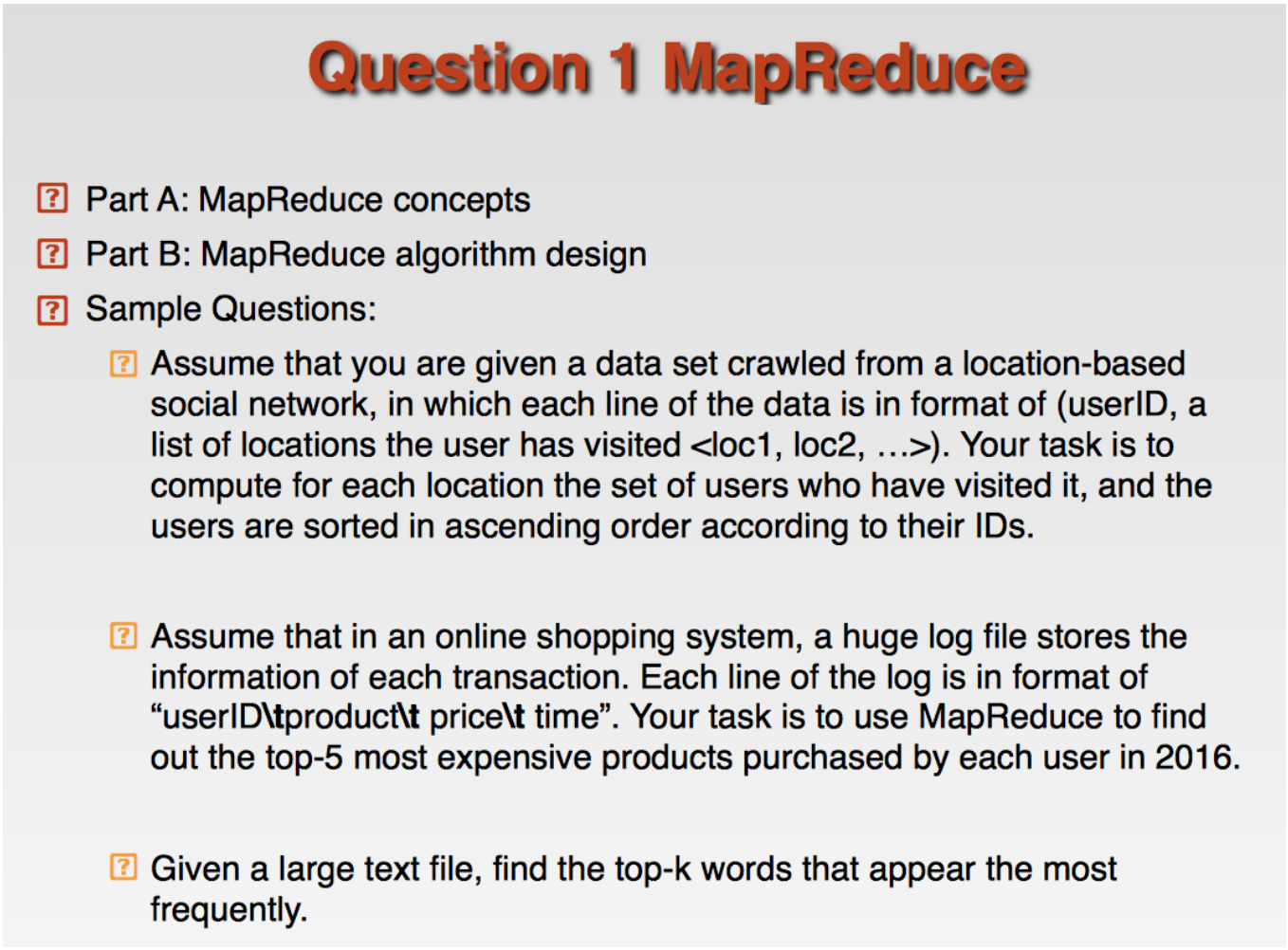
COMP9313 考点复习

`COMP9313` `复习`

- **Author:** 尚学教育
- **18S2 开课时间**
  - 2018.11.09 5:00-9:30
  - 2018.11.11 5:30-10:00

Q1 MapReduce



map

reduce

combiner

- version 1

```
1: class Mapper
2:     method Map(string t, integer r)
3:         Emit(string t, integer r)

1: class Reducer
2:     method Reduce(string t, integers [r_1, r_2, ...])
3:         sum ← 0
4:         cnt ← 0
5:         for all integer r ∈ integers [r_1, r_2, ...] do
6:             sum ← sum + r
7:             cnt ← cnt + 1
8:         r_avg ← sum/cnt
9:         Emit(string t, integer r_avg)
```

- version 2

```
1: class Mapper
2:     method Map(string t, integer r)
3:         Emit(string t, integer r)

1: class Combiner
2:     method Combine(string t, integers [r_1, r_2, ...])
3:         sum ← 0
4:         cnt ← 0
5:         for all integer r ∈ integers [r_1, r_2, ...] do
6:             sum ← sum + r
7:             cnt ← cnt + 1
8:         Emit(string t, pair (sum, cnt))          ▷ Separate sum and count

1: class Reducer
2:     method Reduce(string t, pairs [(s_1, c_1), (s_2, c_2) ...])
3:         sum ← 0
4:         cnt ← 0
5:         for all pair (s, c) ∈ pairs [(s_1, c_1), (s_2, c_2) ...] do
6:             sum ← sum + s
7:             cnt ← cnt + c
8:         r_avg ← sum/cnt
9:         Emit(string t, integer r_avg)
```

ıy doesn't this work? Combiners must have the same input and output type, consistent with the input of reducers (output of mappers)

- version 3

```
1: class Mapper
2:     method Map(string t, integer r)
3:         Emit(string t, pair (r, 1))

1: class Combiner
2:     method Combine(string t, pairs [(s_1, c_1), (s_2, c_2) ...])
3:         sum ← 0
4:         cnt ← 0
5:         for all pair (s, c) ∈ pairs [(s_1, c_1), (s_2, c_2) ...] do
6:             sum ← sum + s
7:             cnt ← cnt + c
8:         Emit(string t, pair (sum, cnt))

1: class Reducer
2:     method Reduce(string t, pairs [(s_1, c_1), (s_2, c_2) ...])
3:         sum ← 0
4:         cnt ← 0
5:         for all pair (s, c) ∈ pairs [(s_1, c_1), (s_2, c_2) ...] do
6:             sum ← sum + s
7:             cnt ← cnt + c
8:         r_avg ← sum/cnt
9:         Emit(string t, pair (r_avg, cnt))
```

- version 4

```
1: class Mapper
2:     method Initialize
3:         S ← new AssociativeArray
4:         C ← new AssociativeArray
5:     method Map(string t, integer r)
6:         S{t} ← S{t} + r
7:         C{t} ← C{t} + 1
8:     method Close
9:         for all term t ∈ S do
10:            Emit(term t, pair (S{t}, C{t}))
```

Q2 Spark

Question 2 Spark

- Part A: Spark concepts
- Part B: Show output of the given code
- Part C: Spark algorithm design
- Sample Questions:
  - What is the output?
    - val lines = sc.parallelize(List("hello world", "this is a scala program", "to create a pair RDD", "in spark"))

      val pairs = lines.map(x => (x.split(" ")(0), x))

      pairs.filter {case (key, value) => key.length <3}.foreach(println)
  - Problem 1 of Lab 8: Compute the average length of words starting with each letter.

A

B output

Create a pair RDD from existing RDDs

```
val pairs = sc.parallelize( List( (“This”, 2), (“is”, 3), (“Spark”, 5), (“is”, 3) ) )
pairs.collect().foreach(println)
```

Output?

reduceByKey() function: reduce key-value pairs by key using give *func*

```
val pair1 = pairs.reduceByKey((x,y) => x + y)
pairs1.collect().foreach(println)
```

mapValues() function: work on values only

```
val pair2 = pairs.mapValues( x => x -1 )
pairs2.collect().foreach(println)
```

groupByKey() function: When called on a dataset of (K, V) pairs, returns a dataset of (K, Iterable<V>) pairs

```
pairs.groupByKey().collect().foreach(println)
```

53

C lab8

Question:

- Write a Spark program which outputs the number of words that start with each letter. This means that for every letter we want to count the total number of words that start with that letter. In your implementation ignore the letter case, i.e., consider all words as lower case. You can ignore all non-alphabetic characters.

Hint: File -> an array of words starting with 'a-z' (flatMap and filter) -> an array of pairs (first letter, 1) (map) -> an array of pairs (first letter, total count) (reduceByKey)

```
1.  package comp9313.lab7
2.
3.  import org.apache.spark.SparkContext
4.  import org.apache.spark.SparkContext._
5.  import org.apache.spark.SparkConf
6.
7.  object LetterCount {
8.    def main(args: Array[String]) {
9.      val inputFile = args(0)
10.     val outputFolder = args(1)
11.     val conf = new SparkConf().setAppName("letter count").setMaster("l
    ocal")
12.     val sc = new SparkContext(conf)
13.     val textFile = sc.textFile(inputFile)
14.     val words = textFile.flatMap(_.split("[\\s*$&#/\"'\\,.:;?!\\[\\](
    {}<>~\\-_]+"))
15.
16.     val counts = words.filter(x => x.length >= 1).map(x => x.toLowerCas
    e).
17.       filter(x => x.charAt(0) <= 'z' && x.charAt(0) >= 'a').map(x => (x
    .charAt(0), 1)).reduceByKey(_+_).sortByKey()
18.
19.       counts.foreach(println)
20.     //counts.saveAsTextFile(outputFolder)
21.   }
22. }
```

```
1.
2.  package comp9313.lab8
3.
4.  import org.apache.spark.SparkContext
```

```scala
import org.apache.spark.SparkContext._
import org.apache.spark.SparkConf

object WordCount {
  def main(args: Array[String]) {
    val inputFile = args(0)
    val outputFolder = args(1)
    val conf = new SparkConf().setAppName("adf")
    val sc = new SparkContext(conf)
    val textFile = sc.textFile(inputFile)
    val words = textFile.flatMap(_.split("[\\s*$&#/\"'\\,.:;?!\\[\\](){}<>~\\-_]+"))

    val counts = words.filter(x => x.length >= 1).map(x => x.toLowerCase).
      filter(x => x.charAt(0) <='z' && x.charAt(0) >='a').map(x => (x.charAt(0), (1, x.length())))

    val aggr = counts.reduceByKey( (a, b) => (a._1+b._1, a._2+b._2)).sortByKey()
    aggr.foreach(x => println(x._1, x._2._2.toDouble/x._2._1))
    //aggr.map(x => (x._1, x._2._2.toFloat/x._2._1)).saveAsTextFile(outputFolder)
    aggr.map(x => s"${x._1},${x._2._2.toDouble/x._2._1}").saveAsTextFile(outputFolder)
  }
}

```

(i). Find the top-5 VoteTypeIds that have the most distinct posts. You need to output the VoteTypeId and the number of posts. The results are ranked in descending order according to the number of posts, and each line is in format of: VoteTypeId\tNumber of posts.

(ii). Find all posts that are favoured by more than 10 users. You need to output both PostId and the list of UserIds, and each line is in format of:

PostId#UserId1,UserId2,UserId3,…,UserIdn

The lines are sorted according to the NUMERIC values of the PostIds in ascending order. Within each line, the UserIds are sorted according to their NUMERIC values in ascending order.

```scala
package comp9313.lab8

import org.apache.spark.SparkContext
import org.apache.spark.SparkContext._
import org.apache.spark.SparkConf
import org.apache.spark.rdd._

object Problem2 {
  var Id = 0
  var PostId = 1
  var VoteTypeId = 2
  var UserId = 3
  var CreationTime = 4

  def Question2(textFile: RDD[Array[String]]) {

    val puPair = textFile.filter(_(VoteTypeId) == "5").map(x => (x(PostId), x(UserId))).distinct

    val res = puPair.groupByKey().filter(_._2.size>10).mapValues(x => x.toList.sortBy(_.toInt)).sortBy(_._1.toInt)
    val fmtres = res.map{case (x, y) => s"""$x#${y.mkString(",")}"""}
    fmtres.foreach(println)
  }

  def Question1(textFile: RDD[Array[String]]) {

    val pvPair = textFile.map(x => (x(VoteTypeId), x(PostId))).distinct
    //val res = pvPair.countByKey().toSeq.sortWith(_._2 > _._2).take(5).map{case (x, y)=> s"$x\t$y"}
    val res = pvPair.countByKey().toSeq.sortWith(_._2 > _._2).take(5)

    res.foreach(x => println(x._1+"\t"+x._2))
  }


  def main(args: Array[String]) {
    val inputFile = args(0)
    val outputFolder = args(1)
    val conf = new SparkConf().setAppName("lab8").setMaster("local")
    val sc = new SparkContext(conf)

    val textFile = sc.textFile(inputFile).map(_.split(","))
    Question1(textFile)
    Question2(textFile)
```

```
43.     }
44.   }
```

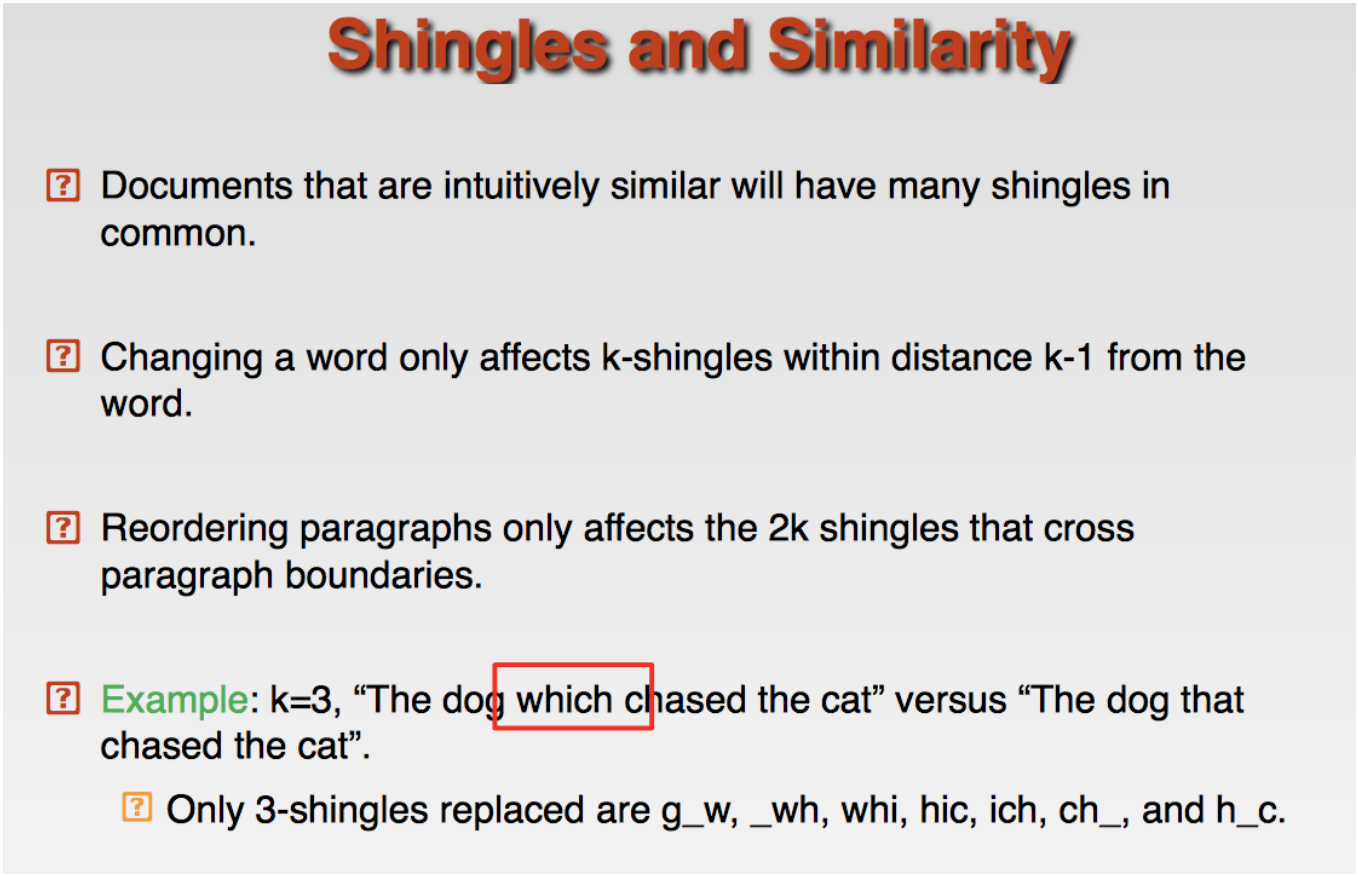
Q3 Finding Simlar items

Question 3 Finding Similar Items

- Shingling
- Min Hashing
- LSH
- Sample Questions:
  - Question 2 in the last assignment

3.1 Shingling

- **Example:** **k=2**; document $\mathbf{D_1}$ = abcab
  Set of 2-shingles: $\mathbf{S(D_1)}$ = {ab, bc, ca}
  - **Option:** Shingles as a bag (multiset), count ab twice: $\mathbf{S'(D_1)}$ = {ab, bc, ca, ab}



3.2 Min Hashing

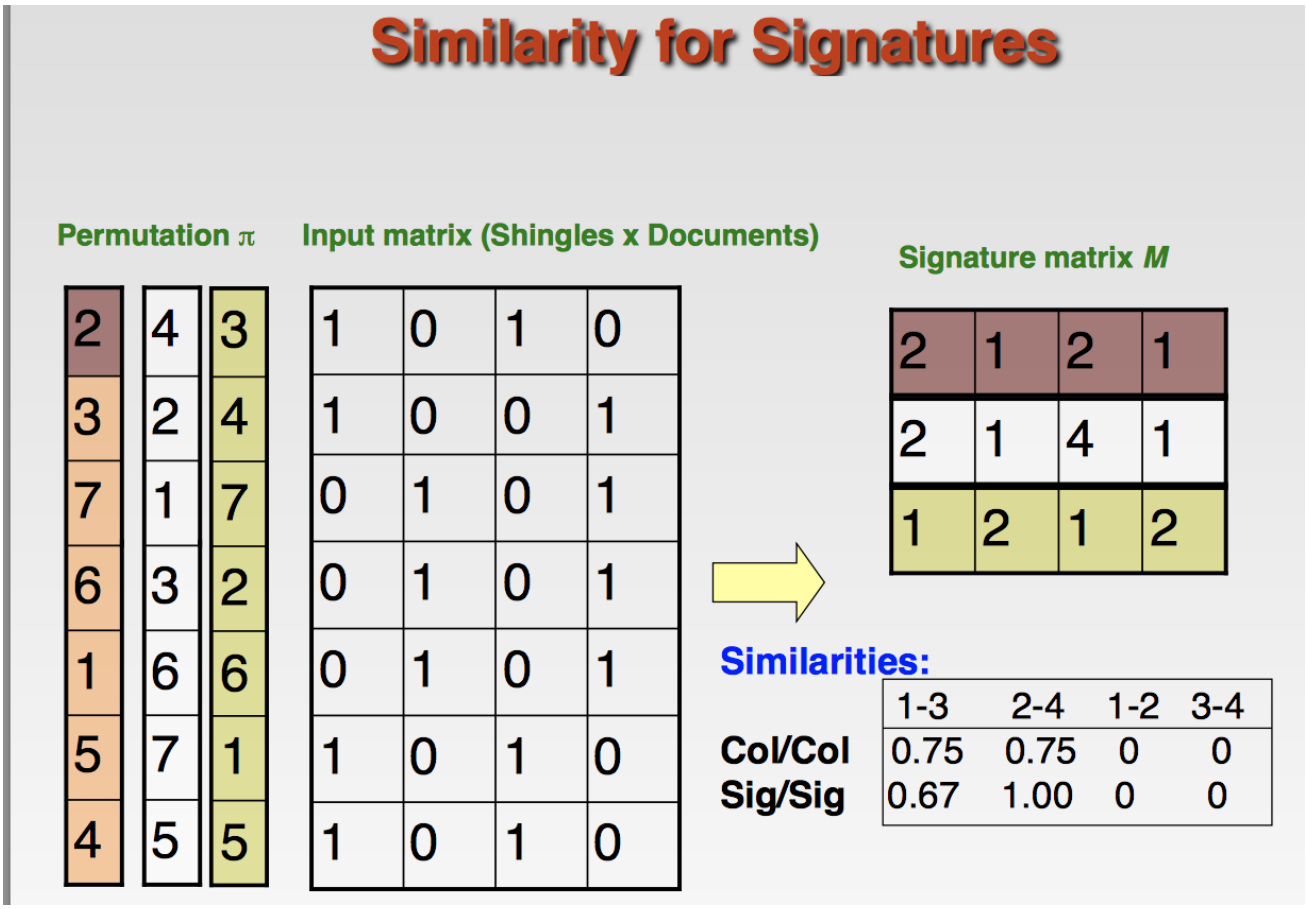
- jaccard sim

Example: S_1 = {a, d}, S_2 = {c}, S_3 = {b, d, e}, and S_4 = {a, c, d}

Element	S_1	S_2	S_3	S_4
a	1	0	0	1
b	0	0	1	0
c	0	1	0	1
d	1	0	1	1
e	0	0	1	0

- Size of intersection = 1; size of union = 4, Jaccard similarity (not distance) = 1/4
- **$d(S_1, S_2)$ = 1 – (Jaccard similarity) = 3/4**

- sim for signature



3.3 LSH

- Locality-Sensitive Hashing:

C_1, C_2 are 80% Similar

- **Find pairs of ≥ *s*=0.8 similarity, set b=20, r=5**
- **Assume:** sim(C_1, C_2) = 0.8
  - Since sim(C_1, C_2) ≥ **s**, we want C_1, C_2 to be a **candidate pair**: We want them to hash to at **least 1 common bucket** (at least one band is identical)
- **Probability C_1, C_2 identical in one particular band:** $(0.8)^5 = 0.328$
- Probability C_1, C_2 are ***not*** similar in all of the 20 bands: $(1-0.328)^{20} = 0.00035$
  - i.e., about 1/3000th of the 80%-similar column pairs are **false negatives** (we miss them)
  - **We would find 99.965% pairs of truly similar documents**

C_1, C_2 are 30% Similar

- **Find pairs of ≥ *s*=0.8 similarity, set b=20, r=5**
- **Assume:** sim(C_1, C_2) = 0.3
  - Since sim(C_1, C_2) < **s** we want C_1, C_2 to hash to **NO common buckets** (all bands should be different)
- **Probability C_1, C_2 identical in one particular band:** $(0.3)^5 = 0.00243$
- Probability C_1, C_2 identical in at least 1 of 20 bands: $1 - (1 - 0.00243)^{20} = 0.0474$
  - In other words, approximately 4.74% pairs of docs with similarity 0.3% end up becoming **candidate pairs**
    - They are **false positives** since we will have to examine them (they are candidate pairs) but then it will turn out their similarity is below threshold **s**

52

- original b=20, r=5
- If we had only 15 bands of 5 rows, the number of false positives would go down, but the number of false negatives would go up

Q4 Mining Data Streams

Question 4 Mining Data Streams

- Sampling
- DGIM
- Filtering
- Sample Question:
  - Question 3 in the last assignment

4.1 Sampling

8.1.1 Sampling a Fixed Proportion 固定比例取样

- 普通方法:**Problem 1**: Sampling fixed proportion
- x queries once
- d queries twice
- TATAL : x+2d
- 问： what fraction of queries by an average serach engine user are duplicates 重复查询的比例?
  - 正确答案：d/（x+d)
  - 分母是distinct number
- **处理方法： keep 10% of the queries**
  - sample contain x/10 singleton queries & 2d/10 of the duplicate queries at least once 存了2d/10个sample, 但其中有的sample就不是duplicate的了，所以sample中distinct d 应该会小于d/10
  - 结果只有d/100 是重复的
  - d/100 = 1/10* 1/10 *d
  - the d duplicates 18d/100 appear once 不重复的有这么多
  - 18d/100 = (1/10*9/10 +1/10*9/10) *d
  - So the sample-based answer is $\frac{\frac{d}{100}}{\frac{x}{10}+\frac{d}{100}+\frac{18d}{100}}=\frac{d}{10x+19d}$ ≠d/(x+d)
- 解决方法：sample users rather than queries
  - Pick 1/10th of users and take all their searches in the sample
  - Use a hash function that hashes the user name or user id uniformly into 10 buckets

4.2 DGIM

- Divide the window into buckets
  - Different from hash buckets
- For each bucket stores
  - The timestamp of its rightmost bit (most recent)
  - The number of 1's in the bucket (size of the bucket)
    - Select the bucket so it size will be a power of two
    - Example: total 7 bits, divide into buckets of size 1,2,4
- Total bits per bucket
  - Timestamp $log_2 N\ bits$
  - Size of the bucket $log_2 log_2 N$
    - Bucket size $i = 2^j$, need only represent j
    - Maximum value of $i\ is\ N$, maximum value of $j\ is\ log_2 N$
    - Total $log_2 log_2 N$ bits for representing every value of j
- Complexity
  - At most $2log_2 N$ buckets to combine, total $O(logN)$

4.3 Filtering

Generalization

- S has m members, array has n bits and there are k hash function
  - Targets x=n
  - Number of darts y=km
- Want proportion of 0 be large
  - So non-S will hash to zero at least once
  - Choose k to be n/m or less
    - Probability of 0 $e^{-\frac{y}{x}} = e^{-\frac{km}{n}} = 0.37 = 37\%$
    - Probability of 1 is $1 - e^{-\frac{km}{n}}$
    - Probability of false positive $\left(1 - e^{-\frac{km}{n}}\right)^k$

Q5 Recommender Systems

Question 5 Recommender Systems

- Content-based recommendation
- Collaborative recommendation
  - User-user
  - Item-item
- Sample Question:
  - Consider three users u_1, u_2, and u_3, and four movies m_1, m_2, m_3, and m_4. The users rated the movies using a 4-point scale: -1: bad, 1: fair, 2: good, and 3: great. A rating of 0 means that the user did not rate the movie. The three users' ratings for the four movies are: u_1 = (3, 0, 0, -1), u_2 = (2, -1, 0, 3), u_3 = (3, 0, 3, 1)
    - Which user has more similar taste to u_1 based on cosine similarity, u_2 or u_3? Show detailed calculation process.
    - User u_1 has not yet watched movies m_2 and m_3. Which movie(s) are you going to recommend to user u_1, based on the user-based collaborative filtering approach? Justify your answer.
  - Question 4 in the last assignment

- pro & con

a	Pros	Cons
Content-base	No community required, comparison between items possible	Content descriptions necessary, cold start for new users, no surprises
Collaborative	No knowledge- engineering effort, serendipity of results, learns market segments	Requires some form of rating feedback, cold start for new users and new items

5.1 Content-based recommendation

- Main idea: Recommend items to customer x similar to previous items rated highly by x
- for each item, create an item

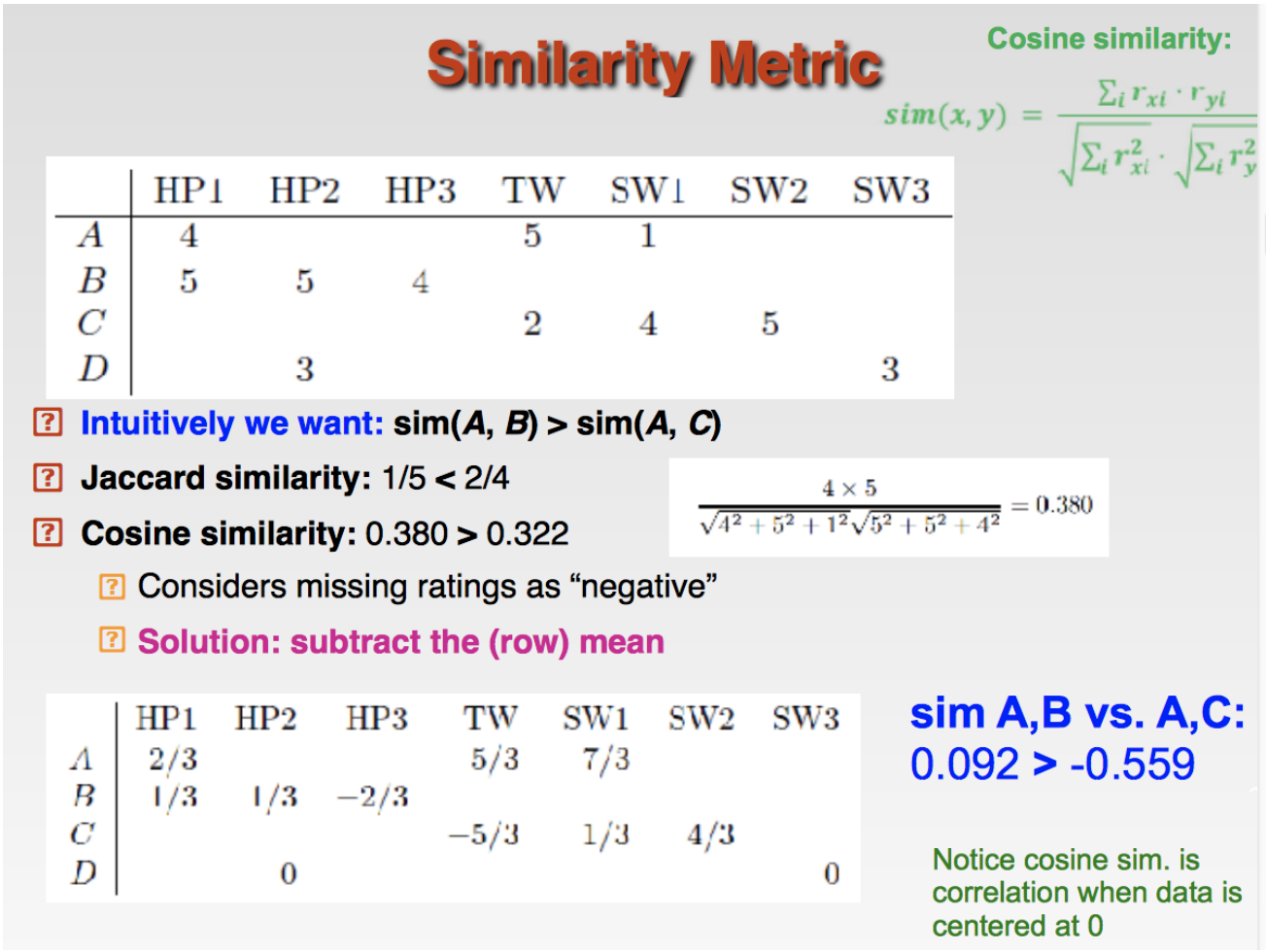
profile ,ie. movies: author,title, actor,director

- how to pick the important features: TF-IDF
  - 一种
    词频（TF）=某关键词出现次数/文章中关键词总数
  - 或者
    词频（TF）=某关键词出现次数/文章中出现最多次数关键词的出现次数
    接着，计算逆文档频率（IDF）。计算IDF需要一个语料库，它的计算公式很简单
  - 逆文档频率（IDF）=log（语料库文档总数/（包含该词的文档数+1）），之所以要+1是为了防止分母为0。由此可见，当一个词被越多的文档包含，则IDF值就越小，也就是所这个词很常见，不是最重要的能区分文章特性的关键词。
- probs
  - +: No need for data on **other users**
    - No cold-start or sparsity problems
  - Able to recommend to users with **unique tastes**
  - Able to **recommend new & unpopular items**
    - No first-rater problem
  - Able to provide **explanations**
    - Can provide explanations of recommended items by listing content-features that caused an item to be recommended
- cons
  - : Finding the appropriate **features** is hard
    - E.g., images, movies, music
  - Recommendations for **new users**
    - How to build a user profile?
  - Overspecialization
    - Never recommends items outside user’ s content profile
    - People might have multiple interests
    - Unable to exploit quality judgments of other users

5.2 Collaborative recommendation

- probs

    - Works for any kind of item
- cons
    - Cold Start: Need enough users in the system to find a match
    - Sparsity
    - First rater
    - Popularity bias
- similary



	HP1	HP2	HP3	TW	SW1	SW2	SW3
A	4			5	1		
B	5	5	4				
C				2	4	5	
D		3					3

	HP1	HP2	HP3	TW	SW1	SW2	SW3
A	2/3			5/3	7/3		
B	1/3	1/3	−2/3				
C				−5/3	1/3	4/3	
D		0					0

5.2.1 User-user

Similarity Metric (Cont')

A popular similarity measure in user-based CF: **Pearson correlation**

$$sim(x,y) = \frac{\sum_{s \in S_{xy}} (r_{xs} - \overline{r_x})(r_{ys} - \overline{r_y})}{\sqrt{\sum_{s \in S_{xy}} (r_{xs} - \overline{r_x})^2} \sqrt{\sum_{s \in S_{xy}} (r_{ys} - \overline{r_y})^2}}$$

Possible similarity values between -1 and 1;

	Item1	Item2	Item3	Item4	Item5
Alice	5	3	4	4	?
User1	3	1	2	3	3
User2	4	3	4	3	5
User3	3	3	1	5	4
User4	1	5	5	2	1

sim = 0,85
sim = 0,70
sim = -0,79

算sim时候不算在内

$$r_{xi} = \frac{\sum_{y \in N} s_{xy} \cdot r_{yi}}{\sum_{y \in N} s_{xy}}$$

Shorthand:
$s_{xy} = sim(x,y)$

5.2.2 Item-item

	Item1	Item2	Item3	Item4	Item5
Alice	5	3	4	4	?
User1	3	1	2	3	3
User2	4	3	4	3	5
User3	3	3	1	5	4
User4	1	5	5	2	1

- item-item often works better than user-user
- Why? Items are simpler, users have multiple tastes

18S2 开课时间

- 2018.11.09 5:00-9:30
- 2018.11.11 5:30-10:00

尚学教育

扫一扫上面的二维码图案，加我微信