y=8.448252983756689\*x+11.340423279535571"

(c) Four (Lambda, E\_in, E\_cv) Triplets:

• (0.1, 1889.7310944042079, 106.9815171126517)

• (1.0, 1890.3601689618079, 157.05226934044097)

• (10.0, 1896.6509145378081, 401.32686083017074)

• (100.0, 1959.5583702978108, 1282.1449583669728)

(d) Final Lambda:

• 0.1

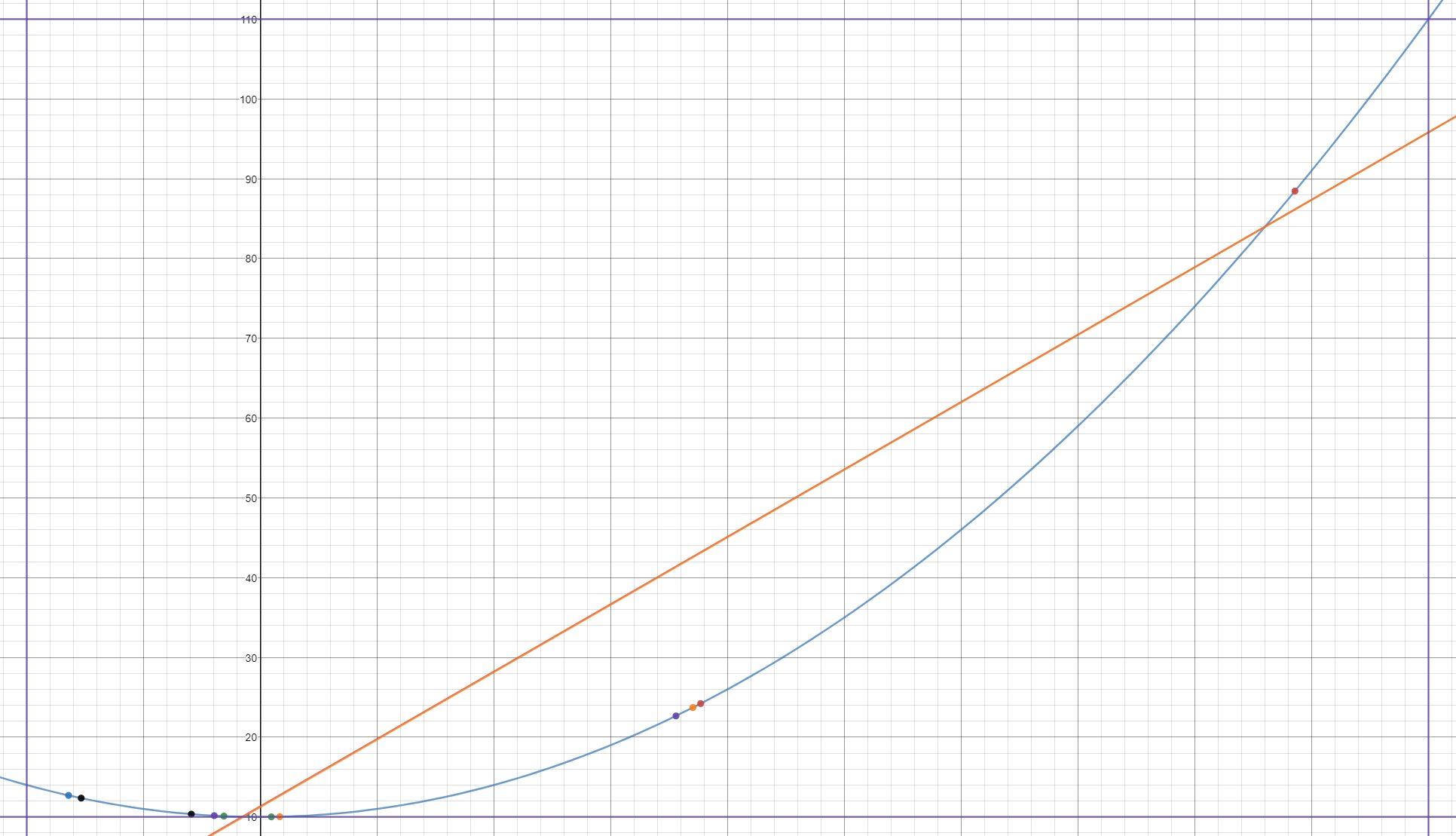
(e) Regularized Regression Line:

• "y=8.457204785853733\*x+11.227216616508398"

Final E\_in:

• 1889.7310944042079

1. It looks like a wide range of X values were randomly generated (this is ideal) which resulted in many different Y values being computed via the provided function f(x) = x2 + 10. You can see these points plotted in the graph. The target function is the blue parabolic line. As expected, every point falls on the target function’s line.
2. Using standard linear regression calculations I have determined that the regression line for this data set is ~ y=8.448\*x+11.34. This line is denoted in orange on the graph, but note that you cannot locate it due to a red line covering it up (at this scale). More on this line later.
3. The four lambda triplets yielded the relationship of: As the value of the lambda increases, the value of the in sample error and cross-validation estimate (each of which was determined through three-fold cross-validation) also goes up, which leads to my next point.
4. The final lambda selected is 0.1, which is because we choose our lambda based on that which minimizes the cross-validation estimate, and as I said before it appears that as lambda increases, so too does the cross-validation estimate, therefore we want the lambda corresponding to the smallest lambda, which would be 0.1.
5. Equipped with these details, I recalculate the regression line using the new lambda to arrive at my final regularized regression line equation of ~y=8.457\*x+11.227. Note that this equation is very close to the original regression line, differing only slightly in both slope and intercept. This seems natural due to the small value selected for lambda. I plotted this new line in red, but because of the similarity between the original regression line and this line, the two are nearly identical and thus on top of each other (at this scale), so you can see almost entirely just the red line (regularized linear regression).



(Source code on next page)

Source Code

import org.ejml.simple.SimpleMatrix;  
  
import java.util.Arrays;  
  
public class Project5 {  
 private static final double *RANDOM\_DOUBLE\_LOWER\_BOUND* = -2;  
 private static final double *RANDOM\_DOUBLE\_UPPER\_BOUND* = 10;  
  
 static final int *N* = 12;  
 static final int *K* = 3;  
  
 public static void main(final String[] *args*) throws Exception {  
 final double[][] *XData* = new double[*N*][];  
 for (int *i* = 0; *i* < Project5.*N*; *i*++) {  
 *XData*[*i*] = new double[]{  
 LinearRegression.*X\_0*,  
 Project5.*randomDouble*(Project5.*RANDOM\_DOUBLE\_LOWER\_BOUND*, Project5.*RANDOM\_DOUBLE\_UPPER\_BOUND*)  
 };  
 }  
 final SimpleMatrix *X\_matrix* = new SimpleMatrix(*XData*);  
  
 final double[][] *YData* = new double[*N*][];  
 for (int *i* = 0; *i* < Project5.*N*; *i*++) {  
 *YData*[*i*] = new double[]{  
 *randomSampleFunction*(*XData*[*i*][1])  
 };  
 }  
 final SimpleMatrix *Y\_matrix* = new SimpleMatrix(*YData*);  
  
 final LinearRegression *linearRegression* = new LinearRegression(*X\_matrix*, *Y\_matrix*);  
 System.*out*.println(*linearRegression*.getLinearRegressionResult().toResultString());  
 }  
  
 private static double randomDouble(final double *lowerBoundInclusive*, final double *upperBoundInclusive*) {  
 return *lowerBoundInclusive* + (Math.*random*() \* ((*upperBoundInclusive* - *lowerBoundInclusive*) + 1));  
 }  
  
 private static double randomSampleFunction(final double *X*) {  
 return Math.*pow*(*X*, 2) + 10;  
 }  
}  
  
class LinearRegression {  
 static final double *X\_0* = 1.0;  
 private static final double *LAMBDA\_1* = 0.1;  
 private static final double *LAMBDA\_2* = 1.0;  
 private static final double *LAMBDA\_3* = 10.0;  
 private static final double *LAMBDA\_4* = 100.0;  
 private static final double[] *LAMBDAS* = new double[]{*LAMBDA\_1*, *LAMBDA\_2*, *LAMBDA\_3*, *LAMBDA\_4*};  
  
 private final LinearRegressionResult linearRegressionResult;  
 private final SimpleMatrix X\_matrix;  
 private final SimpleMatrix Y\_matrix;  
 private final String regressionLine;  
 private final String regularizedRegressionLine;  
  
 LinearRegression(final SimpleMatrix *X\_matrix*, final SimpleMatrix *Y\_matrix*) throws Exception {  
 this.linearRegressionResult = new LinearRegressionResult();  
 this.linearRegressionResult.setLambdas(LinearRegression.*LAMBDAS*);  
  
 if (*X\_matrix*.numRows() == *Y\_matrix*.numRows()) {  
 this.X\_matrix = *X\_matrix*;  
 this.linearRegressionResult.set\_\_X\_matrix(this.X\_matrix);  
  
 this.Y\_matrix = *Y\_matrix*;  
 this.linearRegressionResult.set\_\_Y\_matrix(this.Y\_matrix);  
  
 this.regressionLine = this.calculateRegressionLine();  
 this.linearRegressionResult.setRegressionLine(this.regressionLine);  
  
 SimpleMatrix *weightMatrix* = new SimpleMatrix(new double[][]{  
 new double[]{  
 Math.*random*() // Bias Weight  
 },  
 new double[]{  
 Math.*random*() // X\_1 Weight  
 }  
 });  
 this.regularizedRegressionLine = this.calculateRegularizedRegressionLine(*weightMatrix*);  
 this.linearRegressionResult.setRegularizedRegressionLine(this.regularizedRegressionLine);  
 } else {  
 throw new Exception("Number of X\_matrix and number of Y\_matrix differ, cannot preform linear regression.");  
 }  
 }  
  
 LinearRegressionResult getLinearRegressionResult() {  
 return linearRegressionResult;  
 }  
  
 @Override  
 public String toString() {  
 return "LinearRegression{" +  
 "linearRegressionResult=" + this.linearRegressionResult +  
 ", X\_matrix=" + this.X\_matrix +  
 ", Y\_matrix=" + this.Y\_matrix +  
 ", regressionLine='" + this.regressionLine + '\'' +  
 ", regularizedRegressionLine='" + this.regularizedRegressionLine + '\'' +  
 '}';  
 }  
  
 private String calculateRegressionLine() {  
 SimpleMatrix *XSwordMatrix* = this.calculate\_X\_sword(this.X\_matrix);  
 SimpleMatrix *W\_lin* = this.calculate\_W\_lin(*XSwordMatrix*, this.Y\_matrix);  
  
 return "y=" + *W\_lin*.get(1, 0) + "\*x+" + *W\_lin*.get(0, 0);  
 }  
  
 private SimpleMatrix calculate\_X\_sword(final SimpleMatrix *X\_matrix*) {  
 return *X\_matrix*.transpose().mult(*X\_matrix*).invert().mult(*X\_matrix*.transpose());  
 }  
  
 private SimpleMatrix calculate\_W\_lin(final SimpleMatrix *XSwordMatrix*, final SimpleMatrix *Y\_matrix*) {  
 return *XSwordMatrix*.mult(*Y\_matrix*);  
 }  
  
 private String calculateRegularizedRegressionLine(final SimpleMatrix *weightMatrix*) {  
 System.*out*.println("Calculating regularized regression line...");  
  
 String *regularizedRegressionLine*;  
  
 double *optimalLambda* = this.calculateOptimalLambdaViaCrossValidation(*weightMatrix*);  
 this.linearRegressionResult.setFinalLambda(*optimalLambda*);  
  
 final double *final\_E\_in* = this.calculate\_E\_aug(*weightMatrix*, this.X\_matrix, this.Y\_matrix, *optimalLambda*);  
 this.linearRegressionResult.setFinal\_E\_in(*final\_E\_in*);  
  
 SimpleMatrix *regularized\_X\_sword\_matrix* = this.calculate\_regularized\_X\_sword(this.X\_matrix, *optimalLambda*);  
 SimpleMatrix *regularized\_W\_lin* = this.calculate\_W\_lin(*regularized\_X\_sword\_matrix*, this.Y\_matrix);  
  
 *regularizedRegressionLine* = "y=" + *regularized\_W\_lin*.get(1, 0) + "\*x+" + *regularized\_W\_lin*.get(0, 0);  
  
 System.*out*.println("Regularized regression line is: \"" + *regularizedRegressionLine* + "\"");  
 return *regularizedRegressionLine*;  
 }  
  
 private SimpleMatrix calculate\_regularized\_X\_sword(final SimpleMatrix *X\_matrix*, final double *lambda*) {  
 return *X\_matrix*.transpose().mult(*X\_matrix*).plus(SimpleMatrix.*identity*(*X\_matrix*.numCols()).scale(*lambda*)).invert().mult(*X\_matrix*.transpose());  
 }  
  
 private double calculateOptimalLambdaViaCrossValidation(final SimpleMatrix *weightMatrix*) {  
 System.*out*.println("--Calculate optimal lambda (min(for each lambda: E\_cv(lambda))...");  
  
 double *smallest\_E\_cv* = Double.*MAX\_VALUE*;  
 double *optimalLambda* = LinearRegression.*LAMBDAS*[0];  
  
 for (int *i* = 0; *i* < LinearRegression.*LAMBDAS*.length; *i*++) {  
 System.*out*.println("----Trying lambda \"" + LinearRegression.*LAMBDAS*[*i*] + "\"...");  
  
 double *current\_E\_cv* = this.calculate\_E\_cv(LinearRegression.*LAMBDAS*[*i*]);  
  
 double[] *temporaryResult\_E\_ins* = this.linearRegressionResult.getE\_ins();  
 *temporaryResult\_E\_ins*[*i*] = this.calculate\_E\_aug(*weightMatrix*, this.X\_matrix, this.Y\_matrix, LinearRegression.*LAMBDAS*[*i*]);  
 this.linearRegressionResult.setE\_ins(*temporaryResult\_E\_ins*);  
  
 double[] *temporaryResult\_E\_Cvs* = this.linearRegressionResult.getE\_cvs();  
 *temporaryResult\_E\_Cvs*[*i*] = *current\_E\_cv*;  
 this.linearRegressionResult.setE\_cvs(*temporaryResult\_E\_Cvs*);  
  
 if (*current\_E\_cv* < *smallest\_E\_cv*) {  
 System.*out*.println("----This lambda's E\_cv (lambda: \"" + LinearRegression.*LAMBDAS*[*i*] + "\", E\_cv: \"" + *current\_E\_cv* + "\") is better than the best lambda's E\_cv so far (lambda: \"" + *optimalLambda* + "\", E\_cv: \"" + *smallest\_E\_cv* + "\"), reassigning it.");  
  
 *smallest\_E\_cv* = *current\_E\_cv*;  
 *optimalLambda* = LinearRegression.*LAMBDAS*[*i*];  
 } else {  
 System.*out*.println("----This lambda's E\_cv (lambda: \"" + LinearRegression.*LAMBDAS*[*i*] + "\", E\_cv: \"" + *current\_E\_cv* + "\") is NOT better than the best lambda's E\_cv so far (lambda: \"" + *optimalLambda* + "\", E\_cv: \"" + *smallest\_E\_cv* + "\").");  
 }  
 }  
  
 System.*out*.println("--Calculated optimal lambda to be \"" + *optimalLambda* + "\".");  
 return *optimalLambda*;  
 }  
  
 private double calculate\_E\_cv(final double *lambda*) {  
 System.*out*.println("------Calculating E\_cv for lambda \"" + *lambda* + "\" ((sum(for each leaveNOut set: leaveNOut\_E\_aug(lambda))) / n)...");  
  
 double *E\_cv*;  
 double[] *all\_E\_val* = new double[Project5.*K*];  
  
 for (int *i* = 0; *i* < Project5.*K*; *i*++) {  
 final double[][] *leaveNInXData* = new double[Project5.*N* - Project5.*K*][];  
 final double[][] *leaveNOutXData* = new double[Project5.*K*][];  
 final double[][] *leaveNInYData* = new double[Project5.*N* - Project5.*K*][];  
 final double[][] *leaveNOutYData* = new double[Project5.*K*][];  
  
 int *k\_in* = 0;  
 int *k\_out* = 0;  
 for (int *j* = 0; *j* < Project5.*N*; *j*++) {  
 if (*j* < *i* \* Project5.*K* || *j* >= (*i* + 1) \* Project5.*K*) {  
 *leaveNInXData*[*k\_in*] = new double[]{  
 this.X\_matrix.get(*j*, 0),  
 this.X\_matrix.get(*j*, 1)  
 };  
 *leaveNInYData*[*k\_in*] = new double[]{  
 this.Y\_matrix.get(*j*, 0)  
 };  
  
 *k\_in*++;  
 } else {  
 *leaveNOutXData*[*k\_out*] = new double[]{  
 this.X\_matrix.get(*j*, 0),  
 this.X\_matrix.get(*j*, 1)  
 };  
 *leaveNOutYData*[*k\_out*] = new double[]{  
 this.Y\_matrix.get(*j*, 0)  
 };  
 *k\_out*++;  
 }  
 }  
 SimpleMatrix *leaveNIn\_X\_matrix* = new SimpleMatrix(*leaveNInXData*);  
 SimpleMatrix *leaveNIn\_Y\_matrix* = new SimpleMatrix(*leaveNInYData*);  
 SimpleMatrix *leaveNOut\_X\_matrix* = new SimpleMatrix(*leaveNOutXData*);  
 SimpleMatrix *leaveNOut\_Y\_matrix* = new SimpleMatrix(*leaveNOutYData*);  
  
 SimpleMatrix *leaveNInXSwordMatrix* = this.calculate\_regularized\_X\_sword(*leaveNIn\_X\_matrix*, *lambda*);  
 SimpleMatrix *leaveNIn\_W\_lin* = this.calculate\_W\_lin(*leaveNInXSwordMatrix*, *leaveNIn\_Y\_matrix*);  
  
 *all\_E\_val*[*i*] = this.calculate\_E\_val(*leaveNIn\_W\_lin*, *leaveNOut\_X\_matrix*, *leaveNOut\_Y\_matrix*, *lambda*, Project5.*K*);  
 }  
  
 Double *sum\_E\_val* = 0.0;  
 for (final double *E\_val* : *all\_E\_val*) {  
 *sum\_E\_val* += *E\_val*;  
 }  
  
 *E\_cv* = *sum\_E\_val* / Project5.*K*;  
 System.*out*.println("------Calculated E\_cv for lambda \"" + *lambda* + "\" to be \"" + *E\_cv* + "\".");  
 return *E\_cv*;  
 }  
  
 private double calculate\_E\_val(final SimpleMatrix *W\_lin*, final SimpleMatrix *leaveNOut\_X\_matrix*, final SimpleMatrix *leaveNOut\_Y\_matrix*, final double *lambda*, final int *K*) {  
 System.*out*.println("--------Calculating E\_val for lambda \"" + *lambda* + "\" on data set of size \"" + *K* + "\" (E\_aug / K)...");  
  
 double *E\_val* = calculate\_E\_aug(*W\_lin*, *leaveNOut\_X\_matrix*, *leaveNOut\_Y\_matrix*, *lambda*) / *K*;  
 System.*out*.println("--------Calculated E\_val for lambda \"" + *lambda* + "\" on data set of size \"" + *K* + "\" to be \"" + *E\_val* + "\".");  
 return *E\_val*;  
 }  
  
 private double calculate\_E\_aug(final SimpleMatrix *weightMatrix*, final SimpleMatrix *X\_matrix*, final SimpleMatrix *Y\_matrix*, final double *lambda*) {  
 System.*out*.println("----------Calculating E\_aug for lambda \"" + *lambda* + "\" (E\_in + lambda \* wTw)...");  
  
 double *E\_aug*;  
 double *wTw* = 0.0;  
  
 for (int *i* = 0; *i* < *weightMatrix*.numRows(); *i*++) {  
 *wTw* += Math.*pow*(*weightMatrix*.get(*i*, 0), 2);  
 }  
  
 *E\_aug* = this.calculate\_E\_in(*weightMatrix*, *X\_matrix*, *Y\_matrix*) + *lambda* \* *wTw*; // Ridge regression happens here.  
 System.*out*.println("----------Calculated E\_aug for lambda \"" + *lambda* + "\" to be \"" + *E\_aug* + "\".");  
 return *E\_aug*;  
 }  
  
 private double calculate\_E\_in(final SimpleMatrix *weightMatrix*, final SimpleMatrix *X\_matrix*, final SimpleMatrix *Y\_matrix*) {  
 System.*out*.println("------------Calculating E\_in ((sum((wTx - y)^2))/n)...");  
  
 double *E\_in*;  
  
 double *sumE\_in* = 0.0;  
  
 for (int *i* = 0; *i* < *X\_matrix*.numRows(); *i*++) {  
 final double *X\_i* = *X\_matrix*.get(*i*, 1);  
 final double *Y\_i* = *Y\_matrix*.get(*i*, 0);  
  
 *sumE\_in* += Math.*pow*((this.calculate\_wTx(*weightMatrix*, *X\_i*) - *Y\_i*), 2);  
 }  
  
 *E\_in* = *sumE\_in* / Project5.*N*;  
 System.*out*.println("------------Calculated E\_in to be \"" + *E\_in* + "\".");  
 return *E\_in*;  
 }  
  
 private double calculate\_wTx(final SimpleMatrix *weightMatrix*, final double *X\_1*) {  
 System.*out*.println("--------------Calculating wTx (" + *weightMatrix*.get(0, 0) + " \* " + LinearRegression.*X\_0* + " + " + *weightMatrix*.get(1, 0) + " \* " + *X\_1* + ")...");  
  
 double *wTx* = *weightMatrix*.get(0, 0) \* LinearRegression.*X\_0* + *weightMatrix*.get(1, 0) \* *X\_1*;  
 System.*out*.println("--------------Calculated wTx to be \"" + *wTx* + "\".");  
 return *wTx*;  
 }  
}  
  
class LinearRegressionResult {  
 private SimpleMatrix X\_matrix;  
 private SimpleMatrix Y\_matrix;  
 private String regressionLine;  
 private double[] lambdas;  
 private double[] E\_ins = new double[4];  
 private double[] E\_cvs = new double[4];  
 private double finalLambda;  
 private String regularizedRegressionLine;  
 private double final\_E\_in;  
  
 LinearRegressionResult() {  
 }  
  
 void set\_\_X\_matrix(final SimpleMatrix *X\_matrix*) {  
 this.X\_matrix = *X\_matrix*;  
 }  
  
 void set\_\_Y\_matrix(final SimpleMatrix *Y\_matrix*) {  
 this.Y\_matrix = *Y\_matrix*;  
 }  
  
 void setRegressionLine(final String *regressionLine*) {  
 this.regressionLine = *regressionLine*;  
 }  
  
 void setLambdas(final double[] *lambdas*) {  
 this.lambdas = *lambdas*;  
 }  
  
 double[] getE\_ins() {  
 return E\_ins;  
 }  
  
 void setE\_ins(final double[] *E\_ins*) {  
 this.E\_ins = *E\_ins*;  
 }  
  
 double[] getE\_cvs() {  
 return E\_cvs;  
 }  
  
 void setE\_cvs(final double[] *E\_cvs*) {  
 this.E\_cvs = *E\_cvs*;  
 }  
  
 void setFinalLambda(final double *finalLambda*) {  
 this.finalLambda = *finalLambda*;  
 }  
  
 void setRegularizedRegressionLine(final String *regularizedRegressionLine*) {  
 this.regularizedRegressionLine = *regularizedRegressionLine*;  
 }  
  
 void setFinal\_E\_in(final double *final\_E\_in*) {  
 this.final\_E\_in = *final\_E\_in*;  
 }  
  
 @Override  
 public String toString() {  
 return "LinearRegressionResult{" +  
 "\_X\_matrix=" + this.X\_matrix +  
 ", Y\_matrix=" + this.Y\_matrix +  
 ", regressionLine='" + this.regressionLine + '\'' +  
 ", lambdas=" + Arrays.*toString*(this.lambdas) +  
 ", E\_ins=" + Arrays.*toString*(this.E\_ins) +  
 ", E\_cvs=" + Arrays.*toString*(this.E\_cvs) +  
 ", finalLambda=" + this.finalLambda +  
 ", regularizedRegressionLine='" + this.regularizedRegressionLine + '\'' +  
 ", final\_E\_in=" + this.final\_E\_in +  
 '}';  
 }  
  
 String toResultString() {  
 StringBuilder *output* = new StringBuilder("\n========================= RESULTS =========================");  
  
 *output*.append("\n(a) Twelve (X, Y) coordinate pairs: ");  
 for (int *i* = 0; *i* < Project5.*N*; *i*++) {  
 *output*.append("\n • (").append(this.X\_matrix.get(*i*, 1)).append(", ").append(this.Y\_matrix.get(*i*, 0)).append(")");  
 }  
  
 *output*.append("\n(b) Original Regression Line:");  
 *output*.append("\n • \"").append(this.regressionLine).append("\"");  
  
 *output*.append("\n(c) Four (Lambda, E\_in, E\_cv) Triplets:");  
 *output*.append("\n • (").append(this.lambdas[0]).append(", ").append(this.E\_ins[0]).append(", ").append(this.E\_cvs[0]).append(")");  
 *output*.append("\n • (").append(this.lambdas[1]).append(", ").append(this.E\_ins[1]).append(", ").append(this.E\_cvs[1]).append(")");  
 *output*.append("\n • (").append(this.lambdas[2]).append(", ").append(this.E\_ins[2]).append(", ").append(this.E\_cvs[2]).append(")");  
 *output*.append("\n • (").append(this.lambdas[3]).append(", ").append(this.E\_ins[3]).append(", ").append(this.E\_cvs[3]).append(")");  
  
 *output*.append("\n(d) Final Lambda:");  
 *output*.append("\n • ").append(this.finalLambda);  
  
 *output*.append("\n(e) Regularized Regression Line:");  
 *output*.append("\n • \"").append(this.regularizedRegressionLine).append("\"");  
 *output*.append("\n Final E\_in:");  
 *output*.append("\n • ").append(this.final\_E\_in);  
  
 return *output*.toString();  
 }  
}