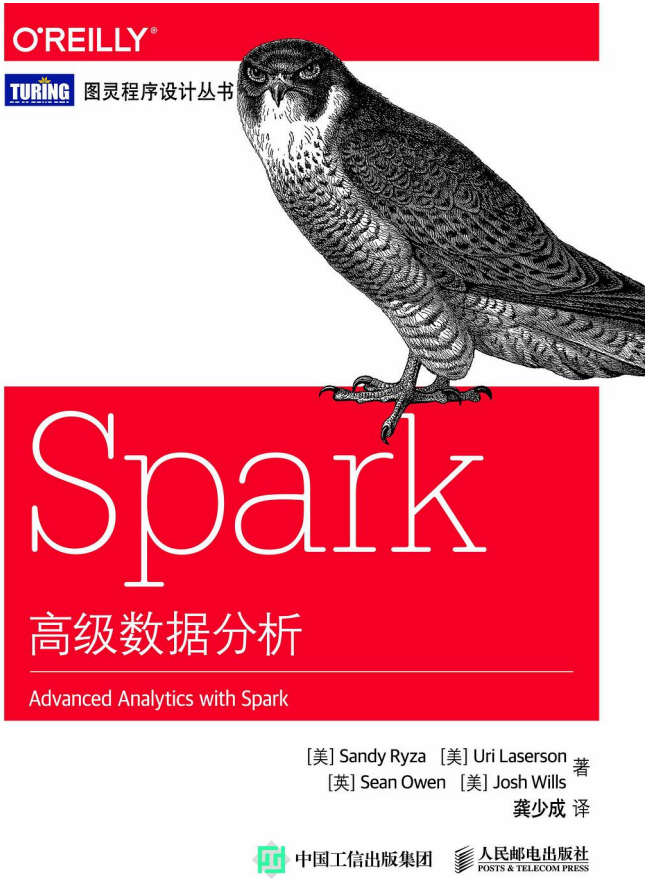
# Spark高级数据分析案例

# 推荐





https://github.com/sryza/aas

# 摘要

- 音乐推荐
- 基于潜在语义分析算法分析维基百科
- 基于蒙特卡罗模拟的金融风险评估

# 摘要

- 音乐推荐
- 基于潜在语义分析算法分析维基百科
- 基于蒙特卡罗模拟的金融风险评估


# 音乐推荐

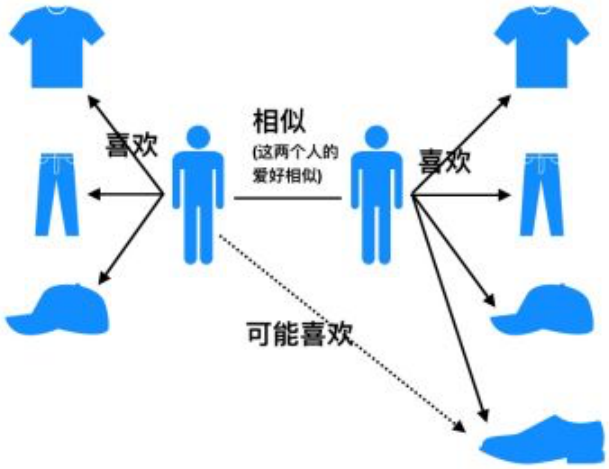


- 数据集
  - Audioscrobbler：last.fm的音乐推荐系统
  - 只记录播放数据，涵盖了更多的用户和艺术家，单条记录的信息较少，称为“隐式反馈数据”（用户和艺术家的关系是通过其他行动隐含体现的，而不是通过显式的评分或点赞得到的）。
  - user_artist_data.txt：141000个用户和160万个艺术家，记录2420万条用户播放艺术家歌曲的信息，包括播放次数。
  - artist_data.txt：每个艺术家的ID和对应的名字。

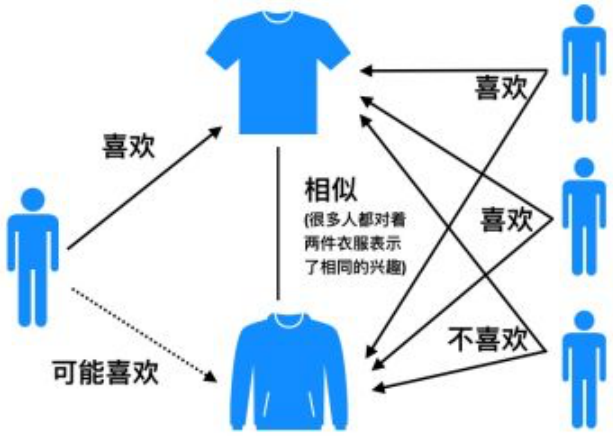
# 协同过滤



- 协同过滤的思路是通过群体的行为来找到某种相似性（用户之间的相似性或者标的物之间的相似性），通过该相似性来为用户做决策和推荐。




“人以群分”的基于用户的协同过滤






“物以类聚”的基于物品的协同过滤


# 协同过滤



- 邻域模型
  - UserBase CF：基于用户的协同过滤
  - ItemBase CF：基于物品的协同过滤
- 潜在因素模型（隐语义模型）
  - 发掘已有评分数据中的隐藏因子。比如通过对user-iterm 评分矩阵进行奇异值分解（Singular Value Decomposition，SVD）推断出模型。
  - 矩阵分解


# 协同过滤



隐语义模型也是基于矩阵分解的，但是和 SVD 不同，它是把原始矩阵分解成两个矩阵相乘而不是三个。

$$R = XY^T$$

现在的问题就变成了确定 $X$ 和 $Y$ ，我们把 $X$ 叫做用户因子矩阵， $Y$ 叫做物品因子矩阵。通常上式不能达到精确相等的程度，我们要做的就是要最小化它们之间的差距，从而又变成了一个最优化问题。


# 协同过滤



通常一个隐语义模型为每个用户 $u$ 定义一个用户因子向量 $x_u \in R^f$，为每一个物品 $i$ 定义物品因子向量 $y_i \in R^f$。通过计算两个向量的内积得到预测结果，如 $\hat{r}_{ui} = x_u^T y_i$。

优化目标是最小化代价函数，即:

$$\min_{x_*, y_*} \sum_{r_{ui}\ is\ known} (r_{ui} - x_u^T y_i)^2 + \lambda(||x_u||^2 + ||y_i||^2)$$

其中 $\lambda$ 用作模型正则化。


# 协同过滤



- 求解算法
  - 梯度下降法（SDG）：简单，快速
  - 交替最小二乘法（ALS）：并行性能较好，可以较好地处理稀疏数据。
    - Collaborative Filtering for Implicit Feedback Datasets
    - Large-scale Parallel Collaborative Filtering for the Netflix Prize


# 协同过滤



## 偏好(Preference)

布尔型变量，表示用户 $u$ 对物品 $i$ 的感情偏好。定义如下:

$$p_{ui} = \begin{cases} 1 & r_{ui} > 0 \\ 0 & r_{ui} = 0 \end{cases}$$

如果用户 $u$ 消费过某物品 $i$，即 $r_{ui} > 0$，这暗示用户 $u$ 喜爱物品 $i$；另一方面，如果用户 $u$ 从未消费过物品 $i$，我们认为用户 $u$ 对该物品 $i$ 没有偏好。

## 置信度(Confidence)

置信度用于衡量对偏好值 $p_{ui}$ 的信心。定义如下:

$$c_{ui} = 1 + \alpha r_{ui}$$

我们的目标是发现每一个用户 $u$ 的向量 $x_u \in R^f$ 和每一个物品 $i$ 的向量 $y_i \in R^f$ 。分别称为用户因子 *user-factor* 和 物品因子 *item-factor*。


# 协同过滤



## 代价函数

$$\min_{x_*,y_*} \sum_{u,i} c_{ui}(p_{ui} - x_u^T y_i)^2 + \lambda(||x_u||^2 + ||y_i||^2)$$

## 迭代求解

$$x_u = (Y^T C^u Y + \lambda I)^{-1} Y^T C^u p(u) \tag{1}$$

$$y_i = (X^T C^i X + \lambda I)^{-1} X^T C^i p(i) \tag{2}$$

随机初始化 $Y$，利用公式 $(1)$ 更新得到 $X$，然后利用公式 $(2)$ 更新 $Y$，直到误差值变化很小或者达到最大迭代次数。

通过迭代的方式交替计算两个公式，最终得到一个存储用户因子的矩阵 $X$ 和 存储物品因子的矩阵 $Y$，进而用于相似性发现和推荐结果生成。

# ALS Example



- MovieLens (基于spark.ml.*)

```
val ratings =
spark.read.textFile("data/mllib/als/sample_movielens_ratings.txt").map(parseRating)
    .toDF()
val Array(training, test) = ratings.randomSplit(Array(0.8, 0.2))
// Build the recommendation model using ALS on the training data
val als = new ALS()
    .setMaxIter(5)
    .setRegParam(0.01)
    .setUserCol("userId")
    .setItemCol("movieId")
    .setRatingCol("rating")
val model = als.fit(training)
```


# ALS Example



```
// Evaluate the model by computing the RMSE on the test data
// Note we set cold start strategy to 'drop' to ensure we don't get NaN
// evaluation metrics
model.setColdStartStrategy("drop")
val predictions = model.transform(test)
val evaluator = new RegressionEvaluator()
      .setMetricName("rmse")
      .setLabelCol("rating")
      .setPredictionCol("prediction")
val rmse = evaluator.evaluate(predictions)
println(s"Root-mean-square error = $rmse")
val userRecs = model.recommendForAllUsers(10)
val movieRecs = model.recommendForAllItems(10)
```


# 音乐推荐



```
val als = new ALS()
    .setSeed(Random.nextLong())
    .setImplicitPrefs(true)
    .setRank(10)
    .setRegParam(0.01)
    .setAlpha(1.0)
    .setMaxIter(5)
    .setUserCol("user")
    .setItemCol("artist")
    .setRatingCol("count")
    .setPredictionCol("prediction")
val model = als.fit(trainData)
```


# 选择超参数



- setRank(10)
  - 模型的潜在因素的个数，即“用户-特征”和“产品-特征”矩阵的列数，一般来说，它也是矩阵的阶。
- setMaxIter(5)
  - 矩阵分解迭代的次数
- setRegParam(0.01)
  - 标准的过拟合参数，通常也被称为λ，值越大越不容易产生过拟合，但值太大会降低分解的准确率。
- setAlpha(1.0)
  - 控制矩阵分解时，被观察到的“用户-物品”交互相对没被观察到的交互的权重。

# 摘要

- 音乐推荐
- 基于潜在语义分析算法分析维基百科
- 基于蒙特卡罗摩尼的金融风险评估

# 潜在语义分析



- 潜在语义分析（latent semantic analysis，LSA）是一种自然语言处理和信息检索技术，其目的是更好地理解文档语料库以及文档中词项的关系。它将语料库提炼成一组相关概念，每个概念捕捉了数据中一个不同的主题，且通常与语料库讨论的主题相符合。
- 概念可以由3个属性组成：语料库中文档的相关度、语料库中词项的相关度，以及概念对描述主题的重要性评分。


# 潜在语义分析

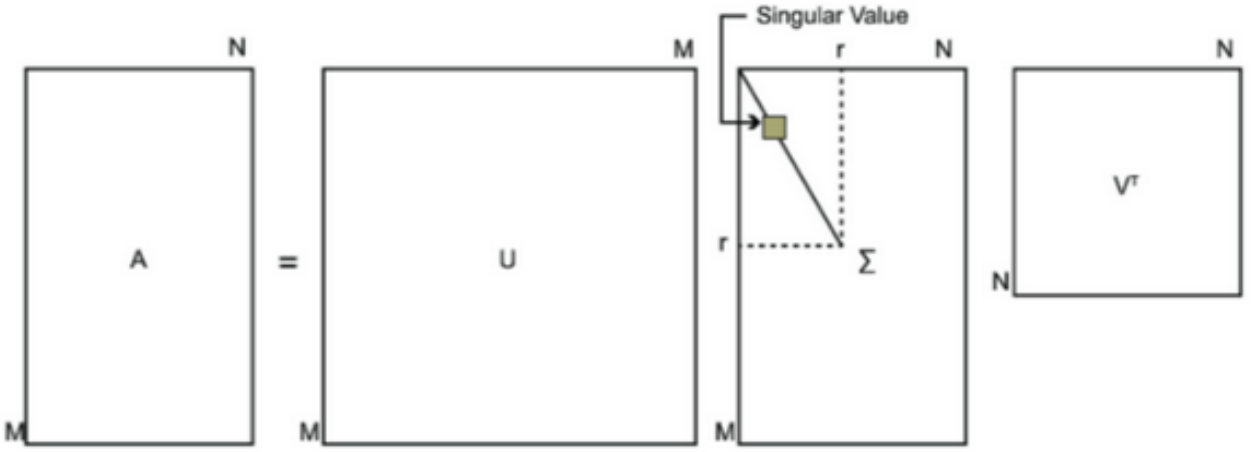


- 潜在语义分析的基本观点是：把高维的向量空间模型（VSM）表示中的文档映射到低维的潜在语义空间中。这个映射是通过对项/文档矩阵的奇异值分解（SVD）来实现的。
- LSA 的应用：信息滤波、文档索引、视频检索、文本分类与聚类、图像检索、信息抽取等。

# 奇异值分解SVD



- $A = U \sum V^T$
- 其中U是一个m×m的矩阵，∑是一个m×n的矩阵，除了主对角线上的元素以外全是0，主对角线上的每个元素都成为奇异值，V是一个n×n的矩阵。U和V都是酉矩阵，即满足$U^TU = I$，$V^TV = I$。




# 奇异值分解SVD



- 根据词项在每个文档中的出现次数构造“**文档-词项**”矩阵。矩阵中每个文档对应一列，每个词项对应一行，矩阵中的每个元素代表某个词项在对应文档中的重要性。
- SVD将矩阵分解成三个矩阵：其中一个矩阵代表文档中出现的概念，另一个代表词项对应的概念，还有一个代表每个概念的重要度。


# 文档-词项矩阵



- 每列代表语料库中出现的一个词项，每行代表一篇文档。不严格地讲，矩阵中每个元素值代表了相应列上的词项相对于相应行上的文档的权重。用的最多的是**词项频率/文档频率**，简写为TF-IDF（term frequency times inverse document frequency）
  - 一个词项在文档中出现的次数越多，它相对于文档的重要性越高；
  - 词项是不平等的。文档中出现语料库中罕见词项的意义比出现常见词项更大，因此指标就是词项在所有语料库中出现次数的倒数。为防止罕见词的权重过大，算法对逆文档频率取对数。


# 词形归并



- 停词 stop word
  - 比如the, is等
- 词干还原(stemming)或词形归并(lemmatization)
  - monkey和monkeys
  - draw和drew
  - nationalize和nationalization
- 工具： Stanford Core NLP项目

# 计算TF-IDF



- 使用spark.ml.feature.{CountVectorizer, IDF}

```
val termsDF = terms.toDF("title", "terms")
val filtered = termsDF.where(size($"terms")>1)

val numTerms = 20000
val countVectorizer = new CountVectorizer().setInputCol("terms").setOutputCol
("termFreqs").setVocabSize(numTerms)
val vocabModel = countVectorizer.fit(filtered)
val docTermFreqs = vocabModel.transform(filtered)

val idf = new IDF().setInputCol("termFreqs").setOutputCol("tfidfVec")
val idfModel = idf.fit(docTermFreqs)
val docTermMatrix = idfModel.transform(doTermFreqs).select("title","tfidfVec")
```

# SVD



- 使用org.apache.spark.mllib.linalg.SingularValueDecomposition
- 要做转换： spark.ml.linalg.Vector -> spark.mllib.linalg.Vector

```
val vecRdd = docTermMatrix.select("tfidfVec").rdd.map { row =>
Vectors. fromML(row.getAs[MLVector]("tfidfVec"))}
vecRdd.cache()
val mat = new RowMatrix(vecRdd)
val k = 1000
val svd = mat.computerSVD(k, computeU=true)
```

# 找出重要的概念



- SVD算法的输出是一组数值。V矩阵表示了词项对概念的重要程度。每个概念都对应V中的一列，每个词项都对应V中一行。每个元素可以理解为词项对于概念的相关度。

```
val v = svd.V
val topTerms = new ArrayBuffer[Seq[(String, Doubl)]]()
val arr = v.toArray
for (i <- 0 until numConcepts) {
        val offs = i * v.numRows
        val termWeights = arr.slice(offs, offs + v.numRows).zipWithIndex
        val sorted = termWeights.sortBy(-_._1)
        topTerms += sorted.take(numTerms).map{
          case (score, id) => (termIds(id), score)
        }
}
```

# 摘要

- 音乐推荐
- 基于潜在语义分析算法分析维基百科
- 基于蒙特卡罗模拟的金融风险评估


# 风险价值



- 风险价值（Value at Risk，VaR）是一个金融统计概念，它度量在一定条件下的期望损失大小。
  - 三个参数：投资组合，时间跨度，p值
- 条件风险价值（CVaR），表示的是期望损失而不是截止值。
- 蒙特卡罗模拟（Monte Carlo Simulation）给出数千个甚至数百万个随机的市场状况，并观察这些状况对投资组合的影响。Spark本身具有高并行性，非常适合进行蒙特卡罗模拟。

# 术语



- 金融工具
  - 可交易的资产，比如债券、贷款、期权或股票。金融工具在任意时刻都可以用一个值来表示，也就是资产的卖出价。
- 投资组合
  - 金融机构持有的金融工具的组合。
- 回报
  - 一段时间内金融工具或投资组合的价值变化

# 术语



- 损失
  - 负的回报
- 指数
  - 一个假设的金融工具组合。比如纳斯达克综合指数包含了美国和世界上其他国家主要公司的约3000只股票和金融工具。
- 市场因素
  - 给定时间点的宏观金融环境指标，比如美国的GDP指标就是一个市场因素，又如美元对欧元的汇率也是一个市场因素。

# VaR计算方法



- 方差-协方差法（variance-covariance）
  - 假设每个金融工具的回报服从正态分布
- 历史模拟法
  - 使用历史数据的分布推断风险值，不依赖概要统计量
- 蒙特卡洛模拟法
  - 评估概率密度函数（probability density function, PDF）
  - 对服从该概率分布的简单随机变量进行重复采用，并对采样结果进行汇总统计

# 蒙特卡罗模拟法



- 定义市场条件与每个金融工具的回报之间的关系，该关系表现为拟合历史数据的模型；
- 为那些容易采样的市场条件定义分布，这些分布也拟合历史数据；
- 在随机市场条件下进行试验；
- 计算每次试验的投资组合总体损失，用这些损失定义损失的经验分布。

# 设计模型



- 蒙特卡罗风险模型通常把每个金融工具的回报分解为一组市场因素的组合。常用的市场因素包括标普500指数、美国GDP和货币汇率等。接着需要一个模型根据这些市场条件来预测每个金融工具的回报。
- 模型：给定试验t的市场因素向量$\mathbf{m}_t$，通过某个转换函数∅得到特征向量$\mathbf{f}_t$，即$\mathbf{f}_t = \emptyset(m_t)$

# 设计模型



- 为每个金融工具训练一个模型，该模型给每个特征赋予一个权重。回报计算公式：

$$r_{it} = c_i + \sum_{j=1}^{|w_i|} w_{ij} * f_{tj}$$

- $r_{it}$为试验t中工具i的回报，$c_i$为金融工具i的截距项，$w_{ij}$为特征j在金融工具i上的回归权重，$f_{tj}$为特征j在试验t中产生的随机值。
- 即每个金融工具的回报等于所有市场因素特征的回报与金融工具的权重的乘积之和。

# 设计模型



- 模拟市场因素，假设市场因素回报服从正态分布，为了考虑市场因素之间的相关性，使用多元正态分布，其协方差矩阵是非对角阵：

$$\mathrm{m}_t \sim \mathcal{N}\left(\mu, \sum\right)$$

- $\mu$代表因素回报经验平均向量，$\sum$代表市场因素回报经验的协方差矩阵。


# 流程



- 获取数据
- 数据预处理
- 确定市场因素的权重
- 采样
- 多元正态分布
- 运行
- 回报分布的可视化
- 评估


# 确定市场因素的权重



- 将价格的时间序列转换成间隔为2周的价格移动交叠序列。

```
def twoWeekReturns(history: Array[(LocalDate, Double)]): Array[Double] = {
  history.sliding(10).map { window =>
    val next = window.last._2
    val prev = window.head._2
    (next - prev) / prev
  }.toArray
}
```

# 确定市场因素的权重



- 附加特征：市场因素的平方以及平方根

```
def featurize(factorReturns: Array[Double]): Array[Double] = {
    val squaredReturns = factorReturns.map(x => math.signum(x) * x * x)
    val squareRootedReturns = factorReturns.map(x => math.signum(x) *
math.sqrt(math.abs(x)))
    squaredReturns ++ squareRootedReturns ++ factorReturns
  }
```

- 拟合线性模型（Apache Commons Math）
  - OLSMultipleLinearRegression
  - estimateRegressionParameters：找到每个工具的模型参数


# 确定市场因素的权重



```
def linearModel(instrument: Array[Double], factorMatrix: Array[Array[Double]])
    : OLSMultipleLinearRegression = {
    val regression = new OLSMultipleLinearRegression()
    regression.newSampleData(instrument, factorMatrix)
    regression
  }

  def computeFactorWeights(
    stocksReturns: Seq[Array[Double]],
    factorFeatures: Array[Array[Double]]): Array[Array[Double]] = {
    stocksReturns.map(linearModel(_,
factorFeatures)).map(_.estimateRegressionParameters()).toArray
  }
```


# 采样



- 生成随机回报因素来模拟市场条件，需要确定因素回报向量的一个概率分布，并从该分布上采样。
- 可视化工具：breeze-viz
- 核密度估计（kernel density estimation）：一种对直方图进行平滑处理的方法。
- org.apache.spark.mllib.stat.KernelDensity


# 多元正态分布



- 多元正态分布的每个样本时一个向量，在其他所有维度的值都确定的情况下，对于给定维度的值服从正态分布。
- 多元正态分布的参数为对应每个维度的均值向量和一个矩阵，该矩阵描述了任意两个维度之间的协方差。

```
val multivariateNormal = new MultivariateNormalDistribution(rand, factorMeans,
    factorCovariances)
```

# 并行化



- 对试验进行并行化：提取一组风险因素样本，用该样本预测每个金融工具的回报，然后将所有回报相加得到总体试验损失，通常需要运行千次甚至数百万次试验。
- 对金融工具进行并行化：数据按照金融工具对RDD进行分区，对每个金融工具进行flatMap转换得到每次试验的损失。对所有任务采用相同随机种子意味着生成的试验序列是相同的。reduceByKey操作把同一个试验的对应的所有损失汇总到一起。

# 并行化



- 生成一个随机种子组成的RDD，希望每个分区的随机种子都不一样，这样每个分区将产生不同的试验。

```
// Generate different seeds so that our simulations don't all end up with the same results
val parallelism = 1000
val baseSeed = 1001L
val seeds = (baseSeed until baseSeed + parallelism)
val seedDS = seeds.toDS().repartition(parallelism)

// Main computation: run simulations and compute aggregate return for each
seedDS.flatMap(trialReturns(_, numTrials / parallelism, factorWeights, factorMeans, factorCov))
```