Spark开源项目：个性化推荐算法实践

# 理论篇

# 1 推荐系统的意义 - 解决信息过载

* 搜索引擎时代

分类导航：雅虎

搜索：谷歌、百度

* 个性化时代(提高用户粘度、增加营收)

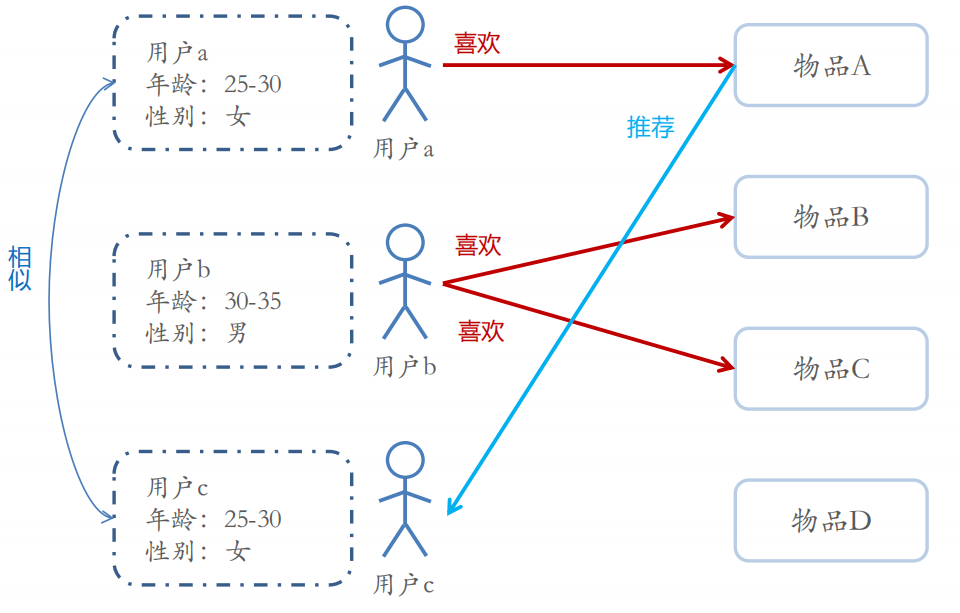
系统自动推荐相关的东西：今日头条、豆瓣、电商

# 2 推荐系统的分类

* 基于人口统计学的推荐
* 基于内容的推荐
* 基于协同过滤的推荐

# 3 基于人口统计学的推荐

基于人口统计学的推荐机制（Demographic-based Recommendation）是一种最易于实现的推荐方法，它只是简单的根据系统用户的基本信息发现用户的相关程度，然后将相似用户喜爱的其他物品推荐给当前用户。



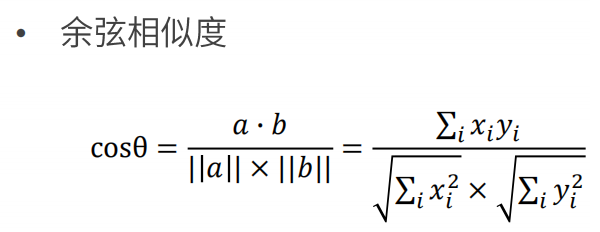
# 4 基于内容的推荐

## 4.1 定义

基于内容的推荐是在推荐引擎出现之初应用最为广泛的推荐机制，它的核心思想是根据推荐物品或内容的元数据，发现物品或者内容的相关性，然后基于用户以往的喜好记录，推荐给用户相似的物品。

## 4.2 算法流程

* 对于物品的特征提取——打标签（tag）
* 对于文本信息的特征提取——关键词
* 生成分词特征向量矩阵
* 计算相似度，常用余弦相似度



## 4.3 核心代码

### 4.3.1 spark TF-IDF

// 核心部分： 用TF-IDF从内容信息中提取电影特征向量  
// 创建一个分词器，默认按空格分词  
val tokenizer = new Tokenizer().setInputCol("genres").setOutputCol("words")  
  
// 用分词器对原始数据做转换，生成新的一列words  
val wordsData = tokenizer.transform(movieTagsDF)  
  
// 引入HashingTF工具，可以把一个词语序列转化成对应的词频  
val hashingTF = new HashingTF().setInputCol("words").setOutputCol("rawFeatures").setNumFeatures(50)  
val featurizedData = hashingTF.transform(wordsData)  
  
// 引入IDF工具，可以得到idf模型  
val idf = new IDF().setInputCol("rawFeatures").setOutputCol("features")  
// 训练idf模型，得到每个词的逆文档频率  
val idfModel = idf.fit(featurizedData)  
// 用模型对原数据进行处理，得到文档中每个词的tf-idf，作为新的特征向量  
val rescaledData = idfModel.transform(featurizedData)

val movieRecs = movieFeatures.cartesian(movieFeatures)  
 .filter{  
 // 把自己跟自己的配对过滤掉  
 case (a, b) => a.\_1 != b.\_1  
 }  
 .map{  
 case (a, b) => {  
 val simScore = this.*consinSim*(a.\_2, b.\_2)  
 ( a.\_1, ( b.\_1, simScore ) )  
 }  
 }  
 .filter(\_.\_2.\_2 > 0.6) // 过滤出相似度大于0.6的  
 .groupByKey()  
 .map{  
 case (mid, items) => MovieRecs( mid, items.toList.sortWith(\_.\_2 > \_.\_2).map(x => Recommendation(x.\_1, x.\_2)) )  
 }  
 .toDF()

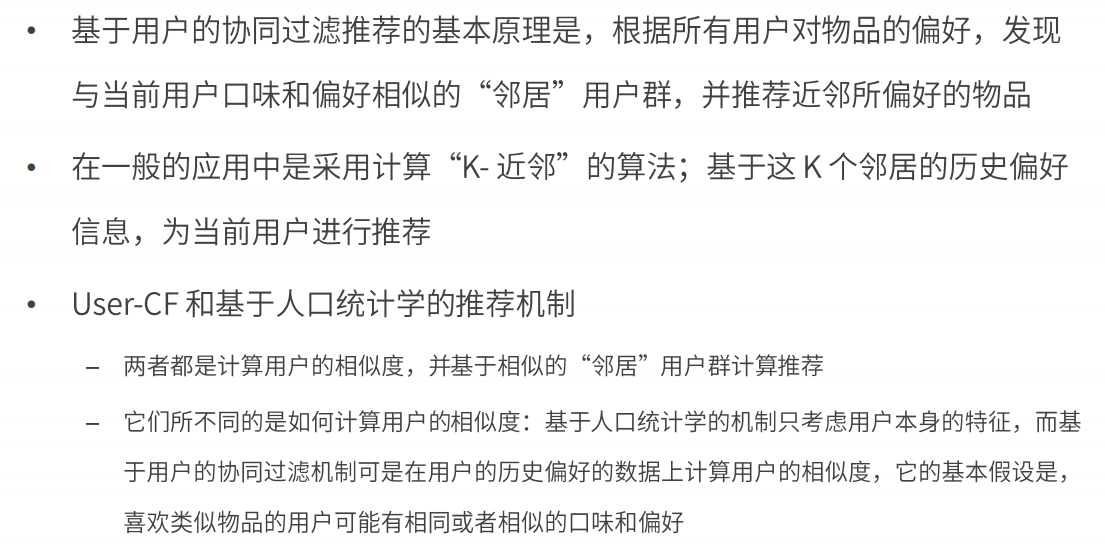
### 4.3.2 ElasticSearch More like this

MoreLikeThisQueryBuilder query = QueryBuilders.*moreLikeThisQuery*(  
 /\*new String[]{"name", "descri", "genres", "actors", "directors", "tags"},\*/  
 new MoreLikeThisQueryBuilder.Item[]{new MoreLikeThisQueryBuilder.Item(Constant.*ES\_INDEX*,  
 Constant.*ES\_MOVIE\_TYPE*, String.*valueOf*(mid))});

# 5 基于协同过滤的推荐

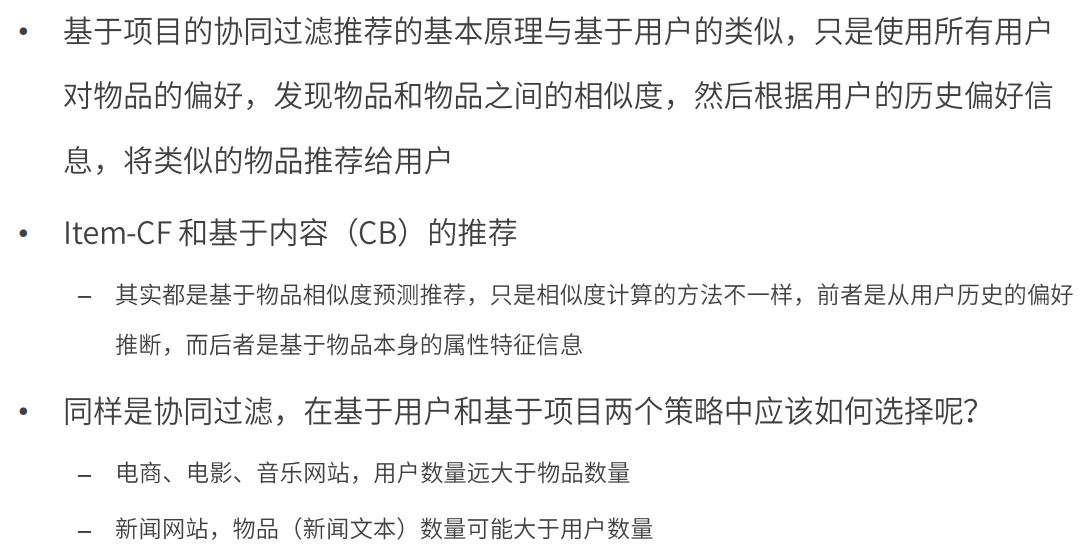
## 5.1基于用户的协同过滤(UserCF)

计算用户的相似度，推荐相似用户的喜好

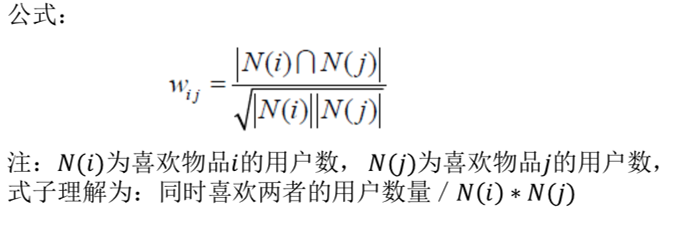


## 5.2 基于物品的协同过滤(ItemCF重点)

计算物品的相似度，推荐相似度高的物品(不同于基于内容的推荐)



### 5.2.1核心算法：计算同现相似度

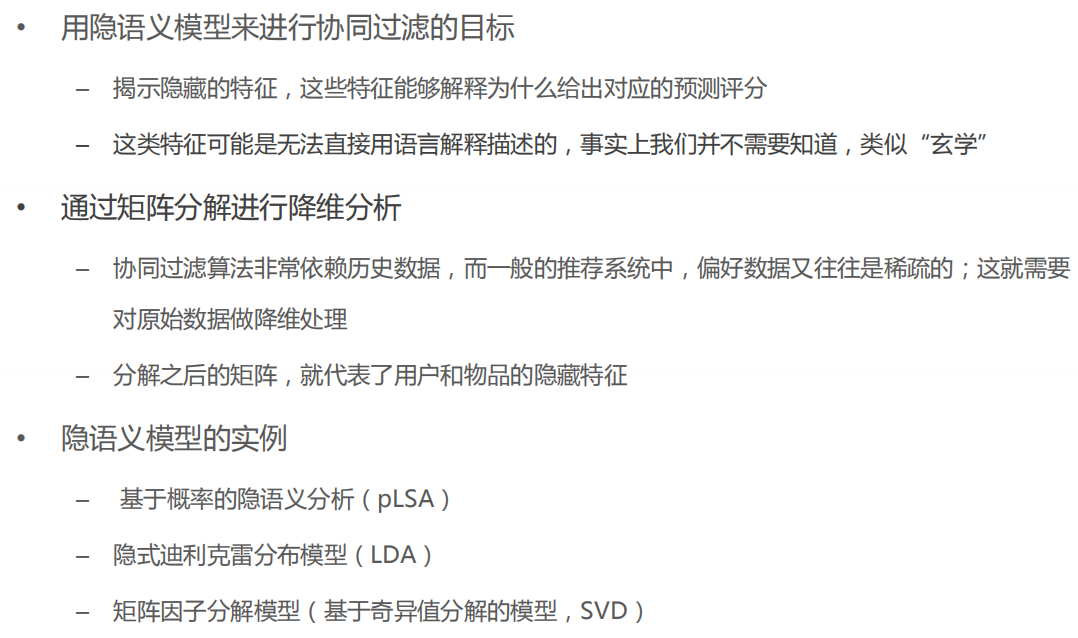


### 5.2.2 核心算法实例

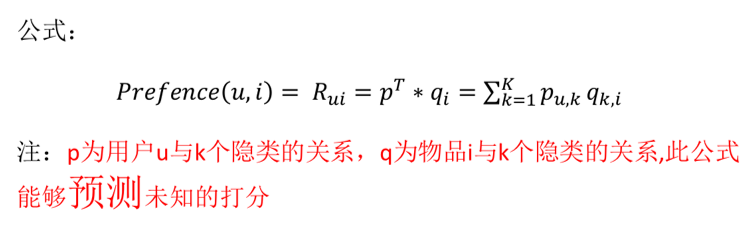
// ( mid1, (mid2, score) )  
val simDF = cooccurrenceDF.map{  
 row =>  
 val coocSim = *cooccurrenceSim*( row.getAs[Long]("cocount"), row.getAs[Long]("count1"),  
 row.getAs[Long]("count2") )  
 ( row.getInt(0), ( row.getInt(1), coocSim ) )  
}  
 .*rdd* .groupByKey()  
 .map{  
 case (mid, recs) =>  
 *MoviesRecs*( mid, recs.toList.sortWith(\_.\_2>\_.\_2).take(*MAX\_RECOMMENDATION*)  
 .map(x=>*Recommendation*(x.\_1,x.\_2)) )  
 }  
 .toDF()

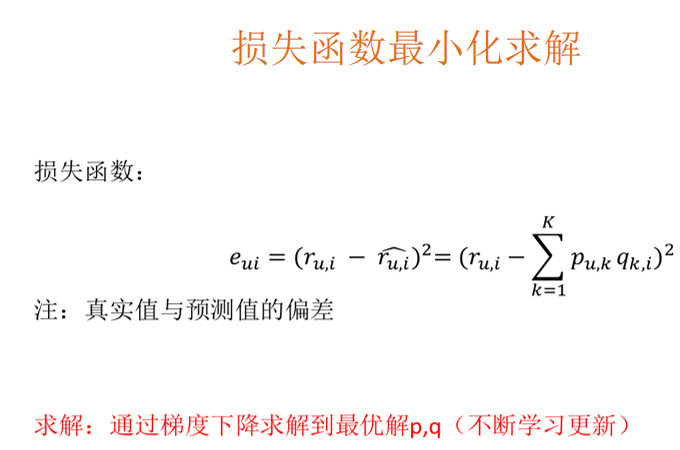
## 5.3 基于隐语义算法模型推荐

### 5.3.1 思想：找到隐藏因子，可以对user和item进行关联



### 5.3.2 算法公式



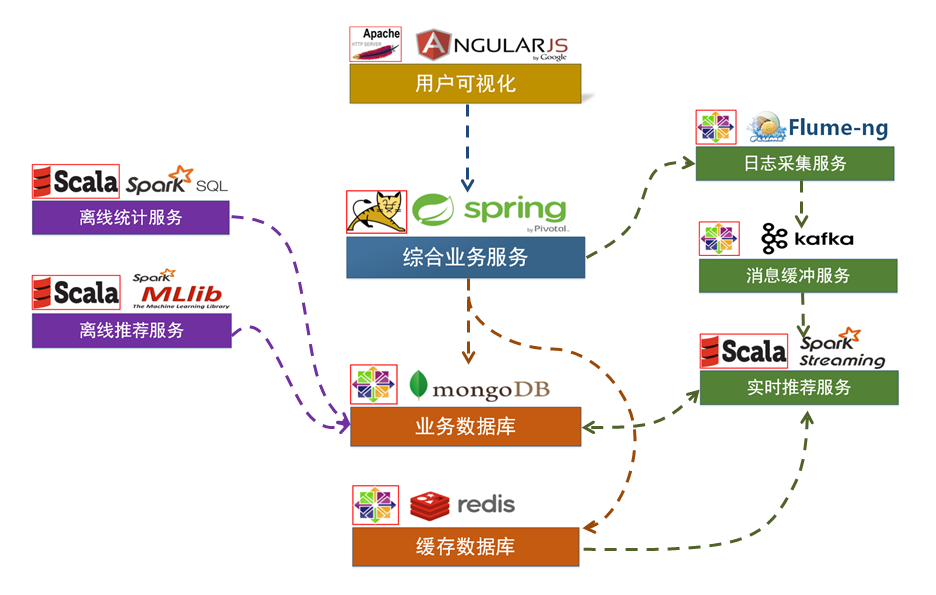


### 5.3.3 核心算法实例

// 训练隐语义模型：Rating(user:Int, product:Int, rating:Double)  
val trainData = ratingRDD.map(x => *Rating*(x.\_1, x.\_2, x.\_3))  
// 多个变量赋值  
val (rank, iterations, lambda) = (200, 5, 0.1)  
val model = ALS.*train*(trainData, rank, iterations, lambda)  
  
// 从rating数据中提取所有的uid和mid，并去重  
val userRDD = ratingRDD.map(\_.\_1).distinct()  
val movieRDD = ratingRDD.map(\_.\_2).distinct()  
val userMovies = userRDD.cartesian(movieRDD)  
  
// 调用model的predict方法预测评分  
val preRatings = model.predict(userMovies)  
  
val userRecs = preRatings  
 .filter(\_.rating > 0)  
 .map(rating => (rating.user, (rating.product, rating.rating))) // Rating->(uid, (mid, score))  
 .groupByKey()  
 .map {  
 case (uid, recs) => *UserRecs*(uid, recs.toList.sortWith(\_.\_2 > \_.\_2).take(*USER\_MAX\_RECOMMENDATION*).map(x => *Recommendation*(x.\_1, x.\_2)))  
 }  
 .toDF()

# 实战篇

# 1 项目技术架构



# 2 项目涉及关键技术

* Redis：存储用户最近评测队列
* Mongdb：BI可视化查询
* Elastic Search：文本关键词模糊检索索引、类别完全匹配检索、More like this基于内容推荐api
* Flume：实时评测数据采集
* Kafka：采集数据中间消息通道
* Kafka stream：消息转发中间管道
* Spark：spark sql、spark stream、spark M数据统计、加载数据源引擎、机器学习模型
* ScalaNLP：JAVA矩阵计算