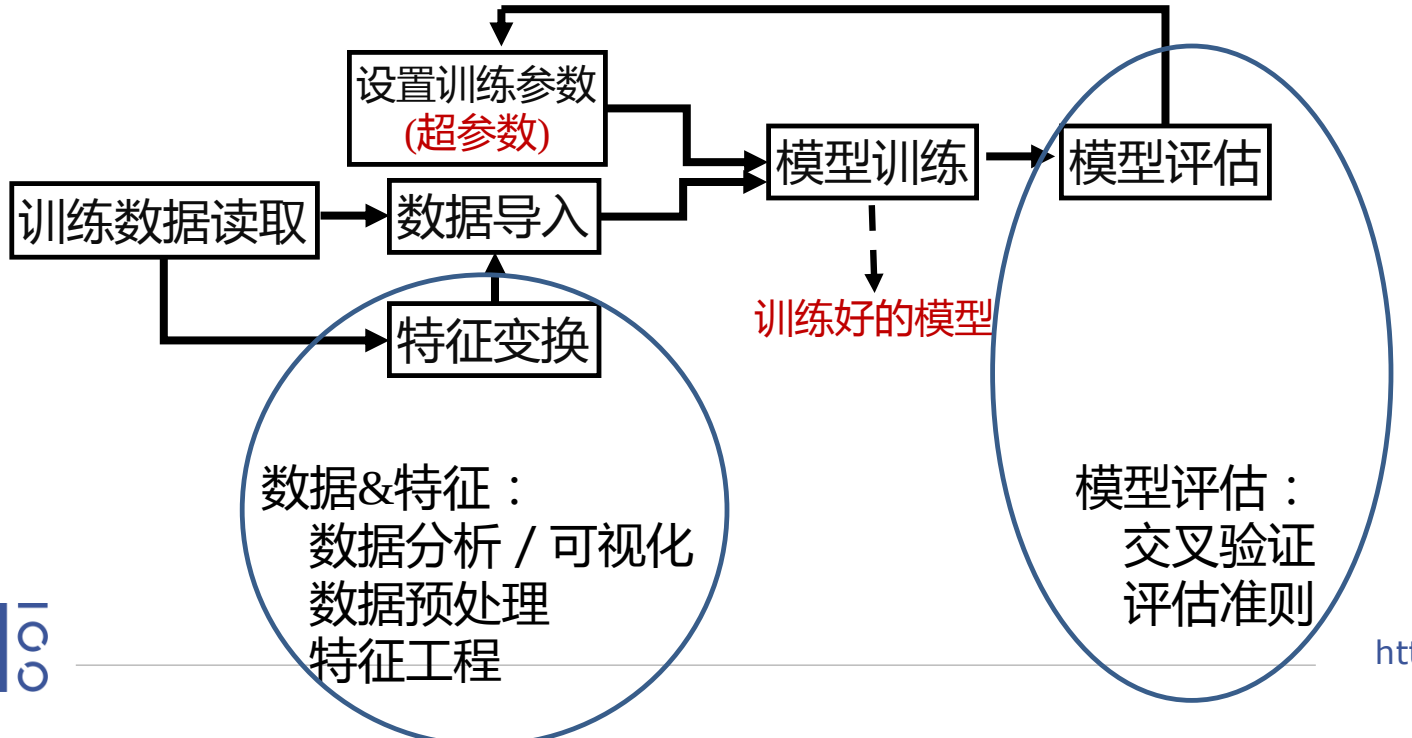
第四讲：XGBoost实战

AI100学院

2017年6月

Recall：处理数据科学任务的一般流程

- 模型训练



Roadmap

- 特征工程
- 评估准则
- XGBoost参数调优
- XGBoost多线程*

一、特征工程

数据分析

- 对数据进行探索性的分析的工具包：pandas、 matplotlib / seaborn

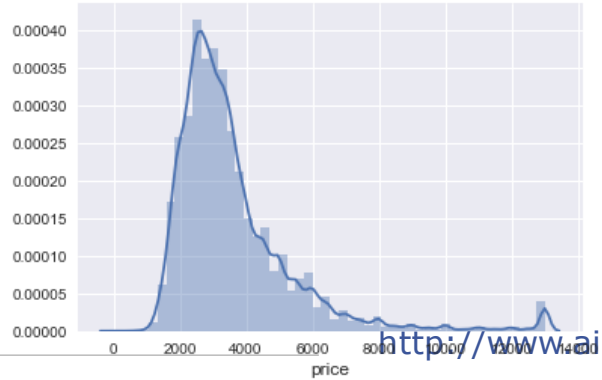
- 读取训练数据，取少量样本进行观测，并查看数据规模和数据类型
  - 标签、特征意义、特征类型等
- 分析每列特征的分布
  - 直方图
  - 包括标签列（对分类问题，可看出类别样本是否均衡）
  - 检测奇异点（outliers）
- 分析每两列特征之间的相关性
  - 特征与特征之间信息是否冗余
  - 特征与标签是否线性相关
- 特征工程

案例分析：
Two Sigma Connect: Rental Listing Inquiries

- 根据公寓的listing 内容，预测纽约市某公寓租赁listing的受欢迎程度
- 标签： interest_level，该listing被咨询的次数
  - 有三个取值：: 'high', 'medium', 'low'
  - 是一个多类分类任务
- Listing内容有：
  - 浴室和卧室的数目bathrooms， bedrooms
  - 地理位置（ longitude 、latitude ）
  - 地址： display_address、street_address
  - building_id、 listing_id、manager_id
  - Created：创建日期
  - Description：更多描述信息
  - features: 公寓的一些特征描述
  - photos: a list of photo links
  - 价格：price
  - https://www.kaggle.com/c/two-sigma-connect-rental-listing-inquiries/dat

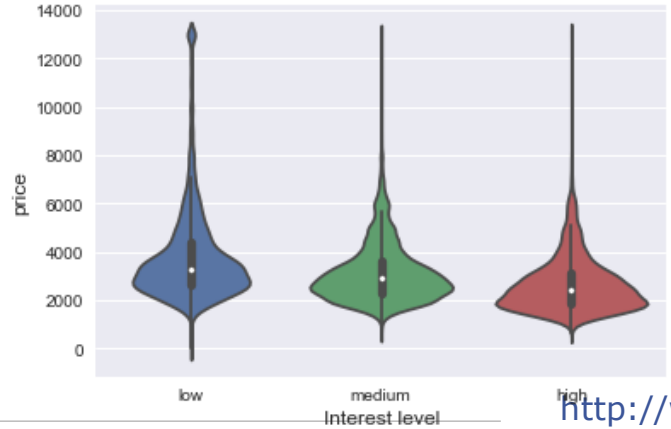
直方图

- 直方图：每个取值在数据集中出现的次数，可视为概率函数（PDF）的估计（seaborn可视化工具比较简单）
  - import seaborn as sns
  - %matplotlib inline（ seaborn 是基于matplotlib 的）
  - sns.distplot(train.price.values, bins=50, kde=True)
- 核密度估计
  - Kernel Density Estimation, KDE
  - 对直方图的加窗平滑



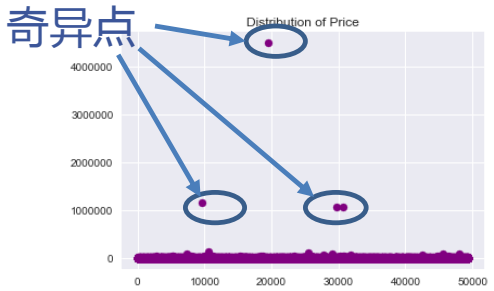
直方图（cont.）

- 在分类任务中，我们关心不同类别的特征分布
  - 核密度估计
  - order = ['low', 'medium', 'high']
  - sns.violinplot(x='interest_level', y='price', data=train, order = order)



奇异点

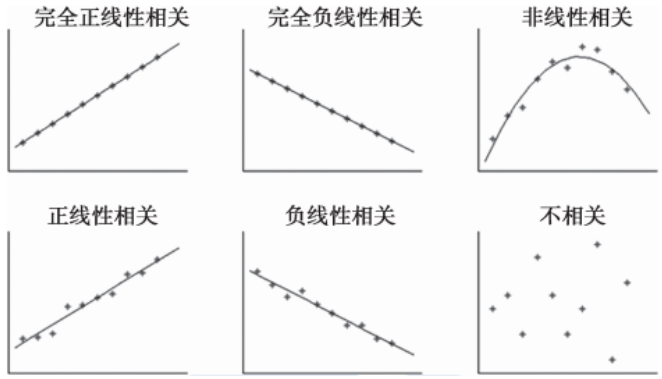
- 奇异点：或称离群点，指远离大多数样本的样本点。通常认为这些点是噪声，对模型有坏影响
- 可以通过直方图或散点图发现奇异点
  - 直方图的尾巴
  - 散点图上孤立的点



- 可以通过只保留某些分位数内的点去掉奇异点
  - 如0.5%-99.5%，或>99%
  - ulimit = np.percentile(train.price.values, 99)
  - train['price'].ix[train['price']>ulimit] = ulimit

相关性

- 相关性可以通过计算相关系数或打印散点图来发现
- 相关系数：两个随机变量x,y之间的线性相关程度
  - $r = \frac{\sum_{i=0}^{n}(x_i-\bar{x})(y_i-\bar{y})}{\sqrt{\sum_{i=0}^{n}(x_i-\bar{x})^2 \sum_{i=0}^{n}(y_i-\bar{y})^2}}$
  - $-1 \le r \le 1$
  - 通常$|r| > 0.5$，认为两者相关性比较强
  - $\begin{cases} r = 0 & \text{完全不线性相关} \\ r > 0 & \text{正相关} \\ r < 0 & \text{负相关} \end{cases}$
  - 不线性相关并不代表不相关，可能高阶相关，如$y = x^2$



相关性(cont.)

- 我们希望特征与标签强相关
  - 分类直方图可以从某种程度上看出特征与标签的相关性：不同类别的直方图差异大
- 特征与特征之间强相关的话意味着信息冗余
  - 可以两个特征可以只保留一个特征
  - 或采用主成分分析（PCA）等降维
- 示例：4_corr_AllstateClaimsSeverity.ipynb

数据类型

- XGBoost 模型内部将所有的问题都建模成一个回归预测问题，输入特征只能是数值型。
- 如果给定的数据是不同的类型，必须先将数据变成数值型。

类别型特征（ categorical attributes ）

- LabelEncoder： 对不连续的数字或者文本进行编号
  - 可用在对字符串型的标签编码（测试结果需进行反变换）
  - 编号默认有序数关系
  - 存储量小
- 如不希望有序数关系： OneHotEncoder：将类别型整数输入从1维→K维的稀疏编码
  - ☺：对XGBoost，OneHotEncoder不是必须，因为XGBoost对特征进行排序从而进行分裂建树；如果用OneHotEncoder得到稀疏编码，XGBoost建树过程中对稀疏特征处理速度块
  - 输入必须是数值型数据（对字符串输入，先调用LabelEncoder变成数字，再用OneHotEncoder ）
  - 存储要求高

类别型特征（cont.）

- 低基数（low-cardinality ）类别型特征： OneHotEncoder
  - 1维→*K*维， *K*为该特征不同的取值数目
  - 所以通常在*K* <10的情况下采用
- 高基数（high-cardinality）类别型特征：通常有成百上千个不同的取值，可先降维
  - 如邮政编码、街道名称…
  - 聚类（Clustering）： 1 维 → *K*维，*K*为聚类的类别数
  - 主成分分析（principle component analysis, PCA）：但对大矩阵操作费资源
  - 均值编码：在贝叶斯的架构下，利用标签变量，有监督地确定最适合特定特征的编码方式

Reference: Mean Encoding: A Preprocessing Scheme for High-Cardinality Categorical Features
http://helios.mm.di.uoa.gr/~rouvas/ssi/sigkdd/sigkdd.vol3.1/barreca.pdf

均值编码

- 均值编码：将特征每个可能的取值k(共有K种取值)，编码为它对应的标签取c值的概率，即$p(y=c|x=k)$
  - 有监督编码方式，可用于分类和回归
- 以分类问题为例，设标签取值共有C类，则编码特征为C-1维
  - $\sum_{c=1}^{C} p(y=c|x=k)=1$,所以少一个自由度
  - 后验概率$p(y=c|x=k)$估计：
  - $p(y=c|x=k)=\frac{p(y=c,x=k)}{p(x=k)}=\frac{(y=c,且x=k)的样本数量}{(x=k)的样本数量}$

均值编码（cont.）

- 真正编码采用的是先验概率$p(y=c)$和后验概率$p(y=c|x=$

参考代码：https://zhuanlan.zhihu.com/p/26308272

日期型特征

- 日期特征：年月日
- 时间特征：小时分秒
- 时间段：早中晚
- 星期，工作日 / 周末

```
import pandas as pd
train['created'] = pd.to_datetime(train['created'])
train['date'] = train['created'].dt.date
train["year"] = train["created"].dt.year
train['month'] = train['created'].dt.month
train['day'] = train['created'].dt.day
train['hour'] = train['created'].dt.hour
train['weekday'] = train['created'].dt.weekday
train['week'] = train['created'].dt.week
train['quarter'] = train['created'].dt.quarter
train['weekend'] = ((train['weekday'] == 5) & (train['weekday'] == 6))train['wd'] = ((train['weekday'] != 5) & (train['weekday'] != 6))
```

文本型特征

- 可用词云（wordcloud）可视化
  - 文本词频统计函数，自动统计词的个数，以字典形式内部存储，在显示的时候词频大的词的字体更大
  - from wordcloud import WordCloud
  - wordcloud = WordCloud(background_color='white', width=600, height=300, max_font_size=50, max_words=40)
  - wordcloud.generate(text)

Rent List Inquries任务中的features特征

特征工程小结

- 如果知道数据的物理意义（领域专家），可能可以设计更多特征
  - 如Higgs Boson任务中有几维特征是物理学家设计的，还有些有高能物理研究经验的竞赛者设计了其他一些特征
  - 如房屋租赁任务中，利用常识可设计出一些特征
- 如果不是领域专家，一些通用的规则：
  - 字符串型特征：Label编码
  - 时间特征：年月日、时间段（早中晚）…
  - 数值型特征：加减乘除，多项式，log, exp
  - 低基数类别特征：one-hot编码
  - 高基数类别特征：先降维，再one-hot编码；均值编码

特征工程小结（cont.）

- 非结构化特征
  - 文本
  - 语音
  - 图像 / 视频
  - fMRI
  - …
- 利用领域知识设计特征
  - 如曾经流行的图像目标检测特征HOG…
- 利用深度学习从数据中学习特征表示
  - 采用end-to-end方式一起学习特征和分类 / 回归 / 排序
  - 学习好特征可以送入XGBoost学习器

数据预处理

- from sklearn.preprocessing import …
  - 数据标准化
  - 数据归一化
  - 数据二值化
  - 数据缺失
  - ☺：XGBoost对数据预处理要求少，以上操作都不是必须

信息泄漏

- 信息泄漏：训练数据特征不应该包含标签的信息
  - 如Rent Listing Inquries任务中图片压缩文件里文件夹的创造时间：加入这个特征后，模型普遍能提高0.01的public LB分数

案例分析：
Two Sigma Connect: Rental Listing Inquiries

- 案例代码： 4_FE_RentListingInqueries.ipynb

二、评价指标

回归问题的评价指标

- 第二讲提到的损失函数可以作为评价指标
- L1: mean absolute error (MAE)
  - $MAE = \frac{1}{N}\sum_{i=0}^{N}|\hat{y}_i - y_i|$
- L2：Root Mean Squared Error (RMSE)
  - $RMSE = \sqrt{\frac{1}{N}\sum_{i=0}^{N}(\hat{y}_i - y_i)^2}$
- Root Mean Squared Logarithmic Error（RMSLE）
  - https://www.kaggle.com/wiki/RootMeanSquaredLogarithmicError
  - $RMSLE = \sqrt{\frac{1}{N}\sum_{i=0}^{N}\left(log(\hat{y}_i + 1) - log(y_i + 1)\right)^2}$
  - 当不想在预测值 / 真值很大时惩罚很大时采用RMSLE

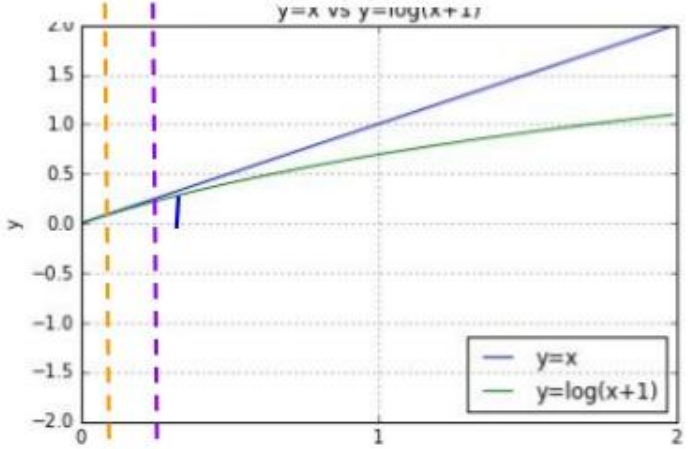
Cost Functions

Root Mean Squared Error (RMSE)

$$\sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

Root Mean Squared Log Error (RMSLE)

$$\sqrt{\frac{1}{n}\sum_{i=1}^{n}(\log(p_i + 1) - \log(a_i + 1))^2}$$



When predicted and actual is small:

For the same predicted & actual, RMSE & RMSLE is same (the blue vertical line)

分类任务的评价指标

- 第二讲提到的损失函数可以作为评价指标
- logistic / 负log似然损失：
  - $\text{logloss} = -\frac{1}{N}\sum_{i=0}^{N}\sum_{j=0}^{M} y_{ij} log p_{ij}$,
    - M为类别数，y_{ij}为二值，当第i个样本为第j类时$y_{ij} = 1$，否则取0；p_{ij}为模型预测的第i个样本为第j类的概率
    - 当M=2时，$\text{logloss} = -\frac{1}{N}\sum_{i=0}^{N}\left(y_i log p_i + (1-y_i) log(1-p_i)\right)$
      - y_i为第i个样本类别，p_i为模型预测的第i个样本为第1类的概率

- 0-1损失对应的Mean Consequential Error (MCE)
  - $\text{MCE} = -\frac{1}{N}\sum_{\hat{y}_i \neq y_i} 1$

两类分类任务中更多评价指标

- ROC / AUC
- PR曲线
- MAP@n

False Positive & False Negative

- 0-1损失：假设两种错误的代价相等
  - False Positive（FP）& False Negative（FN）
- 有些任务中可能某一类错误的代价更大
  - 如蘑菇分类中将毒蘑菇误分为可食用代价更大
  - 因此单独列出每种错误的比例：混淆矩阵
- 混淆矩阵（confusion matrix）
  - 真正的正值（true positives）
  - 假的正值（false positives）
  - 真正的负值（true negatives）
  - 假的负值（false negatives ）

	$\hat{y}=1$	$\hat{y}=0$	Σ
$y=1$	#TP	#FN	N_+
$y=0$	#FP	#TN	N_-
Σ	$\hat{N}_+$	$\hat{N}_-$	

confusion matrix

- Scikit-learn实现了多类分类任务的混淆矩阵
- sklearn.metrics.confusion_matrix(*y_true*, *y_pred*, *labels=None*, *sample_weight=None*)
  - *y_true*：*N*个样本的标签真值
  - *y_pred*：*N*个样本的预测标签值
  - *labels*：*C*个类别在矩阵的索引顺序，缺省为*y_true*或*y_pred*类别出现的顺序
  - *sample_weight*：*N*个样本的权重

Receiver Operating Characteristic (ROC)

	$\hat{y}=1$	$\hat{y}=0$	Σ
$y=1$	#TP	#FN	N_+
$y=0$	#FP	#FN	N_-
Σ	$\hat{N}_+$	$\hat{N}_-$	

$$\text{accuracy} = \frac{TP + TN}{N}$$

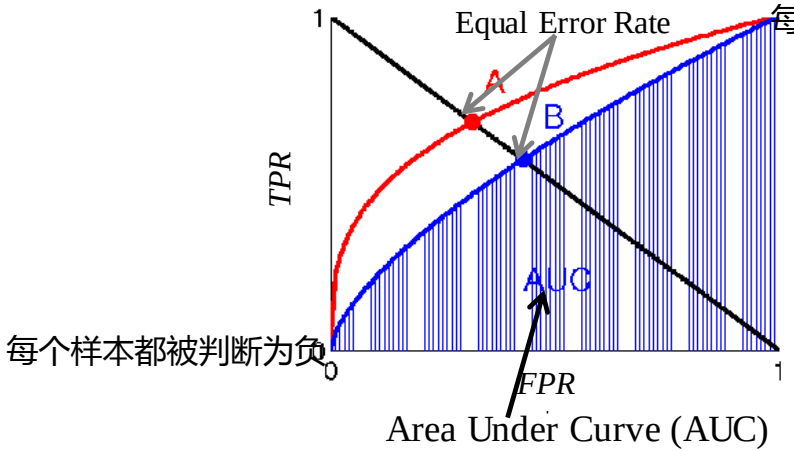
$$\text{error rate} = \frac{FP + TN}{N}$$

	$y=1$	$y=0$	$y=1$	$y=0$
$\hat{y}=1$	$TP/\hat{N}_+$=precision	$FP/\hat{N}_+$=FDP	TP/N_+=TPR	FP/N_-=FPR
$\hat{y}=0$	$FN/\hat{N}_-$	$TN/\hat{N}_-$=NPV	FN/N_+=FNR	TN/N_-=TNR

PPV - positive predictive value, **precision**
TPR - true positive rate, **sensitivity, recall**, hit rate
FPR – False positive rate, **false alarm**, fallout

Receiver Operating Characteristic (ROC)

- 上面我们讨论给定阈值τ的TPR和FPR
- 如果不是只考虑一个阈值，而是在一些列阈值上运行检测器，并画出TPR和FPR为阈值τ的隐式函数，得到ROC曲线在此处键入公式。



$$TPR = \frac{TP}{N_+}$$

$$FPR = \frac{FP}{N_-}$$

PR曲线

- Precision and Recall (PR曲线)：用于稀有事件检测，如目标检测、信息检索
  - 负样本非常多，因此$FPR = FP/N_$很小，比较TPR和FPR不是很有信息（ROC曲线中只有左边很小一部分有意义）→ 只讨论正样本
  - Precision（精度，查准率，正确 率）：以信息检索为例，对于一个查询，返回了一系列的文档，正确率指的是返回结果中相关文档占的比例
    - precision=返回结果中相关文档的数目/返回结果的数目
  - Recall（召回率，查全率）：返回结果中相关文档占所有相关文档的比例
    - Recall=返回结果中相关文档的数目/所有相关文档的数目

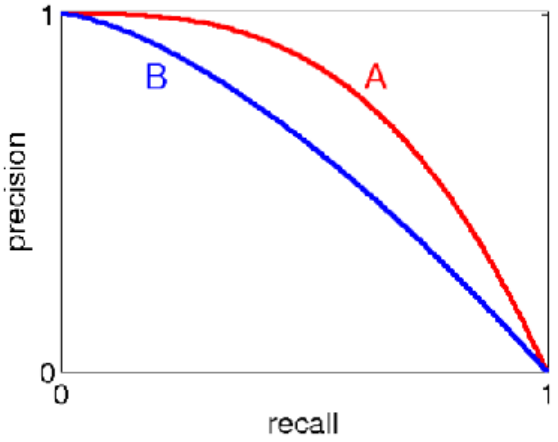
PR曲线(cont.)

- Precision and Recall (PR曲线)
  - 阈值变化时的P和R

Precision: the fraction of our detection are actually positive
Recall: the fraction of the positives we actually detected

$\text{Precsion} = TP/\hat{N}_+$:检测结果真正为正的比例

$\text{Recall} = TP/N_+$:被正确检测到的正样本的比例



AP

- Precision只考虑了返回结果中相关文档的个数，没有考虑文档之间的序。
- 对一个搜索引擎或推荐系统而言，返回的结果必然是有序的，而且越相关的文档排的越靠前越好，于是有了AP的概念。
- AP: Average Precision，对不同召回率点上的正确率进行平均
  - $AP = \int_0^1 p(k)dr = \sum_{k=0}^{n} p(k)\Delta r(k)$
  - 即PR曲线下的面积（Recall： AUC为ROC下的面积）
  - 其中k为返回文档中的序位，n为返回文档的数目，$p(k)$为列表中k截止点的precision， $\Delta r(k)$表示从k-1到k Recall的变化。
- 上述离散求和表示等价于：$AP = \sum_{k=0}^{n} p(k)rel(k)/$相关文档数目，其中$rel(k)$为示性函数，即第$k$个位置为相关文档取1，否则取0.
  - 计算每个位置上的precision，如果该位置的文档是不相关的则该位置 precision=0
  - 然后对所有的位置的precision再做average

Mean Average Precision

- 多个查询的AP平均：
  - $MAP = \left(\sum_{q=0}^{Q} AP(q)\right)/(Q)$，
  - 其中Q为查询的数目，n为文档数目

MAP@K（MAP_K）

- 在现代web信息检索中，recall其实已经没有意义，因为相关文档有成千上万个，很少有人会关心所有文档。
- Precision@K：在第K个位置上的Precision，
  - 对于搜索引擎，考虑到大部分作者只关注前一、两页的结果，所以Precision @10， Precision @20对大规模搜索引擎非常有效

- MAP@K：多个查询Precision@K的平均

F1 分数

- 亦被称为F1 score, balanced F-score or F-measure
- Precision 和 Recall 加权平均：$F1 = \frac{2*(\text{Precision}*\text{Recall})}{(\text{Precision}+\text{Recall})}$
  - 最好为1，最差为0
  - 多类：每类的F1平均值

Scikit-Learn: Scoring

- 用交叉验证（cross_val_score和GridSearchCV）评价模型性能时，用scoring参数定义评价指标。
- 评价指标是越高越好，因此用一些损失函数当评价指标时，需要再加负号，如neg_log_loss，neg_mean_squared_error
- 详见sklearn文档：
  - http://scikit-learn.org/stable/modules/model_evaluation.html#log-loss

Scoring	Function	Comment
Classification		
‘accuracy’	metrics.accuracy_score	
‘average_precision’	metrics.average_precision_score	
‘f1’	metrics.f1_score	for binary targets
‘f1_micro’	metrics.f1_score	micro-averaged
‘f1_macro’	metrics.f1_score	macro-averaged
‘f1_weighted’	metrics.f1_score	weighted average
‘f1_samples’	metrics.f1_score	by multilabel sample
‘neg_log_loss’	metrics.log_loss	requires predict_proba support
‘precision’ etc.	metrics.precision_score	suffixes apply as with ‘f1’
‘recall’ etc.	metrics.recall_score	suffixes apply as with ‘f1’
‘roc_auc’	metrics.roc_auc_score	
Clustering		
‘adjusted_rand_score’	metrics.adjusted_rand_score	
Regression		
‘neg_mean_absolute_error’	metrics.mean_absolute_error	
‘neg_mean_squared_error’	metrics.mean_squared_error	
‘neg_median_absolute_error’	metrics.median_absolute_error	
‘r2’	metrics.r2_score	

Scikit-Learn：sklearn.metrics

- metrics模块还提供为其他目的而实现的预测误差评估函数
  - 分类任务的评估函数如表所示，其他任务评估函数请见：http://scikit-learn.org/stable/modules/classes.html#module-sklearn.metrics

metrics.accuracy_score(y_true, y_pred[, ...])	Accuracy classification score.
metrics.auc(x, y[, reorder])	Compute Area Under the Curve (AUC) using the trapezoidal rule
metrics.average_precision_score(y_true, y_score)	Compute average precision (AP) from prediction scores
metrics.brier_score_loss(y_true, y_prob[, ...])	Compute the Brier score.
metrics.classification_report(y_true, y_pred)	Build a text report showing the main classification metrics
metrics.cohen_kappa_score(y1, y2[, labels, ...])	Cohen's kappa: a statistic that measures inter-annotator agreement.
metrics.confusion_matrix(y_true, y_pred[, ...])	Compute confusion matrix to evaluate the accuracy of a classification
metrics.f1_score(y_true, y_pred[, labels, ...])	Compute the F1 score, also known as balanced F-score or F-measure
metrics.fbeta_score(y_true, y_pred, beta[, ...])	Compute the F-beta score
metrics.hamming_loss(y_true, y_pred[, ...])	Compute the average Hamming loss.
metrics.hinge_loss(y_true, pred_decision[, ...])	Average hinge loss (non-regularized)
metrics.jaccard_similarity_score(y_true, y_pred)	Jaccard similarity coefficient score
metrics.log_loss(y_true, y_pred[, eps, ...])	Log loss, aka logistic loss or cross-entropy loss.
metrics.matthews_corrcoef(y_true, y_pred[, ...])	Compute the Matthews correlation coefficient (MCC) for binary classes
metrics.precision_recall_curve(y_true, ...)	Compute precision-recall pairs for different probability thresholds
metrics.precision_recall_fscore_support(...)	Compute precision, recall, F-measure and support for each class
metrics.precision_score(y_true, y_pred[, ...])	Compute the precision
metrics.recall_score(y_true, y_pred[, ...])	Compute the recall
metrics.roc_auc_score(y_true, y_score[, ...])	Compute Area Under the Curve (AUC) from prediction scores
metrics.roc_curve(y_true, y_score[, ...])	Compute Receiver operating characteristic (ROC)
metrics.zero_one_loss(y_true, y_pred[, ...])	Zero-one classification loss.

XGBoost支持的目标函数

- Objective： 定义学习任务及相应的学习目标，可选的目标函数如下：
  - “reg:linear” –线性回归。
  - “reg:logistic” –逻辑回归。
  - “binary:logistic” –二分类的逻辑回归问题，输出为概率。
  - “binary:logitraw” –二分类的逻辑回归问题，输出的结果为$\mathbf{w}^T\mathbf{x}$。
  - “count:poisson”–计数问题的poisson回归，输出结果为poisson分布。
  - “multi:softmax” –让XGBoost采用softmax目标函数处理多分类问题
  - “multi:softprob” –和softmax一样，但是输出的是ndata * nclass的向量，可以将该向量reshape成ndata行nclass列的矩阵。没行数据表示样本所属于每个类别的概率。
  - “rank:pairwise” –set XGBoost to do ranking task by minimizing the pairwise loss

XGBoost自定义目标函数

- 在GBDT训练过程，当每步训练得到一棵树，要调用目标函数得到其梯度作为下一棵树拟合的目标
- XGBoost在调用obj函数时会传入两个参数：preds和dtrain
  - preds为当前模型完成训练时，所有训练数据的预测值
  - dtrain为训练集，可以通过dtrain.get_label()获取训练样本的label
  - 同时XGBoost规定目标函数需返回当前preds基于训练label的一阶和二阶梯度

- # user define objective function, given prediction, return gradient and second order gradient
- # this is log likelihood loss
- def logregobj(preds, dtrain): #自定义损失函数
  - labels = dtrain.get_label()
  - preds = 1.0 / (1.0 + np.exp(-preds))
  - grad = preds - labels #梯度
  - hess = preds * (1.0-preds) #2阶导数
  - return grad, hess

- 参数：obj= 'logregobj'

https://github.com/dmlc/xgboost/blob/master/demo/guide-python/custom_objective.py

XGBoost支持的评价函数

- **'eval_metric' The choices are listed below，评估指标:**
  - "rmse": root mean square error
  - "logloss": negative log-likelihood
  - "error": Binary classification error rate. It is calculated as #(wrong cases)/#(all cases). For the predictions, the evaluation will regard the instances with prediction value larger than 0.5 as positive instances, and the others as negative instances.
  - "merror": Multiclass classification error rate. It is calculated as #(wrong cases)/#(all cases).
  - "mlogloss": Multiclass logloss
  - "auc": Area under the curve for ranking evaluation.
  - "ndcg":Normalized Discounted Cumulative Gain
  - "map":Mean average precision
  - "ndcg@n","map@n": n can be assigned as an integer to cut off the top positions in the lists for evaluation.
  - "ndcg-","map-","ndcg@n-","map@n-": In XGBoost, NDCG and MAP will evaluate the score of a list without any positive samples as 1. By adding "-" in the evaluation metric XGBoost will evaluate these score as 0 to be consistent under some conditions.

XGBoost自定义评价函数

- # user defined evaluation function, return a pair metric_name, result
- # NOTE: when you do customized loss function, the default prediction value is margin
- # this may make builtin evaluation metric not function properly
- # for example, we are doing logistic loss, the prediction is score before logistic transformation
- # the builtin evaluation error assumes input is after logistic transformation
- # Take this in mind when you use the customization, and maybe you need write customized evaluation function
- def f1error(preds, dtrain):
  - labels = dtrain.get_label()
  - preds = 1.0 / (1.0 + np.exp(-preds))
  - pred = [int (i>0.5) for i in preds]

- 参数： feval =‘evalerror’

http://blog.csdn.net/lujiandong1/article/details/52791117

三、参数调优

XGBoost参数列表

参数	说明
max_depth	树的最大深度。树越深通常模型越复杂，更容易过拟合
learning_rate	学习率或收缩因子。学习率和迭代次数 / 弱分类器数目n_estimators相关。 缺省：0.1
n_estimators	弱分类器数目. 缺省:100
slient	参数值为1时，静默模式开启，不输出任何信息
objective	待优化的目标函数，常用值有： binary:logistic 二分类的逻辑回归，返回预测的概率 multi:softmax 使用softmax的多分类器，返回预测的类别(不是概率)。 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。支持用户自定义目标函数
nthread	用来进行多线程控制。 如果你希望使用CPU全部的核，那就不用缺省值-1，算法会自动检测它。
booster	选择每次迭代的模型，有两种选择： gbtree：基于树的模型，为缺省值。 gbliner：线性模型
gamma	节点分裂所需的最小损失函数下降值
min_child_weight	叶子结点需要的最小样本权重（hessian）和
max_delta_step	允许的树的最大权重

).ai/

XGBoost参数列表

参数	说明
subsample	构造每棵树的所用样本比例（样本采样比例），同GBM
colsample_bytree	构造每棵树的所用特征比例
colsample_bylevel	树在每层每个分裂的所用特征比例
reg_alpha	L1/L0正则的惩罚系数
reg_lambda	L2正则的惩罚系数
scale_pos_weight	正负样本的平衡
base_score	每个样本的初始估计，全局偏差
random_state	随机种子
seed	随机种子
missing	当数据缺失时的填补值。缺省为np.nan
kwargs	XGBoost Booster的Keyword参数

参数类别

- 通用参数：这部分参数通常我们不需要调整，默认值就好
- 学习目标参数：与任务有关，定下来后通常也不需要调整
- booster参数：弱学习器相关参数，需要仔细调整，会影响模型性能

通用参数

- booster：弱学习器类型
  - 可选gbtree（树模型）或gbliner（线性模型）
  - 默认为gbtree（树模型为非线性模型，能处理更复杂的任务）
- silent：是否开启静默模式
  - 1：静默模式开启，不输出任何信息
  - 默认值为0：输出一些中间信息，以助于我们了解模型的状态
- nthread：线程数
  - 默认值为-1，表示使用系统所有CPU核

学习目标参数

- objective: 损失函数
  - 支持分类 / 回归 / 排序
- eval_metric：评价函数
- seed：随机数的种子
  - 默认为0
  - 设置seed可复现随机数据的结果，也可以用于调整参数

booster参数

- 弱学习器的参数，尽管有两种booster可供选择，这里只介绍gbtree
- 1. learning_rate：收缩步长 vs. n_estimators：树的数目
  - 较小的学习率通常意味着更多弱分学习器
  - 通常建议学习率较小（ $\eta < 0.1$ ），弱学习器数目n_estimators大

$$f_m(\mathbf{x}_i) = f_{m-1}(\mathbf{x}_i) + \eta\beta_m\phi_m(\mathbf{x}_i)$$

  - 可以设置较小的学习率，然后用交叉验证确定n_estimators

booster参数

- 2. 行（*subsample*）列（*colsample_bytree*、*colsample_bylevel*）下采样比例
  - 默认值均为1，即不进行下采样，使用所有数据
  - 随机下采样通常比用全部数据的确定性过程效果更好，速度更快
  - 建议值：0.3 - 0.8

booster参数

- 3. 树的最大深度： max_depth
  - max_depth越大，模型越复杂，会学到更具体更局部的样本
  - 需要使用交叉验证进行调优，默认值为6，建议3-10

- 4. min_child_weight ：孩子节点中最小的样本权重和
  - 如果一个叶子节点的样本权重和小于min_child_weight则分裂过程结束

Kaggle竞赛优胜者的建议

- Tong He（XGBoost R语言版本开发者）： 三个最重要的参数为：树的**数目**、树的**深度**和**学习率**。建议参数调整策略为：
  - 采用默认参数配置试试
  - 如果系统过拟合了，降低学习率
  - 如果系统欠拟合，加大学习率

Kaggle竞赛优胜者的建议（cont.）

- Owen Zhang （常使用XGBoost）建议：
  - n_estimators和learning_rate：固定n_estimators为100（数目不大，因为树的深度较大，每棵树比较复杂），然后调整learning_rate
  - 树的深度max_depth：从6开始，然后逐步加大
  - min_child_weight ： $1/sqrt{\text{rare_events}}$，其中rare_events 为稀有事件的数目
  - 列采样colsample_bytree / colsample_bylevel：在[0.3, 0.5]之间进行网格搜索
  - 行采样subsample：固定为1
  - gamma: 固定为0.0

Kaggle竞赛案例分析：
Otto Group Product Classification Challenge

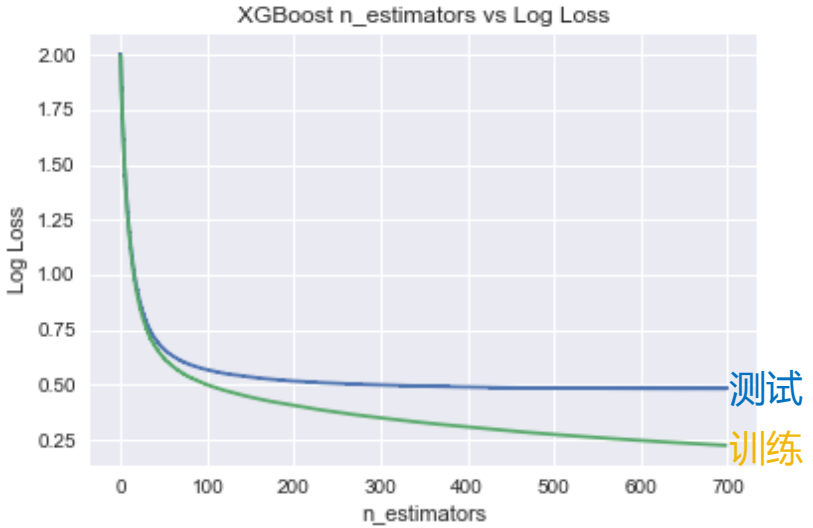
- 竞赛官网：https://www.kaggle.com/c/otto-group-product-classification-challenge
- 电商商品分类：
  - Target：共9个商品类别
  - 93个特征：整数型特征

参数调优的一般方法

- 1. 选择较高的学习率(learning rate)，并选择对应于此学习率的理想的树数量
  - 学习率以工具包默认值为0.1。
  - XGBoost直接引用函数“cv”可以在每一次迭代中使用交叉验证，并返回理想的树数量（因为交叉验证很慢，所以可以import两种XGBoost：直接引用xgboost（用“cv”函数调整树的数目）和XGBClassifier —xgboost的sklearn包（用GridSearchCV调整其他参数 ）。
- 2. 对于给定的学习率和树数量，进行树参数调优(max_depth, min_child_weight, gamma, subsample, colsample_bytree, colsample_bylevel)
- 3. xgboost的正则化参数(lambda, alpha)的调优
- 4. 降低学习率，确定理想参数

可参考：http://blog.csdn.net/u010657489/article/details/51952785

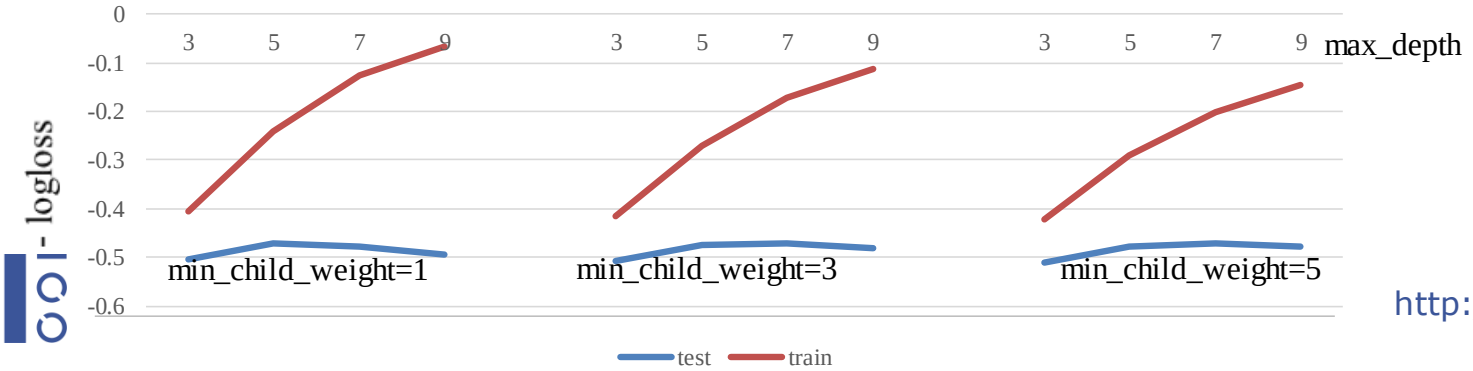
- 1. 采用缺省参数,此时learning_rate =0.1（较大）， 观察n_estimators的合适范围
  - 参数设为1000，earlystop = 50
  - cv函数在n_estimators =699时停止
  - cv测试误差为0.482744




```
max_depth=5,
min_child_weight=1,
gamma=0,
subsample=0.3,
colsample_bytree=0.8,
colsample_bylevel=0.7,
```

- 2.1 max_depth 和 min_child_weight 参数调整
  - 这两个参数对结果影响很大
  - 我们先大范围地粗调参数（步长为2），再小范围微调
  - max_depth = range(3,10,2)
  - min_child_weight = range(1,6,2)

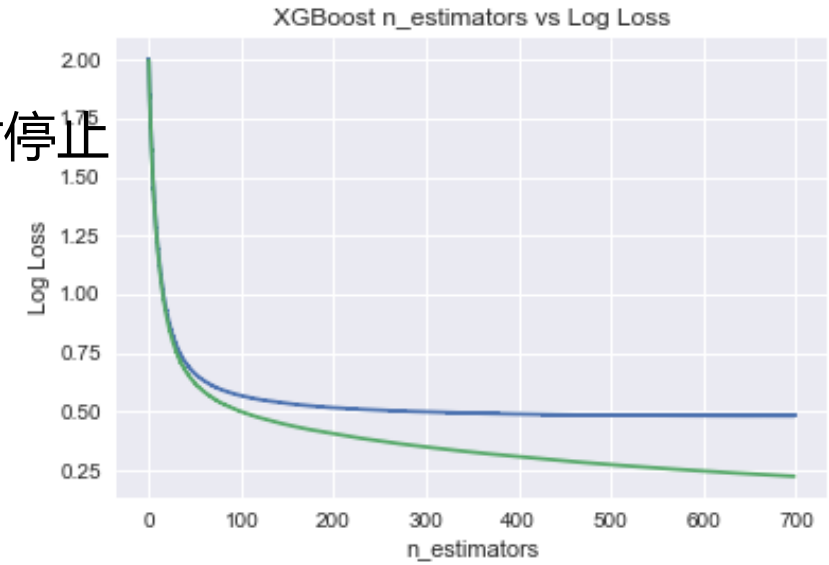
最小交叉验证测试误差：-0.471956
max_depth: 7
min_child_weight: 5



- 2.2 max_depth 和 min_child_weight 参数微调
  - 在max_depth=7和min_child_weight=5周围微调
  - max_depth = [6,7,8]
  - min_child_weight = [4,5,6]

  - 最小交叉验证测试误差： -0.471302
  - max_depth: 6
  - min_child_weight: 4

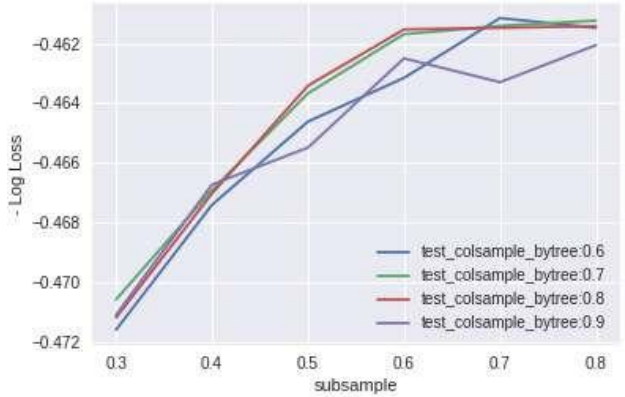
- 2.3 调整max_depth=6 和 min_child_weight=4后，再次调整n_estimators
  - 参数设为1000，earlystop = 50
  - cv函数在n_estimators =645时停止



- 3. gamma参数调整
  - 通常缺省值（0）表现还不错，如计算资源允许，可以调整（略）

- 4. 行列采样参数： subsample和colsample_bytree
  - 这两个参数可分别对样本和特征进行采样，从而增加模型的鲁棒性
  - 现在[0.3-1.0]之间调整，粗调时步长0.1
  - 微调时步长为0.05（略）

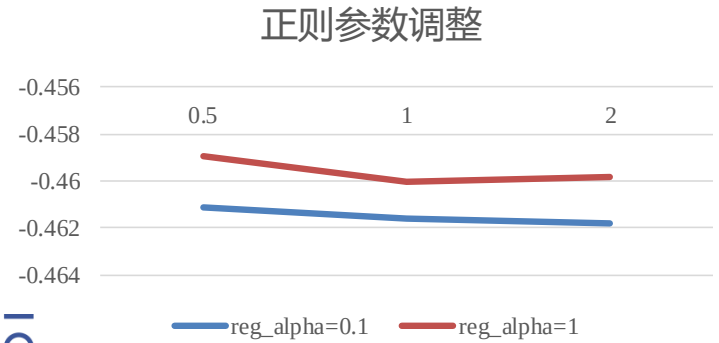
Best: -0.461137
subsample: 0.7,
colsample_bytree: 0.6



  - 注： colsample_bylevel通常0.7-0.8可以得到较好的结果。如计算资源允许，也可以进一步调整

$$\Omega(\phi(\mathbf{x})) = \gamma T + \frac{1}{2}\lambda \sum_{t=1}^{T} w_t^2$$

- 5. 正则参数调整： reg_alpha（L2）和reg_lambda(L0)
  - reg_alpha = [0.1, 1] # L1, default = 0
  - reg_lambda = [0.5, 1, 2] # L2, default = 1
  - Best: -0.458950 using {'reg_alpha': 1, 'reg_lambda': 0.5)



reg_lambda似乎越小越好
reg_alpha似乎越大越好

如计算资源允许，可进一步增大reg_alpha，减小reg_lambda

- 6. 降低学习率，调整树的数目
  - 调用xgboost的cv函数
    - 0.1:　669棵树收敛，cv测试误差为0.469456
    - 0.01：2000棵树还没有收敛，cv测试误差为0.501644
    - 0.05:　1386棵树收敛，cv测试误差为0.465813

```
xgb6 = XGBClassifier( learning_rate =0.05,
                      n_estimators=2000,
                      max_depth=6,
                      min_child_weight=4,
                      subsample=0.7,
                      colsample_bytree=0.6,
                      colsample_bylevel=0.7,
                      reg_alpha = 1,
                      reg_lambda = 0.5,
                      objective= 'multi:softprob',       seed=3)
```

四、XGBoost并行处理

XGBoost速度秘籍

- XGBoost用C++实现，显示地采用OpenMP API做并行处理
  - 建单棵树时并行（Random Forest在建不同树时并行，但Boosting增加树是一个串行操作）
- 在准备建树数据时高效（近似建树、稀疏、Cache、数据分块）
- 交叉验证也支持并行（由scikit-learn 提供支持）

并行处理——建树

- XGBoost用C++实现，显示地采用OpenMP API做并行处理
  - 建单棵树时并行
- XGBoost的scikit-learn接口中的参数nthread 可指定线程数
  - -1 表示使用系统所有的核资源
  - model = XGBClassifier(nthread=-1)

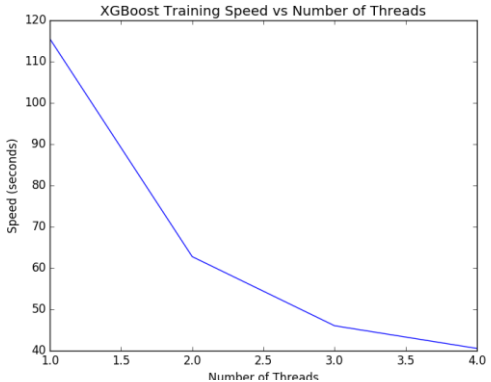
并行处理——建树（cont.）

- 确认系统支持多线程：建立几个不同线性的模型，观察不同模型的速度是否有差异

```
# evaluate the effect of the number of threads
results = []
num_threads = [1, 2, 3, 4]
for n in num_threads:
    start = time.time()
    model = XGBClassifier(nthread=n)
    model.fit(X_train, y_train)
    elapsed = time.time() - start
    print(n, elapsed)
results.append(elapsed)
```

Kaggle竞赛的Otto数据集上的结果：

(1, 115.51652717590332)
(2, 62.7727689743042)
(3, 46.042901039123535)
(4, 40.55334496498108)



并行处理——交叉验证

- scikit-learn 支持的*k*折交叉验证也支持多线程
  - cross_val_score() 函数中的参数：n_ jobs
    - -1 表示使用系统所有的CPU核
  - results = cross_val_score(model, X, label_encoded_y, cv=kfold, scoring= 'neg_log_loss' , n_jobs=-1, verbose=1)

并行处理——建树 vs. 交叉验证

- 并行处理的三种配置
  - 交叉验证并行，XGBoost建树不并行
  - 交叉验证不并行，XGBoost建树并行
  - 交叉验证并行，XGBoost建树并行

- Otto数据集上的10折交叉验证实验结果：
  - Single Thread XGBoost, Parallel Thread CV: 359.854589
  - Parallel Thread XGBoost, Single Thread CV: 330.498101
  - Parallel Thread XGBoost and CV: 313.382301

并行XGBoost比并行交叉验证好，两者都并行更好

四、结束语

XGBoost总结

- XGBoost是一个用于监督学习的非参数模型
  - 目标函数（损失函数、正则项）
  - 参数（树的每个分支分裂特征及阈值）
  - 优化：梯度下降

- 参数优化
  - 决定模型复杂度的重要参数：learning_rate, n_estimators, max_depth, min_child_weight, gamma, reg_alpha, reg_lamba
  - 随机采样参数也影响模型的推广性： subsample, colsample_bytree, colsample_bylevel

其他未涉及的部分

- 分布式XGBoost
  - AWS
  - YARN Cluster
  - …
- GPU加速
- 并行计算与内存优化的细节
  - 课堂内容主要关注XGBoost的对外接口

XGBoost资源

- XGBoost官方文档：https://xgboost.readthedocs.io/en/latest/
  - Python API：http://xgboost.readthedocs.io/en/latest/python/python_api.html
- Github： https://github.com/dmlc/xgboost
  - 很多有用的资源：https://github.com/dmlc/xgboost/blob/master/demo/README.md
  - GPU加速：https://github.com/dmlc/xgboost/blob/master/plugin/updater_gpu/README.md
- XGBoost原理：XGBoost: A Scalable Tree Boosting System
  - https://arxiv.org/abs/1603.02754

其他资源

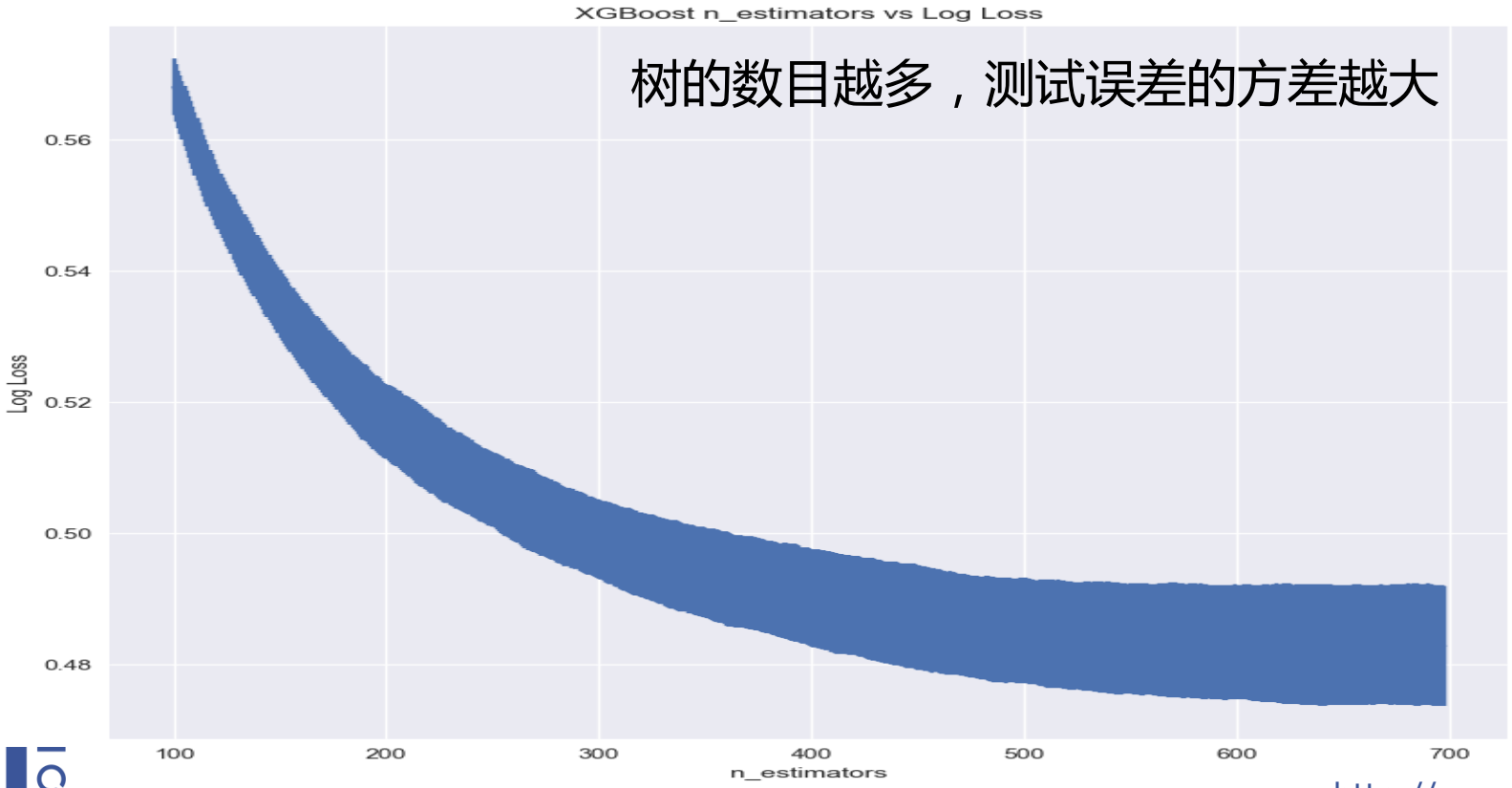
- XGBoost参数调优：
  - https://www.analyticsvidhya.com/blog/2016/03/complete-guide-parameter-tuning-xgboost-with-codes-python/
  - 中文版：http://blog.csdn.net/u010657489/article/details/51952785
- Owen Zhang, Winning Data Science Competitions
  - https://www.slideshare.net/OwenZhang2/tips-for-data-science-competitions?from_action=save
- XGBoost User Group：
  - https://groups.google.com/forum/#!forum/xgboost-user/

THANK YOU

北京智百科技有限公司

实战

- 建议代码风格
  - 特征处理为一个文件，每维特征处理单独定义一个函数
    - 训练数据和测试数据处理尽量分开
    - 如果自己编写特征处理函数，尽量参照sklearn中Encoder的风格：初始化、用训练数据进行fit，对训练数据和测试数据进行transform
  - 学习器参数调整 / 训练为一个文件
  - 系统测试一个文件
- 学习优胜者的经验，体会对特征的细致处理
  - Rent Listing Inquires任务中对如features特征


XGBoost n_estimators vs Log Loss
树的数目越多，测试误差的方差越大
Log Loss
0.56
0.54
0.52
0.50
0.48
100
200
300
400
500
600
700
n_estimators