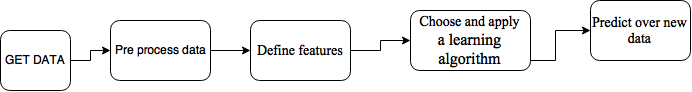
**FLIGHT DELAY PREDICTION DOCUMENTATION**

*Can an app be created to predict flight delays due to weather related issues?*

The following paper tries to model solutions to predict flight delays due to weather related issues. The proposed models for flight delay prediction uses supervised machine learning techniques. The goal of this research is to be able to predict departure delays 2-24 hrs in advance. The paper analyzes the performance of the following algorithms on the flight delay prediction model namely – Simple Logistic Regression, Naive Bayyes classification and Support Vector Machines. The end goal is to choose the best performing algorithm and implement it to predict the delay of flights in advance. The objective of this research is to answer the question mentioned above with concrete results and explanation of the chosen solution.

The working model of the paper is explained with the aid of the following diagram



Step 1:- Get Data

The dataset for this model was collected from the following websites for the month (April – June) for the year 2013.

1. http://gallery.cortanaintelligence.com/Experiment/Binary-Classification-Flight-delay-prediction-3?share=1.

2. https://www.ncdc.noaa.gov/cdo-web/datasets.

3. http://www.transtats.bts.gov/DL\_SelectFields.asp?Table\_ID=236&DB\_Short\_Name=On-Time.

4. https://www.wunderground.com/.

In order to maintain the accuracy of the dataset for the flight and weather, the datasets were cross checked for accuracy by validating the information from the above mentioned websites.

Step 2:- Pre-Process Data

The dataset obtained from the above sources was combined and a fresh dataset was created. The initial dataset contained 18 features combined of (flight + weather). We have chosen to work on a small cluster of data to optimize accuracy and get a better feel of the performance of the algorithm. Since, our main objective is to apply machine learning and the saying goes “There is no fixed size of dataset for better performance in Machine learning. The more, the merrier”. So, our initial goal is to work on 344 datapoints and analyze the results. We can always add more data in our dataset in case of redundancy or conflicting results.

To select the most useful features for the prediction model, we used the Weka software.

* Check for Missing values in the database using the Weka Software.

To check for missing values in the dataset, the dataset is loaded in the Weka software. Then each column is checked in the Weka to identify any missing values. The dataset used in the current experiment doesn’t contain any missing values.

Step 3:- Get Features

* Identify the most important features for the classification algorithms using Weka.

**Algorithm 1:- Simple Logistic Regression**

Available Features:-

1. Month

2. Day

3. Time

4. Timegroup

5. Airportid

6. Temperature (temperature of the origin airport).

7. DewPoint (of the origin airport)

8. humidity (of the origin airport)

9. Pressure (of the origin airport)

10. WindSpeed (of the origin airport)

11. destination airport ID

12. desttemperature (of the destination airport)

13. destdewpoint (of the destination airport)

14. destinationPressure (of the destination airport)

15. destwindspeed (of the destination airport)

16. Prediction (Yes/No).

Our first algorithm that we wanted to experiment was Simple Logistic Regression classification. We wanted to Weka to help us select the best features available for the Simple Logistic Regression Classification Model. Weka returned us with the following results.

=== Run information ===

Evaluator: weka.attributeSelection.ClassifierSubsetEval -B weka.classifiers.functions.SimpleLogistic -T -H "Click to set hold out or test instances" -E acc -- -I 0 -M 500 -H 50 -W 0.0

Search: weka.attributeSelection.BestFirst -D 1 -N 5

Relation: r-weka.filters.unsupervised.attribute.Remove-R14-weka.filters.unsupervised.attribute.Remove-R10-weka.filters.unsupervised.attribute.Remove-R11

Instances: 344

Attributes: 16

Month

Day

Time

Timegroup

Airportid

Temperature

DewPoint

humidity

Pressure

WindSpeed

destination airport ID

desttemperature

destdewpoint

destinationPressure

destwindspeed

Prediction

Evaluation mode: evaluate on all training data

=== Attribute Selection on all input data ===

Search Method:

Best first.

Start set: no attributes

Search direction: forward

Stale search after 5 node expansions

Total number of subsets evaluated: 127

Merit of best subset found: 0.299

Attribute Subset Evaluator (supervised, Class (nominal): 16 Prediction):

Classifier Subset Evaluator

Learning scheme: weka.classifiers.functions.SimpleLogistic

Scheme options: -I 0 -M 500 -H 50 -W 0.0

Hold out/test set: Training data

Subset evaluation: classification error

Selected attributes: 1,3,4,7,8,13,14 : 7

Month

Time

Timegroup

DewPoint

humidity

destdewpoint

destinationPressure

The next step was to run the Simple Logistic Regression on the selected attributes as suggested by Weka. We ran the Simple Logistic Regression on the whole dataset first and then wanted to compare the results with the Simple Logistic Regression on the selected attributes of the dataset.

Results are displayed as below:-

1. Whole dataset (70% training set; 30% test set; 16 features)

=== Run information ===

Scheme: weka.classifiers.functions.SimpleLogistic -I 0 -M 500 -H 50 -W 0.0

Relation: r-weka.filters.unsupervised.attribute.Remove-R14-weka.filters.unsupervised.attribute.Remove-R10-weka.filters.unsupervised.attribute.Remove-R11

Instances: 344

Attributes: 16

Month

Day

Time

Timegroup

Airportid

Temperature

DewPoint

humidity

Pressure

WindSpeed

destination airport ID

desttemperature

destdewpoint

destinationPressure

destwindspeed

Prediction

Test mode: split 70.0% train, remainder test

=== Classifier model (full training set) ===

SimpleLogistic:

Class 0 :

0.84 +

[Timegroup] \* -0.46

Class 1 :

-0.84 +

[Timegroup] \* 0.46

Time taken to build model: 0.19 seconds

=== Evaluation on test split ===

Time taken to test model on training split: 0 seconds

=== Summary ===

Correctly Classified Instances 63 61.165 %

Incorrectly Classified Instances 40 38.835 %

Kappa statistic 0.2429

Mean absolute error 0.4498

Root mean squared error 0.4802

Relative absolute error 88.9816 %

Root relative squared error 94.5167 %

Total Number of Instances 103

=== Detailed Accuracy By Class ===

TP Rate FP Rate Precision Recall F-Measure MCC ROC Area PRC Area Class

0.761 0.509 0.547 0.761 0.636 0.258 0.733 0.701 no

0.491 0.239 0.718 0.491 0.583 0.258 0.733 0.697 yes

Weighted Avg. 0.612 0.360 0.642 0.612 0.607 0.258 0.733 0.699

=== Confusion Matrix ===

a b <-- classified as

35 11 | a = no

29 28 | b = yes

2. Selected attributes (70% training set; 30% test set; 7 features)

=== Run information ===

Scheme: weka.classifiers.functions.SimpleLogistic -I 0 -M 500 -H 50 -W 0.0

Relation: r-weka.filters.unsupervised.attribute.Remove-R14-weka.filters.unsupervised.attribute.Remove-R10-weka.filters.unsupervised.attribute.Remove-R11-weka.filters.unsupervised.attribute.Remove-R2,5-6,9-12,15

Instances: 344

Attributes: 8

Month

Time

Timegroup

DewPoint

humidity

destdewpoint

destinationPressure

Prediction

Test mode: split 70.0% train, remainder test

=== Classifier model (full training set) ===

SimpleLogistic:

Class 0 :

3.04 +

[Month] \* -0.11 +

[Time] \* 0 +

[Timegroup] \* -0.58 +

[DewPoint] \* -0 +

[humidity] \* 0.12 +

[destdewpoint] \* -0.02 +

[ destinationPressure] \* -0.1

Class 1 :

-3.04 +

[Month] \* 0.11 +

[Time] \* -0 +

[Timegroup] \* 0.58 +

[DewPoint] \* 0 +

[humidity] \* -0.12 +

[destdewpoint] \* 0.02 +

[ destinationPressure] \* 0.1

Time taken to build model: 0.14 seconds

=== Evaluation on test split ===

Time taken to test model on training split: 0 seconds

=== Summary ===

Correctly Classified Instances 69 66.9903 %

Incorrectly Classified Instances 34 33.0097 %

Kappa statistic 0.3512

Mean absolute error 0.4598

Root mean squared error 0.4846

Relative absolute error 90.9621 %

Root relative squared error 95.3704 %

Total Number of Instances 103

=== Detailed Accuracy By Class ===

TP Rate FP Rate Precision Recall F-Measure MCC ROC Area PRC Area Class

0.783 0.421 0.600 0.783 0.679 0.364 0.723 0.725 no

0.579 0.217 0.767 0.579 0.660 0.364 0.723 0.673 yes

Weighted Avg. 0.670 0.308 0.693 0.670 0.669 0.364 0.723 0.696

=== Confusion Matrix ===

a b <-- classified as

36 10 | a = no

24 33 | b = yes

So, comparing the results from the classification model, we can see that using the features selected by Weka gives us a ~5% boost in the prediction accuracy. We shall be using the 7 features as suggested by Weka for the Simple Logistic Regression classification program which will be mentioned later.

**Algorithm 2:- Naive Bayyes Classification**

Our second proposed classification algorithm is the Naive Bayyes algorithm. As mentioned above, we wanted to get the best features for our Naive Bayyes model and Weka provided us with the following suggestions.

=== Run information ===

Evaluator: weka.attributeSelection.ClassifierSubsetEval -B weka.classifiers.bayes.NaiveBayes -T -H "Click to set hold out or test instances" -E acc

Search: weka.attributeSelection.BestFirst -D 1 -N 5

Relation: r-weka.filters.unsupervised.attribute.Remove-R14-weka.filters.unsupervised.attribute.Remove-R10-weka.filters.unsupervised.attribute.Remove-R11-weka.filters.unsupervised.attribute.Remove-R2,5-6,9-12,15

Instances: 344

Attributes: 8

Month

Time

Timegroup

DewPoint

humidity

destdewpoint

destinationPressure

Prediction

Evaluation mode: evaluate on all training data

=== Attribute Selection on all input data ===

Search Method:

Best first.

Start set: no attributes

Search direction: forward

Stale search after 5 node expansions

Total number of subsets evaluated: 43

Merit of best subset found: 0.305

Attribute Subset Evaluator (supervised, Class (nominal): 8 Prediction):

Classifier Subset Evaluator

Learning scheme: weka.classifiers.bayes.NaiveBayes

Scheme options:

Hold out/test set: Training data

Subset evaluation: classification error

Selected attributes: 3,4,6 : 3

Timegroup

DewPoint

Destdewpoint

So, our chosen attributes for the Naive Bayyes Classification Model are Timegroup,DewPoint and Destdewpoint. We wanted to run a test on the Naive Bayyes classification model on the selected attributes. The results are mentioned below.

-----------------------------------------------------------------------------------------------------

=== Run information ===

Scheme: weka.classifiers.bayes.NaiveBayes

Relation: r-weka.filters.unsupervised.attribute.Remove-R14-weka.filters.unsupervised.attribute.Remove-R10-weka.filters.unsupervised.attribute.Remove-R11-weka.filters.unsupervised.attribute.Remove-R2,5-6,9-12,15-weka.filters.unsupervised.attribute.Remove-R1-2,5,7

Instances: 344

Attributes: 4

Timegroup

DewPoint

destdewpoint

Prediction

Test mode: split 70.0% train, remainder test

=== Classifier model (full training set) ===

Naive Bayes Classifier

Class

Attribute no yes

(0.52) (0.48)

===============================

Timegroup

mean 1.5363 1.9394

std. dev. 0.6789 0.5689

weight sum 179 165

precision 1 1

DewPoint

mean -5.8049 -5.4939

std. dev. 6.1858 6.314

weight sum 179 165

precision 0.6167 0.6167

destdewpoint

mean 13.1389 14.8783

std. dev. 7.6673 8.2039

weight sum 179 165

precision 0.678 0.678

Time taken to build model: 0 seconds

=== Evaluation on test split ===

Time taken to test model on training split: 0 seconds

=== Summary ===

Correctly Classified Instances 70 67.9612 %

Incorrectly Classified Instances 33 32.0388 %

Kappa statistic 0.3664

Mean absolute error 0.4594

Root mean squared error 0.4733

Relative absolute error 90.8908 %

Root relative squared error 93.1475 %

Total Number of Instances 103

=== Detailed Accuracy By Class ===

TP Rate FP Rate Precision Recall F-Measure MCC ROC Area PRC Area Class

0.761 0.386 0.614 0.761 0.680 0.375 0.719 0.730 no

0.614 0.239 0.761 0.614 0.680 0.375 0.719 0.670 yes

Weighted Avg. 0.680 0.305 0.695 0.680 0.680 0.375 0.719 0.696

=== Confusion Matrix ===

a b <-- classified as

35 11 | a = no

22 35 | b = yes

**Algorithm 3:- Support Vector Machine Classification**

Our third proposed classification algorithm is the Support Vector Machine classification algorithm. We ran our dataset through Weka and wanted to come up with the best available features for our Support Vector Machine Classification Model.

Results are displayed as below:-

=== Run information ===

Evaluator: weka.attributeSelection.ClassifierSubsetEval -B weka.classifiers.functions.SMO -T -H "Click to set hold out or test instances" -E acc -- -C 1.0 -L 0.001 -P 1.0E-12 -N 0 -V -1 -W 1 -K "weka.classifiers.functions.supportVector.PolyKernel -E 1.0 -C 250007" -calibrator "weka.classifiers.functions.Logistic -R 1.0E-8 -M -1 -num-decimal-places 4"

Search: weka.attributeSelection.BestFirst -D 1 -N 5

Relation: r-weka.filters.unsupervised.attribute.Remove-R14-weka.filters.unsupervised.attribute.Remove-R10-weka.filters.unsupervised.attribute.Remove-R11-weka.filters.unsupervised.attribute.Remove-R2,5-6,9-12,15

Instances: 344

Attributes: 8

Month

Time

Timegroup

DewPoint

humidity

destdewpoint

destinationPressure

Prediction

Evaluation mode: evaluate on all training data

=== Attribute Selection on all input data ===

Search Method:

Best first.

Start set: no attributes

Search direction: forward

Stale search after 5 node expansions

Total number of subsets evaluated: 31

Merit of best subset found: 0.308

Attribute Subset Evaluator (supervised, Class (nominal): 8 Prediction):

Classifier Subset Evaluator

Learning scheme: weka.classifiers.functions.SMO

Scheme options: -C 1.0 -L 0.001 -P 1.0E-12 -N 0 -V -1 -W 1 -K weka.classifiers.functions.supportVector.PolyKernel -E 1.0 -C 250007 -calibrator weka.classifiers.functions.Logistic -R 1.0E-8 -M -1 -num-decimal-places 4

Hold out/test set: Training data

Subset evaluation: classification error

Selected attributes: 1,2,3,5,6,7 : 6

Month

Time

Timegroup

humidity

destdewpoint

destinationPressure

The next step was to run the Support Vector Classifier on the dataset based on the selected attributes. The results are displayed as below:-

=== Run information ===

Scheme: weka.classifiers.functions.SMO -C 1.0 -L 0.001 -P 1.0E-12 -N 0 -V -1 -W 1 -K "weka.classifiers.functions.supportVector.PolyKernel -E 1.0 -C 250007" -calibrator "weka.classifiers.functions.Logistic -R 1.0E-8 -M -1 -num-decimal-places 4"

Relation: r-weka.filters.unsupervised.attribute.Remove-R14-weka.filters.unsupervised.attribute.Remove-R10-weka.filters.unsupervised.attribute.Remove-R11-weka.filters.unsupervised.attribute.Remove-R2,5-6,9-12,15-weka.filters.unsupervised.attribute.Remove-R4

Instances: 344

Attributes: 7

Month

Time

Timegroup

humidity

destdewpoint

destinationPressure

Prediction

Test mode: split 70.0% train, remainder test

=== Classifier model (full training set) ===

SMO

Kernel used:

Linear Kernel: K(x,y) = <x,y>

Classifier for classes: no, yes

BinarySMO

Machine linear: showing attribute weights, not support vectors.

0.3218 \* (normalized) Month

+ 0.0813 \* (normalized) Time

+ 2.8137 \* (normalized) Timegroup

+ -0.1539 \* (normalized) humidity

+ 1.1437 \* (normalized) destdewpoint

+ 0.9435 \* (normalized) destinationPressure

- 2.6875

Number of kernel evaluations: 10243 (73.184% cached)

Time taken to build model: 0.03 seconds

=== Evaluation on test split ===

Time taken to test model on training split: 0 seconds

=== Summary ===

Correctly Classified Instances 69 66.9903 %

Incorrectly Classified Instances 34 33.0097 %

Kappa statistic 0.3405

Mean absolute error 0.3301

Root mean squared error 0.5745

Relative absolute error 65.3019 %

Root relative squared error 113.082 %

Total Number of Instances 103

=== Detailed Accuracy By Class ===

TP Rate FP Rate Precision Recall F-Measure MCC ROC Area PRC Area Class

0.696 0.351 0.615 0.696 0.653 0.343 0.672 0.564 no

0.649 0.304 0.725 0.649 0.685 0.343 0.672 0.665 yes

Weighted Avg. 0.670 0.325 0.676 0.670 0.671 0.343 0.672 0.620

=== Confusion Matrix ===

a b <-- classified as

32 14 | a = no

20 37 | b = yes

Step 4:- Choose and Apply a Learning Algorithm

The results of the classification algorithms are mentioned below. Each of the algorithms were run on 70% training set and 30% test set in Weka.

|  |  |  |  |
| --- | --- | --- | --- |
| Algorithm | Accuracy (%) | Error (%) | Completion Time |
| Simple Logistic Regression | 66.9903 | 32.0388 | 0.14 seconds |
| Naive Bayyes | 67.9612 | 32.0388 | 0 seconds |
| Support Vector Machines | 66.9903 | 33.0097 | 0.03 seconds |

The above results suggest that Naive Bayyes is our best bet but we wanted to have an open mind and create a prediction program for all the three algorithms. Our preferred programming language is Java and we are building a simple classifier program for the above algorithms.

We chose java and decided to come up with desktop software for now. Java has a write once, run everywhere policy. This makes our program platform independent. Our preferred IDE is eclipse.

The code for each of the above algorithms is in a separate folder in the GITHUB account of the team and also is attached below at the end of the paper.

Conclusion:-

The aim of this research was to answer the question if an app can be created to predict flight delays in advance based on weather related issues. Based on our above research, we can firmly say that Yes, an app can be created to predict flight delays based on weather data of the past. We applied Machine learning and data mining for this project. Machine learning is one of the fastest growing areas of research in our modern era and is being implemented by some of the top notch companies namely- Google, Microsoft, Facebook, NASA.

We wanted to go a step further and come up with a way to estimate the time of delay along with the prediction (yes/no). To achieve that we first identify if a flight is likely to be delayed based on the weather data. After that we wanted to apply K-means clustering to group together data of the similar type of that specific weather data. We look at the amount of time in departure delays in the dataset of the previous data. Then we take the mean of the departure delay times of the group formed by K-means clustering and suggest it as an approximate time of delay for that particular date.

**Java code for Naive Bayes Classifier algorithm**

Jar files required – weka.jar

**import** java.io.FileNotFoundException;

**import** java.io.FileReader;

**import** java.io.BufferedReader;

**import** java.io.IOException;

**import** java.util.Random;

**import** weka.classifiers.evaluation.\*;

**import** weka.classifiers.bayes.NaiveBayes;

**import** weka.classifiers.bayes.net.\*;

**import** weka.core.Instance;

**import** weka.core.Instances;

**import** weka.core.converters.ConverterUtils;

**public** **class** JavaWeka {

**public** **static** **void** main(String[] args) {

// **TODO** Auto-generated method stub

ConverterUtils.DataSource source1;

**try** {

source1 = **new** ConverterUtils.DataSource("traindata.arff");

Instances train = source1.getDataSet();

// setting class attribute if the data format does not provide this information

// For example, the XRFF format saves the class attribute information as well

**if** (train.classIndex() == -1)

train.setClassIndex(train.numAttributes() - 1);

ConverterUtils.DataSource source2 = **new** ConverterUtils.DataSource("testdata.arff");

Instances test = source2.getDataSet();

// setting class attribute if the data format does not provide this information

// For example, the XRFF format saves the class attribute information as well

**if** (test.classIndex() == -1)

test.setClassIndex(train.numAttributes() - 1);

NaiveBayes naiveBayes = **new** NaiveBayes();

naiveBayes.buildClassifier(train);

Evaluation eval = **new** Evaluation(train);

eval.evaluateModel(naiveBayes,test);

System.***out***.println(eval.toSummaryString("\nResults\n####\n",**true**));

System.***out***.println(eval.fMeasure(1)+ "" + eval.precision(1)+ "" +eval.recall(1));

} **catch** (Exception e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

}

}

Output-

Results

####

Correctly Classified Instances 65 62.5 %

Incorrectly Classified Instances 39 37.5 %

Kappa statistic 0.245

K&B Relative Info Score 1013.5341 %

K&B Information Score 10.1229 bits 0.0973 bits/instance

Class complexity | order 0 103.8896 bits 0.9989 bits/instance

Class complexity | scheme 101.6568 bits 0.9775 bits/instance

Complexity improvement (Sf) 2.2328 bits 0.0215 bits/instance

Mean absolute error 0.4585

Root mean squared error 0.4909

Relative absolute error 91.8479 %

Root relative squared error 98.2601 %

Total Number of Instances 104

0.58064516129032260.6279069767441860.54

**Java code for Support Vector Machine**

Jar files required – JAVAML, WEKA, LibSVM

**import** java.io.File;

**import** java.io.IOException;

**import** libsvm.LibSVM;

**import** net.sf.javaml.classification.Classifier;

**import** net.sf.javaml.core.Dataset;

**import** net.sf.javaml.core.Instance;

**import** net.sf.javaml.tools.data.FileHandler;

**public** **class** SVMweka {

**public** **static** **void** main(String[] args) **throws** IOException {

// **TODO** Auto-generated method stub

Dataset data = FileHandler.*loadDataset*(**new** File("traindata.csv"),3,",");

/\*

\* Contruct a LibSVM classifier with default settings.

\*/

Classifier svm = **new** LibSVM();

svm.buildClassifier(data);

/\*

\* Load a data set, this can be a different one, but we will use the

\* same one.

\*/

Dataset dataForClassification = FileHandler.*loadDataset*(**new** File("testdata.csv"),3,",");

/\* Counters for correct and wrong predictions. \*/

**int** correct = 0, wrong = 0;

/\* Classify all instances and check with the correct class values \*/

**for** (Instance inst : dataForClassification) {

Object predictedClassValue = svm.classify(inst);

Object realClassValue = inst.classValue();

**if** (predictedClassValue.equals(realClassValue))

correct++;

**else**

wrong++;

}

System.***out***.println("Correct predictions " + correct);

System.***out***.println("Wrong predictions " + wrong);

}

}

Output:-

Correct predictions 72

Wrong predictions 33

**Java code for Simple Logistic Regression**

import java.io.BufferedReader;

import java.io.File;

import java.io.FileNotFoundException;

import java.io.FileReader;

import java.io.IOException;

import java.util.ArrayList;

import java.util.Arrays;

import java.util.List;

import java.util.Scanner;

/\*\*

public class SLR {

/\*\* the learning rate \*/

private double rate;

/\*\* the weight to learn \*/

private double[] weights;

/\*\* the number of iterations \*/

private int ITERATIONS = 3000;

public SLR(int n) {

this.rate = 0.0001;

weights = new double[n];

}

private static double sigmoid(double z) {

return 1.0 / (1.0 + Math.exp(-z));

}

public void train(List<Instance> instances) {

for (int n=0; n<ITERATIONS; n++) {

double lik = 0.0;

for (int i=0; i<instances.size(); i++) {

double[] x = instances.get(i).x;

double predicted = classify(x);

int label = instances.get(i).label;

for (int j=0; j<weights.length; j++) {

weights[j] = weights[j] + rate \* (label - predicted) \* x[j];

}

// not necessary for learning

lik += label \* Math.log(classify(x)) + (1-label) \* Math.log(1- classify(x));

}

System.out.println("iteration: " + n + " " + Arrays.toString(weights) + " mle: " + lik);

}

}

private double classify(double[] x) {

double logit = .0;

for (int i=0; i<weights.length;i++) {

logit += weights[i] \* x[i];

}

return sigmoid(logit);

}

public static class Instance {

public int label;

public double[] x;

public Instance(int label, double[] x) {

this.label = label;

this.x = x;

}

}

public static List<Instance> readDataSet(String file) throws FileNotFoundException {

List<Instance> dataset = new ArrayList<Instance>();

Scanner scanner = null;

try {

scanner = new Scanner(new File(file));

while(scanner.hasNextLine()) {

String line = scanner.nextLine();

String[] columns = line.split("\\s+");

// skip first column and last column is the label

int i = 1;

double[] data = new double[columns.length-2];

for (i=1; i<columns.length-2; i++) {

data[i-1] = Double.parseDouble(columns[i]);

}

int label = Integer.parseInt(columns[i]);

Instance instance = new Instance(label, data);

dataset.add(instance);

}

} finally {

if (scanner != null)

scanner.close();

}

return dataset;

}

public static void main(String... args) throws FileNotFoundException {

List<Instance> instances = readDataSet("editedata.txt");

SLR logistic = new SLR(3);

logistic.train(instances);

String month,day,time,origin,destination;

Scanner input = new Scanner(System.in);

System.out.println("Enter the month of travel");

month = input.nextLine();

System.out.println("Enter the day of travel");

day = input.nextLine();

System.out.println("Enter the time of travel");

time = input.nextLine();

System.out.println("Enter the origin airport code of travel");

origin = input.nextLine();

System.out.println("Enter the destination airport code of travel");

destination = input.nextLine();

String csvFile = "fdata.txt";

BufferedReader br = null;

String line = "";

String cvsSplitBy = ",";

try {

Scanner x = new Scanner(new File(csvFile));

while(x.hasNext())

{

String month1,day1,time1,timegroup,airportid,temperature,dewpoint,humidity,windspeed,destinationairportid;

month1 = x.next();

day1 = x.next();

time1 = x.next();

timegroup = x.next();

airportid = x.next();

temperature = x.next();

dewpoint = x.next();

humidity = x.next();

windspeed = x.next();

destinationairportid = x.next();

if(month.compareTo(month1)==0)

{

if(day.compareTo(day1)==0)

{

if(time.compareTo(time1)==0)

{

if(origin.compareTo(airportid)==0)

{

if(destination.compareTo(destinationairportid)==0)

{

System.out.println(month1 + "\t" + day1 +"\t"+ time1 + "\t"+ timegroup+"\t"+ airportid+"\t"+temperature +"\t"+dewpoint+"\t"+humidity+

"\t"+windspeed+"\t"+destinationairportid);

int t1 = Integer.parseInt(timegroup);

int m1 = Integer.parseInt(month1);

Double tmp = Double.parseDouble(temperature);

System.out.println(t1 + "" + m1 + " " + tmp);

double[] z = {m1,tmp,t1};

System.out.println("Done");

System.out.println("prob(1|x) = " + logistic.classify(z));

double res = logistic.classify(z);

if(res>0.5)

{

System.out.println("Flight might delay");

}

else

System.out.println("Flight is on time");

}

}

}

}

}

}

x.close();

}

catch (FileNotFoundException e) {

e.printStackTrace();

} catch (IOException e) {

e.printStackTrace();

} finally {

if (br != null) {

try {

br.close();

} catch (IOException e) {

e.printStackTrace();

}

}

}

System.out.println("Done");

}

}

Output from the Simple Logistic Regression program

Enter the month of travel

4

Enter the day of travel

7

Enter the time of travel

1552

Enter the origin airport code of travel

10140

Enter the destination airport code of travel

11259

4 7 1552 1 10140 23.3 -11.1 0.09 7.2 11259

14 23.3

Done

prob(1|x) = 0.9606277435328205

Flight might delay\*\*\*

Done

Enter the month of travel

4

Enter the day of travel

3

Enter the time of travel

852

Enter the origin airport code of travel

10140

Enter the destination airport code of travel

11259

4 3 852 1 10140 6.1 -1.7 0.58 3.6 11259

14 6.1

Done

prob(1|x) = 0.02986981914996907

Flight is on time\*\*\*

References:- https://github.com/tpeng/logistic-regression