Worcester Polytechnic Institute

Independent Study Report

Online Karaoke Data Analytics

Wa Gao, Feng Zheng, Xiaoting Cui
Advisor: Yanhua Li
Technical Supporter: Hengyu Xu

Spring 2018

一、项目目标

用户快速找到目标信息有两种途径，搜索引擎和推荐系统。在用户对自己需求相对明确的时候，用搜索引擎很方便的通过关键字搜索很快的找到自己需要的信息。同时，在很多情况下，用户其实并不明确自己的需要，或者他们的需求很难用简单的关键字来表述。又或者他们需要更加符合他们个人口味和喜好的结果，因此出现了推荐系统既推荐引擎。

推荐引擎是主动发现用户当前或潜在需求的定律，并主动推送信息给用户的信息网络。推荐引擎综合利用户的行为、属性，对象的属性、内容、分类，以及用户之间的社交关系等等，挖掘用户的喜好和需求，主动向用户推荐其感兴趣或者需要的对象。天籁推荐引擎在分析和学习用户听歌，点唱和分享歌曲的基础上，提前预测用户的需求，从而为用户推荐更加精准和个性化的内容个性化的推荐内容，提高新用户的留存率和老用户的活跃度，从而达到商业目标。

二、数据描述及预处理

在天籁用户的数据中，总共有 44,827 名用户的 6,697,037 条记录，总共有 user, song, singer, user work 的 50 个特征。

在数据预处理的过程中，若 user 的记录少于 5 条，且 user work 的记录数小于 3 条，则忽略该 user 和 user work 以降低数据稀疏性。另外，根据 user id 和 user work id 将记录分类。

三、Predictive Model 算法设计

(一) XGBoost

1. XGBoost 算法简介

XGBOOST 的全称是 Extreme Gradient Boosting，是属于 boosting 迭代型、树类算法，该算法适用于分类和回归问题，Boosting 翻译过来就是提升的意思，通过研究如果将许多个弱分类器集成在一起提升为一个强分类器就是多数 boosting 算法所研究的内容。由于 XGBoost 是提升树模型，所以它与决策树是息息相关的，它通过将很多的决策树集成起来，从而得到一个很强的分类器。

2. XGBoost 算法的优点

(1) 正则化：对叶节点权重进行了惩罚，相当于添加了正则项，防止过拟合；

(2) 并行数理：XGBoost 可以实现并行处理，相比 GBM 有了速度的飞跃。同时也支持 Hadoop 实现；

(3) 高度的灵活性：XGBoost 允许用户定义自定义优化目标和评价标准；
(4) 缺失值处理：XGBoost 内置处理缺失值的规则，用户需要提供一个和其它样本不同的值，然后把它作为一个参数传进去，以此来作为缺失值的取值。XGBoost 在不同节点遇到缺失值时采用不同的处理方法，并且会学习未来遇到缺失值时的处理方法；
(5) 剪枝：XGBoost 会一直分裂到指定的最大深度(max_depth)，然后回过头来剪枝。如果某个节点之后不再有正值，它会去除这个分裂；
(6) 内置交叉验证：XGBoost 允许在每一轮 boosting 迭代中使用交叉验证。因此，可以方便地获得最优 boosting 迭代次数；
(7) 在已有的模型基础上继续: XGBoost 可以在上一轮的结果上继续训练。这个特性在某些特定的应用上是一个巨大的优势学习到一棵树后，对其权重进行缩减，从而降低该棵树的作用，提升可学习空间。

3. XGBoost 的参数

XGBoost 的参数主要有：通用参数：宏观函数控制；Booster 参数：控制每一步的 booster(tree/regression)；学习目标参数：控制训练目标的表现。

(A) 通用参数

这些参数用来控制 XGBoost 的宏观功能。

(1) booster[默认 gbtree]

选择每次迭代的模型，有两种选择： gbtree：基于树的模型；gbliner：线性模型

(2) silent[默认 0]

当这个参数值为 1 时，静默模式开启，不会输出任何信息。一般这个参数就保持默认的 0，因为这样能帮我们更好地理解模型。

(3) nthread[默认值为最大可能的线程数]

这个参数用来进行多线程控制，应当输入系统的核数。如果你希望使用 CPU 全部的核，那就不要输入这个参数，算法会自动检测它。

还有两个参数，XGBoost 会自动设置，目前你不用管它。接下来咱们一起看 booster 参数。

(B) booster 参数

尽管有两种 booster 可供选择，我这里只介绍 tree booster，因为它的表现远

远胜过 linear booster，所以 linear booster 很少用到。

(1) eta[默认 0.3]

和 GBM 中的 learning rate 参数类似。通过减少每一步的权重，可以提高模型的鲁棒性。

典型值为 0.01-0.2。

(2) min_child_weight[默认 1]

决定最小叶子节点样本权重和。和 GBM 的 min_child_leaf 参数类似，但不完全一样。

XGBoost 的这个参数是最小样本权重的和，而 GBM 参数是最小样本总数。这个参数用于避免过拟合。当它的值较大时，可以避免模型学习到局部的特殊样本。但是如果这个值过高，会导致欠拟合。这个参数需要使用 CV 来调整。

(3) max_depth[默认 6]

和 GBM 中的参数相同，这个值为树的最大深度。这个值也是用来避免过拟合的。

max_depth 越大，模型会学到更具体更局部的样本。需要使用 CV 函数来进行调优。典型值：3-10。

(4) max_leaf_nodes

树上最大的节点或叶子的数量。可以替代 max_depth 的作用。因为如果生成的是二叉树，一个深度为 n 的树最多生成 n2 个叶子。

(5) gamma[默认 0]

在节点分裂时，只有分裂后损失函数的值下降了，才会分裂这个节点。Gamma 指定了节点分裂所需的最小损失函数下降值。这个参数的值越大，算法越保守。这个参数的值和损失函数息息相关，所以是需要调整的。

(6) max_delta_step[默认 0]

这参数限制每棵树权重改变的最大步长。如果这个参数的值为 0，那就意味着没有约束。如果它被赋予了某个正值，那么它会让这个算法更加保守。通常，这个参数不需要设置。但是当各类别的样本十分不平衡时，它对逻辑回归是很有帮助的。这个参数一般用不到，但是你可以挖掘出来它更多的用处。

(7) subsample[默认 1]

这个参数控制对于每棵树，随机采样的比例。减小这个参数的值，算法会更加保守，避免过拟合。但是，如果这个值设置得过小，它可能会导致欠拟合。典型值：0.5-1。

(8) colsample_bytree[默认 1]

用来控制每棵随机采样的列数的占比(每一列是一个特征)。典型值：0.5-1。

(9) colsample_bylevel[默认 1]

用来控制树的每一级的每一次分裂，对列数的采样的占比。

(10) lambda[默认 1]

权重的 L2 正则化项。(和 Ridge regression 类似)。这个参数是用来控制 XGBoost 的正则化部分的。虽然大部分数据科学家很少用到这个参数，但是这个参数在减少过拟合上还是可以挖掘出更多用处的。

(11) alpha[默认 0]

权重的 L1 正则化项。(和 Lasso regression 类似)。可以应用在很高维度的情况下，使得算法的速度更快。

(12) scale_pos_weight[默认 1]

在各类别样本十分不平衡时，把这个参数设定为一个正值，可以使算法更快收敛。

(C) 学习目标参数

这个参数用来控制理想的优化目标和每一步结果的度量方法。

(1) objective[默认 reg:linear]

这个参数定义需要被最小化的损失函数。最常用的值有 :binary:logistic 二分类的逻辑回归，返回预测的概率(不是类别)。multi:softmax 使用 softmax 的多分类器，返回预测的类别(不是概率)。在这种情况下，你还需要多设一个参数：num_class(类别数目). multi:softprob 和 multi:softmax 参数一样，但是返回的是每个数据属于各个类别的概率。

(2) eval_metric[默认值取决于 objective 参数的取值]

对于有效数据的度量方法.对于回归问题，默认值是 rmse，对于分类问题，默认值是 error。

典型值有：

rmse 均方根误差（）

mae 平均绝对误差（）

logloss 负对数似然函数值

error 二分类错误率(阈值为 0.5)

merror 多分类错误率

mlogloss 多分类 logloss 损失函数

auc 曲线下面积

(3) seed(默认 0)

随机数的种子，设置它可以复现随机数据的结果，也可以用于调整参数。

4. 天籁项目-XGBoost

(1) 数据预处理

新增 Score 项，即根据用户对歌曲的行为打分

temp.loc[temp.activity==u'送礼物','score']=12

temp.loc[temp.activity==u'评论','score']=10

temp.loc[temp.activity==u'播放','score']=2

temp.loc[temp.activity==u’下载’,'score']=6

(2) Feature selection(特征值选择)

‘gender_x', 'constellation_x', 'region_x', 'user_age_x', 'user_age_y', 'region_y', 'gender_y', 'constellation_y’, 'song_lang', 'song_genre', 'singer_m', 'singer_f', 'singer_g', 'singer_sex', 'score'共 15 个特征值。

(3) 参数设置

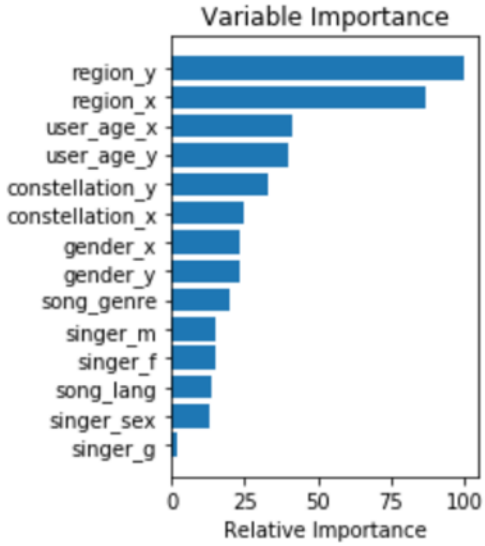
'n_estimators': 1000, 'max_depth': 100, 'min_samples_split': 2, 'learning_rate': 0.01, 'loss':

'huber','min_samples_leaf': 50, 'max_features': ‘sqrt', 'subsample': 0.8

(4) 预测结果：

MSE: 2494.2659

(5) feature 相关性：



(二) RANDOM FOREST

1. Random Forest 算法简介

随机森林(Random Forest)就是通过集成学习的思想将多棵树集成的一种算法，它的基本单元是决策树，而它的本质属于机器学习的一大分支——集成学习(Ensemble Learning)方法。

2. Random Forest 算法的优点

(1) 在目前的算法中，随机森林具有很好的准确率；

(2) 可以有效地运行在大数据集上；

(3) 能够处理具有高维特征的输入样本，而且不需要降维；

(4) 能够评估各个特征在分类问题上的重要性；

(5) 在生成过程中，能够获取到内部生成误差的一种无偏估计；

(6) 对于缺省值问题也能够获得很好得结果。

3. Random Forest 的相关定义

分类器：分类器就是给定一个样本的数据，判定这个样本属于哪个类别的算法。例如在股票涨跌预测中，我们认为前一天的交易量和收盘价对于第二天的涨

跌是有影响的，那么分类器就是通过样本的交易量和收盘价预测第二天的涨跌情况的算法。

分裂：在决策树的训练过程中，需要一次次的将训练数据集分裂成两个子数据集，这个过程就叫做分裂。

特征：在分类问题中，输入到分类器中的数据叫做特征。以上面的股票涨跌预测问题为例，特征就是前一天的交易量和收盘价。

待选特征：在决策树的构建过程中，需要按照一定的次序从全部的特征中选取特征。待选特征就是在目前的步骤之前还没有被选择的特征的集合。例如，全部的特征是 ABCDE，第一步的时候，待选特征就是 ABCDE，第一步选择了 C，那么第二步的时候，待选特征就是 ABDE。

分裂特征：接待选特征的定义，每一次选取的特征就是分裂特征，例如，在上面的例子中，第一步的分裂特征就是 C。因为选出的这些特征将数据集分成了一个个不相交的部分，所以叫它们分裂特征。

4. Random Forest 的实现随机森林算法分 bagging 和决策树。

bagging 的过程是将所有 user 的 training data 进行随机选择，并对其进行 collaborative filtering，得到的结果是 base estimator。之后继续对 feature 进行 bagging 操作，随机选出部分特征进行训练。这种方法可以减小误差。

决策树是一种基本的分类器，一般是将特征分为两类。构建好的决策树呈树形结构，可以认为是 if-then 规则的集合，主要优点是模型具有可读性，分类速度快。

通过 bagging + decision trees，我们可以得到随机森林。将决策树作为 base estimator，然后采用 bagging 技术训练一大堆小决策树，最后将这些小决策树组合起来，这样就得到了一片森林(随机森林)。

5. 随机森林的参数

参数	参数类型	RandomForestClassifier	RandomForestRegressor	GradientBoostingClassifier	GradientBoostingRegressor
loss	目标	/	/	损失函数	损失函数

alpha	目标	/	/	损失函数为 huber 或 quantile 时，alpha 为损失函数中的参数	损失函数为 huber 或 quantile 时，alpha 为损失函数中的参数
class_weight	目标	类别权值	/	/	/
n_estimators	性能	子模型数量 -int: 个数 -默认 10	子模型数量 -int: 个数 -默认 10	子模型数量 -int: 个数 -默认 100	子模型数量 -int: 个数 -默认 100
learning_rate	性能	/	/	学习率	学习率
criterion	性能	计算方法，判断节点是否继续分裂 -entropy -gini	计算方法，判断节点是否继续分裂 -mse	/	/

max_features	性能	节点分裂时，参与判断的最大特征数 -int：个数 -float：占特征百分比 -auto：所有特征开方 -sqrt：所有特征开方 -log2：所有特征对数 -None：=所有特征	节点分裂时，参与判断的最大特征数 -int：个数 -float：占特征百分比 -auto：所有特征开方 -sqrt：所有特征开方 -log2：所有特征对数 -None：=所有特征	节点分裂时，参与判断的最大特征数 -int：个数 -float：占特征百分比 -auto：所有特征开方 -sqrt：所有特征开方 -log2：所有特征对数 -None：=所有特征	节点分裂时，参与判断的最大特征数 -int：个数 -float：占特征百分比 -auto：所有特征开方 -sqrt：所有特征开方 -log2：所有特征对数 -None：=所有特征
max_depth	性能	最大深度(若max_leaf_nodes已制定，则忽略) -int：深度 -None：树会生长到所有叶子都分到一个类，或某节点代表的样本数小于min_samples_split	最大深度(若max_leaf_nodes已制定，则忽略) -int：深度 -None：树会生长到所有叶子都分到一个类，或某节点代表的样本数小于min_samples_split	最大深度(若max_leaf_nodes已制定，则忽略) -int：深度 -默认 3	最大深度(若max_leaf_nodes已制定，则忽略) -int：深度 -默认 3

min_samples_split	性能	分裂所需最小样本数 -int: 样本数 -默认 2	分裂所需最小样本数 -int: 样本数 -默认 2	分裂所需最小样本数 -int: 样本数 -默认 2	分裂所需最小样本数 -int: 样本数 -默认 2
min_samples_leaf	性能	叶子节点最小样本数 -int: 样本数 -默认 1	叶子节点最小样本数 -int: 样本数 -默认 1	叶子节点最小样本数 -int: 样本数 -默认 1	叶子节点最小样本数 -int: 样本数 -默认 1
min_weight_fraction_leaf	性能	叶子节点最小样本权重总值 -float: 权重总值 -默认 0	叶子节点最小样本权重总值 -float: 权重总值 -默认 0	叶子节点最小样本权重总值 -float: 权重总值 -默认 0	叶子节点最小样本权重总值 -float: 权重总值 -默认 0
max_leaf_nodes	性能	最大叶节点数 -int: 个数 －None: 不限制叶节点数	最大叶节点数 -int: 个数 －None: 不限制叶节点数	最大叶节点数 -int: 个数 －None: 不限制叶节点数	最大叶节点数 -int: 个数 －None: 不限制叶节点数
bootstrap	性能	是否 bootstrap 对样本抽样 -False -True: 默认值	是否 bootstrap 对样本抽样 -False -True: 默认值	/	/
subsample	性能	/	/	子采样率 -float: 采样率 -默认 1.0	子采样率 -float: 采样率

					-默认 1.0
init	性能	/	/	初始子模型	初始子模型
n_jobs	效率	并行数 -int: 个数 -与 CPU 核数相同时为-1 -默认 1	并行数 -int: 个数 -与 CPU 核数相同时为-1 -默认 1	/	/
warm_start	效率	是否热启动 若使用热启动，则下次训练以追加数形式进行 -bool: 热启动 -默认 False	是否热启动 若使用热启动，则下次训练以追加数形式进行 -bool: 热启动 -默认 False	/	/
presort	效率	/	/	是否预排序(加速查找最佳分裂点) -bool: 预排序 -auto: 非稀疏数据使用预排序，稀疏数据则不使用	是否预排序(加速查找最佳分裂点) -bool: 预排序 -auto: 非稀疏数据使用预排序，稀疏数据则不使用
oob_score	附加	是否计算袋外得分	是否计算袋外得分	/	/

		-默认 False	-默认 False		
random_state	附加	随机器对象	随机器对象	随机器对象	随机器对象
verbose	附加	日志冗长度 Int: 冗长度 -0: 不输出 -1: 偶尔输出 ->1: 对每个子模型都输出	日志冗长度 Int: 冗长度 -0: 不输出 -1: 偶尔输出 ->1: 对每个子模型都输出	日志冗长度 Int: 冗长度 -0: 不输出 -1: 偶尔输出 ->1: 对每个子模型都输出	日志冗长度 Int: 冗长度 -0: 不输出 -1: 偶尔输出 ->1: 对每个子模型都输出

6. 天籁项目-Random Forest

(1) 数据预处理

新增 Score 项，即根据用户对歌曲的行为打分

temp.loc[temp.activity==u'送礼物','score']=12

temp.loc[temp.activity==u'评论','score']=10

temp.loc[temp.activity==u'播放','score']=2

temp.loc[temp.activity==u’下载’,'score']=6

(2) Feature selection(特征值选择)

‘gender_x', 'constellation_x', 'region_x', 'user_age_x', 'user_age_y', 'region_y', 'gender_y', 'constellation_y’, 'song_lang', 'song_genre', 'singer_m', 'singer_f', 'singer_g', 'singer_sex', 'score'共 15 个特征值。

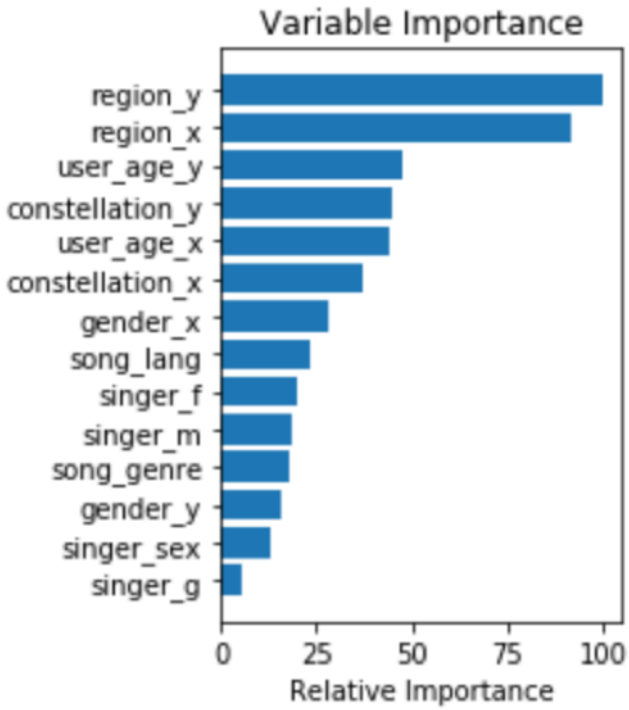
(3) 参数设置

'n_estimators': 1000, 'max_depth': 1000,'random_state': 2, 'random_state': 2

(4) 预测结果

MSE: 2649.2839

(5) feature 相关性



(三) SVR (Support Vector Regression)

1. SVR 简介

SVR 与 SVM 原理相同，唯一不同是目标值是连续的数值。SVM 支持向量机(Support Vector Machine)，是一种监督式学习的方法，可广泛地应用于统计分类以及回归分析。支持向量机属于一般化线性分类器，这族分类器的特点是他们能够同时最小化经验误差与最大化几何边缘区，因此支持向量机也被称为最大边缘区分类器。

2. SVR 的主要思想

(1) 所谓回归（regression），基本上就是拟合，用一个函数拟合 x 与 y 的关系。对于 SVR 来说，x 是向量，y 是标量，拟合的函数形式为 y=W^T*g(x)+b，其中 g(x)为核函数对应的特征空间向量；

(2) SVR 认为，只要估计的 y 在实际的 y 的两侧一个固定的范围(epsilon)之内，就认为是估计正确，没有任何损失；

(3) SVR 的优化目标，是|W|最小，这样 y-x 曲线的斜率最小，这个 function 最 flat，

这样据说可以增加估计的鲁棒性；

(4) 之后的事情就很自然了，和 SVM 一样：可以有 soft margin，用一个小正数控制。用对偶式来解；

(5) 但有一个不同，控制范围的 epsilon 的值难于确定，在最小优化目标中加入一项 C*\nu*\epsilon，其中 epsilon 是一个变量，nu 是一个预先给定的正数。

3. SVR 的参数

(1) booster[默认 gbtree]

选择每次迭代的模型，有两种选择： gbtree：基于树的模型；gbliner：线性模型。

(2) C [默认值为 1.0]

错误项的惩罚系数。C 越大，即对分错样本的惩罚程度越大，因此在训练样本中准确率越高，但是泛化能力降低，也就是对测试数据的分类准确率降低。相反，减小 C 的话，容许训练样本中有一些误分类错误样本，泛化能力强。对于训练样本带有噪声的情况，一般采用后者，把训练样本集中错误分类的样本作为噪声。

(3) kernel [默认为‘rbf’]

算法中采用的核函数类型，可选参数有：

‘linear’：线性核函数

‘poly’:多项式核函数

‘rbf’：径像核函数/高斯核

‘sigmod’: sigmod 核函数

‘precomputed’：核矩阵

(4) degree [默认为 3]

这个参数只对多项式核函数有用，是指多项式核函数的阶数 n。

如果给的核函数参数是其他核函数，则会自动忽略该参数。

(5) gamma [float 参数 默认为 auto]

核函数系数，只对‘rbf’,‘poly’,‘sigmod’有效。

如果 gamma 为 auto，代表其值为样本特征数的倒数，即 1/n_features。

(6) coef0 [float 参数 默认为 0.0]

核函数中的独立项，只有对‘poly’和‘sigmod’核函数有用，是指其中的参数 c

probability： bool 参数 默认为 False

是否启用概率估计。 这必须在调用 fit()之前启用，并且会 fit()方法速度变慢。

shrinking： bool 参数 默认为 True

是否采用启发式收缩方式。

(7) tol [默认为 1e^-3]

svm 停止训练的误差精度。

(8) cache_size [默认为 200]

指定训练所需要的内存，以 MB 为单位，默认为 200MB。

(9) class_weight [默认为 None]

给每个类别分别设置不同的惩罚参数 C，如果没有给，则会给所有类别都给 C=1，即前面参数指出的参数 C.如果给定参数’balance’，则使用 y 的值自动调整与输入数据中的类频率成反比的权重。

(10) verbose [默认为 False]

是否启用详细输出。 此设置利用 libsvm 中的每个进程运行时设置，如果启用，可能无法在多线程上下文中正常工作。一般情况都设为 False，不用管它。

(11) max_iter [默认为-1]

最大迭代次数，如果为-1，表示不限制。

(12) random_state[默认为 None]

伪随机数发生器的种子,在混洗数据时用于概率估计。

4. 天籁项目-SVR

(1) 数据预处理

新增 Score 项，即根据用户对歌曲的行为打分

temp.loc[temp.activity==u'送礼物','score']=12

temp.loc[temp.activity==u'评论','score']=10

temp.loc[temp.activity==u'播放','score']=2

temp.loc[temp.activity==u’下载’,'score']=6

(2) Feature selection(特征值选择)

‘gender_x', 'constellation_x', 'region_x', 'user_age_x', 'user_age_y', 'region_y', 'gender_y', 'constellation_y’, 'song_lang', 'song_genre', 'singer_m', 'singer_f', 'singer_g', 'singer_sex', 'score'共 15 个特征值。

(3) 参数设置

'C': 2.0, 'cache_size': 300,'coef0': 0.0, 'degree': 5 , 'epsilon': 0.9, 'gamma': 'auto','kernel': 'rbf', 'max_iter': -1, 'shrinking': True,'tol': 0.01,'verbose': False

(4) 预测结果

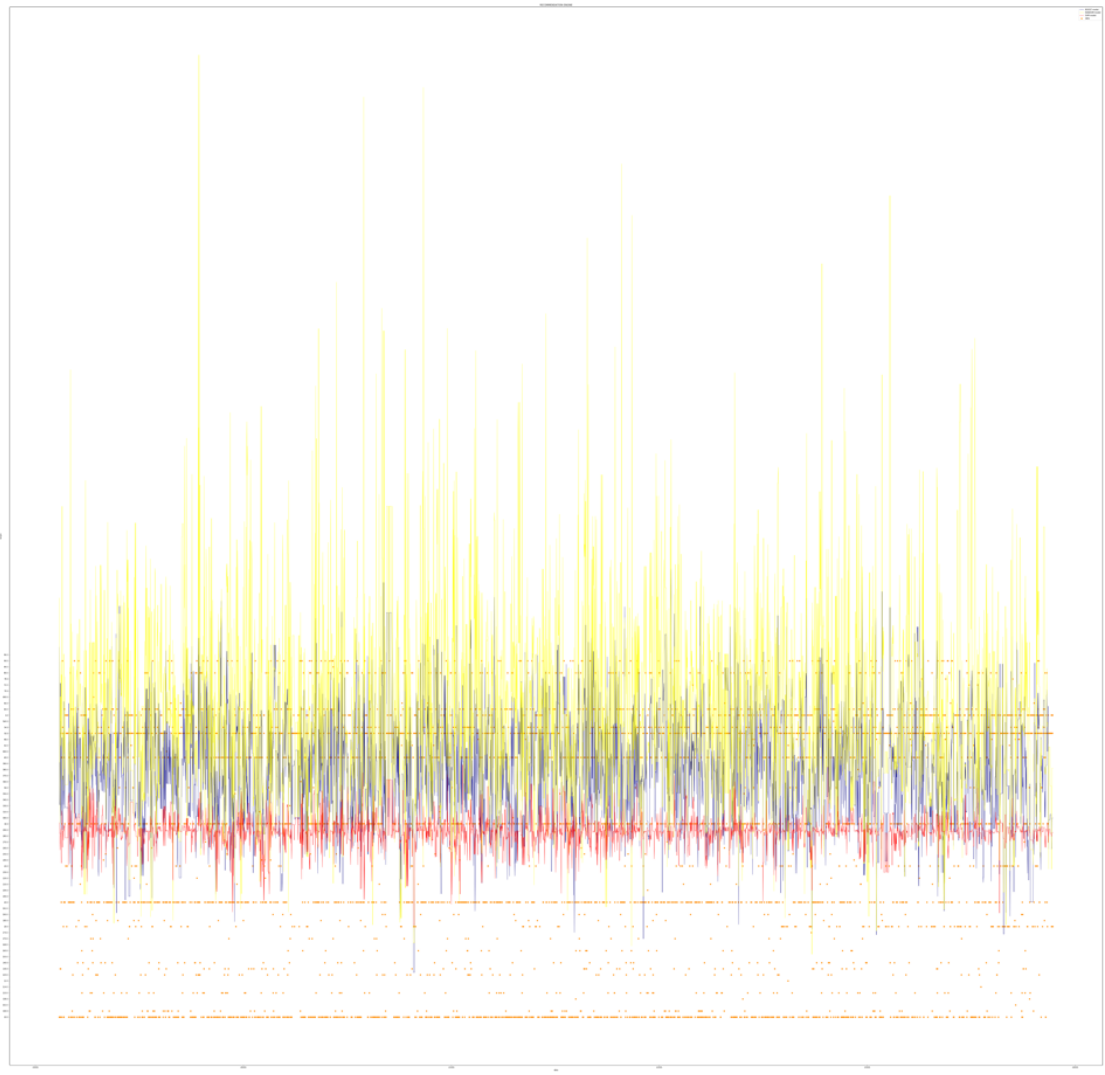
MSE: 2671.6001

(四) XGBoost, Random Forest 和 SVR 的结果对比

蓝色曲线－XGBoost

黄色曲线－Random Forest

红色曲线－SVR



四、Collaborative Filtering 算法

1. Collaborative Filtering 简介

协同过滤(Collaborative Filtering)推荐算法是诞生最早，并且较为著名的推荐算法。主要的功能是预测和推荐。算法通过对用户历史行为数据的挖掘发现用户的偏好，基于不同的偏好对用户进行群组划分并推荐品味相似的商品。协同过滤推荐算法分为两类，分别是基于用户的协同过滤算法(user-based collaborative filtering)，和基于物品的协同过滤算法(item-based collaborative filtering)。简单的说就是：人以类聚，物以群分。

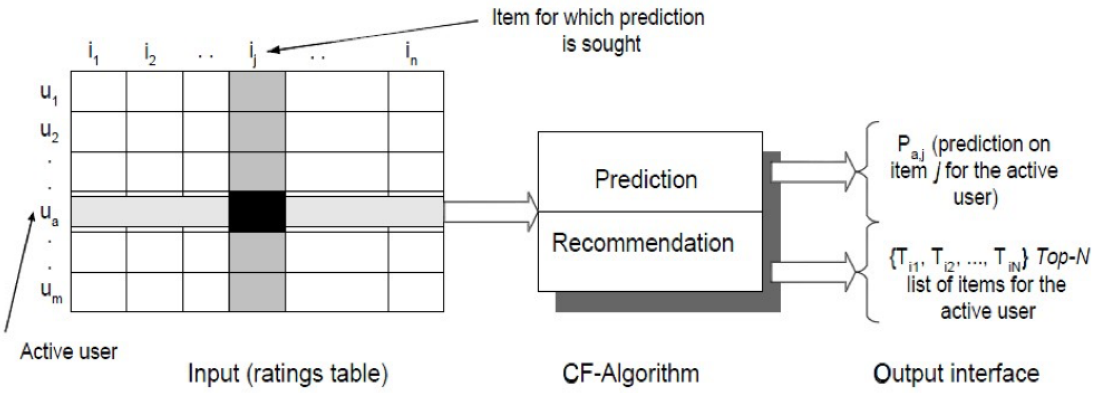


Figure 1: The Collaborative Filtering Process.

如图 1 所示，在 CF 中，用 m×n 的矩阵表示用户对物品的喜好情况，一般用打分表示用户对物品的喜好程度，分数越高表示越喜欢这个物品，0 表示没有买过该物品。图中行表示一个用户，列表示一个物品，Uij 表示用户 i 对物品 j 的打分情况。CF 分为两个过程，一个为预测过程，另一个为推荐过程。预测过程是预测用户对没有购买过的物品的可能打分值，推荐是根据预测阶段的结果推荐用户最可能喜欢的一个或 Top-N 个物品。

2. User-based 算法与 Item-based 算法对比

CF 算法分为两大类，一类为基于 memory 的(Memory-based)，另一类为基于 Model 的(Model-based)，User-based 和 Item-based 算法均属于 Memory-based 类型。

User-based 的基本思想是如果用户 A 喜欢物品 a，用户 B 喜欢物品 a、b、c，用户 C 喜欢 a 和 c，那么认为用户 A 与用户 B 和 C 相似，因为他们都喜欢 a，而喜欢 a 的用户同时也喜欢 c，所以把 c 推荐给用户 A。该算法用最近邻居 (nearest-neighbor)算法找出一个用户的邻居集合，该集合的用户和该用户有相似的喜好，算法根据邻居的偏好对该用户进行预测。

User-based 算法存在两个重大问题：

1. 数据稀疏性。一个大型的电子商务推荐系统一般有非常多的物品，用户可能买的其中不到 1%的物品，不同用户之间买的物品重叠性较低，导致算法无法找到一个用户的邻居，即偏好相似的用户；

2. 算法扩展性。最近邻居算法的计算量随着用户和物品数量的增加而增加，不适合数据量大的情况使用。

Iterm-based 的基本思想是预先根据所有用户的历史偏好数据计算物品之间的相似性，然后把与用户喜欢的物品相类似的物品推荐给用户。还是以之前的例

子为例，可以知道物品 a 和 c 非常相似，因为喜欢 a 的用户同时也喜欢 c，而用户 A 喜欢 a，所以把 c 推荐给用户 A。

因为物品直接的相似性相对比较固定, 所以可以预先在线下计算好不同物品之间的相似度，把结果存在表中，当推荐时进行查表，计算用户可能的打分值，可以同时解决上面两个问题。

3. Item-based 算法详细过程

(1) 相似度计算

Item-based 算法首选计算物品之间的相似度, 计算相似度的方法有以下几种：

(a) 基于余弦（Cosine-based）的相似度计算，通过计算两个向量之间的夹角余弦值来计算物品之间的相似性，公式如下：

$$sim(i,j) = \cos(\vec{i},\vec{j}) = \frac{\vec{i}\cdot\vec{j}}{||\vec{i}||_2 * ||\vec{j}||_2}$$

其中分子为两个向量的内积，即两个向量相同位置的数字相乘。

(b) 基于关联（Correlation-based）的相似度计算，计算两个向量之间的 Pearson-r 关联度，公式如下：

$$sim(i,j) = \frac{\sum_{u\in U}(R_{u,i}-\bar{R}_i)(R_{u,j}-\bar{R}_j)}{\sqrt{\sum_{u\in U}(R_{u,i}-\bar{R}_i)^2}\sqrt{\sum_{u\in U}(R_{u,j}-\bar{R}_j)^2}}.$$

其中$R_{u,i}$表示用户 u 对物品 i 的打分， 表示第 i 个物品打分的平均值。

(c) 调整的余弦（Adjusted Cosine）相似度计算，由于基于余弦的相似度计算没有考虑不同用户的打分情况，可能有的用户偏向于给高分，而有的用户偏向于给低分，该方法通过减去用户打分的平均值消除不同用户打分习惯的影响，公式如下：

$$sim(i,j) = \frac{\sum_{u\in U}(R_{u,i}-\bar{R}_u)(R_{u,j}-\bar{R}_u)}{\sqrt{\sum_{u\in U}(R_{u,i}-\bar{R}_u)^2}\sqrt{\sum_{u\in U}(R_{u,j}-\bar{R}_u)^2}}.$$

其中$\bar{R}_u$表示用户 u 打分的平均值。

(2) 预测值计算

根据之前算好的物品之间的相似度，接下来对用户未打分的物品进行预测，有两种预测方法：

(a) 加权求和

用过对用户 u 已打分的物品的分数进行加权求和，权值为各个物品与物品 i 的相似度，然后对所有物品相似度的和求平均，计算得到用户 u 对物品 i 打分，公式如下：

$$P_{u,i} = \frac{\sum_{\text{all similar items, N}}(s_{i,N} * R_{u,N})}{\sum_{\text{all similar items, N}}(|s_{i,N}|)}$$

其中$s_{i,N}$为物品 i 与物品 N 的相似度，$R_{u,N}$为用户 u 对物品 N 的打分。

(b) 回归

和上面加权求和的方法类似，但回归的方法不直接使用相似物品 N 的打分值$R_{u,N}$，因为用余弦法或 Pearson 关联法计算相似度时存在一个误区，即两个打分向量可能相距比较远（欧氏距离），但有可能有很高的相似度。因为不同用户的打分习惯不同，有的偏向打高分，有的偏向打低分。如果两个用户都喜欢一样的物品，因为打分习惯不同，他们的欧式距离可能比较远，但他们应该有较高的相似度。在这种情况下用户原始的相似物品的打分值进行计算会造成糟糕的预测结果。通过用线性回归的方式重新估算一个新的$R_{u,N}$值，运用上面同样的方法进行预测。重新计算$R_{u,N}$的方法如下：

$$R'_N = \alpha \bar{R}_i + \beta + \epsilon$$

其中物品 N 是物品 i 的相似物品，α和β通过对物品 N 和 i 的打分向量进行线性回归计算得到，ϵ为回归模型的误差。具体怎么进行线性回归文章里面没有说明，需要查阅另外的相关文献。

4. 天籁项目-Collaborative Filtering

我们尝试通过 user-based 和 item-based 算法实现推荐系统。

因为这个算法没有参数，我们尝试了不同的相似度计算方法。

(1) 创建 user-item 的 matrix：

计算每一位用户对一个翻唱一共进行了多少分的活动。如下表：

用户 / 翻唱	1	2	3	4
1	34	53	77	49
2	48	12	169	29
3	79	74	25	19
4	105	44	36	81

(2) 创建 user-feature 的 matrix：

对每一个用户，对这个用户的 activity 按照 user-work 的特点进行分类。比如下表所示，用户 1 对女歌手翻唱的歌曲进行了 30 分的活动，对男歌手翻唱的歌曲进行了 35 分的活动，对摇滚类歌曲进行了 36 分的活动，对流行歌曲进行了 27 分的活动。这样用户 1 的 vector 是[30,35,36,27]。对其他的用户以此类推然后计算用户间的相似度。

用户	翻唱用户性别女	翻唱用户性别男	翻唱歌曲种类-摇滚	翻唱歌曲种类-流行
1	30	35	36	27
2	20	16	74	11
3	18	76	21	104
4	3	88	34	89

五、结论

根据所应用的算法，可得到如下结果：

算法名称	Mean Square Error	算法名称	Mean Square Error
XGBoost	2649	User-user CF	712
Random Forest	2494	Item-item CF	806
SVR	2671	User-user CF (user feature matrix)	858

(Lower is better)

根据表格中的结果，我们认为 Collaborative Filtering 比传统的机器学习算法在天籁项目推荐系统上的表现更优。

六、未来的工作

1. 完成特征选择，特征提取，并明确特征变量的数据类型：数值，范围，名词标签；

2. 完成缺失数据的处理；

3. 应用算法模型 GBDT (Gradient Boosting Decision Tree), Factorization Machine (隐因子子分解机)和 DNN (Deep Neural Network)。