Week4-学习笔记心得

## 协同过滤

是推荐系统的主流思想之一：基于动态的，有一定的群体数据基础  
1。 基于邻域的协同过滤(也叫做基于内存的推荐)  
 UserCF  
 ItemCF  
2. 基于模型的协同过滤（通过机器学习的方式）  
 1）隐语义模型（LFM, Latent Factor Model）  
 矩阵分解（MF）  
 LDA,LSA,pLSA  
 2）基于贝叶斯网络  
 3）基于SVM

## 什么是隐语义模型

用户与物品之间存在着隐含的关系，通过隐含特征联系用户兴趣和物品，  
隐含特征--k比较大，说明分的比较细，隐特征比较多。  
隐特征可解释性较差，隐含特征计算机能理解就好，但是对于人来说，是不好解释的，但机器比较清楚。就像我们不知道机器是怎么把这些聚合在一起的，相比之下itemCF可解释性较强。

## 推荐系统的两大应用场景

### 1. 评分预测

根据用户给的显示评分。

### 2. TopN推荐

常用于购物网站，拿不到显示评分，通过用户的隐式反馈为用户提供给一个可能感兴趣的列表

## 矩阵分解MF

A screen shot of a social media post

Description automatically generated

该矩阵表示收集到的用户数据，12个用户，9个电影

矩阵分解要做的是预测出矩阵中缺失的评分，使得预测评分能反映用户的喜欢程度

可以把预测评分最高的前K个电影推荐给用户了。

Chart, table, bar chart, waterfall chart

Description automatically generated

此时k=3

所以本12\*9的矩阵就变成了12\*3和3\*9的两个稠密的矩阵

两个稠密矩阵相乘的结果也是稠密的，这样就完成了矩阵补全的任务

（原来这个矩阵是稀疏的，通过拆解成两个稠密的矩阵，最后稀疏矩阵中空的就被填上了，变成了稠密的矩阵）

如何验证这个空填的好不好呢？

我们知道机器学习最核心的部分就是定义loss（定义损失函数）

（机器学习中所有的算法都需要最大化或最小化一个函数，这个函数被称为“目标函数”。其中，我们一般把最小化的一类函数，称为“损失函数”。它能根据预测结果，衡量出模型预测能力的好坏）

损失函数大致分为两类：分类问题的损失函数和回归任务的损失函数

Diagram, shape, polygon

Description automatically generated

回归任务的损失函数用的是哪一个损失函数呢？MSE

### 如何从评分矩阵中分解出User矩阵和Item矩阵？

1.只有左侧的评分矩阵R是已知的

2. User矩阵和Item矩阵是未知

3.学习出User矩阵和Item矩阵，使得User矩阵\*Item矩阵与评分矩阵中已知的评分差异最小 => 最优化问题

Table

Description automatically generated

观察User矩阵：用户的听歌爱好体现在User向量上

观察Item矩阵，电影的风格也会体现在Item向量上

MF用user向量和item向量的内积去拟合评分矩阵中该user对该item的评分，内积的大小反映了user对item的喜欢程度。内积大匹配度高，内积小匹配度低。

隐含特征个数k，k越大，隐类别分得越细，计算量越大。

### 矩阵分解的目标函数

Text

Description automatically generated

这里用MSE Text

Description automatically generated 做损失函数，预测值就是User矩阵\*Item矩阵

实际值/已知值就是，表示只计算有分数的;正则化带了两个参数，x和y

这里是L2正则是二阶项，x的平方和y的平方，如果是L1正则，就是一阶项。

即如下：

Text

Description automatically generated

*一般情况下，我们会用如上公式在除以N，即MSE/N*

预测过程（User矩阵和Item矩阵）是不用计算损失函数的，只有测试集可以计算损失函数

### Summary

MF是一种隐语义模型，它通过隐类别匹配用户和item来做推荐。

1.MF对原有的评分矩阵R进行了降维，分成了两个小矩阵：User矩阵和Item矩阵，User矩阵每一行代表一个用户的向量，Item矩阵的每一列代表一个item的向量。将User矩阵和Item矩阵的维度降低到隐类别个数的维度。

2.根据用户行为，矩阵分解分为显式矩阵分解和隐式矩阵

在显式MF中，用户向量和物品向量的内积拟合的是用户对物品的实际评分

在隐式MF中，用户向量和物品向量的内积拟合的是用户对物品的偏好(0或1)，拟合的强度由置信度控制，置信度又由行为的强度决定

怎么去做矩阵分解：

先设损失函数，设完用ALS求解

## 矩阵分解的目标函数最优化问题的工程解决

### 1. 交替最小二乘法 ALS (Alternating Least Squares)--可以进行并行化处理

#### 原理 ALS进行矩阵分解的原理

1。step1 固定Y优化X  
 2. step2 固定X优化Y  
 重复step1和step2,直到X和Y收敛。每次固定一个矩阵，优化另一个矩阵，都是最小二乘问题。

为什么要用平均数，而不是中位数，或者是几何平均数？

Text, schematic

Description automatically generated

#### 为什么用算数平均值做为实际值？

要使得结果最小，对它求导，最后求得的y就是算数平均值

Text, letter

Description automatically generated

Text, letter

Description automatically generated

Text, letter

Description automatically generated

求解Y的时候同理。

#### ALS隐式反馈

除了针对显示评分矩阵，ALS还可以对隐式数据进行分解：  
 将评分看作行为的强度，比如浏览次数，阅读时间  
 当rui>0时，用户u对商品i有行为；  
 当rui<0时，用户u对商品i没有行为。  
 Pui称为用户偏好，

  
 当rui>0 ， 即用户u对商品i有行为；认为用户u对物品i感兴趣，此时Pui=1;  
 当rui<=0 ， 即用户u对商品i没有行为；认为用户u对物品i不感兴趣，此时Pui=0;

隐式行为可能有多次，我们会把这个算成一个强度

对隐式矩阵进行分解：

引入置信度，代表weight,最小值为1，这里加1的目的就是说明最小值为1，强度越大，拟合程度越好



当rui>0时，cui与rui线性递增

当rui=0时，cui=1，也就是cui最小值为1

目标函数



xu, yi都为k维列向量，k为隐特征的个数

将目标函数转化为矩阵形式，并进行求导

Step1，固定Y优化X

同理，求解得





Λu 为用户u 对所有物品的置信度cui 构成的对角阵

Step2，固定X优化Y

同理，求解得





Λi 为所有用户对物品i 的偏好的置信度构成的对角矩阵

ALS是一种重要的数据拟合技术，可应用于线形回归、非线形回归

Python代码

https://github.com/tushushu/imylu/blob/master/imylu/recommend/als.py

#### ALS工具

spark mllib库（spark3.0版本后废弃）  
 支持常见的算法，包括 分类、回归、聚类和协同过滤  
 from pyspark.mllib.recommendation import ALS, Rating, MatrixFactorizationModel  
 spark ml库（官方推荐）

功能更全面更灵活，ml在DataFrame上的抽象级别更高，数据和操作耦合度更低，使用起来想sklearn

from pyspark.ml.recommendation import ALS

ALS函数中的参数rank代表降成多少维

Spark安装很多坑，需要慢慢来

实训平台可以直接用，不用装spark

### 2. 随机梯度下降 SGD (Stochastic Gradient Descent)--可以进行并行化处理

基本思路是以随机方式遍历训练集中的数据，并给出每个已知评分的预测评分。用户和物品特征向量的调整就沿着评分误差越来越小的方向迭代进行，直到误差达到要求。所以，SGD不需要遍历所有的样本即可完成特征向量的求解。

Chart, line chart

Description automatically generated

--> K值增加到 RMSE 下降幅度很小的时候，K值再增加的意义不大，即RMSE基本不再下降的时候。

梯度下降法：

我们处于山中的某个位置，不知道极值点在哪里

每一步，我们都以下降最多的路线来下山

a表示学习率（步长）

A picture containing shape

Description automatically generated

Chart, surface chart

Description automatically generated  
 曲面上方向导数的最大值的方向代表了梯度的方向。沿着梯度方向，会让h值升高，因此我们需要沿着梯度的反方向，也就是采用梯度下降的方式进行权重更新





整个梯度下降有三种方法：  
1.BGD（批量梯度下降

2.SGD（随机梯度下降）速度最快

3.MBGD（mini-batch梯度下降）

区别：  
BGD:在每次更新时用所有样本；稳定；收敛慢

SGD:m个样本中随机抽取一个样本（方向可能会来回换），用这一个样本来近似所有的样本；更快收敛；最终解在全局最优解附近  
MBGD:每次更新用b个样本，折中方法；速度较快

### 3. ALS-WR算法，加入了正则化

可以解决过拟合问题，当隐特征个数很多的时候，也不会造成过拟合

### 4. 旋转混合式求解方法

Facebook把ALS和SGD两算法进行了揉合，提出了旋转混合式求解方法（Giraph），可以处理1000亿数据，效率比普通Spark MLlib快了10倍

Chart, line chart

Description automatically generated

### Summary

1.ALS和SGD都是数学上的优化方法，可以解决最优化问题（损失函数最小化）

2.ALS-WR算法，可以解决过拟合问题，当隐特征个数很多的时候也不会造成过拟合

3.ALS，SGD都可以进行并行化处理

4.SGD方法可以不需要遍历所有的样本即可完成特征向量的求解

5.Facebook把SGD和ALS两个算法进行了揉合，提出了旋转混合式求解方法，可以处理1000亿数据，效率比普通的Spark MLlib快了10倍

## 推荐系统工具Surprise和LightFM

Surprise

1．Surprise是scikit系列中的一个推荐系统库

文档：<https://surprise.readthedocs.io/en/stable/>

pip install scikit-surprise

2.数据集：可以使用内置数据集（Movielens等），也可以自定义数据集

3.优化算法：支持多种优化算法，ALS，SGD

4.预测算法：包括基线算法，邻域方法，矩阵分解，SlopeOne等

5.相似性度量：内置cosine，MSD，pearson等

6.scikit家族，可以使用GridSearchCV自动调参，方便比较各种算法结果

LightFM

Python推荐算法库，具有隐式和显式反馈的多种推荐算法实现。

易用、快速（通过多线程模型估计），能够产生高质量的结果。

## Surprise工具

### Surprise中常用算法

from surprise import BaselineOnly, KNNBasic, KNNBaseline, SlopeOne

1。Baseline算法  
2。基于邻域的协同过滤 KNNBasic  
3。矩阵分解：SVD，SVD++，PMF，NMF

from surprise import SVD,SVDpp,NMF  
4。SlopeOne协同过滤算法

Graphical user interface, text

Description automatically generated

NormalPredictor() 基于统计的推荐系统预测打分，假定用户打分的分布是基于正态分布的  
BaselineOnly 基于统计的基准预测线打分  
knns.KNNBasic 基本的协同过滤算法  
knns.KNNWithMeans 协同过滤算法的变种，考虑每个用户的平均评分  
knns.KNNWithZScore 协同过滤算法的变种，考虑每个用户评分的归一化操作  
knns.KNNBaseline 协同过滤算法的变种，考虑每个用户评分的基线  
matrix\_factorzation.SVD SVD 矩阵分解算法  
matrix\_factorzation.SVDpp SVD++ 矩阵分解算法  
matrix\_factorzation.NMF 一种非负矩阵分解的协同过滤算法  
SlopeOne SlopeOne 协同过滤算法

## Surprise中的Baseline算法

这里surprise中的baseline 和前面的spark中的ALS是两个model,虽然baseline中也有ALS优化方法，因为Spark中ALSshi有k的，baseline中k=1

MovieLens

<https://www.kaggle.com/jneupane12/movielens/download>

Baseline算法：基于统计的基准预测线打分

bui=u+bu+bi  
bui预测值  
bu用户对整体的偏差  
bi商品对整体的偏差

Graphical user interface, text, application

Description automatically generated  
  
Baseline算法可以理解为是比矩阵分解更简单的一种方法  
矩阵分解中用户的偏好是对每一类（k值）的偏好，（对于动作片一种偏好，爱情片一种偏好）；  
Baseline算法把用户的偏好归位一类偏好--》bu--可以理解为k=1  
bi也是一样，是对所有商品的一种偏好

这里的bu和bi是一个值，所以本身和ALS求解的方法是不一样的，这里没有k的概念，所以ALS 方法算起来更准确，因为有多个维度，baseline方法算起来更快，因为相当于k=1的情况  
  
Baseline算法：

使用ALS进行优化  
step1,固定bu,求bi  
step2,固定bi,求buText, letter

Description automatically generated

## 使用Surprise的BaselineOnly对MovieLens电影进行推荐

from surprise import Reader  
 from surprise import Dataset  
 from surprise import BaselineOnly,NormalPredictor  
 from surprise.model\_selection import KFold  
 from surprise import accuracy  
 #数据读取  
 reader = Reader(line\_format='user item rating timestamp',sep=',',skip\_lines=1)  
 data = Dataset.load\_from\_file('./ratings.csv',reader=reader)  
 train\_set = data.build\_full\_trainset()  
  
 #ALS优化  
 #bsl\_options = {'method': 'als','n\_epochs':5,'reg\_u':12,'reg\_i':5}  
 '''  
 'n\_epochs':5, 迭代次数，默认是10  
 'reg\_u':12, 用户的正则化参数，默认是15  
 'reg\_i':5 物品的正则化参数，默认是10  
 '''  
 ''''  
 Baseline算法ALS 优化  
 Estimating biases using als...  
 RMSE: 0.8643  
 Estimating biases using als...  
 RMSE: 0.8629  
 Estimating biases using als...  
 RMSE: 0.8645  
 user: 196 item: 302 r\_ui = 4.00 est = 3.96 {'was\_impossible': False}  
 '''  
  
 # SGD 优化  
 '''  
 参数：  
 reg: 代价函数的正则化参数默认0.02  
 learnng\_rate:学习率，默认0.005  
 n\_epochs, 迭代次数，默认是10  
 '''  
 bsl\_options ={'method':'sgd','n\_epoch':5}  
 '''  
 Baseline算法SGD 优化  
 Estimating biases using sgd...  
 RMSE: 0.8650  
 Estimating biases using sgd...  
 RMSE: 0.8648  
 Estimating biases using sgd...  
 RMSE: 0.8638  
 user: 196 item: 302 r\_ui = 4.00 est = 4.18 {'was\_impossible': False}  
 '''  
  
 #algo = BaselineOnly(bsl\_options=bsl\_options)  
  
 #algo = BaselineOnly()  
 '''  
 Estimating biases using als...  
 RMSE: 0.8657  
 Estimating biases using als...  
 RMSE: 0.8662  
 Estimating biases using als...  
 RMSE: 0.8659  
 user: 196 item: 302 r\_ui = 4.00 est = 4.19 {'was\_impossible': False}  
 '''  
  
 algo = NormalPredictor()  
 '''  
 RMSE: 1.4326  
 RMSE: 1.4333  
 RMSE: 1.4316  
 user: 196 item: 302 r\_ui = 4.00 est = 4.84 {'was\_impossible': False}  
 '''  
  
 # 定义K折交叉验证迭代器，K=3  
 # k 值越大，泛化结果越好，但是时间也越长，一般取5到10就好  
 kf = KFold(n\_splits=3)  
  
 for trainset,testset in kf.split(data):  
 # 训练并测试  
 algo.fit(trainset)  
 predictions = algo.test(testset)  
 # 计算RSME  
 accuracy.rmse(predictions,verbose=True)  
  
 # 196这个用户对于302这个电影的预测分数是怎么样的  
 uid = str(196)  
 iid = str(302)  
 #输出uid对iid的预测结果  
 pred = algo.predict(uid,iid,r\_ui=4,verbose=True)

## Surprise中的SlopeOne算法

**SlopeOne算法**：  
 2005 年提出的一个Item-Based 的协同过滤推荐算法  
 最大优点在于算法很简单, 易于实现, 效率高且推荐准确度较高。

如下C对商品2的评分=4-((5-3)+(4-3))/2=2.5

Table

Description automatically generated

Text

Description automatically generated  
  
 举例：  
 a b c d  
 A 5 3.5  
 B 2 5 4 2  
 C 4.5 3.5 1 4  
 解释： 用户A 对商品a 的评分是5，对商品b的品分是3.5，没有对c和d的，预测出对c和d的，并按预测的分数高低对A进行推荐。  
  
 Step1，计算Item之间的评分差的均值  
 b与a：((3.5-5)+(5-2)+(3.5-4.5))/3=0.5/3  
 c与a：((4-2)+(1-4.5))/2=-1.5/2  
 d与a：((2-2)+(4-4.5))/2=-0.5/2  
 c与b：((4-5)+(1-3.5))/2=-3.5/2  
 d与b：((2-5)+(4-3.5))/2=-2.5/2  
 d与c：((2-4)+(4-1))/2=1/2

得到下表：  
 a b c d  
 a  
 b 0.17  
 c -0.75 -1.75  
 d -0.25 -1.25 0.5  
  
 Step2，预测用户A对商品c和d的评分  
 a对c评分=((-0.75+5)+(-1.75+3.5))/2==3  
 a对d评分=((-0.25+5)+(-1.25+3.5))/2=3.5  
  
 Step3，将预测评分排序，推荐给用户  
 推荐顺序为{d, c}

**加权算法 Weighted Slope One**  
如果有10个用户对Item1和Item2都打过分, 有100个用户对Item3和Item2也打过分，显然这两个rating差的权重是不一样的，因此计算方法为：  
(10\*(Rating 1 to 2) + 100(Rating 3 to 2)) / (10 + 100)  
  
SlopeOne算法的特点：  
适用于item更新不频繁，数量相对较稳定  
item数<<user数  
算法简单，易于实现，执行效率高  
依赖用户行为，存在冷启动问题和稀疏性问题  
  
 import pandas as pd  
 from surprise import Reader,Dataset  
 from surprise import SlopeOne  
  
 # 数据读取  
 reader = Reader(line\_format='user item rating timestamp',sep=',',skip\_lines=1)  
 data = Dataset.load\_from\_file('ratings.csv',reader=reader)  
 train\_set = data.build\_full\_trainset()  
  
 #使用SlopeOne算法  
 algo = SlopeOne()  
 algo.fit(train\_set)  
  
 #对指定用户和商品进行评分预测  
 uid = str(196)  
 iid = str(302)  
 pred = algo.predict(uid,iid,r\_ui=4,verbose=True)  
  
 #user: 196 item: 302 r\_ui = 4.00 est = 4.32 {'was\_impossible': False}

## 原理 K折交叉验证

交叉验证（Cross Validation）为CV。

Chart, bar chart

Description automatically generated  
 基本思想：

1.将训练集数据划分为K份，使用其中的K-1份作为训练集，剩余一份作为测试集

2.K次误差的平均值作为泛化误差

3.所有数据都做过训练和测试，更好的利用数据

4.K越大，平均误差作为泛化误差的结果就越可靠，但花费的时间也时间也越长

性能指标。  
 Kfold:  
 原始数据分成K组（一般是均分），将每个子集数据分别做一次验证集，其余的k-1组子集数据作为训练集，  
 这样会得到k个模型，用这k个模型最终的验证集的分类准确率的平均数作为此K-CV下分类器的性能指标。

StratifiedKFlod：  
 是K-Flod的变种，会返回stratified(分层的折叠)；每个小集合中，各个分类的样例比例大致和完整数据集中相同。  
  
 另一种交叉验证：  
  
 参见L2-titanic\_clean.py  
 # 使用K折交叉验证 统计决策树准确率  
 print(u'cross\_val\_score准确率为 %.4lf' % np.mean(cross\_val\_score(clf, train\_features, train\_labels, cv=10)))  
  
 两者区别  
 https://zhuanlan.zhihu.com/p/24825503

## surprise中的评价指标：RMSE，MSE，MAE, FCP

RMSE 均方根误差



MSE 均方误差



MAE 平均绝对误差



FCP，Fraction of Concordant Pairs 一致对的分数

A screenshot of a cell phone

Description automatically generated

## Pandas使用

在数据分析中，Pandas的使用频率很高

Pandas可以说是基于 NumPy 构建的含有更高级数据结构和分析能力的工具包

Series和 DataFrame是两个核心数据结构，分别代表着一维的序列和二维的表结构

基于这两种数据结构，Pandas可以对数据进行导入、清洗、处理、统计和输出

Table

Description automatically generated

### Series

一维数组，是个定长的字典序列

Series有两个基本属性：index 和 values

from pandas import Series, DataFrame

x1 = Series([1,2,3,4])

x2 = Series(data=[1,2,3,4], index=['a', 'b', 'c', 'd'])

# 使用字典来进行创建

d = {'a':1, 'b':2, 'c':3, 'd':4}

x3 = Series(d)

print(x1)

print(x2)

print(x3)

### Dataframe

类似数据库表，包括了行索引和列索引，可以将DataFrame 看成是由相同索引的Series组成的字典类型

from pandas import Series, DataFrame

data = {'Chinese': [66, 95, 93, 90,80], 'Math': [30, 98, 96, 77, 90], 'English': [65, 85, 92, 88, 90]}

df1 = DataFrame(data)

df2 = DataFrame(data, index=['ZhangFei', 'GuanYu', 'LiuBei', 'DianWei', 'XuChu'], columns=['Chinese', 'Math', 'English'])

print(df1)

print(df2)

### Apply函数

apply函数是Pandas中自由度非常高的函数，使用频率高

比如对name列的数值都进行大写转化

df['name'] = df['name'].apply(str.upper)

也可以定义个函数，在apply中进行使用

def double\_df(x):

return 2\*x

df1[u'语文'] = df1[u'语文'].apply(double\_df)

### Map函数

map(function, iterable, ...)

Python 内置的函数，它接收一个函数 f 和一个 list，把函数 f 依次作用在 list 的每个元素上，得到一个新的 list 并返回

# 计算列表各个元素的平方

def square(x):

return x \* x

print(list(map(square, [1,2,3,4,5]))) –》 A picture containing text

Description automatically generated

#### Thinking: apply和map的区别是什么？

apply 用在dataframe上，用于对row或者column进行计算

applymap 用于dataframe上，是元素级别的操作

map ，是python自带的，用于series上，是元素级别的操作

### Lambda函数

lambda 参数:操作(参数)

也称为匿名函数。如果你不想在程序中对一个函数使用两次，可以用lambda表达式，作用和普通函数一样

=>一个只用一行就能解决问题的函数

add = lambda x, y: x + y

print(add(5, 6))

# 按照x[1]进行列表排序

a = [(2, 56), (3, 12), (6, 10), (9, 13)]

a.sort(key=lambda x: x[1])

print(a)

A picture containing text

Description automatically generated

练习： 新列表 = 原列表的平方

使用map, lambda两种方法完成

numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9]

输出结果

[1, 4, 9, 16, 25, 36, 49, 64, 81]

===》

# 计算平方数

def square(x):

return x \* x

print(list(map(square, numbers)))

# 使用lambda定义函数

print(list(map(lambda x: x\*x, numbers)))

### Pandas中的统计函数

count() 统计个数，空值NaN不计算

describe() 一次性输出多个统计指标，包括：count, mean, std, min, max等

min()最小值

max()最大值

sum()总和

mean()平均值

median()中位数

var()方差

std()标准差

argmin() 统计最小值的索引位置

argmax()统计最大值的索引位置

idxmin() 统计最小值的索引值

idxmax() 统计最大值的索引值

A picture containing table

Description automatically generated

### Loc函数和iloc函数

loc函数：通过行索引 "Index" 中的具体值来取行数据（如取"Index"为"A"的行）

iloc函数：通过行号来取行数据（如取第二行的数据）

Diagram

Description automatically generated

from pandas import Series, DataFrame

data = {'Chinese': [66, 95, 93, 90,80], 'Math': [30, 98, 96, 77, 90], 'English': [65, 85, 92, 88, 90]}

df = DataFrame(data, index=['ZhangFei', 'GuanYu', 'LiuBei', 'DianWei', 'XuChu'], columns=['Chinese', 'Math', 'English'])

# 提取Index为ZhangFei的行

print(df.loc['ZhangFei'])

# 提取第0行

print(df.iloc[0])

A picture containing text

Description automatically generated

# 提取列为English的所有行

print(df.loc[:,['English']])

# 提取第2列的所有行

print(df.iloc[:,2])

Text

Description automatically generated

# 查看ZhangFei, GuanYu的Chinese Math成绩

print(df.loc[['ZhangFei','GuanYu'], ['Chinese','Math']])

print(df.iloc[[0,1],[0,1]])

A picture containing diagram

Description automatically generated

### Groupby函数

作用是进行数据的分组以及分组后地组内运算

import numpy as np

import pandas as pd

# 因为文件中有中文，所以采用gbk编码读取

data = pd.read\_csv('heros2.csv', encoding='gbk')

result = data.groupby('role').agg([np.sum, np.mean])

print(result)

A picture containing diagram

Description automatically generated

Text

Description automatically generated

## Project：基于Syntax Tree的对话机器人

# 定语从句语法

grammar = '''

战斗 => 施法 ， 结果 。

施法 => 主语 动作 技能

结果 => 主语 获得 效果

主语 => 张飞 | 关羽 | 赵云 | 典韦 | 许褚 | 刘备 | 黄忠 | 曹操 | 鲁班七号 | 貂蝉

动作 => 施放 | 使用 | 召唤

技能 => 一骑当千 | 单刀赴会 | 青龙偃月 | 刀锋铁骑 | 黑暗潜能 | 画地为牢 | 守护机关 | 狂兽血性 | 龙鸣 | 惊雷之龙 | 破云之龙 | 天翔之龙

获得 => 损失 | 获得

效果 => 数值 状态

数值 => 1 | 1000 |5000 | 100

状态 => 法力 | 生命

'''

# 得到语法字典

def getGrammarDict(gram, linesplit = "\n", gramsplit = "=>"):

#定义字典

result = {}

for line in gram.split(linesplit):

# 去掉首尾空格后，如果为空则退出

if not line.strip():

continue

expr, statement = line.split(gramsplit)

result[expr.strip()] = [i.split() for i in statement.split("|")]

return result

# 生成句子

def generate(gramdict, target, isEng = False):

if target not in gramdict:

return target

find = random.choice(gramdict[target])

blank = ''

# 如果是英文中间间隔为空格

if isEng:

blank = ' '

return blank.join(generate(gramdict, t, isEng) for t in find)

gramdict = getGrammarDict(grammar)

print(generate(gramdict,"战斗"))

print(generate(gramdict,"战斗", True))

Text

Description automatically generated

## Paper Reading

知云文献翻译

院士的论文引用次数至少要达到500次

Paper Reading是个痛苦&收获的过程

文字太多，代码就一句话的事

经典&有影响力的论文都是英语，阅读障碍

原理看不懂，遇到不会的地方就放弃了

论文的价值

* 严谨性，一篇论文的产生可能需要3-6个月的周期
* 完整性，从问题背景，他人方案，解决方案，实验结果，后续工作
* 启发性，在阅读过程中会带有自己的思考，新idea的产生

**代码可以给你交付感，Paper给你坚实的基础。想要有更好的提升，Paper Reading是必经之路**

阅读论文方法

* 先看Survey（对于新领域）

对研究的内容有个深入广泛的了解，知道最新的研究进展（state of the art），奠基性的成果有哪些（经典paper，数据集benchmark），还有哪些未来值得研究的地方，哪些技术比较成熟可以直接使用

* 关注重要的国际会议

相比来说文章比期刊更短，时效性更强。应用性更强

* 作者的代码及Blog

国外很多论文都提供源代码，如果没有在网站上，可以写email给作者

很多作者是大学教授，或者在读博士生，会有个人主页，关于他论文list及研究方向

阅读论文方法

* 粗读

一次性读完，即使有很多不懂的地方

如果一遍没读懂，可以多读几遍，不要放弃。

如果是经典的论文，往往有其他人的解读，可以作为理解的补充

粗读：不需要一次性都读明白，可以多读几遍

* 精读

带着想法读论文，可能会产生新的idea，记录下来

对论文自己做个summary或者思维导图，转化为自己的知识，语言体系

精读：针对重要的部分，反复读直到理解为止

* 跳读

找到一篇文章中找到重要的部分，进行精读

跳读：带着问题去思考，在文章中寻找自己的答案

如何下载论文

* 学校图书馆内网
* 百度学术/Google Scholar
* arXiv.org

一个收录科学文献预印本的在线数据库，每个人都可以免费地访问全文数据

超过50万篇文章，每个月增长5000篇

包含：数学，物理，计算机，非线性科学，定量生物学，定量财务以及统计学等

## 思考与练习

### Thinking1 ALS都有哪些应用场景

1.ALS是一种重要的数据拟合技术，可应用于线形回归、非线形回归

2.ALS是一种数学的优化方法，可以解决最优化问题。

在推荐的场景中使用较多。

ALS分为显示反馈和隐式反馈两种：显示反馈ALS、隐式反馈ALS-WR。

ALS适用于解决由明确评分矩阵的应用场景；

ALS-WR通过置信度权重c来解决用户没有明确反馈对物品的偏好，即对于更确信用户偏好的项赋予较大的权重，对于没有反馈的项，赋予较小的权重。

### Thinking2 ALS进行矩阵分解的时候，为什么可以并行化处理

ALS是先固定X求解Y，再固定Y求解X，在矩阵求解过程中，比如固定Y，求解X的时候，目标评分矩阵A，X的每一行可以独立求解，X的第I行和Y计算得到A的第i行，这样对每一步的X或者Y的行或者列都可以独立并行求解，故ALS可以进行并行化计算。

### Thinking3 梯度下降法中的批量梯度下降（BGD），随机梯度下降（SGD），和小批量梯度下降有什么区别（MBGD）

区别在于每次迭代使用的样本量不同

1.BGD是最原始的形式，每次迭代时使用所有样本来进行梯度的更新。

优点：

1）一次迭代是对所有样本进行计算，比较稳定，利用矩阵进行操作，实现了并行；

2）由全数据集确定的方向能够更好地代表样本总体，能更准确地朝极值所在的方向。当目标函数是凸函数时，BGD能够得到全局最优。

缺点：

当样本数目m很大时，每迭代一步都需要对所有样本计算，训练过程很慢，收敛慢。

2.SGD是每次迭代从m个样本随机抽取一个样本来对参数进行更新，使得训练速度加快。优点：

1）由于不是在全部训练数据上的损失函数，而是在每轮迭代中，随机优化某一条训练数据上的目标函数，这样每一轮参数的更新速度大大加快，收敛快，最终解在全局最优解附近。

缺点：

1）准确度下降，即使在目标函数为强凸函数的情况下，SGD仍旧无法做到线性收敛；2）可能会收敛到局部最优，由于单个样本并不能代表全体样本的趋势；

3）不易于并行实现。

3.MBGD是综合了BGD和SGD的优缺点，每次迭代只使用一部分样本batch\_size来进行更新。

优点：

1）通过矩阵运算，每次在一个batch上优化参数并不会比单个数据慢太多；

2）每次使用一个batch可以大大减小收敛所需要的迭代次数；

3）可实现并行化。

缺点：batch\_size的不当选择可能会带来的一些问题

### Thinking4 你阅读过和推荐系统/计算广告/预测相关的论文么？有哪些论文是你比较推荐的，可以分享到微信群中

之前对于这方面的论文阅读确实很少，推荐大家一个我之前只是粗略阅读过的文章，题目是《Variational Autoencoders for Collaborative Filtering 》

A close up of a newspaper

Description automatically generated

## Action1：对MovieLens数据集进行评分预测

数据集：MovieLens

工具：可以使用Surprise或者其他

说明使用的模型，及简要原理

## Action2：Paper Reading（1周）

Slope one predictors for online rating-based collaborative filtering. Daniel Lemire and Anna Maclachlan, 2007. http://arxiv.org/abs/cs/0702144.

积累，总结笔记，自己的思考及idea

## Action3：设计你自己的句子生成器

grammar = '''

战斗 => 施法 ， 结果 。

施法 => 主语 动作 技能

结果 => 主语 获得 效果

主语 => 张飞 | 关羽 | 赵云 | 典韦 | 许褚 | 刘备 | 黄忠 | 曹操 | 鲁班七号 | 貂蝉

动作 => 施放 | 使用 | 召唤

技能 => 一骑当千 | 单刀赴会 | 青龙偃月 | 刀锋铁骑 | 黑暗潜能 | 画地为牢 | 守护机关 | 狂兽血性 | 龙鸣 | 惊雷之龙 | 破云之龙 | 天翔之龙

获得 => 损失 | 获得

效果 => 数值 状态

数值 => 1 | 1000 |5000 | 100

状态 => 法力 | 生命

'''

host = """

host = 寒暄 报数 询问 具体业务 结尾

报数 = 我是工号 数字 号 ,

数字 = 单个数字 | 数字 单个数字

单个数字 = 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

寒暄 = 称谓 打招呼 | 打招呼

称谓 = 人称 ,

人称 = 先生 | 女士 | 小朋友

打招呼 = 你好 | 您好

询问 = 请问你要 | 您需要

具体业务 = 办信用卡| 账单分期| 充话费| 买手机

结尾 = 吗？"""

这节课感觉内容超多，尤其是安装环境和读论文真的还挺费时间的，导致很多原理性的东西还没有好好研究很深刻，我觉得学习之后一定要好好复习，及时巩固！这里就先整理到这里，以后在好好补充！

老师辛苦，这里谢谢老师！

最后，继续加油！希望以后找到大厂推荐算法工程师的工作！