Justin Reusnow

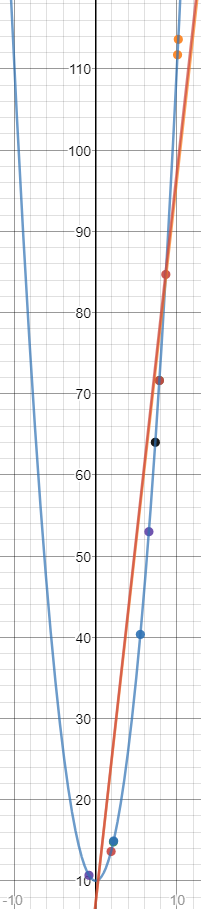
Professor Chakraborty

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 most 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:

• (8.64239933536798, 84.6910662719689)

• (5.5089048899719195, 40.34803308675653)

• (2.1883353237783627, 14.788811489296151)

• (10.180882259792751, 113.65036358776275)

• (-0.8237717040811703, 10.678599820444795)

• (1.90053445017955, 13.612031196319284)

• (2.215616602059776, 14.908956927322908)

• (7.850783101923859, 71.63479531345322)

• (10.086729657708892, 111.74211518770414)

• (6.557675631887632, 53.00310969305285)

• (7.349616946722655, 64.01686926355285)

• (7.849824705699229, 71.61974791020597)

(b) Original Regression Line:

• "y=8.896610169491526\*x+7.384745762711863"

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

• (0.1, 3832.6740094092124, 3832.721740805104)

• (1.0, 3832.6740094092124, 3833.1513233681258)

• (10.0, 3832.6740094092124, 3837.4471489983466)

• (100.0, 3832.6740094092124, 3880.40540530056)

(d) Final Lambda:

• 0.1

(e) Regularized Regression Line:

• "y=8.996610169491525\*x+7.2847457627118635"

Final E\_in:

• 3832.7217408051038

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 fall 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=8.897\*x+7.385. 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 same in sample error (this does not seem correct, hence why I am worrisome about my data), and a relationship of: As the value of the lambda increases, the value of the cross-validation estimate 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=8.997\*x+7.285. Note that this equation is very close to the original regression line, differing only slightly in both slope and intercept. I plotted this new line in red, but because of the similarity, the two lines 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 is on the next page)

Source Code

import java.util.ArrayList;  
  
public class Project5 {  
 private static final int *QUANTITY\_RANDOM\_DOUBLES* = 12;  
 private static final double *RANDOM\_DOUBLE\_LOWER\_BOUND* = -2;  
 private static final double *RANDOM\_DOUBLE\_UPPER\_BOUND* = 10;  
  
 public static void main(final String[] *args*) throws Exception {  
 final ArrayList<Double> *Xs* = new ArrayList<>();  
 for (int *i* = 0; *i* < Project5.*QUANTITY\_RANDOM\_DOUBLES*; *i*++) {  
 *Xs*.add(Project5.*randomDouble*(Project5.*RANDOM\_DOUBLE\_LOWER\_BOUND*, Project5.*RANDOM\_DOUBLE\_UPPER\_BOUND*));  
 }  
  
 final ArrayList<Double> *Ys* = new ArrayList<>();  
 for (final double *X* : *Xs*) {  
 *Ys*.add(*randomSampleFunction*(*X*));  
 }  
  
 final LinearRegression *linearRegression* = new LinearRegression(*Xs*, *Ys*);  
 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 {  
 private static final double *X\_0* = 1.0;  
 private static final ArrayList<Double> *LAMBDAS* = new ArrayList<>();  
 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;  
  
 static {  
 LinearRegression.*LAMBDAS*.add(*LAMBDA\_1*);  
 LinearRegression.*LAMBDAS*.add(*LAMBDA\_2*);  
 LinearRegression.*LAMBDAS*.add(*LAMBDA\_3*);  
 LinearRegression.*LAMBDAS*.add(*LAMBDA\_4*);  
 }  
  
 private final LinearRegressionResult linearRegressionResult;  
 private final ArrayList<Double> Xs;  
 private final ArrayList<Double> Ys;  
 private final ArrayList<Double> weights;  
 private final int n;  
 private final int sumX;  
 private final int sumY;  
 private final int sumXY;  
 private final int sumXSquared;  
 private final double a;  
 private final double b;  
 private final String regressionLine;  
 private final String regularizedRegressionLine;  
  
 LinearRegression(final ArrayList<Double> *Xs*, final ArrayList<Double> *Ys*) throws Exception {  
 this.linearRegressionResult = new LinearRegressionResult();  
 this.linearRegressionResult.setLambdas(LinearRegression.*LAMBDAS*);  
  
 if (*Xs*.size() == *Ys*.size()) {  
 this.Xs = *Xs*;  
 this.linearRegressionResult.setXs(this.Xs);  
  
 this.Ys = *Ys*;  
 this.linearRegressionResult.setYs(this.Ys);  
  
 this.weights = new ArrayList<>();  
 this.n = this.Xs.size();  
  
 this.weights.add(Math.*random*()); // Bias Weight  
 this.weights.add(Math.*random*()); // X 1 Weight  
  
 this.sumX = this.calculateSumX();  
 this.sumY = this.calculateSumY();  
 this.sumXY = this.calculateSumXY();  
 this.sumXSquared = this.calculateSumXSquared();  
  
 this.a = this.calculateA();  
 this.b = this.calculateB();  
  
 this.regressionLine = "y=" + this.a + (this.b >= 0 ? "\*x+" + this.b : "\*x-" + -this.b);  
 this.linearRegressionResult.setRegressionLine(this.regressionLine);  
  
 this.regularizedRegressionLine = this.calculateRegularizedRegression();  
 this.linearRegressionResult.setRegularizedRegressionLine(this.regularizedRegressionLine);  
 } else {  
 throw new Exception("Number of Xs and number of Ys differ, cannot preform linear regression.");  
 }  
 }  
  
 LinearRegressionResult getLinearRegressionResult() {  
 return linearRegressionResult;  
 }  
  
 @Override  
 public String toString() {  
 return "LinearRegression{" +  
 "linearRegressionResult=" + linearRegressionResult +  
 ", Xs=" + this.Xs +  
 ", Ys=" + this.Ys +  
 ", weights=" + this.weights +  
 ", n=" + this.n +  
 ", sumX=" + this.sumX +  
 ", sumY=" + this.sumY +  
 ", sumXY=" + this.sumXY +  
 ", sumXSquared=" + this.sumXSquared +  
 ", a=" + this.a +  
 ", b=" + this.b +  
 ", regressionLine='" + this.regressionLine + '\'' +  
 ", regularizedRegressionLine='" + this.regularizedRegressionLine + '\'' +  
 '}';  
 }  
  
 private int calculateSumX() {  
 int *sumX* = 0;  
  
 for (int *i* = 0; *i* < this.n; *i*++) {  
 *sumX* += this.Xs.get(*i*);  
 }  
  
 return *sumX*;  
 }  
  
 private int calculateSumY() {  
 int *sumY* = 0;  
  
 for (int *i* = 0; *i* < this.n; *i*++) {  
 *sumY* += this.Ys.get(*i*);  
 }  
  
 return *sumY*;  
 }  
  
 private int calculateSumXY() {  
 int *sumXY* = 0;  
  
 for (int *i* = 0; *i* < this.n; *i*++) {  
 *sumXY* += this.Xs.get(*i*) \* this.Ys.get(*i*);  
 }  
  
 return *sumXY*;  
 }  
  
 private int calculateSumXSquared() {  
 int *sumXSquared* = 0;  
  
 for (int *i* = 0; *i* < this.n; *i*++) {  
 *sumXSquared* += Math.*pow*(this.Xs.get(*i*), 2);  
 }  
  
 return *sumXSquared*;  
 }  
  
 private double calculateA() {  
 return (this.n \* this.sumXY - this.sumX \* this.sumY) / (this.n \* this.sumXSquared - Math.*pow*(this.sumX, 2));  
 }  
  
 private double calculateB() {  
 return (this.sumY - this.a \* this.sumX) / this.n;  
 }  
  
 private String calculateRegularizedRegression() {  
 System.*out*.println("Calculating regularized regression line...");  
  
 String *regularizedRegressionLine*;  
  
 double *optimalLambda* = this.calculateOptimalLambdaViaCrossValidation();  
 this.linearRegressionResult.setFinalLambda(*optimalLambda*);  
  
 final double *final\_E\_in* = this.calculate\_E\_aug(this.weights, this.Xs, this.Ys, *optimalLambda*);  
 this.linearRegressionResult.setFinal\_E\_in(*final\_E\_in*);  
  
 *regularizedRegressionLine* = "y=" + (this.a + *optimalLambda*) + (this.b >= 0 ? "\*x+" + (this.b - *optimalLambda*) : "\*x-" + -(this.b - *optimalLambda*));  
 System.*out*.println("Regularized regression line is: \"" + *regularizedRegressionLine* + "\"");  
 return *regularizedRegressionLine*;  
 }  
  
 private double calculateOptimalLambdaViaCrossValidation() {  
 System.*out*.println("--Calculate optimal lambda (min(for each lambda: E\_cv(lambda))...");  
  
 double *smallest\_E\_cv* = Double.*MAX\_VALUE*;  
 double *optimalLambda* = LinearRegression.*LAMBDAS*.get(0);  
  
 for (int *i* = 0; *i* < LinearRegression.*LAMBDAS*.size(); *i*++) {  
 System.*out*.println("----Trying lambda \"" + LinearRegression.*LAMBDAS*.get(*i*) + "\"...");  
  
 double *current\_E\_cv* = this.calculate\_E\_cv(this.weights, LinearRegression.*LAMBDAS*.get(*i*));  
  
 ArrayList<Double> *temporaryResult\_E\_ins* = this.linearRegressionResult.getE\_ins();  
 *temporaryResult\_E\_ins*.add(*i*, this.calculate\_E\_in(this.weights, this.Xs, this.Ys));  
 this.linearRegressionResult.setE\_ins(*temporaryResult\_E\_ins*);  
  
 ArrayList<Double> *temporaryResult\_E\_Cvs* = this.linearRegressionResult.getE\_cvs();  
 *temporaryResult\_E\_Cvs*.add(*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*.get(*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*.get(*i*);  
 } else {  
 System.*out*.println("----This lambda's E\_cv (lambda: \"" + LinearRegression.*LAMBDAS*.get(*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 ArrayList<Double> *weights*, final double *lambda*) {  
 System.*out*.println("------Calculating E\_cv for lambda \"" + *lambda* + "\" ((sum(for each leaveOneOut set: leaveOneOut\_E\_aug(lambda))) / n)...");  
  
 double *E\_cv*;  
 ArrayList<Double> *allLeaveOneOut\_E\_aug* = new ArrayList<>();  
  
 for (int *i* = 0; *i* < this.n; *i*++) {  
 final ArrayList<Double> *leaveOneOutXs* = new ArrayList<>();  
 final ArrayList<Double> *leaveOneOutYs* = new ArrayList<>();  
  
 for (int *j* = 0; *j* < this.n; *j*++) {  
 if (*i* != *j*) {  
 *leaveOneOutXs*.add(this.Xs.get(*j*));  
 *leaveOneOutYs*.add(this.Ys.get(*j*));  
 }  
 }  
  
 *allLeaveOneOut\_E\_aug*.add(calculate\_E\_aug(*weights*, *leaveOneOutXs*, *leaveOneOutYs*, *lambda*));  
 }  
  
 Double *sumLeaveOneOut\_E\_aug* = 0.0;  
 for (final Double *leaveOneOut\_E\_aug* : *allLeaveOneOut\_E\_aug*) {  
 *sumLeaveOneOut\_E\_aug* += *leaveOneOut\_E\_aug*;  
 }  
  
 *E\_cv* = *sumLeaveOneOut\_E\_aug* / this.n;  
 System.*out*.println("------Calculated E\_cv for lambda \"" + *lambda* + "\" to be \"" + *E\_cv* + "\".");  
 return *E\_cv*;  
 }  
  
 private double calculate\_E\_aug(final ArrayList<Double> *weights*, final ArrayList<Double> *Xs*, final ArrayList<Double> *Ys*, final double *lambda*) {  
 System.*out*.println("--------Calculating E\_aug for lambda \"" + *lambda* + "\" (E\_in + lambda \* wTw)...");  
  
 double *E\_aug*;  
 double *wTw* = 0.0;  
  
 for (final double *weight* : *weights*) {  
 *wTw* += Math.*pow*(*weight*, 2);  
 }  
  
 *E\_aug* = this.calculate\_E\_in(*weights*, *Xs*, *Ys*) + *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 ArrayList<Double> *weights*, final ArrayList<Double> *Xs*, final ArrayList<Double> *Ys*) {  
 System.*out*.println("----------Calculating E\_in ((sum((wTx - y)^2))/n)...");  
  
 double *E\_in*;  
 final int *n* = *Xs*.size();  
  
 double *sumE\_in* = 0.0;  
  
 for (int *i* = 0; *i* < *n*; *i*++) {  
 final double *X\_i* = *Xs*.get(*i*);  
 final double *Y\_i* = *Ys*.get(*i*);  
  
 *sumE\_in* += Math.*pow*((this.calculate\_wTx(*weights*, *X\_i*) - *Y\_i*), 2);  
 }  
  
 *E\_in* = *sumE\_in* / *n*;  
 System.*out*.println("----------Calculated E\_in to be \"" + *E\_in* + "\".");  
 return *E\_in*;  
 }  
  
 private double calculate\_wTx(final ArrayList<Double> *weights*, final double *X\_1*) {  
 System.*out*.println("------------Calculating wTx (" + *weights*.get(0) + " \* " + LinearRegression.*X\_0* + " + " + *weights*.get(1) + " \* " + *X\_1* + ")...");  
  
 double *wTx* = *weights*.get(0) \* LinearRegression.*X\_0* + *weights*.get(1) \* *X\_1*;  
 System.*out*.println("------------Calculated wTx to be \"" + *wTx* + "\".");  
 return *wTx*;  
 }  
}  
  
class LinearRegressionResult {  
 private ArrayList<Double> Xs;  
 private ArrayList<Double> Ys;  
 private String regressionLine;  
 private ArrayList<Double> lambdas;  
 private ArrayList<Double> E\_ins = new ArrayList<>();  
 private ArrayList<Double> E\_cvs = new ArrayList<>();  
 private double finalLambda;  
 private String regularizedRegressionLine;  
 private double final\_E\_in;  
  
 LinearRegressionResult() {  
 }  
  
 void setXs(final ArrayList<Double> *xs*) {  
 Xs = *xs*;  
 }  
  
 void setYs(final ArrayList<Double> *ys*) {  
 Ys = *ys*;  
 }  
  
 void setRegressionLine(final String *regressionLine*) {  
 this.regressionLine = *regressionLine*;  
 }  
  
 void setLambdas(final ArrayList<Double> *lambdas*) {  
 this.lambdas = *lambdas*;  
 }  
  
 ArrayList<Double> getE\_ins() {  
 return E\_ins;  
 }  
  
 void setE\_ins(final ArrayList<Double> *E\_ins*) {  
 this.E\_ins = *E\_ins*;  
 }  
  
 ArrayList<Double> getE\_cvs() {  
 return E\_cvs;  
 }  
  
 void setE\_cvs(final ArrayList<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{" +  
 "Xs=" + this.Xs +  
 ", Ys=" + this.Ys +  
 ", regressionLine='" + this.regressionLine + '\'' +  
 ", lambdas=" + this.lambdas +  
 ", E\_ins=" + this.E\_ins +  
 ", E\_cvs=" + 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* < this.Xs.size(); *i*++) {  
 *output*.append("\n • (").append(this.Xs.get(*i*)).append(", ").append(this.Ys.get(*i*)).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.get(0)).append(", ").append(this.E\_ins.get(0)).append(", ").append(this.E\_cvs.get(0)).append(")");  
 *output*.append("\n • (").append(this.lambdas.get(1)).append(", ").append(this.E\_ins.get(1)).append(", ").append(this.E\_cvs.get(1)).append(")");  
 *output*.append("\n • (").append(this.lambdas.get(2)).append(", ").append(this.E\_ins.get(2)).append(", ").append(this.E\_cvs.get(2)).append(")");  
 *output*.append("\n • (").append(this.lambdas.get(3)).append(", ").append(this.E\_ins.get(3)).append(", ").append(this.E\_cvs.get(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();  
 }  
}