June 2, 2015

Bhushan Kothari

Santa clara university

bkkothari@scu.edu

Market Basket Analysis

using pyspark

**Market Basket Analysis**

It’s a modelling technique, based upon theory that if you buy certain item, you are mor or less likely to buy certain set of items.

**Important terms:**

**Itemsets**: The set of items bought by the customer are called itemsets.

**Support Factor**: Probability that the customer will buy x item is called the support for that rule. If I is a set of items, the support for I is the number of baskets for which I is a subset.

**Confidence**: Probability that customer buying x, will also buy y is called its confidence.

Combinations is an inbuilt python library to create the combination of n elements.

***from itertools import combinations***

We read the input file using the textFile() method. ***inputRDD = sc.textFile('Documents/mBasket\_IP')***

We split the input on ‘ ‘. ***transactions = inputRDD.map(lambda x:(x.split(' ')))***

[[u'beer', u'vodka'], [u'pizza', u'beer', u'chicken', u'soda'], [u'pizza', u'coke', u'bread'], [u'bread', u'milk', u'meat'], [u'banana', u'milk', u'bread'], [u'banana', u'apples', u'lettuce', u'beer'], [u'coke', u'chicken', u'vodka'], [u'cracker', u'icecream', u'beer'], [u'chicken', u'pizza', u'coke', u'bread'], [u'baguette', u'soda', u'hering', u'cracker', u'beer', u''], [u'bourbon', u'coke', u'turkey'], [u'sardines', u'beer', u'chicken', u'coke'], [u'apples', u'peppers', u'avocado', u'steak'], [u'sardines', u'apples', u'peppers', u'avocado', u'steak']]

All the records are then sorted so that only one combination of (A,B) is created instead of (A,B) and (B,A)

***sortedTransactions = transactions.sortBy(lambda x: x)  
sortedTransactions = sortedTransactions.map(lambda x:sorted(x))***

[[u'apples', u'avocado', u'peppers', u'steak'], [u'', u'baguette', u'beer', u'cracker', u'hering', u'soda'], [u'apples', u'banana', u'beer', u'lettuce'], [u'banana', u'bread', u'milk'], [u'beer', u'vodka'], [u'bourbon', u'coke', u'turkey'], [u'bread', u'meat', u'milk'], [u'bread', u'chicken', u'coke', u'pizza'], [u'chicken', u'coke', u'vodka'], [u'beer', u'cracker', u'icecream'], [u'beer', u'chicken', u'pizza', u'soda'], [u'bread', u'coke', u'pizza'], [u'apples', u'avocado', u'peppers', u'sardines', u'steak'], [u'beer', u'chicken', u'coke', u'sardines']]

Now we create the combinations of 2 items using the following syntax. ***itemSets = sortedTransactions.map(lambda x: list(combinations(x,3)))***

***itemSets.collect()***

[[(u'apples', u'avocado', u'peppers'), (u'apples', u'avocado', u'steak'), (u'apples', u'peppers', u'steak'), (u'avocado', u'peppers', u'steak')], [(u'', u'baguette', u'beer'), (u'', u'baguette', u'cracker'), (u'', u'baguette', u'hering'), (u'', u'baguette', u'soda'), (u'', u'beer', u'cracker'), (u'', u'beer', u'hering'), (u'', u'beer', u'soda'), (u'', u'cracker', u'hering'), (u'', u'cracker', u'soda'), (u'', u'hering', u'soda'), (u'baguette', u'beer', u'cracker'), (u'baguette', u'beer', u'hering'), (u'baguette', u'beer', u'soda'), (u'baguette', u'cracker', u'hering'), (u'baguette', u'cracker', u'soda'), (u'baguette', u'hering', u'soda'), (u'beer', u'cracker', u'hering'), (u'beer', u'cracker', u'soda'), (u'beer', u'hering', u'soda'), (u'cracker', u'hering', u'soda')], [(u'apples', u'banana', u'beer'), (u'apples', u'banana', u'lettuce'), (u'apples', u'beer', u'lettuce'), (u'banana', u'beer', u'lettuce')], [(u'banana', u'bread', u'milk')], [], [(u'bourbon', u'coke', u'turkey')], [(u'bread', u'meat', u'milk')], [(u'bread', u'chicken', u'coke'), (u'bread', u'chicken', u'pizza'), (u'bread', u'coke', u'pizza'), (u'chicken', u'coke', u'pizza')], [(u'chicken', u'coke', u'vodka')], [(u'beer', u'cracker', u'icecream')], [(u'beer', u'chicken', u'pizza'), (u'beer', u'chicken', u'soda'), (u'beer', u'pizza', u'soda'), (u'chicken', u'pizza', u'soda')], [(u'bread', u'coke', u'pizza')], [(u'apples', u'avocado', u'peppers'), (u'apples', u'avocado', u'sardines'), (u'apples', u'avocado', u'steak'), (u'apples', u'peppers', u'sardines'), (u'apples', u'peppers', u'steak'), (u'apples', u'sardines', u'steak'), (u'avocado', u'peppers', u'sardines'), (u'avocado', u'peppers', u'steak'), (u'avocado', u'sardines', u'steak'), (u'peppers', u'sardines', u'steak')], [(u'beer', u'chicken', u'coke'), (u'beer', u'chicken', u'sardines'), (u'beer', u'coke', u'sardines'), (u'chicken', u'coke', u'sardines')]]

For every combination generated we append 1 to it, to count the number of instances when those products were bought together. ***itemSetAppended = itemSets.flatMap(lambda x:[(x[i],1) for i in range (0,len(x))])  
itemSetCount= itemSetAppended.reduceByKey(lambda x,y: x+y)***

***itemSetCount.collect()***

[((u'apples', u'avocado', u'peppers'), 2), ((u'beer', u'pizza', u'soda'), 1), ((u'', u'beer', u'soda'), 1), ((u'apples', u'beer', u'lettuce'), 1), ((u'baguette', u'cracker', u'hering'), 1), ((u'avocado', u'peppers', u'steak'), 2), ((u'chicken', u'coke', u'pizza'), 1), ((u'beer', u'hering', u'soda'), 1), ((u'avocado', u'peppers', u'sardines'), 1), ((u'baguette', u'beer', u'cracker'), 1), ((u'', u'beer', u'hering'), 1), ((u'baguette', u'cracker', u'soda'), 1), ((u'bread', u'chicken', u'pizza'), 1), ((u'', u'baguette', u'soda'), 1), ((u'peppers', u'sardines', u'steak'), 1), ((u'bread', u'chicken', u'coke'), 1), ((u'apples', u'peppers', u'sardines'), 1), ((u'apples', u'peppers', u'steak'), 2), ((u'chicken', u'coke', u'sardines'), 1), ((u'baguette', u'hering', u'soda'), 1), ((u'beer', u'cracker', u'soda'), 1), ((u'apples', u'avocado', u'sardines'), 1), ((u'', u'cracker', u'soda'), 1), ((u'bourbon', u'coke', u'turkey'), 1), ((u'', u'beer', u'cracker'), 1), ((u'', u'baguette', u'beer'), 1), ((u'apples', u'banana', u'lettuce'), 1), ((u'cracker', u'hering', u'soda'), 1), ((u'banana', u'bread', u'milk'), 1), ((u'beer', u'chicken', u'coke'), 1), ((u'beer', u'cracker', u'icecream'), 1), ((u'apples', u'banana', u'beer'), 1), ((u'baguette', u'beer', u'soda'), 1), ((u'baguette', u'beer', u'hering'), 1), ((u'apples', u'sardines', u'steak'), 1), ((u'beer', u'chicken', u'sardines'), 1), ((u'avocado', u'sardines', u'steak'), 1), ((u'', u'cracker', u'hering'), 1), ((u'banana', u'beer', u'lettuce'), 1), ((u'beer', u'coke', u'sardines'), 1), ((u'chicken', u'coke', u'vodka'), 1), ((u'beer', u'chicken', u'pizza'), 1), ((u'', u'baguette', u'cracker'), 1), ((u'bread', u'coke', u'pizza'), 2), ((u'chicken', u'pizza', u'soda'), 1), ((u'', u'baguette', u'hering'), 1), ((u'beer', u'chicken', u'soda'), 1), ((u'', u'hering', u'soda'), 1), ((u'bread', u'meat', u'milk'), 1), ((u'apples', u'avocado', u'steak'), 2), ((u'beer', u'cracker', u'hering'), 1)]

We use the sortByKey to sort the input in descending order. ***itemSetCntDesc = itemSetCount.sortBy(lambda x: x[1],False)***

***itemSetCntDesc.collect()***

[((u'apples', u'avocado', u'peppers'), 2), ((u'avocado', u'peppers', u'steak'), 2), ((u'apples', u'peppers', u'steak'), 2), ((u'bread', u'coke', u'pizza'), 2), ((u'apples', u'avocado', u'steak'), 2), ((u'beer', u'pizza', u'soda'), 1), ((u'', u'beer', u'soda'), 1), ((u'apples', u'beer', u'lettuce'), 1), ((u'baguette', u'cracker', u'hering'), 1), ((u'chicken', u'coke', u'pizza'), 1), ((u'beer', u'hering', u'soda'), 1), ((u'avocado', u'peppers', u'sardines'), 1), ((u'baguette', u'beer', u'cracker'), 1), ((u'', u'beer', u'hering'), 1), ((u'baguette', u'cracker', u'soda'), 1), ((u'bread', u'chicken', u'pizza'), 1), ((u'', u'baguette', u'soda'), 1), ((u'peppers', u'sardines', u'steak'), 1), ((u'bread', u'chicken', u'coke'), 1), ((u'apples', u'peppers', u'sardines'), 1), ((u'chicken', u'coke', u'sardines'), 1), ((u'baguette', u'hering', u'soda'), 1), ((u'beer', u'cracker', u'soda'), 1), ((u'apples', u'avocado', u'sardines'), 1), ((u'', u'cracker', u'soda'), 1), ((u'bourbon', u'coke', u'turkey'), 1), ((u'', u'beer', u'cracker'), 1), ((u'', u'baguette', u'beer'), 1), ((u'apples', u'banana', u'lettuce'), 1), ((u'cracker', u'hering', u'soda'), 1), ((u'banana', u'bread', u'milk'), 1), ((u'beer', u'chicken', u'coke'), 1), ((u'beer', u'cracker', u'icecream'), 1), ((u'apples', u'banana', u'beer'), 1), ((u'baguette', u'beer', u'soda'), 1), ((u'baguette', u'beer', u'hering'), 1), ((u'apples', u'sardines', u'steak'), 1), ((u'beer', u'chicken', u'sardines'), 1), ((u'avocado', u'sardines', u'steak'), 1), ((u'', u'cracker', u'hering'), 1), ((u'banana', u'beer', u'lettuce'), 1), ((u'beer', u'coke', u'sardines'), 1), ((u'chicken', u'coke', u'vodka'), 1), ((u'beer', u'chicken', u'pizza'), 1), ((u'', u'baguette', u'cracker'), 1), ((u'chicken', u'pizza', u'soda'), 1), ((u'', u'baguette', u'hering'), 1), ((u'beer', u'chicken', u'soda'), 1), ((u'', u'hering', u'soda'), 1), ((u'bread', u'meat', u'milk'), 1), ((u'beer', u'cracker', u'hering'), 1)]

***totalCount = transactions.count()***

***totalCount  
14***

Now we check for all the rows having confidence greater than or equal to 2. (As our sample data is small) In case of ideal data set we specify the confidence here. For retail industry items are considered to be frequent if they occur in more than 1% of the baskets. Thus we can calculate the total number of baskets using RDD transactions and then divide the support by this total count and check if its percentage value is greater than 1. However as sample data in this case is too small, I have used hardcoded value 2.

**afterConfidence = itemSetCntDesc.filter(lambda x: x[1]>=2)  
afterConfidence.collect()**

[((u'apples', u'avocado', u'peppers'), 2), ((u'avocado', u'peppers', u'steak'), 2), ((u'apples', u'peppers', u'steak'), 2), ((u'bread', u'coke', u'pizza'), 2), ((u'apples', u'avocado', u'steak'), 2)]

Similarly when we run the code for generating frequent itemsets of 2 items we get the following output.

***from itertools import combinations  
inputRDD = sc.textFile('Documents/mBasket\_IP')  
transactions = inputRDD.map(lambda x:(x.split(' ')))  
sortedTransactions = transactions.sortBy(lambda x: x)  
sortedTransactions = sortedTransactions.map(lambda x:sorted(x))  
itemSets = sortedTransactions.map(lambda x: list(combinations(x,2)))  
itemSetAppended = itemSets.flatMap(lambda x:[(x[i],1) for i in range (0,len(x))])  
itemSetCount= itemSetAppended.reduceByKey(lambda x,y: x+y)  
itemSetCntDesc = itemSetCount.sortBy(lambda x: x[1],False)  
totalCount = transactions.count()  
afterConfidence = itemSetCntDesc.map(lambda x:(x[0],float(x[1]/totalCount)))  
afterConfidenceHardcoded = itemSetCntDesc.filter(lambda x: x[1]>=2)  
afterConfidenceHardcoded.collect()***

[((u'chicken', u'coke'), 3), ((u'', u'coke'), 3), ((u'bread', u'milk'), 2), ((u'', u'beer'), 2), ((u'beer', u'soda'), 2), ((u'beer', u'cracker'), 2), ((u'apples', u'steak'), 2), ((u'peppers', u'steak'), 2), ((u'avocado', u'peppers'), 2), ((u'', u'bread'), 2), ((u'apples', u'peppers'), 2), ((u'bread', u'pizza'), 2), ((u'coke', u'pizza'), 2), ((u'avocado', u'steak'), 2), ((u'chicken', u'pizza'), 2), ((u'', u'pizza'), 2), ((u'apples', u'avocado'), 2), ((u'bread', u'coke'), 2), ((u'beer', u'chicken'), 2), ((u'', u'chicken'), 2)]

And for 4:

***from itertools import combinations  
inputRDD = sc.textFile('Documents/mBasket\_IP')  
transactions = inputRDD.map(lambda x:(x.split(' ')))  
sortedTransactions = transactions.sortBy(lambda x: x)  
sortedTransactions = sortedTransactions.map(lambda x:sorted(x))  
itemSets = sortedTransactions.map(lambda x: list(combinations(x,4)))  
itemSetAppended = itemSets.flatMap(lambda x:[(x[i],1) for i in range (0,len(x))])  
itemSetCount= itemSetAppended.reduceByKey(lambda x,y: x+y)  
itemSetCntDesc = itemSetCount.sortBy(lambda x: x[1],False)  
totalCount = transactions.count()  
afterConfidence = itemSetCntDesc.map(lambda x:(x[0],float(x[1]/totalCount)))  
afterConfidenceHardcoded = itemSetCntDesc.filter(lambda x: x[1]>=2)  
afterConfidenceHardcoded.collect()***

[((u'apples', u'avocado', u'peppers', u'steak'), 2)]

Sample Input:

beer vodka  
pizza beer chicken soda  
pizza coke bread   
bread milk meat  
banana milk bread  
banana apples lettuce beer  
coke chicken vodka  
cracker icecream beer  
chicken pizza coke bread   
baguette soda hering cracker beer   
bourbon coke turkey  
sardines beer chicken coke   
apples peppers avocado steak   
sardines apples peppers avocado steak