Spark Machine Learning Library (mllib package)

Setup

MLLIB:

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

Set up:

- Top left menu, Open a terminal:
  Applications => System Tools => Terminal

- Type:

sudo easy_install numpy==1.4.1

- Hit enter

Open PySpark

```
PYSPARK_DRIVER_PYTHON=ipython 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])
```

A row

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

A column

- MLLIB package: Vectors

```
from pyspark.mllib.linalg import Vectors

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

numpy arrays 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 releases

- 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

Example: document-word

Document ID Document Text

1,A long time ago in a galaxy far ...

2,Another episode of star …

3,There are far and away many stars …

4,A galloping horse using two coconuts..

5,My kingdom for a horse …

…

Goal: Make Doc-Word RDD of Vectors

1 column per word

Word counts

1 row per Document

long	far	star	horse	many	…
1	2	1	0	0	
1	1	1	0	0	
0	0	1	0	1	
1	0	0	2	0	
…					

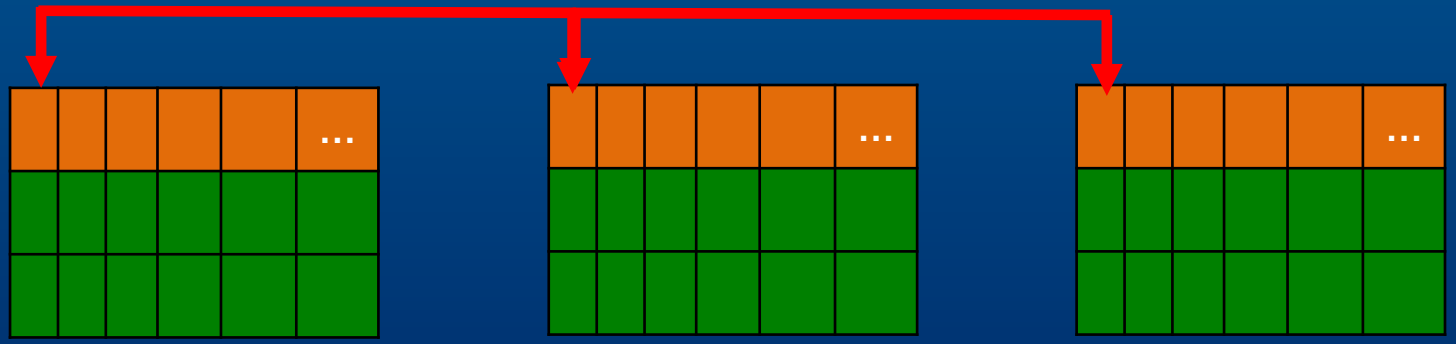
- Get Data into RDD
  - Use Spark DataFrames and RDD maps functions

i.e. groupby <doc,word> and get counts

(details covered in course 3,
see attachment)

Note: <doc-word> key ensures row data is at a partition

Issue: Word columns have to match across partitions



- ‘reshaping’ row values into columns
  - Python has ‘pivot’ function

But not for RDD!

- Get Word counts into columns for RDD

1. Create global word-to-col_index list

2. Create a function to make a Vector entry for each

   <doc, word> , count

3. Map DataFrame into RDD of Vectors

Spark MLLIB Classification

- MLLIB package: NaiveBayes

NaïveBayes:

estimate most likely class
using probabilities

Can work with one pass thru data

- MLLIB package:NaiveBayes

```
from pyspark.mllib.classification import  NaiveBayes
```

model object

train function

RDD of labeled points

```
my_nbmodel =
            NaiveBayes.train(data_rdd)
```

No parameters necessary

- MLLIB package: NaiveBayes

model object

predict function

RDD of vectors

```
test_pred = my_nbmodel.predict(test_point.features)
```

Class prediction returned

- MLLIB package: NaiveBayes

```
my_nbmodel.theta
```

returns log of P(attribute|class)

```
my_nbmodel.pi
```

returns log of class priors

Not that useful as probability estimates, just discrimination

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

Convert categorical data (e.g. ‘sunny’ = [1 0 0], etc..) and make labeled points

Reading attachment has code sample

Confusion Matrix:

Loop over training points

```
predctn = my_nbmodel.predict(train_pt.features)

cf_mat[train_pt.label][predctn]+=1
```

Save 2x2 matrix: true-class X predicted-class

```
cf_mat=  [[ 3.  2.]
          [ 0.  9.]]
```

12 of 14 correct

- 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 computers

- MLLIB package:DecisionTree

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

- MLLIB package:DecisionTree

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

Number of classes

```
my  dt = DecisionTree.trainClassifier(data_rdd, numClasses=2,
          impurity='entropy',
          maxDepth=5,
          maxBins=32,
          minInstancesPerNode=2)
```

'gini' or 'entropy'

controls tree size,
may need cross validation
to optimize

- MLLIB package:DecisionTree

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

Number of classes, or do Regression

```
my_dt = DecisionTree.trainClassifier(data_rdd,
        impurity='entropy',
        maxDepth=5,
        maxBins=32,
        minInstancesPerNode=2)
```

Confusion Matrix:

```
predctn = my_dt.predict(train_pt.features)

cf_mat[train_pt.label][predctn]+=1
```

Also, 12 of 14 correct
(but a bit different than NaiveBayes)

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

Decision Tree output:

IF-THEN-ELSE
rules are nodes

```
print dt_model.toDebugString()

Out[]: DecisionTreeModel classifier of depth 3 with 9 nodes
  If (feature 1 <= 0.0)
    If (feature 4 <= 80.0)
      If (feature 3 <= 68.0)
        Predict: 0.0
      Else (feature 3 > 68.0)
        Predict: 1.0
        …
        …
```

feat 1 is ‘sunny’

leaf node is prediction