Spark Machine Learning Library (mllib package)

SDSC SAN DIEGO SUPERCOMPUTER CENTER

UC San Diego

MLLIB:

- Spark package for Machine Learning with resilient distributed datasets (RDD)
- Partly based on Python’s ‘scipy’ package
- Some key ML algorithms (and growing)

[screen 0: bash] etrain102@comet-05-25:~/SI2016ML/SPML

```
Kill is control-H (^H).
[etrain102@comet-05-25 ~]$ ls
SI2016ML  vis
[etrain102@comet-05-25 ~]$ cd SI2016ML/SPML/
[etrain102@comet-05-25 SPML]$ source Setup_pyspark.sh
```

Enter:

\> pyspark

after startup logs …

```
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /__ / .__/\_,_/_/ /_/\_\   version 1.3.0
      /_/

Using Python version 2.6.6 (r266:84292, Feb 22 2013 00:00:18)
SparkContext available as sc, HiveContext available as sqlCtx.

In [1]:
```

Spark MLLIB Data Types

- **MLLIB works with RDD of:**
  - **Arrays**
  - **Vectors**
  - **Labeled Points**

- **Numpy package: Arrays**

```
import numpy as np


x =  np.array([1,2,3,4])
x[0]
Out[]: 1
```

- **Array of arrays**

```
x = np.array([[1,2],[3,4]])

x[0]
Out[]: array([1, 2])

X[:,1]
Out[]: array([2,4])
```

A row

A column

- **MLLIB package: Vectors**

```
from pyspark.mllib.linalg import Vectors

x =  Vectors.dense([1,2,3,4])
x[0]
Out[]: 1
```

numpy arrays are interchangeable with mllib Vectors

- **MLLIB package: RDD of Vectors**

```
x = [ Vectors.dense([1,2,3,4]),
      Vectors.dense([5,6,7,8])]
xrdd =  sc.parallelize(x)
xrdd
Out[]: <Python RDD …. >
```

now ‘xrdd’ has RDD actions available

- **MLLIB linalg package notes:**

SparseVectors also possible

Distributed Matrix support in later pyspark releases
(but some available in Scala)

- **MLLIB package: LabeledPoint**

```
from pyspark.mllib.regression import LabeledPoint

my_pt = LabeledPoint(1.0, Vectors.dense([1.0, 0.0, 3.0]))

my_pt.label
Out[]: 1.0
my_pt.features
Out[]: [1.0, 0.0, 3.0]
```

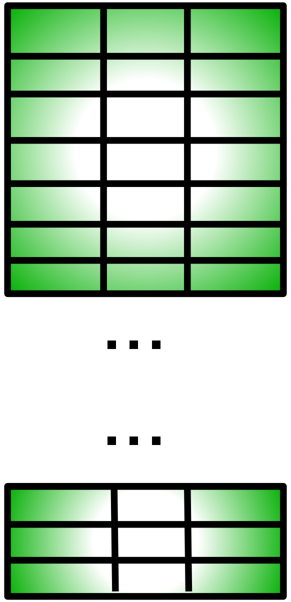
Class Label

Array

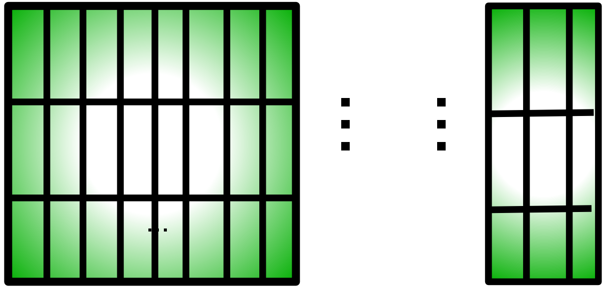
use this for setting up a class variable

RDD partitioning is along rows:

Large number of rows



Large number of Columns



Assume: data partitioned on rows, and 1 row fits in 1 computer memory

Spark MLLIB Clustering

- **MLLIB package: Kmeans**

Assign points to ‘closest’ cluster mean,

Update cluster mean,

Iterate until assignments converge

Needs to iterate over data,

and calculate distance to cluster centers

- Generating Random data:

```
from pyspark.mllib.random import RandomRDDs


c1_v=RandomRDDs.normalVectorRDD(
       sc,20,2,
       numPartitions=2,
       seed=1L).
      map(lambda v:np.add([1,5],v))
```

normal distribution

20 rows, 2 columns

center points around [1,5] by adding [1,5] to each point

- **Generating Random data:**

Ask for stats of the RandomRDD

```
print c1_v.stats()

Out[]: (count: 20, mean: [ 1.15426378  4.90223615],
stdev: [ 1.10801385  1.40049822],
max: [ 3.01083638  8.46783831],
min: [-1.08338413  2.83934928])
```

You get basic stats by column

- **Generate 2 more classes and concatenate:**

```
c2_v=RandomRDDs  ….   np.add([5,1],v))

c3_v=RandomRDDs  …    np.add([4,6],v))

c12        =c1_v.union(c2_v)
my_data=c12.union(c3_v)

```

- **MLLIB package:Kmeans**

```
from pyspark.mllib.cluster import Kmeans, Kmeansmodel
```

- **MLLIB package:Kmeans**

```
my_kmmodel = KMeans.train(my_data,
        k=3,
        maxIterations=10,
        runs=2,
        initializationMode='k-means||')
```

number of clusters

use k-means over small, sample
to initialize (other option is ‘random’)

- **Kmeans model functions:**

```
#Sum Square Error of points to their cluster's center
my_kmmodel.computeCost(my_data)

# get cluster centers
my_kmmodel.clusterCenters

Out: [array([ 5.0476959 ,  1.27729209]), array([ 3.99839705,  6.28073879]), array([ 0.95767935,  4.69770646])]
```

Note: with big data you sometimes only keep the cluster centers for further analysis

Spark MLLIB Classification

- **MLLIB package: Decision Tree**

Decision Tree induction:

At each node, partition data into bins based on attribute values

- **MLLIB package: Decision Tree**

Decision Tree induction:

At each node, partition data into bins based on attribute values

Needs to iterate over data,

collect metrics,

choose nodes,

update current tree across

weather data example

```
import numpy as np
from pyspark.mllib.linalg import Vectors
from pyspark.mllib.regression import LabeledPoint
```

Import modules

```
rawdata=[
        ['sunny',85,85,'FALSE',0
        ['sunny',80,90,'TRUE',0],
        ['overcast',83,86,'FALSE',1],
        ['rainy',70,96,'FALSE',1],....
```

Raw data

```
data_df=sqlContext.createDataFrame(rawdata,
                                    ['outlook','temp','humid','windy','play'])

#make RDD of labeled vectors  (ie using only numbers!)
#  build a dictionary to map outlook to new values
out2index={'sunny':0,'overcast':1,'rainy':2}

def newrow(dfrow):
    outnum = out2index.get(dfrow[0])
    outrow=list([outnum])
    outrow.append(dfrow[1])   #temp
                       ….
     return (LabeledPoint(dfrow[4],outrow))

datax_rdd=data_df.map(newrow)
```

Convert categorical data (e.g. outlook= 1,2 or 3) and make labeled points

- **MLLIB package:DecisionTree**

```
from pyspark.mllib.classification import
DecisionTree
```

- MLLIB package:DecisionTree

```
from pyspark.mllib.classification import  DecisionTree

my_dt = DecisionTree.trainClassifier(data_rdd,
        2,
        {0:3,3:2},
        impurity='entropy',
        maxDepth=5,
        maxBins=32,
        minInstancesPerNode=2)
```

2 classes,
Variable:Number of Categories

controls tree size and splitting (these
may need cross validation to optimize)

Confusion Matrix:

```
predictions = dt_model.predict(datax_rdd.map(lambda
                                           x: x.features))

labelsAndPredictions = datax_rdd.map(lambda lp:
                                  lp.label).zip(predictions

Confusion_mat=  [[ 5.  0.]
                 [ 2.  7.]]
```

12 of 14 correct

Decision Tree output:

```
print dt_model.toDebugString()
```

IF-THEN-ELSE
rules are nodes

```
Out[]: DecisionTreeModel classifier of depth 3 with 9 nodes
  If (feature 2 <= 80.0)
    If (feature 1 <= 65.0)
          Predict: 0.0
    Else (feature 1 > 65.0)
          Predict: 1.0
     Else (feature 2 > 80.0)
     If (feature 0 in {0.0})
         ...
```

feat 2 is humid,
feat 1 is 'temp'

leaf node is
prediction

pause