Week10 - Clustering With Spark

November 1, 2015

1 Week 10 - Clustering with Spark

Assignment is presented in three sections. The first section shows how to do a basic job of coutning primes using Spark and Python. I used this link as a guide on how to setup PySpark and IPython notebook integration, and for the code regarding primes.

The second section is a basic K-Means clustering example. The code here is from the example on this page.

The last section basically runs the K-Means algorithm on the data from the previous weeks and presents results

1.1 Intro and Basic Setup

In [112]:
```
# Create PySpark context
from pyspark import  SparkContext
sc = SparkContext( 'local', 'pyspark')
```

In [97]:
```
# Simple example of distributed computation of the number of primes till 1000000
def isprime(n):
    """
    check if integer n is a prime
    """
    # make sure n is a positive integer
    n = abs(int(n))
    # 0 and 1 are not primes
    if n < 2:
        return False
    # 2 is the only even prime number
    if n == 2:
        return True
    # all other even numbers are not primes
    if not n & 1:
        return False
    # range starts with 3 and only needs to go up the square root of n
    # for all odd numbers
    for x in range(3, int(n**0.5)+1, 2):
        if n % x == 0:
            return False
    return True

# Create an RDD of numbers from 0 to 1,000,000
nums = sc.parallelize(xrange(1000000))

# Compute the number of primes in the RDD
print nums.filter(isprime).count()
```

```
78498
```

```
In [98]: # Load in shakespear text file
         text = sc.textFile("shakespeare.txt")
         print text
```

```
MapPartitionsRDD[17] at textFile at NativeMethodAccessorImpl.java:-2
```

```
In [99]: # Define function to take in string an return list of tokens. Could use nltk here
         def tokenize(text):
             return text.split()

         # Use function above and map it to every entry in the RDD
         words = text.flatMap(tokenize)
         print words
```

```
PythonRDD[18] at RDD at PythonRDD.scala:43
```

```
In [100]: # Map each word to a tuple of (word, 1) to indiate that this word appeared once
          wc = words.map(lambda x: (x,1))
          print wc.toDebugString()
```

```
(1) PythonRDD[19] at RDD at PythonRDD.scala:43 []
 |  MapPartitionsRDD[17] at textFile at NativeMethodAccessorImpl.java:-2 []
 |  shakespeare.txt HadoopRDD[16] at textFile at NativeMethodAccessorImpl.java:-2 []
```

```
In [101]: # Reduce the generated keys, using add to add the resulting values together for every key
          from operator import add
          counts = wc.reduceByKey(add)
          print counts.toDebugString()
```

```
(1) PythonRDD[24] at RDD at PythonRDD.scala:43 []
 |  MapPartitionsRDD[23] at mapPartitions at PythonRDD.scala:374 []
 |  ShuffledRDD[22] at partitionBy at NativeMethodAccessorImpl.java:-2 []
 +-(1) PairwiseRDD[21] at reduceByKey at <ipython-input-101-f50e20538e62>:3 []
    |  PythonRDD[20] at reduceByKey at <ipython-input-101-f50e20538e62>:3 []
    |  MapPartitionsRDD[17] at textFile at NativeMethodAccessorImpl.java:-2 []
    |  shakespeare.txt HadoopRDD[16] at textFile at NativeMethodAccessorImpl.java:-2 []
```

```
In [102]: # Save counts to folder wc, and look into output folder
          counts.saveAsTextFile("wc")
          !ls -al wc
```

```
total 6076
drwxrwxr-x 2 james james    4096 Nov  1 15:46 .
drwxrwxr-x 5 james james    4096 Nov  1 15:46 ..
-rw-r--r-- 1 james james 6159858 Nov  1 15:46 part-00000
-rw-rw-r-- 1 james james   48132 Nov  1 15:46 .part-00000.crc
-rw-r--r-- 1 james james       0 Nov  1 15:46 _SUCCESS
-rw-rw-r-- 1 james james       8 Nov  1 15:46 ._SUCCESS.crc
```

```
In [103]: # Show output from word count and stop the spark context
          !head wc/part-00000
          sc.stop()
```

```
(u'kingrichardiii@18311', 1)
(u'troilusandcressida@83747', 1)
(u'considered.', 2)
(u'kinghenryviii@7731', 1)
(u'hamlet@141843', 1)
(u'othello@36737', 1)
(u'kinghenryviii@7732', 1)
(u'othello@36738', 1)
(u'romeoandjuliet@1862', 1)
(u'coriolanus@166868', 1)
```

1.2 Clustering with K-Means Example

Here we do a kmeans clistering of data distributed with the Spark installation. This shows how to use Mllib from Python.

In [104]:
```
from pyspark import  SparkContext
sc = SparkContext('local', 'pyspark')

from pyspark.mllib.clustering import KMeans, KMeansModel
from numpy import array
from math import sqrt

# Load and parse the data
data = sc.textFile("%s/data/mllib/kmeans_data.txt" % os.environ['SPARK_HOME'])
parsedData = data.map(lambda line: array([float(x) for x in line.split(' ')]))

# Build the model (cluster the data)
clusters = KMeans.train(parsedData, 2, maxIterations=10,
        runs=10, initializationMode="random")

# Evaluate clustering by computing Within Set Sum of Squared Errors
def error(point):
    center = clusters.centers[clusters.predict(point)]
    return sqrt(sum([x**2 for x in (point - center)]))

WSSSE = parsedData.map(lambda point: error(point)).reduce(lambda x, y: x + y)
print("Within Set Sum of Squared Error = " + str(WSSSE))

# Save and load model
clusters.save(sc, "myModelPath")
sameModel = KMeansModel.load(sc, "myModelPath")
```

```
Within Set Sum of Squared Error = 0.692820323028
```

In [105]:
```
from pprint import pprint

def show_clusters(clusters):
    ''' Show cluster centers '''
    print("Cluster centers: ")
    pprint(clusters.centers)

def show_predictions(examples):
    ''' Show we can predict new arrays '''
    for example in examples:
        print("%s is clustered to: %s" % (example, clusters.centers[clusters.predict(example)]
```

```
In [106]: show_clusters(clusters)
          show_predictions([ array([1, 1, 1]), array([8, 8, 8]) ])
```

```
Cluster centers:
[array([ 0.1,  0.1,  0.1]), array([ 9.1,  9.1,  9.1])]
[1 1 1] is clustered to: [ 0.1  0.1  0.1]
[8 8 8] is clustered to: [ 9.1  9.1  9.1]
```

1.3 K-Means for Mini Project

Lets do the above but for the data from previous weeks

```
In [107]: # Load and parse the data (make sure to convert to floats and create numpy arrays)
          mini_project_data = sc.textFile('data.csv').map(lambda line: array(map(lambda col: float(col)

          # Show data (only first 10 rows)
          mini_project_data.collect()[0:10]
```

```
Out[107]: [array([ 0.,  0.,  0.,  0.,  0.]),
           array([ 0.08,  0.08,  0.1 ,  0.24,  0.9 ]),
           array([ 0.06,  0.06,  0.05,  0.25,  0.33]),
           array([ 0.1 ,  0.1 ,  0.15,  0.65,  0.3 ]),
           array([ 0.08,  0.08,  0.08,  0.98,  0.24]),
           array([ 0.09,  0.15,  0.4 ,  0.1 ,  0.66]),
           array([ 0.1 ,  0.1 ,  0.43,  0.29,  0.56]),
           array([ 0.15,  0.02,  0.34,  0.4 ,  0.01]),
           array([ 0.2 ,  0.14,  0.35,  0.72,  0.25]),
           array([ 0.  ,  0.  ,  0.5 ,  0.2 ,  0.85])]
```

```
In [108]: # Number of clusters to find
          num_clusters = 10

          # Build the model (cluster the data)
          mini_project_clusters = KMeans.train(mini_project_data, num_clusters, maxIterations=10, runs=
```

```
In [109]: # Show some details of results
          #show_clusters(mini_project_clusters)
          mini_project_week10_clusters = pd.DataFrame(mini_project_clusters.centers)
          mini_project_week10_clusters.sort([0])
          #mini_project_week10_clusters
```

```
Out[109]:          0         1         2         3         4
          9  0.238562  0.211250  0.690000  0.271875  0.226875
          4  0.240167  0.287867  0.186667  0.311000  0.250000
          5  0.250970  0.237879  0.371970  0.241818  0.699394
          1  0.252963  0.408111  0.273333  0.713333  0.194074
          7  0.309179  0.375321  0.676429  0.698929  0.235714
          3  0.349778  0.684444  0.596667  0.240741  0.654556
          2  0.407344  0.276469  0.728437  0.286875  0.637812
          8  0.576667  0.213625  0.395417  0.475417  0.225417
          6  0.582952  0.494571  0.607143  0.833810  0.681905
          0  0.620200  0.376500  0.192500  0.327500  0.722000
```

Lets compare this to the R output from Week9:

```
In [110]: import pandas as pd
          mini_project_week9_clusters = pd.DataFrame.from_csv("cluster_centers_week9.csv")
          mini_project_week9_clusters
```

Out[110]:

	X0	X0.1	X0.2	X0.3	X0.4
1	0.200000	0.221300	0.620500	0.414000	0.150000
2	0.207593	0.252593	0.638148	0.307407	0.716296
3	0.256842	0.177632	0.341579	0.751579	0.244211
4	0.272735	0.512324	0.315588	0.565294	0.227353
5	0.337025	0.285325	0.213875	0.234250	0.626500
6	0.373586	0.671724	0.575172	0.242759	0.669759
7	0.470348	0.333739	0.754783	0.761739	0.454783
8	0.527621	0.250759	0.675862	0.217931	0.480000
9	0.545773	0.248545	0.278636	0.453182	0.248182
10	0.682643	0.601071	0.397143	0.810000	0.662143

We get somewhat consistent results. If we look at the clusters side by side, we can see the first row of week10's clusters maps to week9's second row, and vice versa. The 8th cluster (index 0) in week 10 maps to the 8th cluster (index 8) as well. The 7th row (index 2) in week 10 matches to row 6 in week9. Some of the others are harder to match. The fact that there doesn't seem to be more of an alignment (unless I am missing something) suggests that maybe I should reduce k in both situations, as the other clusters might be 'superfluous'

```
In [111]: sc.stop()
```