Data, you have the bridge

About

Using combineByKey in Apache-Spark

Oct 11, 2014

Aggregating data is a fairly straight-forward task, but what if you are working with a distributed data set, one that does not fit in local memory?

In this post I am going to make use of key-value pairs and Apache-Spark's `combineByKey` method to compute the average-by-key. Aggregating-by-key may seem like a trivial task, but it happens to play a major role in the implementation of algorithms such as KMeans, Naive Bayes, and TF-IDF. More importantly, implementing algorithms in a distributed framework such as Spark is an invaluable skill to have.

Average By Key

The example below uses data in the form of a list of key-value tuples: `(key, value)`. I turn that list into a Resilient Distributed Dataset (RDD) with `sc.parallelize`, where `sc` is an instance of `pyspark.SparkContext`.

The next step is to use `combineByKey` to compute the sum and count for each key in `data`. Admittedly, using three lambda-functions as arguments to `combineByKey` makes the code difficult to read. I will explain each lambda-function in the next section. The result, `sumCount`, is an RDD where its values are in the form of `(label, (sum, count))`.

To compute the average-by-key, I use the `map` method to divide the sum by the count for each key.

Finally, I use the `collectAsMap` method to return the average-by-key as a dictionary.

```
data = sc.parallelize( [(0, 2.), (0, 4.), (1, 0.), (1, 10.), (1, 20.)] )

sumCount = data.combineByKey(lambda value: (value, 1),
                             lambda x, value: (x[0] + value, x[1] + 1),
                             lambda x, y: (x[0] + y[0], x[1] + y[1]))

averageByKey = sumCount.map(lambda (label, (value_sum, count)): (label, value_sum / count))

print averageByKey.collectAsMap()
```

Result:

```
{0: 3.0, 1: 10.0}
```

See here for the above example as an executable script.

The combineByKey Method

In order to aggregate an RDD's elements in parallel, Spark's `combineByKey` method requires three functions:

- `createCombiner`
- `mergeValue`
- `mergeCombiner`

Create a Combiner

```
lambda value: (value, 1)
```

The first required argument in the `combineByKey` method is a function to be used as the very first aggregation step for each key. The argument of this function corresponds to the value in a key-value pair. If we want to compute the sum and count using `combineByKey`, then we can create this “combiner” to be a tuple in the form of `(sum, count)`. The very first step in this aggregation is then `(value, 1)`, where `value` is the first RDD value that `combineByKey` comes across and `1` initializes the count.

Merge a Value

```
lambda x, value: (x[0] + value, x[1] + 1)
```

The next required function tells `combineByKey` what to do when a combiner is given a new value. The arguments to this function are a combiner and a new value. The structure of the combiner is defined above as a tuple in the form of `(sum, count)` so we merge the new value by adding it to the first element of the tuple while incrementing `1` to the second element of the tuple.

Merge two Combiners

```
lambda x, y: (x[0] + y[0], x[1] + y[1])
```

The final required function tells `combineByKey` how to merge two combiners. In this example with tuples as combiners in the form of `(sum, count)`, all we need to do is add the first and last elements together.

Compute the Average

```
averageByKey = sumCount.map(lambda (label, (value_sum, count)): (label, value_sum / count))
```

Ultimately the goal is to compute the average-by-key. The result from `combineByKey` is an RDD with elements in the form `(label, (sum, count))`, so the average-by-key can easily be obtained by using the `map` method, mapping `(sum, count)` to `sum / count`.

Note: I do not use `sum` as variable name in the code because it is a built-in function in Python.

Learn More

To learn more about Spark and programming with key-value pairs in Spark, see:

- Spark Documentation Overview
- Spark Programming Guide
- O'Reilly: Learning Spark, Chapter 4
- PySpark Documentation

For an example of using the above calculation in a PySpark implementation of KMeans, see:

- Lloyd's Algorithm in PySpark: sparkmeans.py

Data, you have the bridge

Data, you have the bridge
adam.b.shinn@gmail.com

abshinn
abshinn

Data Scientist at kWh Analytics, Inc.