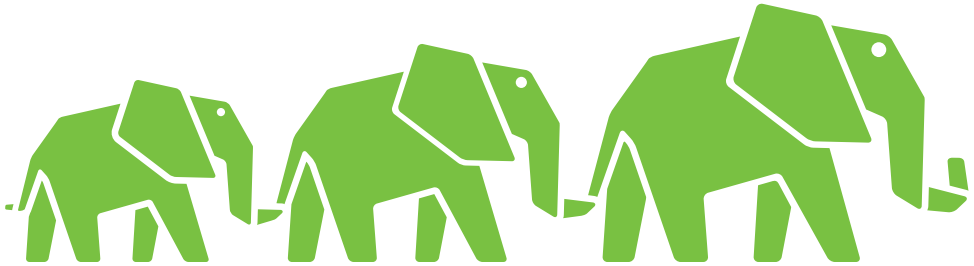

Hortonworks

Spam Classification with Mllib LAB

A Hortonworks University
Hadoop Training Course

Title: LAB GUIDE: Data Science for the Hortonworks Data Platform
Revision 2

Lab: Spam Classifier using Spark MLlib

Objective:	Become familiar with using Spark MLlib to run data science algorithms on a Hadoop cluster.
Successful Outcome:	You will have created a spam classifier with MLlib.
Before You Begin:	Your HDP cluster should be up and running in the classroom VM.

1.1. Before we can begin this lab, we need to set up our environment. Run the following commands from the terminal:

>>> sudo yum install numpy
>>> wget https://www.dropbox.com/s/wpfpl92wrwj58oo/SPAMTrain.txt?dl=1
>>> mv file\?dl\=1 spark/data/spamEmail/SPAMTrain.txt

1.2. Exit your current pyspark instance and restart a new pyspark instance with the following command, this will ensure we have the right amount of resources available for this job:

>>>pyspark --master yarn-client --executor-memory 2g --num-executors 2 --driver-memory 1g

1.3. The labeled data is in your VM locally at

/root/spark/data/spamEmail

The files themselves are not labeled as either spam or not, but there is a separate file, SPAMTrain.txt, that has the label 1 for spam, 0 for not-spam for each filename. It is in the directory "label" at the same path.

Create empty lists to hold the spam/non-spam file names.

```
spamFiles = []
notSpamFiles = []
```

1.4. Read the SPAMTrain.txt file to find out if the file in question is labeled spam or not: spam is labeled 0 , non-spam is labeled 1

Add this code creating the spamFiles and notSpamFiles lists

```
spamFiles=[]
notSpamFiles=[]
f=open('/root/spark/data/spamEmail/SPAMTrain.txt', 'r')
for line in f:
    if int(line[0]) == 1:
        r = line[2:]
        notSpamFiles.append('/root/spark/data/spamEmail/'+r.
rstrip('\n'))
    elif int(line[0]) == 0:
        r = line[2:]
        spamFiles.append('/root/spark/data/spamEmail/'+r.rst
rip('\n'))
print len(notSpamFiles)
print len(spamFiles)
```

```
2949
1378
```

1.5. Create a list with the contents of the files in spamFiles, and for the nonSpamFiles

```
spams = []
for file in spamFiles:
    f = open(file,"r")
    spams.append(f.read())
print len(spams)

notSpams = []
for file in notSpamFiles:
    g = open(file,"r")
    notSpams.append(g.read())
print len(notSpams)
```

Step 1: Import MLlib modules to do Logistic Regression as our classifier. Logistic regression requires as input a labeled training set in the form of a vector of doubles. We'll also make use of the regression library's "LabeledPoint" to prepare our data in this format.

```
from pyspark import SparkContext, SparkConf
from pyspark.mllib.regression import LabeledPoint
from pyspark.mllib.feature import HashingTF
from pyspark.mllib.classification import
    LogisticRegressionWithSGD
```

Step 2: Create training data RDDs

2.1. Create a spam emails object

```
spams = sc.parallelize(spams)
```

2.1. And create a non-spam/normal emails object

```
nonspams = sc.parallelize(notSpams)
```

Step 3: Create a HashingTF instance that we will use to map the email text to uniformly length vectors all with 1,000,000 features (i.e. S =1000000)

```
HTF = HashingTF(numFeatures = 1000000)
```

Step 4: Each email is split into words, and each word is mapped to one feature, either spam or normal.

```
spamHTF = spams.map(lambda x: HTF.transform(x.split(" ")))
nonspamHTF = nonspams.map(lambda x: HTF.transform(x.split("
    ")))
```

Step 5: Create LabeledPoint datasets for positive (spam) and negative (normal) examples.

5.1. The positive and negative LabeledPoint objects are key/value

```
positives = spamHTF.map(lambda x: LabeledPoint(1, x))
negatives = nonspamHTF.map(lambda x: LabeledPoint(0, x))
```

5.2. Union the datasets together into our labeled training data

```
trainingSet = positivs.union(negatives)
```

The last lined is cached since Logistic Regression is an iterative algorithm.

```
trainingSet.take(1)
```

Your result should look something like

```
[LabeledPoint(1.0, (1000000,[5145,8599,17324,28104,28831,28863,28906,28923,28945,28991,31271,33519,335
70,33779,37344,40616,42639,52544,56612,57751,61882,66271,70658,75935,81559,83163,83901,84397,86869,878
61,89421,89991,92710,93676,94330,95543,95817,98176,98178,98197,98205,98336,100050,108076,111948,111954
,112360,113217,113245,115378,115521,115952,116671,116714,118220,118221,118233,118243,118285,118425,118
570,118872,120207,121162,121165,121175,121183,124976,125191,125984,126012,127015,128289,128297,128317,
132916,133937,135213,142758,153983,156479,161885,164701,164751,165666,167169,174767,180391,183777,1848
67,185729,191870,193699,199901,201225,201258,203215,207581,207619,207923,208345,208547,208900,209106,2
16292,216334,216479,216518,216587,226492,229656,231106,231285,231304,234566,235357,237100,237798,23872
9,239238,240846,242219,247650,250447,257615,272370,275543,277230,280718,281190,285491,287071,291510,29
2448,301000,301080,301344,303417,305748,305754,305829,309097,309179,323172,324276,325107,326918,333218
,340374,345688,346507,348857,350893,350901,351024,351034,351589,356233,363588,370868,373646,377210,379
```

Step 6: Run Logistic Regression using the SGD algorithm. Create a model trained on the prepared dataset.

```
classifier = LogisticRegressionWithSGD.train(trainingSet)
```

Step 7: Test on a positive example (spam) and a negative one (normal). We first apply the same HashingTF feature transformation to get vectors (i.e. tf.transform), then apply the model (i.e. model.predict).

```
positiveResult = HTF.transform("Poker for money against real
    playersGet your favorite Poker action!".split(" "))
negativeRessult = HTF.transform("Please report to
    principal's office...".split(" "))
print " Positive test prediction: %g" %
    classifier.predict(positiveResult)
print " Negative test prediction: %g" %
    classifier.predict(negativeResult)
```