In [1]:
```
from pyspark import SparkConf, SparkContext
import pyspark
import sys
from collections import defaultdict
import pandas as pd
import numpy as np
from itertools import combinations, permutations
#configure spark
DATA_PATH = "gs://zw2624-bucket/input/subsample_data_3.csv"
K = 10
```

In [2]:
```
conf = SparkConf()
sc = SparkContext.getOrCreate()
```

In [3]:
```
def parseVector(line):
    line = line.split(",")
    return line[0],(line[1],line[2])

def reduceMean(line):
    uid = line[0]
    mean = sum([p[1] for p in line[1]]) / len(line[1])
    return uid, [(p[0], p[1] - mean) for p in line[1]]

def getItemPairs(user_id,items_with_rating):
    for item1,item2 in permutations(items_with_rating, 2):
        yield (item1[0],item2[0]),(item1[1],item2[1])
    return

def adjustedCosine(p):
    item_pair = p[0]
    pair_rating = p[1]
    up = sum(s[0]*s[1] for s in pair_rating)
    down = np.sqrt(sum(s[0]**2 for s in pair_rating) * sum(s[1]**2 for s in pair_rating))
    return item_pair, up / down

def keyOnFirst(p):
    item_pair,item_sim_data = p[0], p[1]
    (item1_id,item2_id) = item_pair
    return item1_id, (item2_id,item_sim_data)

def KNN(item, k):
    item_id = item[0]
    item_data = item[1]
    item_data.sort(key = lambda x: x[1], reverse=True)
    return item_id, item_data[:k]
```

In [4]:
```
data = sc.textFile(DATA_PATH).map(parseVector) \
                             .filter(lambda line: line[1][1] != 'rating') \
                             .map(lambda x: (x[0],(x[1][0],float(x[1][1]))))
```

In [5]:
```
'''
user_id, [(item1,item1_rating-mean), (item2, item2_rating-mean), ...]
'''
user_item_pairs = data.groupByKey().map(reduceMean).cache()
```

In [ ]:
```
'''
(item1,item2), [(item1_rating,item2_rating), (item1_rating,item2_rating), ...]
'''
pairwise_items = user_item_pairs.flatMap(lambda p: getItemPairs(p[0],p[1])) \
                                .groupByKey().cache()
pairwise_items.take(3)
```

In [ ]:
```
'''
(item1,item2), adjusted_cosine_similarity
'''
item_sims = pairwise_items.map(adjustedCosine).cache()
item_sims.take(3)
```

In [8]:
```
item_neighbors = item_sims.map(keyOnFirst) \
                          .groupByKey().map(lambda x : (x[0], list(x[1]))) \
                          .map(lambda x: KNN(x, K)).cache()
```