**Assignment 1**

**Machine learning algorithms using SciPy**

**General Notes:**

Configuration for dataset file: The dataset file is being read from a folder Assignment1 in “SPARK\_HOME” folder.

Install SciPy and Numpy libraries for python(used the Anaconda distribution for this)

The following steps have been followed for all algorithms before actually building the models:

1. Normalizing features data using StandardScaler to reduce normalization errors.
2. Training to test ratio has been set to 60:40
3. We have tried to implement the following performance metrics:
   1. **Classification algorithms:** 
      1. Accuracy score
      2. Classification report:
         1. Precision: Proportion of instances predicted as positives that were correctly evaluated
         2. Recall: Proportion of positive instances that were correctly evaluated
         3. F1-Score: Harmonic mean of precision and recall
      3. Confusion Matrix: in its (i, j) cell, it shows the number of class instances i that were predicted to be in class j
   2. **Regression algorithms**:
      1. Square root of mean of sum of squares
      2. Explained Variance

How to run the code:

The code for Scipy and Pyspark has been attached as ipython notebook files and the Scala code has been included as jar files built as stand alone applications.

1. The summary of the performance metrics for classification and regression algorithms is given below.

**Performance Metrics for Classification Algorithms**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Algorithm** | **Accuracy Score(%)** | **Precision** | **Recall** | **F1-Score** | **Confusion matrix** |
| **SGD(hinge,l1)** | 79.84 | 0 0.80  1 0.69 | 0 0.98  1 0.14 | 0 0.88  1 0.24 | [[1504 28]  [ 367 61]] |
| **SGD(hinge,l2)** | 80 | 0 0.85  1 0.51 | 0 0.91  1 0.39 | 0 0.88  1 0.44 | [[1412 147]  [ 245 156]] |
| **SGD(log,l1)** | 81.27 | 0 0.82  1 0.70 | 0 0.97  1 0.24 | 0 0.89  1 0.36 | [[1491 43]  [ 324 102]] |
| **SGD(log,l2)** | 80.45 | 0 0.82  1 0.63 | 0 0.96  1 0.25 | 0 0.88  1 0.36 | [[1468 64]  [ 319 109]] |
| **LogisticRegression(l1)** | 80 | 0 0.83  1 0.55 | 0 0.94  1 0.27 | 0 0.88  1 0.37 | [[1455 94]  [ 298 113]] |
| **LogisticRegression(l2)** | 80 | 0 0.83  1 0.55 | 0 0.94  1 0.27 | 0 0.88  1 0.37 | [[1455 94]  [ 298 113]] |

**Performance Metrics for Regression Algorithms**

|  |  |  |
| --- | --- | --- |
| **Algorithm** | **Sqrt mean square error** | **Explained Variance** |
| **Linear Regression** | 0.77 | 0.24 |
| **SGDRegressor(l1)** | 0.77 | 0.28 |
| **SGDRegressor(l2)** | 0.76 | 0.28 |
| **Ridge** | 0.77 | 0.22 |
| **Lasso** | 0.75 | 0.26 |

**Comparison between Scipy and Apache Spark for classification algorithms**

|  |  |  |  |
| --- | --- | --- | --- |
| **Comparison criteria** | **Scipy** | **Spark** | **Comments** |
| **Accuracy Score** | Generally around 79-80 | Around 96 | Spark mllib libraries generally showed a much higher accuracy score for all the algorithms |
| **Consistency** | The results varied a lot with scipy, with a range of +-2 | The results were much more consistent and the variation was negligible | Spark mllib libraries gave much more consistent results |
| **Data Normalization** | Required | Required | Data normalization stabilized the results in both the cases, before normalization the results had a large variation |
| **Metrics / Preprocessing / Helper functions** | Very straightforward and easy to get metrics of the model and perform preprocessing. Good documentation | Rather difficult to implement similar metrics using Spark, had to compute a lot of metrics manually, required conversion between RDD’s for each operation | Scipy API is much mature than Spark. The documentation is clear and easy to follow. Lots of helper functions for metrics and preprocessing are also available in Scipy |

**Comparison between Scipy and Apache Spark for regression algorithms**

|  |  |  |  |
| --- | --- | --- | --- |
| **Comparison criteria** | **Scipy** | **Spark** | **Comments** |
| **Sqrt mean square err** | 0.7 | 0.2 | Spark mllib libraries gave a lower value for sort mean square error and therefore a better model |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

**Classification algorithms:**

**Algorithm: SGDClassifier(with hinge loss and regularization/penalty = l1)**

import numpy as np

from sklearn import linear\_model

from sklearn.cross\_validation import train\_test\_split

from sklearn import metrics

from sklearn import preprocessing

#Load and parse the data

sparkHome = os.environ.get('SPARK\_HOME')

fileLocation = sparkHome + "/Assignment1/winequality-white.csv";

f = open(fileLocation);

f.readline();

data = np.loadtxt(fname=f,delimiter = ';');

X = data[:,0:10];

#Converting it to binary classification

Y = np.where(data[:,11] >= 7, 1, 0);

#Split training/test sets

X\_train,X\_test,Y\_train,Y\_test = train\_test\_split(X,Y,test\_size=0.40);

#Scaling features

scaler = preprocessing.StandardScaler().fit(X\_train);

X\_train = scaler.transform(X\_train);

X\_test = scaler.transform(X\_test);

#Algorithm being used

clf = linear\_model.SGDClassifier(loss="hinge", penalty="l1", n\_iter=100)

#Training the model

clf.fit(X\_train, Y\_train);

print clf.coef\_

print clf.intercept\_

#Predicting on test set

Y\_train\_pred = clf.predict(X\_test);

#Printing out accuracy score

print metrics.accuracy\_score(Y\_test,Y\_train\_pred);

#Printing classification report(precision,recall,f1-score,support)

print metrics.classification\_report(Y\_test,Y\_train\_pred)

#Printing confusion matrix

print metrics.confusion\_matrix(Y\_test,Y\_train\_pred)

**Algorithm: SGDClassifier(with hinge loss and regularization/penalty = l2)**

import numpy as np

from sklearn import linear\_model

from sklearn.cross\_validation import train\_test\_split

from sklearn import metrics

from sklearn import preprocessing

sparkHome = os.environ.get('SPARK\_HOME')

fileLocation = sparkHome + "/Assignment1/winequality-white.csv";

f = open(fileLocation);

f.readline();

data = np.loadtxt(fname=f,delimiter = ';');

X = data[:,0:10];

#Converting it to binary classification

Y = np.where(data[:,11] >= 7, 1, 0);

#Split training/test sets

X\_train,X\_test,Y\_train,Y\_test = train\_test\_split(X,Y,test\_size=0.40);

#Scaling features

scaler = preprocessing.StandardScaler().fit(X\_train);

X\_train = scaler.transform(X\_train);

X\_test = scaler.transform(X\_test);

#Algorithm being used

clf = linear\_model.SGDClassifier(loss="hinge", penalty="l2",n\_iter=100)

#Training the model

clf.fit(X\_train, Y\_train);

print clf.coef\_

print clf.intercept\_

#Predicting on test set

Y\_train\_pred = clf.predict(X\_test);

#Printing out accuracy score

print metrics.accuracy\_score(Y\_test,Y\_train\_pred);

#Printing classification report(precision,recall,f1-score,support)

print metrics.classification\_report(Y\_test,Y\_train\_pred)

#Printing confusion matrix

print metrics.confusion\_matrix(Y\_test,Y\_train\_pred)

**Algorithm: SGDClassifier(with log loss and regularization/penalty = l1)**

import numpy as np

from sklearn import linear\_model

from sklearn.cross\_validation import train\_test\_split

from sklearn import metrics

from sklearn import preprocessing

#Load and parse the data

sparkHome = os.environ.get('SPARK\_HOME')

fileLocation = sparkHome + "/Assignment1/winequality-white.csv";

f = open(fileLocation);

f.readline();

data = np.loadtxt(fname=f,delimiter = ';');

X = data[:,0:10];

#Converting it to binary classification

Y = np.where(data[:,11] >= 7, 1, 0);

#Split training/test sets

X\_train,X\_test,Y\_train,Y\_test = train\_test\_split(X,Y,test\_size=0.40);

#Scaling features

scaler = preprocessing.StandardScaler().fit(X\_train);

X\_train = scaler.transform(X\_train);

X\_test = scaler.transform(X\_test);

#Algorithm being used

clf = linear\_model.SGDClassifier(loss="log", penalty="l1",n\_iter=100)

#Training the model

clf.fit(X\_train, Y\_train);

print clf.coef\_

print clf.intercept\_

#Predicting on test set

Y\_train\_pred = clf.predict(X\_test);

#Printing out accuracy score

print metrics.accuracy\_score(Y\_test,Y\_train\_pred);

#Printing classification report(precision,recall,f1-score,support)

print metrics.classification\_report(Y\_test,Y\_train\_pred)

#Printing confusion matrix

print metrics.confusion\_matrix(Y\_test,Y\_train\_pred)

**Algorithm: SGDClassifier(with log loss and regularization/penalty = l2)**

import numpy as np

from sklearn import linear\_model

from sklearn.cross\_validation import train\_test\_split

from sklearn import metrics

from sklearn import preprocessing

#Load and parse the data

sparkHome = os.environ.get('SPARK\_HOME')

fileLocation = sparkHome + "/Assignment1/winequality-white.csv";

f = open(fileLocation);

f.readline();

data = np.loadtxt(fname=f,delimiter = ';');

X = data[:,0:10];

#Converting it to binary classification

Y = np.where(data[:,11] >= 7, 1, 0);

#Split training/test sets

X\_train,X\_test,Y\_train,Y\_test = train\_test\_split(X,Y,test\_size=0.40);

#Scaling features

scaler = preprocessing.StandardScaler().fit(X\_train);

X\_train = scaler.transform(X\_train);

X\_test = scaler.transform(X\_test);

#Algorithm being used

clf = linear\_model.SGDClassifier(loss="log", penalty="l2",n\_iter=100)

#Training the model

clf.fit(X\_train, Y\_train);

print clf.coef\_

print clf.intercept\_

#Predicting on test set

Y\_train\_pred = clf.predict(X\_test);

#Printing out accuracy score

print metrics.accuracy\_score(Y\_test,Y\_train\_pred);

#Printing classification report(precision,recall,f1-score,support)

print metrics.classification\_report(Y\_test,Y\_train\_pred)

#Printing confusion matrix

print metrics.confusion\_matrix(Y\_test,Y\_train\_pred)

**Algorithm: LogisticRegression(with regularization/penalty = l1)**

import numpy as np

from sklearn import linear\_model

from sklearn.cross\_validation import train\_test\_split

from sklearn import metrics

from sklearn import preprocessing

#Load and parse the data

sparkHome = os.environ.get('SPARK\_HOME')

fileLocation = sparkHome + "/Assignment1/winequality-white.csv";

f = open(fileLocation);

f.readline();

data = np.loadtxt(fname=f,delimiter = ';');

X = data[:,0:10];

#Converting it to binary classification

Y = np.where(data[:,11] >= 7, 1, 0);

#Split training/test sets

X\_train,X\_test,Y\_train,Y\_test = train\_test\_split(X,Y,test\_size=0.40);

#Scaling features

scaler = preprocessing.StandardScaler().fit(X\_train);

X\_train = scaler.transform(X\_train);

X\_test = scaler.transform(X\_test);

#Algorithm being used

clf = linear\_model.LogisticRegression(penalty="l1")

#Training the model

clf.fit(X\_train, Y\_train);

print clf.coef\_

print clf.intercept\_

#Predicting on test set

Y\_train\_pred = clf.predict(X\_test);

#Printing out accuracy score

print metrics.accuracy\_score(Y\_test,Y\_train\_pred);

#Printing classification report(precision,recall,f1-score,support)

print metrics.classification\_report(Y\_test,Y\_train\_pred)

#Printing confusion matrix

print metrics.confusion\_matrix(Y\_test,Y\_train\_pred)

**Algorithm: LogisticRegression(with regularization/penalty = l2)**

import numpy as np

from sklearn import linear\_model

from sklearn.cross\_validation import train\_test\_split

from sklearn import metrics

from sklearn import preprocessing

#Load and parse the data

sparkHome = os.environ.get('SPARK\_HOME')

fileLocation = sparkHome + "/Assignment1/winequality-white.csv";

f = open(fileLocation);

f.readline();

data = np.loadtxt(fname=f,delimiter = ';');

X = data[:,0:10];

#Converting it to binary classification

Y = np.where(data[:,11] >= 7, 1, 0);

#Split training/test sets

X\_train,X\_test,Y\_train,Y\_test = train\_test\_split(X,Y,test\_size=0.40);

#Scaling features

scaler = preprocessing.StandardScaler().fit(X\_train);

X\_train = scaler.transform(X\_train);

X\_test = scaler.transform(X\_test);

#Algorithm being used

clf = linear\_model.LogisticRegression(penalty="l2")

#Training the model

clf.fit(X\_train, Y\_train);

print clf.coef\_

print clf.intercept\_

#Predicting on test set

Y\_train\_pred = clf.predict(X\_test);

#Printing out accuracy score

print metrics.accuracy\_score(Y\_test,Y\_train\_pred);

#Printing classification report(precision,recall,f1-score,support)

print metrics.classification\_report(Y\_test,Y\_train\_pred)

#Printing confusion matrix

print metrics.confusion\_matrix(Y\_test,Y\_train\_pred)

**Regression algorithms:**

**Algorithm: LinearRegression**

import numpy as np

from sklearn import linear\_model

from sklearn.cross\_validation import train\_test\_split

from sklearn import metrics

from sklearn import preprocessing

#Load and parse the data

sparkHome = os.environ.get('SPARK\_HOME')

fileLocation = sparkHome + "/Assignment1/winequality-white.csv";

f = open(fileLocation);

f.readline();

data = np.loadtxt(fname=f,delimiter = ';');

X = data[:,0:10];

Y = data[:,11];

#Split training/test sets

X\_train,X\_test,Y\_train,Y\_test = train\_test\_split(X,Y,test\_size=0.40);

#Scaling features

scaler = preprocessing.StandardScaler().fit(X\_train);

X\_train = scaler.transform(X\_train);

X\_test = scaler.transform(X\_test);

#Algorithm being used

regr = linear\_model.LinearRegression();

#Training the model

regr.fit(X\_train, Y\_train);

print regr.coef\_

print regr.intercept\_

#Predicting and printing out sqrt of mean of sum of squares and Variance

print("Residual sqrt of sum of squares: %.2f"

% np.sqrt(np.mean((regr.predict(X\_test) - Y\_test) \*\* 2)))

# Explained variance score: 1 is perfect prediction

print('Variance score: %.2f' % regr.score(X\_test, Y\_test))

**Algorithm: SGDRegressor(with regularization/penalty =l1)**

import numpy as np

from sklearn import linear\_model

from sklearn.cross\_validation import train\_test\_split

from sklearn import metrics

from sklearn import preprocessing

#Load and parse the data

sparkHome = os.environ.get('SPARK\_HOME')

fileLocation = sparkHome + "/Assignment1/winequality-white.csv";

f = open(fileLocation);

f.readline();

data = np.loadtxt(fname=f,delimiter = ';');

X = data[:,0:10];

Y = data[:,11];

#Split training/test sets

X\_train,X\_test,Y\_train,Y\_test = train\_test\_split(X,Y,test\_size=0.40);

#Scaling features

scaler = preprocessing.StandardScaler().fit(X\_train);

X\_train = scaler.transform(X\_train);

X\_test = scaler.transform(X\_test);

#Algorithm being used

regr = linear\_model.SGDRegressor (penalty='l1',n\_iter=100)

#Training the model

regr.fit(X\_train, Y\_train);

print regr.coef\_

print regr.intercept\_

#Predicting and printing out sqrt of mean of sum of squares and Variance

print("Residual sum of squares: %.2f"

% np.sqrt(np.mean((regr.predict(X\_test) - Y\_test) \*\* 2)))

# Explained variance score: 1 is perfect prediction

print('Variance score: %.2f' % regr.score(X\_test, Y\_test))

**Algorithm: SGDRegressor(with regularization/penalty =l2)**

import numpy as np

from sklearn import linear\_model

from sklearn.cross\_validation import train\_test\_split

from sklearn import metrics

from sklearn import preprocessing

#Load and parse the data

sparkHome = os.environ.get('SPARK\_HOME')

fileLocation = sparkHome + "/Assignment1/winequality-white.csv";

f = open(fileLocation);

f.readline();

data = np.loadtxt(fname=f,delimiter = ';');

X = data[:,0:10];

Y = data[:,11];

#Split training/test sets

X\_train,X\_test,Y\_train,Y\_test = train\_test\_split(X,Y,test\_size=0.40);

#Scaling features

scaler = preprocessing.StandardScaler().fit(X\_train);

X\_train = scaler.transform(X\_train);

X\_test = scaler.transform(X\_test);

#Algorithm being used

regr = linear\_model.SGDRegressor (penalty='l2',n\_iter=100)

#Training the model

regr.fit(X\_train, Y\_train);

print regr.coef\_

print regr.intercept\_

#Predicting and printing out sqrt of mean of sum of squares and Variance

print("Residual sum of squares: %.2f"

% np.sqrt(np.mean((regr.predict(X\_test) - Y\_test) \*\* 2)))

# Explained variance score: 1 is perfect prediction

print('Variance score: %.2f' % regr.score(X\_test, Y\_test))

**Algorithm: Ridge(default regularization=l2)**

import numpy as np

from sklearn import linear\_model

from sklearn.cross\_validation import train\_test\_split

from sklearn import metrics

from sklearn import preprocessing

#Load and parse the data

sparkHome = os.environ.get('SPARK\_HOME')

fileLocation = sparkHome + "/Assignment1/winequality-white.csv";

f = open(fileLocation);

f.readline();

data = np.loadtxt(fname=f,delimiter = ';');

X = data[:,0:10];

Y = data[:,11];

#Split training/test sets

X\_train,X\_test,Y\_train,Y\_test = train\_test\_split(X,Y,test\_size=0.40);

#Scaling features

scaler = preprocessing.StandardScaler().fit(X\_train);

X\_train = scaler.transform(X\_train);

X\_test = scaler.transform(X\_test);

#Algorithm being used

regr = linear\_model.Ridge (alpha = .5)

#Training the model

regr.fit(X\_train, Y\_train);

print regr.coef\_

print regr.intercept\_

#Predicting and printing out sqrt of mean of sum of squares and Variance

print("Residual sum of squares: %.2f"

% np.sqrt(np.mean((regr.predict(X\_test) - Y\_test) \*\* 2)))

# Explained variance score: 1 is perfect prediction

print('Variance score: %.2f' % regr.score(X\_test, Y\_test))

**Algorithm: Lasso(default regularization=l1)**

import numpy as np

from sklearn import linear\_model

from sklearn.cross\_validation import train\_test\_split

from sklearn import metrics

from sklearn import preprocessing

#Load and parse the data

sparkHome = os.environ.get('SPARK\_HOME')

fileLocation = sparkHome + "/Assignment1/winequality-white.csv";

f = open(fileLocation);

f.readline();

data = np.loadtxt(fname=f,delimiter = ';');

X = data[:,0:10];

Y = data[:,11];

#Split training/test sets

X\_train,X\_test,Y\_train,Y\_test = train\_test\_split(X,Y,test\_size=0.40);

#Scaling features

scaler = preprocessing.StandardScaler().fit(X\_train);

X\_train = scaler.transform(X\_train);

X\_test = scaler.transform(X\_test);

#Algorithm being used

regr = linear\_model.Lasso(alpha = 0.001)

#Training the model

regr.fit(X\_train, Y\_train);

print regr.coef\_

print regr.intercept\_

#Predicting and printing out sqrt of mean of sum of squares and Variance

print("Residual sum of squares: %.2f"

% np.sqrt(np.mean((regr.predict(X\_test) - Y\_test) \*\* 2)))

# Explained variance score: 1 is perfect prediction

print('Variance score: %.2f' % regr.score(X\_test, Y\_test))

**Machine learning algorithms using PySpark**

**General Notes:**

Configuration for dataset file: The dataset file is being read from a folder Assignment1 in “SPARK\_HOME” folder.

Install Numpy libraries for python(used the Anaconda distribution for this)

The following steps have been followed for all algorithms before actually building the models:

1. Normalizing features data using Standard Scaler to reduce normalization errors.
2. Training to test ratio has been set to 60:40
3. We have tried to implement the following performance metrics:

**1. Classification algorithms:**

I.Accuracy score

**2. Regression algorithms**:

I.Square root of mean of sum of squares

II.Explained Variance

4. The summary of the performance metrics for classification and regression algorithms is given below.

**Performance Metrics for Classification Algorithms**

|  |  |  |
| --- | --- | --- |
| **Algorithm** | **Accuracy Score(%)** | **Loss** |
| **SGD(hinge,l1)** | 96 | 0.04 |
| **SGD(hinge,l2)** | 97 | 0.03 |
| **SGD(log,l1)** | 96 | 0.04 |
| **SGD(log,l2)** | 97 | 0.03 |
| **LogisticRegression(l1)** | 96 | 0.04 |
| **LogisticRegression(l2)** | 96 | 0.04 |

**Performance Metrics for Regression Algorithms**

|  |  |
| --- | --- |
| **Algorithm** | **Sqrt mean square error** |
| **Linear Regression(l1)** | 0.2 |
| **Linear Regression(l2)** | 0.2 |
| **Ridge** | 0.19 |
| **Lasso** | 0.2 |

**------ SVM with SGD (L1)**

**# Library Imports**

from pyspark.mllib.classification import SVMWithSGD, SVMModel

from pyspark.mllib.regression import LabeledPoint

from pyspark.mllib.feature import StandardScaler

**# Load and parse the data**

def parsePoint(line):

values = [float(x) for x in line.split(',')]

return LabeledPoint(values[11], values[0:10])

data = sc.textFile("Spark/Assignment1/data.csv")

parsedData = data.map(parsePoint)

**#Split training/test sets**

(trainingData, testData) = parsedData.randomSplit([0.6, 0.4])

**#Scaling features**

label = trainingData.map(lambda x: x.label)

features = trainingData.map(lambda x: x.features)

scaler1 = StandardScaler().fit(features)

trainingData = label.zip(scaler1.transform(features))

trainingData = trainingData.map(lambda (x,y) : LabeledPoint(x,y))

**# Build the model with the training for 100 iterations using regularization #parameter as 0.01 and regularization type as** ‘**l1**’ **using the below algorithm**

model = SVMWithSGD.train(trainingData, iterations=100,regParam=0.01,regType='l1')

**# Evaluating the model on test data**

labelsAndPreds = testData.map(lambda p: (p.label, model.predict(p.features)))

**#Calculating the Training error using the following formula**

trainErr = labelsAndPreds.filter(lambda (v, p): v != p).count() / float(testData.count())

**#Calculating the Accuracy using the following formula**

accuracy = labelsAndPreds.filter(lambda (v, p): v == p).count() / float(testData.count())

**#Print the Training Error and Accuracy calculated**

print("Training Error = " + str(trainErr))

print("Accuracy = " + str(accuracy))

**------ SVM with SGD (L2)**

from pyspark.mllib.classification import SVMWithSGD, SVMModel

from pyspark.mllib.regression import LabeledPoint

from pyspark.mllib.feature import StandardScaler

**# Load and parse the data**

def parsePoint(line):

values = [float(x) for x in line.split(',')]

return LabeledPoint(values[11], values[0:10])

data = sc.textFile("Spark/Assignment1/data.csv")

parsedData = data.map(parsePoint)

**#Split training/test sets**

(trainingData, testData) = parsedData.randomSplit([0.6, 0.4])

**#Scaling features**

label = trainingData.map(lambda x: x.label)

features = trainingData.map(lambda x: x.features)

scaler1 = StandardScaler().fit(features)

trainingData = label.zip(scaler1.transform(features))

trainingData = trainingData.map(lambda (x,y) : LabeledPoint(x,y))

**# Build the model with the training for 100 iterations using regularization #parameter as 0.01 and regularization type as** ‘**l2**’ **using the below algorithm**

model = SVMWithSGD.train(trainingData, iterations=100,regParam=0.01,regType='l2')

**# Evaluating the model on test data**

labelsAndPreds = testData.map(lambda p: (p.label, model.predict(p.features)))

**#Calculating the Training error using the following formula**

trainErr = labelsAndPreds.filter(lambda (v, p): v != p).count() / float(testData.count())

**#Calculating the Accuracy using the following formula**

accuracy = labelsAndPreds.filter(lambda (v, p): v == p).count() / float(testData.count())

**#Print the Training Error and Accuracy calculated**

print("Training Error = " + str(trainErr))

print("Accuracy = " + str(accuracy))

**------ Logistic Regression with LBFGS (L1)**

**# Library Imports**

from pyspark.mllib. classification import LabeledPoint, LogisticRegressionWithLBFGS

from numpy import array

from pyspark.mllib.feature import StandardScaler

**# Load and parse the data**

def parsePoint(line):

values = [float(x) for x in line.split(',')]

return LabeledPoint(values[11], values[0:10])

data = sc.textFile("Spark/Assignment1/data.csv")

parsedData = data.map(parsePoint)

**#Split training/test sets**

(trainingData, testData) = parsedData.randomSplit([0.6, 0.4])

**#Scaling features**

label = trainingData.map(lambda x: x.label)

features = trainingData.map(lambda x: x.features)

scaler1 = StandardScaler().fit(features)

trainingData = label.zip(scaler1.transform(features))

trainingData = trainingData.map(lambda (x,y) : LabeledPoint(x,y))

**# Build the model with the parsedData for 100 iterations using regularization #parameter as 0.01 and regularization type as** ‘**l1**’

model=LogisticRegressionWithLBFGS.train(trainingData,iterations=100,regParam=0.01,regType='l1')

**# Evaluate the model on training data**

labelsAndPreds = testData.map(lambda p: (p.label, model.predict(p.features)))

**#Calculating the Training error using the following formula**

trainErr = labelsAndPreds.filter(lambda (v, p): v != p).count() / float(testData.count())

**#Calculating the Accuracy using the following formula**

accuracy = labelsAndPreds.filter(lambda (v, p): v == p).count() / float(testData.count())

**#Print the Training Error and Accuracy calculated**

print("Training Error = " + str(trainErr))

print("Accuracy = " + str(accuracy))

**------ Logistic Regression with LBFGS (L2)**

**# Library Imports**

from pyspark.mllib. classification import LabeledPoint, LogisticRegressionWithLBFGS

from numpy import array

from pyspark.mllib.feature import StandardScaler

**# Load and parse the data**

def parsePoint(line):

values = [float(x) for x in line.split(',')]

return LabeledPoint(values[11], values[0:10])

**# Build the model with the parsedData for 100 iterations using regularization parameter as 0.01 and regularization type as** ‘**l2**’

data = sc.textFile("Spark/Assignment1/data.csv")

parsedData = data.map(parsePoint)

**#Split training/test sets**

(trainingData, testData) = parsedData.randomSplit([0.6, 0.4])

**#Scaling features**

label = trainingData.map(lambda x: x.label)

features = trainingData.map(lambda x: x.features)

scaler1 = StandardScaler().fit(features)

trainingData = label.zip(scaler1.transform(features))

trainingData = trainingData.map(lambda (x,y) : LabeledPoint(x,y))

**# Build the model with the parsedData for 100 iterations using regularization parameter as 0.01 and regularization type as** ‘**l2**’

model = LogisticRegressionWithLBFGS.train(trainingData,iterations=100,regParam=0.01,regType='l2')

**# Evaluate the model on training data**

labelsAndPreds = testData.map(lambda p: (p.label, model.predict(p.features)))

**#Calculating the Training error using the following formula**

trainErr = labelsAndPreds.filter(lambda (v, p): v != p).count() / float(testData.count())

**#Calculating the accuracy using the following formula**

accuracy = labelsAndPreds.filter(lambda (v, p): v == p).count() / float(testData.count())

**#Print the Training Error and Accuracy calculated**

print("Training Error = " + str(trainErr))

print("Accuracy = " + str(accuracy))

**----- Logistic Regression with SGD (L1)**

**# Library Imports**

from pyspark.mllib.classification import LogisticRegressionWithSGD

from pyspark.mllib.regression import LabeledPoint

from pyspark.mllib.evaluation import MulticlassMetrics

from numpy import array

from pyspark.mllib.feature import StandardScaler

**# Load and parse the data**

def parsePoint(line):

values = [float(x) for x in line.split(',')]

return LabeledPoint(values[11], values[0:10])

data = sc.textFile("Spark/Assignment1/data.csv")

parsedData = data.map(parsePoint)

**#Split training/test sets**

(trainingData, testData) = parsedData.randomSplit([0.6, 0.4])

**#Scaling features**

label = trainingData.map(lambda x: x.label)

features = trainingData.map(lambda x: x.features)

scaler1 = StandardScaler().fit(features)

trainingData = label.zip(scaler1.transform(features))

trainingData = trainingData.map(lambda (x,y) : LabeledPoint(x,y))

**# Build the model with the parsedData for 100 iterations using regularization parameter as 0.01 and regularization type as** ‘**l1**’

model = LogisticRegressionWithSGD.train(trainingData, iterations=100,regParam=0.01,regType='l1')

**# Evaluate the model on training data**

labelsAndPreds = testData.map(lambda p: (p.label, model.predict(p.features)))

**#Calculating the Training error using the following formula**

trainErr = labelsAndPreds.filter(lambda (v, p): v != p).count() / float(testData.count())

**#Calculating the Accuracy using the following formula**

accuracy = labelsAndPreds.filter(lambda (v, p): v == p).count() / float(testData.count())

**#Print the Training Error and Accuracy calculated**

print("Training Error = " + str(trainErr))

print("Accuracy = " + str(accuracy))

**----- Logistic Regression with SGD (L2)**

**# Library Imports**

from pyspark.mllib.classification import LogisticRegressionWithSGD

from pyspark.mllib.regression import LabeledPoint

from pyspark.mllib.evaluation import MulticlassMetrics

from numpy import array

from pyspark.mllib.feature import StandardScaler

**# Load and parse the data**

def parsePoint(line):

values = [float(x) for x in line.split(',')]

return LabeledPoint(values[11], values[0:10])

data = sc.textFile("Spark/Assignment1/data.csv")

parsedData = data.map(parsePoint)

**#Split training/test sets**

(trainingData, testData) = parsedData.randomSplit([0.6, 0.4])

**#Scaling features**

label = trainingData.map(lambda x: x.label)

features = trainingData.map(lambda x: x.features)

scaler1 = StandardScaler().fit(features)

trainingData = label.zip(scaler1.transform(features))

trainingData = trainingData.map(lambda (x,y) : LabeledPoint(x,y))

**# Build the model with the parsedData for 100 iterations using regularization #parameter as 0.01 and regularization type as** ‘**l2**’

model = LogisticRegressionWithSGD.train(trainingData, iterations=100,regParam=0.01,regType='l2')

**# Evaluate the model on training data**

labelsAndPreds = testData.map(lambda p: (p.label, model.predict(p.features)))

**#Calculating the Training error using the following formula**

trainErr = labelsAndPreds.filter(lambda (v, p): v != p).count() / float(testData.count())

**#Calculating the Accuracy using the following formula**

accuracy = labelsAndPreds.filter(lambda (v, p): v == p).count() / float(testData.count())

**#Print the Training Error and Accuracy calculated**

print("Training Error = " + str(trainErr))

print("Accuracy = " + str(accuracy))

**----- Linear Regression with SGD (L1)**

**# Library Imports**

from pyspark.mllib.regression import LabeledPoint, LinearRegressionWithSGD

from numpy import array

from pyspark.mllib.feature import StandardScaler

**# Load and parse the data**

def parsePoint(line):

values = [float(x) for x in line.split(',')]

return LabeledPoint(values[11], values[0:10])

data = sc.textFile("Spark/Assignment1/data.csv")

parsedData = data.map(parsePoint)

**#Split training/test sets**

(trainingData, testData) = parsedData.randomSplit([0.6, 0.4])

**#Scaling features**

label = trainingData.map(lambda x: x.label)

features = trainingData.map(lambda x: x.features)

scaler1 = StandardScaler().fit(features)

trainingData = label.zip(scaler1.transform(features))

trainingData = trainingData.map(lambda (x,y) : LabeledPoint(x,y))

**# Build the model with the parsedData for 100 iterations using regularization #parameter as 0.01 and regularization type as** ‘**l1**’

model = LinearRegressionWithSGD.train(trainingData,iterations=100,step=0.0001,regParam=0.1,regType='l1')

**# Evaluate the model on training data**

valuesAndPreds = testData.map(lambda p: (p.label, model.predict(p.features)))

**#Calculating the Sqrt Mean Square Error using the following formula**

MSE = valuesAndPreds.map(lambda (v, p): (v - p)\*\*2).reduce(lambda x, y: x + y) / valuesAndPreds.count()

MSER = (MSE)\*\*0.5

**#Calculating the Variance using the following formula**

meanVal = valuesAndPreds.map(lambda (v,p): p).sum() /valuesAndPreds.count()

variance = valuesAndPreds.map(lambda (v,p): (p - meanVal)\*\*2).sum() /valuesAndPreds.count()

**#Print the Sqrt Mean Square Error and Variance calculated**

print("Mean Squared Root Error =" +str(MSER))

print ("Variance = " + str(variance))

**----- Linear Regression with SGD (L2)**

**# Library Imports**

from pyspark.mllib.regression import LabeledPoint, LinearRegressionWithSGD

from numpy import array

from pyspark.mllib.feature import StandardScaler

**# Load and parse the data**

def parsePoint(line):

values = [float(x) for x in line.split(',')]

return LabeledPoint(values[11], values[0:10])

data = sc.textFile("Spark/Assignment1/data.csv")

parsedData = data.map(parsePoint)

**#Split training/test sets**

(trainingData, testData) = parsedData.randomSplit([0.6, 0.4])

**#Scaling features**

label = trainingData.map(lambda x: x.label)

features = trainingData.map(lambda x: x.features)

scaler1 = StandardScaler().fit(features)

trainingData = label.zip(scaler1.transform(features))

trainingData = trainingData.map(lambda (x,y) : LabeledPoint(x,y))

**# Build the model with the parsedData for 100 iterations using regularization #parameter as 0.01 and regularization type as** ‘**l2**’

model = LinearRegressionWithSGD.train(trainingData,iterations=100,step=0.0001,regParam=0.1,regType='l2')

**# Evaluate the model on training data**

valuesAndPreds = testData.map(lambda p: (p.label, model.predict(p.features)))

**#Calculating the Sqrt Mean Square Error using the following formula**

MSE = valuesAndPreds.map(lambda (v, p): (v - p)\*\*2).reduce(lambda x, y: x + y) / valuesAndPreds.count()

MSER = (MSE)\*\*0.5

**#Calculating the Variance using the following formula**

meanVal = valuesAndPreds.map(lambda (v,p): p).sum() /valuesAndPreds.count()

variance = valuesAndPreds.map(lambda (v,p): (p - meanVal)\*\*2).sum() /valuesAndPreds.count()

**#Print the Sqrt Mean Square Error and Variance calculated**

print("Mean Squared Root Error =" +str(MSER))

print ("Variance = " + str(variance))

**----- Ridge Regression with SGD (L2)**

**# Library Imports**

from pyspark.mllib.regression import LabeledPoint, RidgeRegressionWithSGD

from numpy import array

from pyspark.mllib.feature import StandardScaler

**# Load and parse the data**

def parsePoint(line):

values = [float(x) for x in line.split(',')]

return LabeledPoint(values[11], values[0:10])

data = sc.textFile("Spark/Assignment1/data.csv")

parsedData = data.map(parsePoint)

**#Split training/test sets**

(trainingData, testData) = parsedData.randomSplit([0.6, 0.4])

**#Scaling features**

label = trainingData.map(lambda x: x.label)

features = trainingData.map(lambda x: x.features)

scaler1 = StandardScaler().fit(features)

trainingData = label.zip(scaler1.transform(features))

trainingData = trainingData.map(lambda (x,y) : LabeledPoint(x,y))

**# Build the model with the parsedData for 100 iterations using regularization parameter as 0.01 and regularization type as** ‘**l2**’

model = RidgeRegressionWithSGD.train(trainingData,iterations=100,step=0.0001,regParam=0.1)

**# Evaluate the model on training data**

valuesAndPreds = testData.map(lambda p: (p.label, model.predict(p.features)))

**#Calculating the Sqrt Mean Square Error using the following formula**

MSE = valuesAndPreds.map(lambda (v, p): (v - p)\*\*2).reduce(lambda x, y: x + y) / valuesAndPreds.count()

MSER = (MSE)\*\*0.5

**#Calculating the Variance using the following formula**

meanVal = valuesAndPreds.map(lambda (v,p): p).sum() /valuesAndPreds.count()

variance = valuesAndPreds.map(lambda (v,p): (p - meanVal)\*\*2).sum() /valuesAndPreds.count()

**#Print the Sqrt Mean Square Error and Variance calculated**

print("Mean Squared Root Error =" +str(MSER))

print ("Variance = " + str(variance))

**----- Lasso with SGD (L1)**

**# Library Imports**

from pyspark.mllib.regression import LabeledPoint, LassoWithSGD

from numpy import array

from pyspark.mllib.feature import StandardScaler

**# Load and parse the data**

def parsePoint(line):

values = [float(x) for x in line.split(',')]

return LabeledPoint(values[11], values[0:10])

data = sc.textFile("Spark/Assignment1/data.csv")

parsedData = data.map(parsePoint)

**#Split training/test sets**

(trainingData, testData) = parsedData.randomSplit([0.6, 0.4])

**#Scaling features**

label = trainingData.map(lambda x: x.label)

features = trainingData.map(lambda x: x.features)

scaler1 = StandardScaler().fit(features)

trainingData = label.zip(scaler1.transform(features))

trainingData = trainingData.map(lambda (x,y) : LabeledPoint(x,y))

**# Build the model with the parsedData for 100 iterations using regularization parameter as 0.01 and regularization type as** ‘**l1**’

model=LassoWithSGD.train(trainingData,iterations=100,step=0.0001,regParam=0.1)

**# Evaluate the model on training data**

valuesAndPreds = testData.map(lambda p: (p.label, model.predict(p.features)))

**#Calculating the Sqrt Mean Square Error using the following formula**

MSE = valuesAndPreds.map(lambda (v, p): (v - p)\*\*2).reduce(lambda x, y: x + y) / valuesAndPreds.count()

MSER = (MSE)\*\*0.5

**#Calculating the Variance using the following formula**

meanVal = valuesAndPreds.map(lambda (v,p): p).sum() /valuesAndPreds.count()

variance = valuesAndPreds.map(lambda (v,p): (p - meanVal)\*\*2).sum() /valuesAndPreds.count()

**#Print the Sqrt Mean Square Error and Variance calculated**

print("Mean Squared Root Error =" +str(MSER))

print ("Variance = " + str(variance))

**Machine learning algorithms using Spark - Scala**

**General Notes:**

Configuration for dataset file: The dataset file is being read from a folder Assignment1 in “SPARK\_HOME” folder.

The following steps have been followed for all algorithms before actually building the models:

1. Normalizing features data using Standard Scaler to reduce normalization errors.

2. Training to test ratio has been set to 60:40

3. We have tried to implement the following performance metrics:

1. **Classification algorithms:**

I. Area under RO

2. **Regression algorithms**:

I. Square root of mean of sum of squares

3. The summary of the performance metrics for classification and regression algorithms is given below.

|  |  |  |
| --- | --- | --- |
| **Area under ROC** | **L1 Regularizer** | **L2 Regularizer** |
| **SVMwithSGD** | 0.6833177450896986 | 0.6833177450896992 |

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Confusion Matrix** | **Recall** | **Acuuracy** |
| **LogisticRegressionWithLBFGS – L1** | 1857.0 0.0  76.0 0.0 | 0.9606828763579928 | 0.96068287 |
| **LogisticRegressionWithLBFGS – L2** | 1857.0 0.0  76.0 0.0 | 0.9606828763579928 | 0.96068287 |
| **LogisticRegressionWithSGD – L1** | 1857.0 0.0  76.0 0.0 | 0.9606828763579928 | 0.96068287 |
| **LogisticRegressionWithSGD – L2** | 1857.0 0.0  76.0 0.0 | 0.9606828763579928 | 0.96068287 |

|  |  |
| --- | --- |
|  |  |
| **LinearRegresssionwithSGD – L1** | 0.19828545998637231 |
| **LinearRegressionwithSGD – L2** | 0.1944077258887634 |
| **Ridge Regression** | 0.1944041439212048 |
| **Lasso Regression** | 0.19440564777531041 |

**The scala scripts are explained below:**

**SVMwithSGD with L1 regularizer:**

import org.apache.spark.mllib.classification.{SVMModel, SVMWithSGD}

import org.apache.spark.mllib.evaluation.MulticlassMetrics

import org.apache.spark.mllib.util.MLUtils

import org.apache.spark.mllib.regression.LabeledPoint

import org.apache.spark.mllib.linalg.Vectors

import org.apache.spark.mllib.optimization.L1Updater

import org.apache.spark.mllib.evaluation.BinaryClassificationMetrics

import org.apache.spark.mllib.feature.StandardScaler

import org.apache.spark.SparkContext

import org.apache.spark.SparkContext.\_

import org.apache.spark.SparkConf

object SVMwithSGDL1 {

def main(args: Array[String]) {

val conf = new SparkConf().setAppName("SVMwithSGDL1")

val sc = new SparkContext(conf)

//Declaring the delimitter for the file

val Delimeter = ","

//Load the csv file in a RDD

val textFile = sc.textFile("Assignment1/data.csv")

val data = textFile.map { line =>

val parts = line.split(Delimeter)

LabeledPoint(parts(11).toDouble, Vectors.dense(parts.slice(0,10).map(x => x.toDouble).toArray))

}

// Split data into training (60%) and test (40%).

val splits = data.randomSplit(Array(0.6, 0.4), seed = 11L)

val training = splits(0).cache()

val test = splits(1)

// Setting number of iterations

val numIterations = 100

// Creating new instance for the algorithm

val svmAlg = new SVMWithSGD()

// Setting the parameters for optimizations

svmAlg.optimizer.

setNumIterations(numIterations).

setRegParam(0.1).

setUpdater(new L1Updater)

// normalizing the data

val ss = new StandardScaler().fit(training.map(x=>x.features))

val training1 = training.map(x=>(x.label, ss.transform(x.features)))

val training2 = training1.map(y=> LabeledPoint(y.\_1,y.\_2))

val test1 = test.map(x=>(x.label, ss.transform(x.features)))

val test2 = test1.map(y=> LabeledPoint(y.\_1,y.\_2))

// Training the model with training data

val model = svmAlg.run(training2)

// Clear the default threshold.

model.clearThreshold()

// Calculating a raw score for the test data

val scoreAndLabels = test2.map { point =>

val score = model.predict(point.features)

(score, point.label)

}

// Get evaluation metrics using BinaryClassificationMetrics cLass

val metrics = new BinaryClassificationMetrics(scoreAndLabels)

// Stroing auROC for performance evaluation

val auROC = metrics.areaUnderROC()

val accurtimes = scoreAndLabels.filter(r => r.\_1 == r.\_2)

val accuracy = accurtimes.count()/scoreAndLabels.count()

println("accuracy:" + accuracy)

println("Area under ROC = " + auROC)

}

}

**SVMwithSGD with L2 Regularizer:**

import org.apache.spark.mllib.classification.{SVMModel, SVMWithSGD}

import org.apache.spark.mllib.evaluation.MulticlassMetrics

import org.apache.spark.mllib.util.MLUtils

import org.apache.spark.mllib.regression.LabeledPoint

import org.apache.spark.mllib.linalg.Vectors

import org.apache.spark.mllib.evaluation.BinaryClassificationMetrics

import org.apache.spark.mllib.feature.StandardScaler

import org.apache.spark.SparkContext

import org.apache.spark.SparkContext.\_

import org.apache.spark.SparkConf

object SVMwithSGDL1 {

def main(args: Array[String]) {

val conf = new SparkConf().setAppName("SVMwithSGDL2")

val sc = new SparkContext(conf)

val Delimeter = ","

val textFile = sc.textFile("Assignment1/data.csv")

val data = textFile.map { line =>

val parts = line.split(Delimeter)

LabeledPoint(parts(10).toDouble, Vectors.dense(parts.slice(0,9).map(x => x.toDouble).toArray))

}

// Split data into training (60%) and test (40%).

val splits = data.randomSplit(Array(0.6, 0.4), seed = 11L)

val training = splits(0).cache()

val test = splits(1)

//Normalizing the data

val ss = new StandardScaler().fit(training.map(x=>x.features))

val training1 = training.map(x=>(x.label, ss.transform(x.features)))

val training2 = training1.map(y=> LabeledPoint(y.\_1,y.\_2))

val test1 = test.map(x=>(x.label, ss.transform(x.features)))

val test2 = test1.map(y=> LabeledPoint(y.\_1,y.\_2))

// Run training algorithm to build the model

val numIterations = 100

val model = SVMWithSGD.train(training2, numIterations)

// Clear the default threshold.

model.clearThreshold()

// Calculating a raw score for the test data

val scoreAndLabels = test2.map { point =>

val score = model.predict(point.features)

(score, point.label)

}

// Get evaluation metrics using BinaryClassificationMetrics cLass

val metrics = new BinaryClassificationMetrics(scoreAndLabels)

// Stroing auROC for performance evaluation

val auROC = metrics.areaUnderROC()

val accurtimes = scoreAndLabels.filter(r => r.\_1 == r.\_2)

val accuracy = accurtimes.count()/scoreAndLabels.count()

println("accuracy:" + accuracy)

println("Area under ROC = " + auROC)

}

}

**LogisticRegressionwithLBFGS with L1 regularizer**

import org.apache.spark.mllib.classification.{LogisticRegressionWithLBFGS, LogisticRegressionModel}

import org.apache.spark.mllib.evaluation.MulticlassMetrics

import org.apache.spark.mllib.util.MLUtils

import org.apache.spark.mllib.regression.LabeledPoint

import org.apache.spark.mllib.linalg.Vectors

import org.apache.spark.mllib.feature.StandardScaler

import org.apache.spark.mllib.optimization.L1Updater

import org.apache.spark.SparkContext

import org.apache.spark.SparkContext.\_

import org.apache.spark.SparkConf

object LogisticRegressionWithLBFGSL1 {

def main(args: Array[String]) {

val conf = new SparkConf().setAppName("LBFGSL1")

val sc = new SparkContext(conf)

// Declaring demlitter

val Delimeter = ","

val textFile = sc.textFile("Assignment1/data.csv")

val data = textFile.map { line =>

val parts = line.split(Delimeter)

LabeledPoint(parts(0).toDouble, Vectors.dense(parts.slice(1,10).map(x => x.toDouble).toArray))

}

// Split data into training (60%) and test (40%).

val splits = data.randomSplit(Array(0.6, 0.4), seed = 11L)

val training = splits(0).cache()

val test = splits(1)

// Declaring number of iterations

val numIterations = 100

//Normalizing data

val ss = new StandardScaler().fit(training.map(x=>x.features))

val training1 = training.map(x=>(x.label, ss.transform(x.features)))

val training2 = training1.map(y=> LabeledPoint(y.\_1,y.\_2))

val test1 = test.map(x=>(x.label, ss.transform(x.features)))

val test2 = test1.map(y=> LabeledPoint(y.\_1,y.\_2))

val linearRegSGD = new LogisticRegressionWithLBFGS();

linearRegSGD.optimizer.setNumIterations(numIterations).

setRegParam(0.01).

setUpdater(new L1Updater)

//

val model = linearRegSGD.run(training2)

val predictionAndLabels = test2.map { case LabeledPoint(label, features) =>

val prediction = model.predict(features)

(prediction, label)

}

//calculating accuracy

val accurtimes = predictionAndLabels.filter(r => r.\_1 == r.\_2)

val accuracy = accurtimes.count().toFloat/predictionAndLabels.count()

// Get evaluation metrics.

val metrics = new MulticlassMetrics(predictionAndLabels)

val precision = metrics.precision

val recall = metrics.recall

println(metrics.confusionMatrix)

println("Recall(0):" + metrics.recall(0.0))

println("Recall(1):" + metrics.recall(1.0))

println("Recall:" + metrics.recall)

println("Precision(0):" + metrics.precision(0.0))

println("Precision(1):" + metrics.precision(1.0))

println("accuracy:" + accuracy)

}

}

**LogisticRegressionwithLBFGS with L2 regularizer**

import org.apache.spark.mllib.classification.{LogisticRegressionWithLBFGS, LogisticRegressionModel}

import org.apache.spark.mllib.evaluation.MulticlassMetrics

import org.apache.spark.mllib.util.MLUtils

import org.apache.spark.mllib.regression.LabeledPoint

import org.apache.spark.mllib.linalg.Vectors

import org.apache.spark.mllib.feature.StandardScaler

import org.apache.spark.SparkContext

import org.apache.spark.SparkContext.\_

import org.apache.spark.SparkConf

object LogisticRegressionWithLBFGSL2 {

def main(args: Array[String]) {

val conf = new SparkConf().setAppName("LBFGSL2")

val sc = new SparkContext(conf)

// Declaring demlitter

val Delimeter = ","

val textFile = sc.textFile("Assignment1/data.csv")

val data = textFile.map { line =>

val parts = line.split(Delimeter)

LabeledPoint(parts(0).toDouble, Vectors.dense(parts.slice(1,10).map(x => x.toDouble).toArray))

}

// Split data into training (60%) and test (40%).

val splits = data.randomSplit(Array(0.6, 0.4), seed = 11L)

val training = splits(0).cache()

val test = splits(1)

// Run training algorithm to build the model

val numIterations = 100

//Normalizing the data

val ss = new StandardScaler().fit(training.map(x=>x.features))

val training1 = training.map(x=>(x.label, ss.transform(x.features)))

val training2 = training1.map(y=> LabeledPoint(y.\_1,y.\_2))

val test1 = test.map(x=>(x.label, ss.transform(x.features)))

val test2 = test1.map(y=> LabeledPoint(y.\_1,y.\_2))

// training the model

val model = new LogisticRegressionWithLBFGS().setNumClasses(2).run(training)

val predictionAndLabels = test2.map { case LabeledPoint(label, features) =>

val prediction = model.predict(features)

(prediction, label)

}

val accurtimes = predictionAndLabels.filter(r => r.\_1 == r.\_2)

val accuracy = accurtimes.count().toFloat/predictionAndLabels.count()

// Get evaluation metrics.

val metrics = new MulticlassMetrics(predictionAndLabels)

val precision = metrics.precision

val recall = metrics.recall

println(metrics.confusionMatrix)

println("Recall(0):" + metrics.recall(0.0))

println("Recall(1):" + metrics.recall(1.0))

println("Recall:" + metrics.recall)

println("Precision:" + metrics.precision(0.0))

println("Precision:" + metrics.precision(1.0))

println("accuracy:" + accuracy)

}

}

**LogisticRegressionwithSGD with L1 regularizer:**

import org.apache.spark.SparkContext

import org.apache.spark.mllib.classification.{LogisticRegressionWithSGD, LogisticRegressionModel}

import org.apache.spark.mllib.evaluation.MulticlassMetrics

import org.apache.spark.mllib.regression.LabeledPoint

import org.apache.spark.mllib.linalg.Vectors

import org.apache.spark.mllib.util.MLUtils

import org.apache.spark.mllib.optimization.L1Updater

import org.apache.spark.mllib.feature.StandardScaler

import org.apache.spark.SparkContext

import org.apache.spark.SparkContext.\_

import org.apache.spark.SparkConf

object LogisticRegressionWithSGDL1 {

def main(args: Array[String]) {

val conf = new SparkConf().setAppName("LRGGDL1")

val sc = new SparkContext(conf)

// Declaring demlitter

val Delimeter = ","

val textFile = sc.textFile("Assignment1/data.csv")

val data = textFile.map { line =>

val parts = line.split(Delimeter)

LabeledPoint(parts(11).toDouble, Vectors.dense(parts.slice(0,10).map(x => x.toDouble).toArray))

}

// Split data into training (60%) and test (40%).

val splits = data.randomSplit(Array(0.6, 0.4), seed = 11L)

val training = splits(0).cache()

val test = splits(1)

// Run training algorithm to build the model

val numIterations = 100

//Normalizing the data

val ss = new StandardScaler().fit(training.map(x=>x.features))

val training1 = training.map(x=>(x.label, ss.transform(x.features)))

val training2 = training1.map(y=> LabeledPoint(y.\_1,y.\_2))

val test1 = test.map(x=>(x.label, ss.transform(x.features)))

val test2 = test1.map(y=> LabeledPoint(y.\_1,y.\_2))

// training the model

val lbfgsAlgo = new LogisticRegressionWithSGD()

lbfgsAlgo.optimizer.

setNumIterations(numIterations).

setRegParam(0.1).

setUpdater(new L1Updater)

val model = lbfgsAlgo.run(training2)

val predictionAndLabels = test2.map { case LabeledPoint(label, features) =>

val prediction = model.predict(features)

(prediction, label)

}

val accurtimes = predictionAndLabels.filter(r => r.\_1 == r.\_2)

val accuracy = accurtimes.count().toFloat/predictionAndLabels.count()

// Get evaluation metrics.

val metrics = new MulticlassMetrics(predictionAndLabels)

val precision = metrics.precision

val recall = metrics.recall

println(metrics.confusionMatrix)

println("Recall(0):" + metrics.recall(0.0))

println("Recall(1):" + metrics.recall(1.0))

println("Recall:" + metrics.recall)

println("Precision:" + metrics.precision(0.0))

println("Precision:" + metrics.precision(1.0))

println("accuracy:" + accuracy)

}

}

**LogisticRegressionwithSGD with L2 Regularizer:**

import org.apache.spark.SparkContext

import org.apache.spark.mllib.classification.{LogisticRegressionWithSGD, LogisticRegressionModel}

import org.apache.spark.mllib.evaluation.MulticlassMetrics

import org.apache.spark.mllib.regression.LabeledPoint

import org.apache.spark.mllib.linalg.Vectors

import org.apache.spark.mllib.util.MLUtils

import org.apache.spark.mllib.optimization.L1Updater

import org.apache.spark.mllib.feature.StandardScaler

import org.apache.spark.SparkContext

import org.apache.spark.SparkContext.\_

import org.apache.spark.SparkConf

object LogisticRegressionWithSGDL2 {

def main(args: Array[String]) {

val conf = new SparkConf().setAppName("LRGGDL2")

val sc = new SparkContext(conf)

// Declaring demlitter

val Delimeter = ","

val textFile = sc.textFile("Assignment1/data.csv")

val data = textFile.map { line =>

val parts = line.split(Delimeter)

LabeledPoint(parts(11).toDouble, Vectors.dense(parts.slice(0,10).map(x => x.toDouble).toArray))

}

// Split data into training (60%) and test (40%).

val splits = data.randomSplit(Array(0.6, 0.4), seed = 11L)

val training = splits(0).cache()

val test = splits(1)

// Run training algorithm to build the model

val numIterations = 100

//Normalizing the data

val ss = new StandardScaler().fit(training.map(x=>x.features))

val training1 = training.map(x=>(x.label, ss.transform(x.features)))

val training2 = training1.map(y=> LabeledPoint(y.\_1,y.\_2))

val test1 = test.map(x=>(x.label, ss.transform(x.features)))

val test2 = test1.map(y=> LabeledPoint(y.\_1,y.\_2))

// training the model

val model = LogisticRegressionWithSGD.train(training2,numIterations)

val predictionAndLabels = test2.map { case LabeledPoint(label, features) =>

val prediction = model.predict(features)

(prediction, label)

}

val accurtimes = predictionAndLabels.filter(r => r.\_1 == r.\_2)

val accuracy = accurtimes.count().toFloat/predictionAndLabels.count()

// Get evaluation metrics.

val metrics = new MulticlassMetrics(predictionAndLabels)

val precision = metrics.precision

val recall = metrics.recall

println(metrics.confusionMatrix)

println("Recall(0):" + metrics.recall(0.0))

println("Recall(1):" + metrics.recall(1.0))

println("Recall:" + metrics.recall)

println("Precision:" + metrics.precision(0.0))

println("Precision:" + metrics.precision(1.0))

println("accuracy:" + accuracy)

}

}

**LinearRegressionwithSGD with L1 regularizer:**

import org.apache.spark.SparkContext

import org.apache.spark.mllib.regression.LinearRegressionModel

import org.apache.spark.mllib.regression.LinearRegressionWithSGD

import org.apache.spark.mllib.evaluation.RegressionMetrics

import org.apache.spark.mllib.regression.LabeledPoint

import org.apache.spark.mllib.linalg.Vectors

import org.apache.spark.mllib.util.MLUtils

import org.apache.spark.mllib.optimization.L1Updater

import org.apache.spark.mllib.evaluation.RegressionMetrics

import org.apache.spark.mllib.feature.StandardScaler

import org.apache.spark.SparkContext

import org.apache.spark.SparkContext.\_

import org.apache.spark.SparkConf

object LinearRegressionWithSGDL1 {

def main(args: Array[String]) {

val conf = new SparkConf().setAppName("LinearRegressionWithSGDL1")

val sc = new SparkContext(conf)

// Declaring demlitter

val Delimeter = ","

val textFile = sc.textFile("Assignment1/data.csv")

val data = textFile.map { line =>

val parts = line.split(Delimeter)

LabeledPoint(parts(11).toDouble, Vectors.dense(parts.slice(0,10).map(x => x.toDouble).toArray))

}

// Split data into training (60%) and test (40%).

val splits = data.randomSplit(Array(0.6, 0.4), seed = 11L)

val training = splits(0).cache()

val test = splits(1)

// Run training algorithm to build the model

val numIterations = 100

//Normalizing the data

val ss = new StandardScaler().fit(training.map(x=>x.features))

val training1 = training.map(x=>(x.label, ss.transform(x.features)))

val training2 = training1.map(y=> LabeledPoint(y.\_1,y.\_2))

val test1 = test.map(x=>(x.label, ss.transform(x.features)))

val test2 = test1.map(y=> LabeledPoint(y.\_1,y.\_2))

// training the model

val linearRegSGD = new LinearRegressionWithSGD();

linearRegSGD.optimizer.setNumIterations(numIterations).

setRegParam(0.01).

setUpdater(new L1Updater).

setStepSize(0.00001)

val model = linearRegSGD.run(training)

//val model = LinearRegressionWithSGD.train(training,numIterations,0.00001)

val valuesAndPreds = test2.map { point =>

val prediction = model.predict(point.features)

(point.label, prediction)

}

val metrics = new RegressionMetrics(valuesAndPreds)

val MSE = valuesAndPreds.map{case(v, p) => math.pow((v - p), 2)}.mean()

val meanPred = valuesAndPreds.map{case(v, p) => p}.mean()

val numerator = valuesAndPreds.map{case(v, p) => math.pow((meanPred - p), 2)}.sum()

val varian = numerator.toFloat/valuesAndPreds.count()

println("training Mean Squared Error = " + MSE)

println("Variance:" + varian)

println("RME: "+ metrics.rootMeanSquaredError)

println("Explained Variance: "+metrics.explainedVariance)

}

}

**LinearRegressionwithSGD with L2 regularizer:**

import org.apache.spark.SparkContext

import org.apache.spark.mllib.regression.LinearRegressionModel

import org.apache.spark.mllib.regression.LinearRegressionWithSGD

import org.apache.spark.mllib.evaluation.RegressionMetrics

import org.apache.spark.mllib.regression.LabeledPoint

import org.apache.spark.mllib.linalg.Vectors

import org.apache.spark.mllib.util.MLUtils

//import org.apache.spark.mllib.optimization.L2Updater

import org.apache.spark.mllib.evaluation.RegressionMetrics

import org.apache.spark.mllib.feature.StandardScaler

import org.apache.spark.SparkContext

import org.apache.spark.SparkContext.\_

import org.apache.spark.SparkConf

object LinearRegressionWithSGDL1 {

def main(args: Array[String]) {

val conf = new SparkConf().setAppName("LinearRegressionWithSGDL1")

val sc = new SparkContext(conf)

// Declaring demlitter

val Delimeter = ","

val textFile = sc.textFile("Assignment1/data.csv")

val data = textFile.map { line =>

val parts = line.split(Delimeter)

LabeledPoint(parts(11).toDouble, Vectors.dense(parts.slice(0,10).map(x => x.toDouble).toArray))

}

// Split data into training (60%) and test (40%).

val splits = data.randomSplit(Array(0.6, 0.4), seed = 11L)

val training = splits(0).cache()

val test = splits(1)

// Run training algorithm to build the model

val numIterations = 100

//Normalizing the data

val ss = new StandardScaler().fit(training.map(x=>x.features))

val training1 = training.map(x=>(x.label, ss.transform(x.features)))

val training2 = training1.map(y=> LabeledPoint(y.\_1,y.\_2))

val test1 = test.map(x=>(x.label, ss.transform(x.features)))

val test2 = test1.map(y=> LabeledPoint(y.\_1,y.\_2))

// training the model

val model = LinearRegressionWithSGD.train(training2,numIterations,0.00001)

val valuesAndPreds = test2.map { point =>

val prediction = model.predict(point.features)

(point.label, prediction)

}

val metrics = new RegressionMetrics(valuesAndPreds)

val MSE = valuesAndPreds.map{case(v, p) => math.pow((v - p), 2)}.mean()

val meanPred = valuesAndPreds.map{case(v, p) => p}.mean()

val numerator = valuesAndPreds.map{case(v, p) => math.pow((meanPred - p), 2)}.sum()

val varian = numerator.toFloat/valuesAndPreds.count()

println("training Mean Squared Error = " + MSE)

println("Variance:" + varian)

println("RME: "+ metrics.rootMeanSquaredError)

println("Explained Variance: "+metrics.explainedVariance)

}

}

**RidgeRegression:**

import org.apache.spark.SparkContext

import org.apache.spark.mllib.regression.RidgeRegressionModel

import org.apache.spark.mllib.regression.RidgeRegressionWithSGD

import org.apache.spark.mllib.evaluation.RegressionMetrics

import org.apache.spark.mllib.regression.LabeledPoint

import org.apache.spark.mllib.linalg.Vectors

import org.apache.spark.mllib.util.MLUtils

//import org.apache.spark.mllib.optimization.L2Updater

import org.apache.spark.mllib.evaluation.RegressionMetrics

import org.apache.spark.mllib.feature.StandardScaler

import org.apache.spark.SparkContext

import org.apache.spark.SparkContext.\_

import org.apache.spark.SparkConf

object LinearRegressionWithSGDL1 {

def main(args: Array[String]) {

val conf = new SparkConf().setAppName("LinearRegressionWithSGDL1")

val sc = new SparkContext(conf)

// Declaring demlitter

val Delimeter = ","

val textFile = sc.textFile("Assignment1/data.csv")

val data = textFile.map { line =>

val parts = line.split(Delimeter)

LabeledPoint(parts(11).toDouble, Vectors.dense(parts.slice(0,10).map(x => x.toDouble).toArray))

}

// Split data into training (60%) and test (40%).

val splits = data.randomSplit(Array(0.6, 0.4), seed = 11L)

val training = splits(0).cache()

val test = splits(1)

// Run training algorithm to build the model

val numIterations = 100

//Normalizing the data

val ss = new StandardScaler().fit(training.map(x=>x.features))

val training1 = training.map(x=>(x.label, ss.transform(x.features)))

val training2 = training1.map(y=> LabeledPoint(y.\_1,y.\_2))

val test1 = test.map(x=>(x.label, ss.transform(x.features)))

val test2 = test1.map(y=> LabeledPoint(y.\_1,y.\_2))

// training the model

val rdg = new RidgeRegressionWithSGD()

rdg.optimizer.setNumIterations(numIterations).

setRegParam(0.01).

setStepSize(0.0001)

val model = rdg.run(training2)

val valuesAndPreds = test2.map { point =>

val prediction = model.predict(point.features)

(point.label, prediction)

}

val metrics = new RegressionMetrics(valuesAndPreds)

val MSE = valuesAndPreds.map{case(v, p) => math.pow((v - p), 2)}.mean()

val meanPred = valuesAndPreds.map{case(v, p) => p}.mean()

val numerator = valuesAndPreds.map{case(v, p) => math.pow((meanPred - p), 2)}.sum()

val varian = numerator.toFloat/valuesAndPreds.count()

println("training Mean Squared Error = " + MSE)

println("Variance:" + varian)

println("RME: "+ metrics.rootMeanSquaredError)

println("Explained Variance: "+metrics.explainedVariance)

}

}

**LassoRegression:**

import org.apache.spark.SparkContext

import org.apache.spark.mllib.regression.LassoModel

import org.apache.spark.mllib.regression.LassoWithSGD

import org.apache.spark.mllib.evaluation.RegressionMetrics

import org.apache.spark.mllib.regression.LabeledPoint

import org.apache.spark.mllib.linalg.Vectors

import org.apache.spark.mllib.util.MLUtils

//import org.apache.spark.mllib.optimization.L2Updater

import org.apache.spark.mllib.evaluation.RegressionMetrics

import org.apache.spark.mllib.feature.StandardScaler

import org.apache.spark.SparkContext

import org.apache.spark.SparkContext.\_

import org.apache.spark.SparkConf

object LinearRegressionWithSGDL1 {

def main(args: Array[String]) {

val conf = new SparkConf().setAppName("LinearRegressionWithSGDL1")

val sc = new SparkContext(conf)

// Declaring demlitter

val Delimeter = ","

val textFile = sc.textFile("Assignment1/data.csv")

val data = textFile.map { line =>

val parts = line.split(Delimeter)

LabeledPoint(parts(11).toDouble, Vectors.dense(parts.slice(0,10).map(x => x.toDouble).toArray))

}

// Split data into training (60%) and test (40%).

val splits = data.randomSplit(Array(0.6, 0.4), seed = 11L)

val training = splits(0).cache()

val test = splits(1)

// Run training algorithm to build the model

val numIterations = 100

//Normalizing the data

val ss = new StandardScaler().fit(training.map(x=>x.features))

val training1 = training.map(x=>(x.label, ss.transform(x.features)))

val training2 = training1.map(y=> LabeledPoint(y.\_1,y.\_2))

val test1 = test.map(x=>(x.label, ss.transform(x.features)))

val test2 = test1.map(y=> LabeledPoint(y.\_1,y.\_2))

// training the model

val rdg = new LassoWithSGD()

rdg.optimizer.setNumIterations(numIterations).

setRegParam(0.01).

setStepSize(0.0001)

val model = rdg.run(training2)

val valuesAndPreds = test2.map { point =>

val prediction = model.predict(point.features)

(point.label, prediction)

}

val metrics = new RegressionMetrics(valuesAndPreds)

val MSE = valuesAndPreds.map{case(v, p) => math.pow((v - p), 2)}.mean()

val meanPred = valuesAndPreds.map{case(v, p) => p}.mean()

val numerator = valuesAndPreds.map{case(v, p) => math.pow((meanPred - p), 2)}.sum()

val varian = numerator.toFloat/valuesAndPreds.count()

println("training Mean Squared Error = " + MSE)

println("Variance:" + varian)

println("RME: "+ metrics.rootMeanSquaredError)

println("Explained Variance: "+metrics.explainedVariance)

}

}

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |