Justin Reusnow

Professor Chakraborty

CMP SCI 4340

7 November 2017

Project 3

1. The solution my implementation arrives at (varies each run due to randomly initialized weights) is:
2. Note that the equation predicts the chances of a student failing, not passing, and technically speaking it doesn’t address passing at all, only failing or not failing, so the true answer to these two sub questions would be “N/A”, but I am going to assume passing is equivalent to “not failing”. Thus the probability of a student passing is equal to 1 – p where p is the probability of the student failing after three weeks of inactivity. My implementation outputs: Pr(y = 1.0 | x = 3) = 0.1661, therefore the answer is 1 – 0.1661 = 0.8339.
3. Applying similar thinking from the previous question, we can see that: Pr(y = 1.0 | x = 5) = 0.7235, and thus the answer is 1 – 0.7235 = 0.2765
4. Yes it can. Logistic regression outputs a probability (0.0 <= y <= 1.0) for the entire domain of x. Using this we can set a threshold for classification, and the most neutral of which would be 0.5 Using this threshold we can classify a datum into either “1” or “0” by using 1logistic\_regression(datum)>=0.5.
5. We would still be able to, it would just change the output a bit. Logistic regression outputs a probability, therefore it is normal to find data containing the same X-value and differing Y-values, but if our logistic regression equation is accurate, we should expect to find those which differ to do so proportionally to their expected probability.

Source Code on following pages.

Source Code

//Charles Justin Reusnow • CMP SCI 4340 • Project 3 • 7 November 2017  
  
import java.util.ArrayList;  
  
public class Project3 {  
 public static void main(String[] *args*) throws ImproperlySizedLogisticRegressionDatumException {  
 //Populate the provided data into the Logistic Regression classes.  
 System.*out*.println("Loading data into Logistic Regression...");  
  
 //Datum 1  
 final ArrayList<LogisticRegressionFeature<Integer>> *logisticRegressionFeatures1* = new ArrayList<>();  
 final LogisticRegressionFeature<Integer> *logisticRegressionFeature1\_1* = new LogisticRegressionFeature<>(1);  
 *logisticRegressionFeatures1*.add(*logisticRegressionFeature1\_1*);  
 final LogisticRegressionOutcome *logisticRegressionOutcome1* = new LogisticRegressionOutcome(0);  
 final SingleFeatureLogisticRegressionDatum<Integer> *singleFeatureLogisticRegressionDatum1* = new SingleFeatureLogisticRegressionDatum<>(*logisticRegressionFeatures1*, *logisticRegressionOutcome1*);  
  
 //Datum 2  
 final ArrayList<LogisticRegressionFeature<Integer>> *logisticRegressionFeatures2* = new ArrayList<>();  
 final LogisticRegressionFeature<Integer> *logisticRegressionFeature2\_1* = new LogisticRegressionFeature<>(2);  
 *logisticRegressionFeatures2*.add(*logisticRegressionFeature2\_1*);  
 final LogisticRegressionOutcome *logisticRegressionOutcome2* = new LogisticRegressionOutcome(1);  
 final SingleFeatureLogisticRegressionDatum<Integer> *singleFeatureLogisticRegressionDatum2* = new SingleFeatureLogisticRegressionDatum<>(*logisticRegressionFeatures2*, *logisticRegressionOutcome2*);  
  
 //Datum 3  
 final ArrayList<LogisticRegressionFeature<Integer>> *logisticRegressionFeatures3* = new ArrayList<>();  
 final LogisticRegressionFeature<Integer> *logisticRegressionFeature3\_1* = new LogisticRegressionFeature<>(3);  
 *logisticRegressionFeatures3*.add(*logisticRegressionFeature3\_1*);  
 final LogisticRegressionOutcome *logisticRegressionOutcome3* = new LogisticRegressionOutcome(0);  
 final SingleFeatureLogisticRegressionDatum<Integer> *singleFeatureLogisticRegressionDatum3* = new SingleFeatureLogisticRegressionDatum<>(*logisticRegressionFeatures3*, *logisticRegressionOutcome3*);  
  
 //Datum 4  
 final ArrayList<LogisticRegressionFeature<Integer>> *logisticRegressionFeatures4* = new ArrayList<>();  
 final LogisticRegressionFeature<Integer> *logisticRegressionFeature4\_1* = new LogisticRegressionFeature<>(4);  
 *logisticRegressionFeatures4*.add(*logisticRegressionFeature4\_1*);  
 final LogisticRegressionOutcome *logisticRegressionOutcome4* = new LogisticRegressionOutcome(1);  
 final SingleFeatureLogisticRegressionDatum<Integer> *singleFeatureLogisticRegressionDatum4* = new SingleFeatureLogisticRegressionDatum<>(*logisticRegressionFeatures4*, *logisticRegressionOutcome4*);  
  
 //Datum 5  
 final ArrayList<LogisticRegressionFeature<Integer>> *logisticRegressionFeatures5* = new ArrayList<>();  
 final LogisticRegressionFeature<Integer> *logisticRegressionFeature5\_1* = new LogisticRegressionFeature<>(5);  
 *logisticRegressionFeatures5*.add(*logisticRegressionFeature5\_1*);  
 final LogisticRegressionOutcome *logisticRegressionOutcome5* = new LogisticRegressionOutcome(0);  
 final SingleFeatureLogisticRegressionDatum<Integer> *singleFeatureLogisticRegressionDatum5* = new SingleFeatureLogisticRegressionDatum<>(*logisticRegressionFeatures5*, *logisticRegressionOutcome5*);  
  
 //Datum 6  
 final ArrayList<LogisticRegressionFeature<Integer>> *logisticRegressionFeatures6* = new ArrayList<>();  
 final LogisticRegressionFeature<Integer> *logisticRegressionFeature6\_1* = new LogisticRegressionFeature<>(6);  
 *logisticRegressionFeatures6*.add(*logisticRegressionFeature6\_1*);  
 final LogisticRegressionOutcome *logisticRegressionOutcome6* = new LogisticRegressionOutcome(1);  
 final SingleFeatureLogisticRegressionDatum<Integer> *singleFeatureLogisticRegressionDatum6* = new SingleFeatureLogisticRegressionDatum<>(*logisticRegressionFeatures6*, *logisticRegressionOutcome6*);  
  
 //Datum 7  
 final ArrayList<LogisticRegressionFeature<Integer>> *logisticRegressionFeatures7* = new ArrayList<>();  
 final LogisticRegressionFeature<Integer> *logisticRegressionFeature7\_1* = new LogisticRegressionFeature<>(7);  
 *logisticRegressionFeatures7*.add(*logisticRegressionFeature7\_1*);  
 final LogisticRegressionOutcome *logisticRegressionOutcome7* = new LogisticRegressionOutcome(1);  
 final SingleFeatureLogisticRegressionDatum<Integer> *singleFeatureLogisticRegressionDatum7* = new SingleFeatureLogisticRegressionDatum<>(*logisticRegressionFeatures7*, *logisticRegressionOutcome7*);  
  
 //Datum 8  
 final ArrayList<LogisticRegressionFeature<Integer>> *logisticRegressionFeatures8* = new ArrayList<>();  
 final LogisticRegressionFeature<Integer> *logisticRegressionFeature8\_1* = new LogisticRegressionFeature<>(8);  
 *logisticRegressionFeatures8*.add(*logisticRegressionFeature8\_1*);  
 final LogisticRegressionOutcome *logisticRegressionOutcome8* = new LogisticRegressionOutcome(1);  
 final SingleFeatureLogisticRegressionDatum<Integer> *singleFeatureLogisticRegressionDatum8* = new SingleFeatureLogisticRegressionDatum<>(*logisticRegressionFeatures8*, *logisticRegressionOutcome8*);  
  
 ArrayList<SingleFeatureLogisticRegressionDatum<Integer>> *singleFeatureLogisticRegressionData* = new ArrayList<>();  
 *singleFeatureLogisticRegressionData*.add(*singleFeatureLogisticRegressionDatum1*);  
 *singleFeatureLogisticRegressionData*.add(*singleFeatureLogisticRegressionDatum2*);  
 *singleFeatureLogisticRegressionData*.add(*singleFeatureLogisticRegressionDatum3*);  
 *singleFeatureLogisticRegressionData*.add(*singleFeatureLogisticRegressionDatum4*);  
 *singleFeatureLogisticRegressionData*.add(*singleFeatureLogisticRegressionDatum5*);  
 *singleFeatureLogisticRegressionData*.add(*singleFeatureLogisticRegressionDatum6*);  
 *singleFeatureLogisticRegressionData*.add(*singleFeatureLogisticRegressionDatum7*);  
 *singleFeatureLogisticRegressionData*.add(*singleFeatureLogisticRegressionDatum8*);  
  
 SingleFeatureLogisticRegression<Integer> *singleFeatureLogisticRegression* = new SingleFeatureLogisticRegression<>(*singleFeatureLogisticRegressionData*);  
  
 //Train the Logistic Regression class on the provided data set.  
 System.*out*.println("Training Logistic Regression on Data Set...");  
 *singleFeatureLogisticRegression*.train();  
 System.*out*.println("Training complete!");  
 System.*out*.println("Logistic Regression yeilds the function: Pr(y = 1 | x) = 1/(1 + e^(-" + ((LogisticRegressionWeight) *singleFeatureLogisticRegression*.getLogisticRegressionWeights().get(0)).getValue() + "\*x + " + *singleFeatureLogisticRegression*.getBiasWeight().getValue() + "))");  
  
 //Test the now-trained Logistic Regression class on provided test cases.  
 System.*out*.println("Running tests on Logistic Regression from question a.i. and a.ii. in specification sheet...");  
  
 //Test 1  
 final ArrayList<LogisticRegressionFeature<Integer>> *logisticRegressionFeaturesTest1* = new ArrayList<>();  
 final LogisticRegressionFeature<Integer> *logisticRegressionFeatureTest1\_1* = new LogisticRegressionFeature<>(3);  
 *logisticRegressionFeaturesTest1*.add(*logisticRegressionFeatureTest1\_1*);  
 final LogisticRegressionOutcome *logisticRegressionOutcomeTest1* = new LogisticRegressionOutcome(0);  
 final SingleFeatureLogisticRegressionDatum<Integer> *singleFeatureLogisticRegressionDatumTest1* = new SingleFeatureLogisticRegressionDatum<>(*logisticRegressionFeaturesTest1*, *logisticRegressionOutcomeTest1*);  
  
 System.*out*.println("Test Case 1 - Probability of student failure when that student hadn't done any coursework for 3 weeks:");  
 System.*out*.println("Pr(y = 1.0 | x = 3) = " + *singleFeatureLogisticRegression*.computeProbability(*singleFeatureLogisticRegressionDatumTest1*));  
  
 //Test 2  
 final ArrayList<LogisticRegressionFeature<Integer>> *logisticRegressionFeaturesTest2* = new ArrayList<>();  
 final LogisticRegressionFeature<Integer> *logisticRegressionFeatureTest2\_1* = new LogisticRegressionFeature<>(5);  
 *logisticRegressionFeaturesTest2*.add(*logisticRegressionFeatureTest2\_1*);  
 final LogisticRegressionOutcome *logisticRegressionOutcomeTest2* = new LogisticRegressionOutcome(0);  
 final SingleFeatureLogisticRegressionDatum<Integer> *singleFeatureLogisticRegressionDatumTest2* = new SingleFeatureLogisticRegressionDatum<>(*logisticRegressionFeaturesTest2*, *logisticRegressionOutcomeTest2*);  
  
 System.*out*.println("Test Case 2 - Probability of student failure when that student hadn't done any coursework for 5 weeks:");  
 System.*out*.println("Pr(y = 1.0 | x = 5) = " + *singleFeatureLogisticRegression*.computeProbability(*singleFeatureLogisticRegressionDatumTest2*));  
 }  
}  
  
abstract class LogisticRegression<T extends Number> {  
 private static final Integer *MAXIMUM\_TRAINING\_ITERATIONS* = 1000;  
 private static final Double *MINIMUM\_GRADIENT* = 0.01;  
 private static final Double *LEARNING\_RATE* = 0.1;  
 private static final Double *INITIAL\_BIAS\_WEIGHT\_VALUE* = Math.*random*();  
  
 private final ArrayList<LogisticRegressionDatum<T>> logisticRegressionData;  
 private final ArrayList<LogisticRegressionWeight> logisticRegressionWeights;  
 private final LogisticRegressionWeight biasWeight = new LogisticRegressionWeight(LogisticRegression.*INITIAL\_BIAS\_WEIGHT\_VALUE*);  
  
 private static Double sigmoidFunction(final Double *s*) {  
 //The driver of Logistic Regression, this is the Theta function: "1 / (1 + e^-s)"  
 return 1.0 / (1.0 + Math.*exp*(-*s*));  
 }  
  
 LogisticRegression(final ArrayList<LogisticRegressionDatum<T>> *logisticRegressionData*) {  
 this.logisticRegressionData = *logisticRegressionData*;  
 this.logisticRegressionWeights = new ArrayList<>();  
  
 for (int *i* = 0; *i* < this.getLogisticRegressionData().get(0).getLogisticRegressionFeatures().size(); *i*++) {  
 this.logisticRegressionWeights.add(new LogisticRegressionWeight());  
 }  
 }  
  
 private ArrayList<LogisticRegressionDatum<T>> getLogisticRegressionData() {  
 return this.logisticRegressionData;  
 }  
  
 public ArrayList<LogisticRegressionWeight> getLogisticRegressionWeights() {  
 return this.logisticRegressionWeights;  
 }  
  
 public LogisticRegressionWeight getBiasWeight() {  
 return this.biasWeight;  
 }  
  
 @Override()  
 public String toString() {  
 return "{\"logisticRegressionData\": " + this.getLogisticRegressionData().toString() + ", \"logisticRegressionWeights\": " + this.getLogisticRegressionWeights().toString() + ", \"biasWeight\": " + this.getBiasWeight().toString() + "}";  
 }  
  
 public void train() {  
 Double *gradient* = Double.*MAX\_VALUE*;  
  
 //Reset all the weights anytime we try to train.  
 this.getBiasWeight().setValue(LogisticRegression.*INITIAL\_BIAS\_WEIGHT\_VALUE*);  
 for (LogisticRegressionWeight *logisticRegressionWeight* : this.getLogisticRegressionWeights()) {  
 *logisticRegressionWeight*.setValue(LogisticRegressionWeight.*generateRandomWeight*());  
 }  
  
 int *trainingIterations* = 0;  
 while (*trainingIterations* < LogisticRegression.*MAXIMUM\_TRAINING\_ITERATIONS* && Math.*abs*(*gradient*) > LogisticRegression.*MINIMUM\_GRADIENT*) {  
 //Compute the gradient on the current state.  
 *gradient* = computeGradientForOneTrainingIteration();  
 Double *directionToMove* = -1 \* *gradient*;  
  
 //Update the bias weight.  
 this.getBiasWeight().setValue(this.getBiasWeight().getValue() + *LEARNING\_RATE* \* *directionToMove*);  
  
 //Update every other weight.  
 for (LogisticRegressionDatum<T> *logisticRegressionDatum* : this.getLogisticRegressionData()) {  
 LogisticRegressionWeight *weight* = this.getLogisticRegressionWeights().get(0);  
 Integer *y* = *logisticRegressionDatum*.getLogisticRegressionOutcome().getValueAsInt();  
 Double *prediction* = computeProbability(*logisticRegressionDatum*);  
 Integer *x* = *logisticRegressionDatum*.getLogisticRegressionFeatures().get(0).getValue().intValue();  
  
 *weight*.setValue(*weight*.getValue() + *LEARNING\_RATE* \* (*y* - *prediction*) \* *x*);  
 }  
  
 *trainingIterations*++;  
 }  
 }  
  
 private Double computeGradientForOneTrainingIteration() {  
 Double *gradient* = 0.0;  
  
 //For every datum in the data set, calculate the gradient based on: -1 \* Sigma(n = 0; n < N; (y\_n \* x\_n) / (1 + e^(y\_n \* wT \* x\_n))) / N (Source: Logistic Regression Algorithm, page 95 of textbook)  
 for (int *n* = 0; *n* < this.getLogisticRegressionData().size(); *n*++) {  
 Integer *y* = this.getLogisticRegressionData().get(*n*).getLogisticRegressionOutcome().getValueAsInt();  
 Integer *x* = this.getLogisticRegressionData().get(*n*).getLogisticRegressionFeatures().get(0).getValue().intValue();  
 Double *wT* = this.getLogisticRegressionWeights().get(0).getValue();  
  
 *gradient* += *y* \* *x* / (1 + Math.*exp*(*y* \* *wT* \* *x*));  
 }  
 *gradient* = -1 \* *gradient* / this.getLogisticRegressionData().size();  
  
 return *gradient*;  
 }  
  
 public Double computeProbability(final LogisticRegressionDatum<T> *logisticRegressionDatum*) {  
 //The Theta function accepts "1 / 1 + e^-(a \* x + b)", and this function builds "a \* x + b" and passes it to the sigmoid function.  
  
 //Initialize s to the bias weight to handle "b".  
 Double *s* = -this.getBiasWeight().getValue();  
  
 //Add the value "a \* x: to s.  
 *s* += this.getLogisticRegressionWeights().get(0).getValue() \* *logisticRegressionDatum*.getLogisticRegressionFeatures().get(0).getValue().intValue();  
  
 return LogisticRegression.*sigmoidFunction*(*s*);  
 }  
}  
  
class SingleFeatureLogisticRegression<T extends Number> extends LogisticRegression {  
 SingleFeatureLogisticRegression(final ArrayList<SingleFeatureLogisticRegressionDatum<T>> *singleFeatureLogisticRegressionData*) {  
 super(*singleFeatureLogisticRegressionData*);  
 }  
}  
  
interface VaryingQuantityFeatures {  
 Integer getAnticipatedQuantityFeatures();  
  
 boolean containsAnticipatedQuantityFeatures();  
}  
  
abstract class LogisticRegressionDatum<T extends Number> implements VaryingQuantityFeatures {  
 private final ArrayList<LogisticRegressionFeature<T>> logisticRegressionFeatures;  
 private final LogisticRegressionOutcome logisticRegressionOutcome;  
  
 LogisticRegressionDatum(final ArrayList<LogisticRegressionFeature<T>> *logisticRegressionFeatures*, final LogisticRegressionOutcome *logisticRegressionOutcome*) {  
 this.logisticRegressionFeatures = *logisticRegressionFeatures*;  
 this.logisticRegressionOutcome = *logisticRegressionOutcome*;  
 }  
  
 public ArrayList<LogisticRegressionFeature<T>> getLogisticRegressionFeatures() {  
 return this.logisticRegressionFeatures;  
 }  
  
 public LogisticRegressionOutcome getLogisticRegressionOutcome() {  
 return this.logisticRegressionOutcome;  
 }  
  
 @Override()  
 public String toString() {  
 return "{\"logisticRegressionFeatures\": " + this.getLogisticRegressionFeatures().toString() + ", \"logisticRegressionOutcome\": " + this.getLogisticRegressionOutcome().toString() + "}";  
 }  
}  
  
class SingleFeatureLogisticRegressionDatum<T extends Number> extends LogisticRegressionDatum implements VaryingQuantityFeatures {  
 private static final Integer *ANTICIPATED\_QUANTITY\_FEATURES* = 1;  
  
 SingleFeatureLogisticRegressionDatum(final ArrayList<LogisticRegressionFeature<T>> *logisticRegressionFeatures*, final LogisticRegressionOutcome *logisticRegressionOutcome*) throws ImproperlySizedLogisticRegressionDatumException {  
 super(*logisticRegressionFeatures*, *logisticRegressionOutcome*);  
  
 if (!this.containsAnticipatedQuantityFeatures()) {  
 throw new ImproperlySizedLogisticRegressionDatumException(this);  
 }  
 }  
  
 @Override()  
 public Integer getAnticipatedQuantityFeatures() {  
 return SingleFeatureLogisticRegressionDatum.*ANTICIPATED\_QUANTITY\_FEATURES*;  
 }  
  
 @Override()  
 public boolean containsAnticipatedQuantityFeatures() {  
 return this.getLogisticRegressionFeatures().size() == SingleFeatureLogisticRegressionDatum.*ANTICIPATED\_QUANTITY\_FEATURES*;  
 }  
}  
  
class ImproperlySizedLogisticRegressionDatumException extends Exception {  
 ImproperlySizedLogisticRegressionDatumException(final LogisticRegressionDatum *logisticRegressionDatum*) {  
 super("Improperly sized LogisticRegressionDatum. Expected size \"" + *logisticRegressionDatum*.getAnticipatedQuantityFeatures() + "\", encountered size: \"" + *logisticRegressionDatum*.getLogisticRegressionFeatures().size() + "\", for LogisticRegressionDatum with data \"" + *logisticRegressionDatum*.toString() + "\".");  
 }  
}  
  
class LogisticRegressionWeight {  
 private Double value;  
  
 public static Double generateRandomWeight() {  
 return Math.*random*();  
 }  
  
 LogisticRegressionWeight() {  
 this(LogisticRegressionWeight.*generateRandomWeight*());  
 }  
  
 LogisticRegressionWeight(final Double *value*) {  
 this.value = *value*;  
 }  
  
 public Double getValue() {  
 return this.value;  
 }  
  
 public void setValue(final Double *value*) {  
 this.value = *value*;  
 }  
  
 @Override()  
 public String toString() {  
 return "{\"value\": " + this.getValue().toString() + "}";  
 }  
}  
  
class LogisticRegressionFeature<T extends Number> {  
 private final T value;  
  
 LogisticRegressionFeature(final T *value*) {  
 this.value = *value*;  
 }  
  
 public T getValue() {  
 return this.value;  
 }  
  
 @Override()  
 public String toString() {  
 return "{\"value\": " + this.getValue().toString() + "}";  
 }  
}  
  
class LogisticRegressionOutcome {  
 private final Boolean value;  
  
 LogisticRegressionOutcome(final Boolean *value*) {  
 this.value = *value*;  
 }  
  
 LogisticRegressionOutcome(final Integer *value*) {  
 this.value = *value* != 0;  
 }  
  
 public Boolean getValue() {  
 return this.value;  
 }  
  
 public Integer getValueAsInt() {  
 return this.value ? 1 : 0;  
 }  
  
 @Override()  
 public String toString() {  
 return "{\"value\": " + this.getValueAsInt().toString() + "}";  
 }  
}