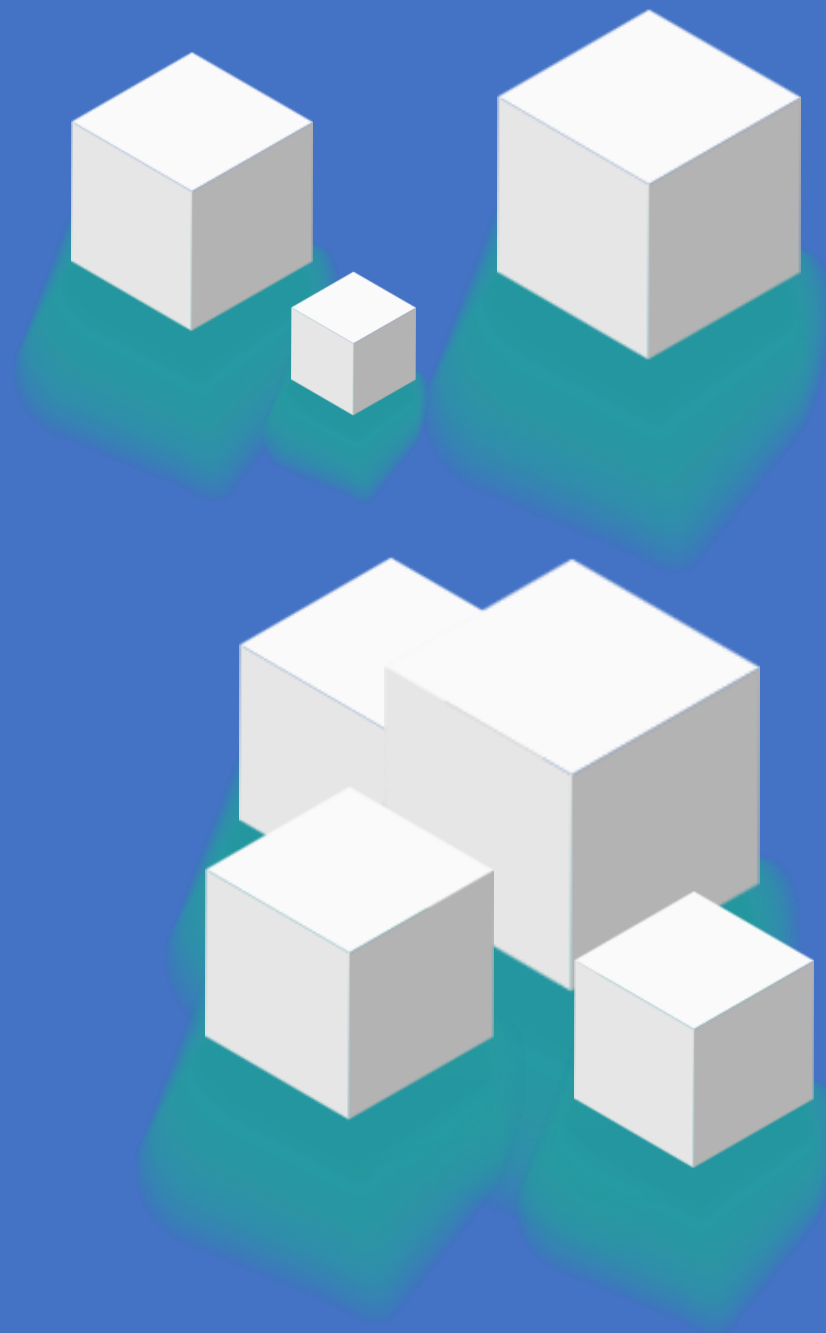
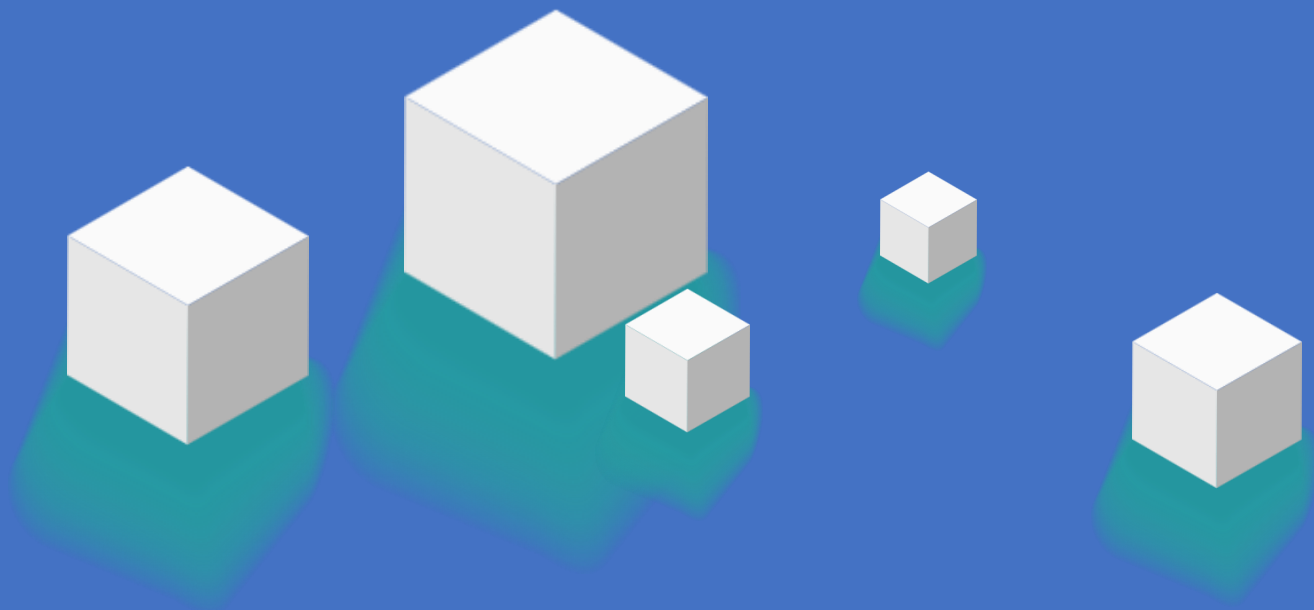


机器学习神器



学习方法

- Thinking: behind the theory, original from the real problem
- Action: solve problems by tools, present the results

>> 今天的学习目标

预测全家桶

- Project A: 员工离职预测
- Project B: 男女声音识别
- 分类算法: LR, SVM, KNN
- 树模型: GBDT, XGBoost, LightGBM, CatBoost, NGBoost

机器学习神器

- 什么是集成学习
- GBDT原理
- XGBoost
- LightGBM
- CatBoost
- 在Project中使用机器学习神器
- AI大赛: 二手车价格预测
- 如何防止模型过拟合

1/2 预测全家桶

Project A: 员工离职预测

员工离职预测

- In Class Competition
- <https://www.kaggle.com/c/bi-attrition-predict/>
- 我们有员工的各种统计信息，以及该员工是否已经离职，统计的信息包括了（工资、出差、工作环境满意度、工作投入度、是否加班、是否升职、工资提升比例等）
- 现在需要你来通过训练数据得出 员工离职预测，并给出你在测试集上的预测结果。我们将给出课程上公开的榜单



Project A: 员工离职预测

数据表字段:

字段	定义
Age	员工年龄
Attrition	员工是否已经离职, Yes表示离职, No表示未离职
BusinessTravel	商务差旅频率, Non-Travel不出差, TravelRarely不经常出差, TravelFrequently经常出差
DailyRate	平均日工资
Department	员工所在部门, Sales销售部, Research & Development研发部, Human Resources人力资源部
DistanceFromHome	公司跟家庭住址的距离, 从1到29, 1表示最近, 29表示最远
Education	员工的教育程度, 从1到5, 5表示教育程度最高
EducationField	员工所学习的专业领域, Life Sciences表示生命科学, Medical表示医疗, Marketing表示市场营销, Technical Degree表示技术学位, Human Resources表示人力资源, Other表示其他
EmployeeNumber	员工号码
EnvironmentSatisfaction	员工对于工作环境的满意程度, 从1到4, 1的满意程度最低, 4的满意程度最高
Gender	员工性别, Male表示男性, Female表示女性

字段	定义
JobInvolvement	员工工作投入度, 从1到4, 1为投入度最低, 4为投入度最高
JobLevel	职业级别, 从1到5, 1为最低级别, 5为最高级别
JobRole	工作角色: Sales Executive销售主管, Research Scientist科学研究员, Laboratory Technician实验室技术员, Manufacturing Director制造总监, Healthcare Representative医疗代表, Manager经理, Sales Representative销售代表, Research Director研究总监, Human Resources人力资源
JobSatisfaction	工作满意度, 从1到4, 1代表满意度最低, 4代表最高
MaritalStatus	员工婚姻状况, Single单身, Married已婚, Divorced离婚
MonthlyIncome	员工月收入, 范围在1009到19999之间
NumCompaniesWorked	员工曾经工作过的公司数
Over18	年龄是否超过18岁
OverTime	是否加班, Yes表示加班, No表示不加班
PercentSalaryHike	工资提高的百分比

Project A: 员工离职预测

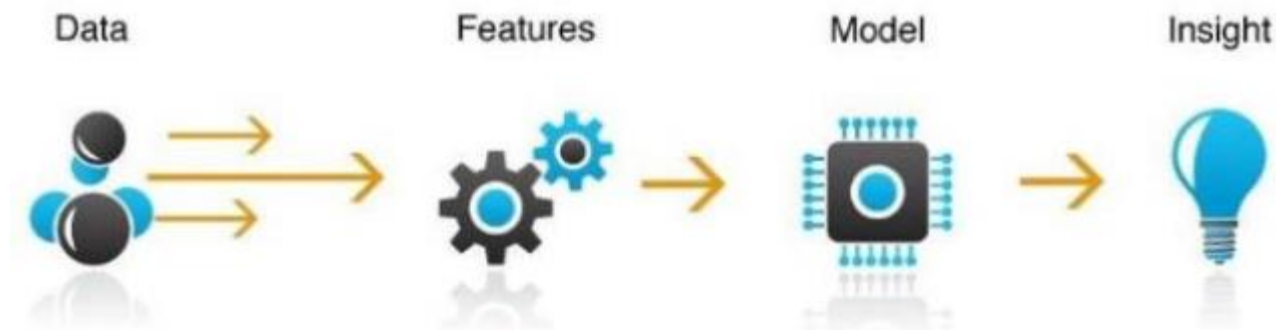
数据表字段:

字段	定义
PerformanceRating	绩效评估
RelationshipSatisfaction	关系满意度, 从1到4, 1表示满意度最低, 4表示满意度最高
StandardHours	标准工时
StockOptionLevel	股票期权水平
TotalWorkingYears	总工龄
TrainingTimesLastYear	上一年的培训时长, 从0到6, 0表示没有培训, 6表示培训时间最长
WorkLifeBalance	工作与生活平衡程度, 从1到4, 1表示平衡程度最低, 4表示平衡程度最高
YearsAtCompany	在目前公司工作年数
YearsInCurrentRole	在目前工作职责的工作年数
YearsSinceLastPromotion	距离上次升职时长
YearsWithCurrManager	跟目前的管理者共事年数

预测全家桶

常用预测（分类，回归）模型：

- 分类算法：LR, SVM, KNN
- 树模型：GBDT, XGBoost, LightGBM, CatBoost, NGBoost



- 特征工程：好的特征工程是拿分的关键
- 模型：懂原理，会调参

常用预测模型

- 数据预处理
- 分类算法: LR, SVM, KNN
- 树模型: GBDT, XGBoost, LightGBM, CatBoost, NGBost

Project A: 数据预处理

Step1, 对数据进行探索

#工离职预测

```
import pandas as pd
```

```
train=pd.read_csv('train.csv',index_col=0)
```

```
test=pd.read_csv('test.csv',index_col=0)
```

```
print(train['Attrition'].value_counts())
```

处理Attrition字段

```
train['Attrition']=train['Attrition'].map(lambda x:1 if x=='Yes' else 0)
```

查看数据中每列是否有空值

```
print(train.isna().sum())
```



```
No    988
Yes    188
Name: Attrition, dtype: int64
Age          0
Attrition     0
BusinessTravel    0
DailyRate       0
Department      0
DistanceFromHome    0
Education       0
EducationField    0
EmployeeCount     0
.....
YearsInCurrentRole    0
YearsSinceLastPromotion  0
YearsWithCurrManager   0
dtype: int64
```

Project A: 数据预处理

Step2, 去掉无用特征, 处理分类特征

去掉没用的列 员工号码, 标准工时 (=80)

```
train = train.drop(['EmployeeNumber', 'StandardHours'], axis=1)
```

```
test = test.drop(['EmployeeNumber', 'StandardHours'], axis=1)
```

对于分类特征进行特征值编码

```
from sklearn.preprocessing import LabelEncoder
```

```
attr=['Age','BusinessTravel','Department','Education','EducationField','Gender',  
, 'JobRole','MaritalStatus','Over18','OverTime']
```

```
for feature in attr:
```

```
    lbe=LabelEncoder()
```

```
    train[feature]=lbe.fit_transform(train[feature])
```

```
    test[feature]=lbe.transform(test[feature])
```

```
train.to_csv('temp.csv')
```

uesr_id	Age	Attritior	BusinessTravel	DailyRate	Department	DistanceFromHome	Education	Education
1374	40	0	2	605	2	21	2	1
1092	27	0	2	950	1	28	2	5
768	22	0	2	300	2	26	2	2
569	18	0	0	1434	2	8	3	1
911	7	1	1	599	2	24	0	1
408	34	0	2	1490	1	4	1	1
1321	29	0	2	207	1	9	3	1
1224	8	0	2	390	1	17	3	3
1061	6	0	0	830	2	13	1	1
530	9	0	2	608	1	1	1	1
1233	12	0	2	793	1	16	0	1
1303	29	0	2	1001	1	4	2	1
347	29	0	1	1309	2	4	0	3
440	16	1	1	988	0	23	2	0
1160	27	0	2	1329	1	2	1	4
826	20	0	2	433	0	1	2	0
61	20	0	1	653	1	29	4	1
519	11	0	1	806	1	1	3	1
1027	16	0	2	401	1	1	2	1
92	12	0	2	1334	2	4	1	3
1169	9	0	2	486	1	8	2	3
620	17	0	2	1343	1	27	0	3
682	14	0	0	1184	1	1	2	1
40	17	0	2	464	1	4	1	4
977	16	0	0	999	1	26	0	5
75	13	0	2	746	1	8	3	1
422	1	1	2	489	0	2	1	5
1264	37	0	2	478	1	2	2	3
703	20	0	0	152	2	10	2	5
662	2	1	2	500	2	2	2	3
247	16	0	2	470	1	2	3	1
621	18	0	2	928	2	1	1	1
1192	31	0	2	464	1	16	2	3
553	22	0	2	804	1	2	0	3
186	22	0	2	989	1	4	0	3
156	33	0	2	1169	1	7	3	3
879	42	0	2	696	2	7	3	2
312	13	0	2	192	1	2	3	1
8	20	0	1	216	1	23	2	1

Project A: 数据预处理

处理前，处理后数据对比

uesr_id	Age	Attrition	BusinessTravel	DailyRate	Department	DistanceFromHome	Education	EducationLevel
1374	58	No	Travel_Rarely	605	Sales	21	3	Life Science
1092	45	No	Travel_Rarely	950	Research	28	3	Technical
768	40	No	Travel_Rarely	300	Sales	26	3	Marketing
569	36	No	Non-Travel	1434	Sales	8	4	Life Science
911	25	Yes	Travel_Frequently	599	Sales	24	1	Life Science
408	52	No	Travel_Rarely	1490	Research	4	2	Life Science
1321	47	No	Travel_Rarely	207	Research	9	4	Life Science
1224	26	No	Travel_Rarely	390	Research	17	4	Medical
1061	24	No	Non-Travel	830	Sales	13	2	Life Science
530	27	No	Travel_Rarely	608	Research	1	2	Life Science
1233	30	No	Travel_Rarely	793	Research	16	1	Life Science
1303	47	No	Travel_Rarely	1001	Research	4	3	Life Science
347	47	No	Travel_Frequently	1309	Sales	4	1	Medical
440	34	Yes	Travel_Frequently	988	Human Resources	23	3	Human Resources
1160	45	No	Travel_Rarely	1329	Research	2	2	Other
826	38	No	Travel_Rarely	433	Human Resources	1	3	Human Resources
61	38	No	Travel_Frequently	653	Research	29	5	Life Science
519	29	No	Travel_Frequently	806	Research	1	4	Life Science
1027	34	No	Travel_Rarely	401	Research	1	3	Life Science
92	30	No	Travel_Rarely	1334	Sales	4	2	Medical
1169	27	No	Travel_Rarely	486	Research	8	3	Medical
620	35	No	Travel_Rarely	1343	Research	27	1	Medical
682	32	No	Non-Travel	1184	Research	1	3	Life Science
40	35	No	Travel_Rarely	464	Research	4	2	Other
977	34	No	Non-Travel	999	Research	26	1	Technical
75	31	No	Travel_Rarely	746	Research	8	4	Life Science
422	19	Yes	Travel_Rarely	489	Human Resources	2	2	Technical
1264	55	No	Travel_Rarely	478	Research	2	3	Medical
703	38	No	Non-Travel	152	Sales	10	3	Technical
662	20	Yes	Travel_Rarely	500	Sales	2	3	Medical
247	34	No	Travel_Rarely	470	Research	2	4	Life Science
621	36	No	Travel_Rarely	928	Sales	1	2	Life Science
1192	49	No	Travel_Rarely	464	Research	16	3	Medical
553	40	No	Travel_Rarely	804	Research	2	1	Medical
186	40	No	Travel_Rarely	989	Research	4	1	Medical
156	51	No	Travel_Rarely	1169	Research	7	4	Medical
879	60	No	Travel_Rarely	696	Sales	7	4	Marketing
312	31	No	Travel_Rarely	192	Research	2	4	Life Science
8	38	No	Travel_Frequently	216	Research	23	3	Life Science



uesr_id	Age	Attrition	BusinessTravel	DailyRate	Department	DistanceFromHome	Education	EducationLevel
1374	40	0	2	605	2	21	2	1
1092	27	0	2	950	1	28	2	5
768	22	0	2	300	2	26	2	2
569	18	0	0	1434	2	8	3	1
911	7	1	1	599	2	24	0	1
408	34	0	2	1490	1	4	1	1
1321	29	0	2	207	1	9	3	1
1224	8	0	2	390	1	17	3	3
1061	6	0	0	830	2	13	1	1
530	9	0	2	608	1	1	1	1
1233	12	0	2	793	1	16	0	1
1303	29	0	2	1001	1	4	2	1
347	29	0	1	1309	2	4	0	3
440	16	1	1	988	0	23	2	0
1160	27	0	2	1329	1	2	1	4
826	20	0	2	433	0	1	2	0
61	20	0	1	653	1	29	4	1
519	11	0	1	806	1	1	3	1
1027	16	0	2	401	1	1	2	1
92	12	0	2	1334	2	4	1	3
1169	9	0	2	486	1	8	2	3
620	17	0	2	1343	1	27	0	3
682	14	0	0	1184	1	1	2	1
40	17	0	2	464	1	4	1	4
977	16	0	0	999	1	26	0	5
75	13	0	2	746	1	8	3	1
422	1	1	2	489	0	2	1	5
1264	37	0	2	478	1	2	2	3
703	20	0	0	152	2	10	2	5
662	2	1	2	500	2	2	2	3
247	16	0	2	470	1	2	3	1
621	18	0	2	928	2	1	1	1
1192	31	0	2	464	1	16	2	3
553	22	0	2	804	1	2	0	3
186	22	0	2	989	1	4	0	3
156	33	0	2	1169	1	7	3	3
879	42	0	2	696	2	7	3	2
312	13	0	2	192	1	2	3	1
8	20	0	1	216	1	23	2	1

LR工具

LR工具:

- `from sklearn.linear_model.logistic import LogisticRegression`

参数:

- `penalty`, 惩罚项, 正则化参数, 防止过拟合, l1或l2, 默认为l2
- `C`, 正则化系数 λ 的倒数, `float`类型, 默认为1.0
- `solver`, 损失函数优化方法, `liblinear` (默认), `lbfgs`, `newton-cg`, `sag`
- `random_state`, 随机数种子
- `max_iter`, 算法收敛的最大迭代次数, 默认为100
- `tol=0.0001`: 优化算法停止条件, 迭代前后函数差小于`tol`则终止
- `verbose=0`: 日志冗长度`int`: 冗长度; 0: 不输出训练过程; 1: 偶尔输出; >1: 对每个子模型都输出
- `n_jobs=1`: 并行数, `int`: 个数; -1: 跟CPU核数一致; 1:默认值

常用方法:

- `fit(X, y, sample_weight=None)`
- `fit_transform(X, y=None, **fit_params)`
- `predict(X)`, 用来预测样本, 也就是分类
- `predict_proba(X)`, 输出分类概率。返回每种类别的概率, 按照分类类别顺序给出。
- `score(X, y, sample_weight=None)`, 返回给定测试集合的平均准确率 (mean accuracy)

LR工具

Step3, 模型参数配置

```
model = LogisticRegression(max_iter=100,  
                           verbose=True,  
                           random_state=33,  
                           tol=1e-4  
                           )
```

```
model.fit(X_train, y_train)
```

```
predict = model.predict_proba(test)[: , 1]
```

```
test['Attrition']=predict
```

转化为二分类输出

```
test['Attrition']=test['Attrition'].map(lambda x:1 if x>=0.5 else 0)
```

```
test[['Attrition']].to_csv('submit_lr.csv')
```



```
iter 25 act 1.052e+01 pre 1.050e+01 delta 2.865e-01 f 3.291e+02 |g| 3.106e+02 CG 9  
iter 26 act 5.049e-02 pre 5.014e-02 delta 2.865e-01 f 3.186e+02 |g| 5.462e+04 CG 1  
iter 27 act 8.230e-03 pre 8.184e-03 delta 2.865e-01 f 3.185e+02 |g| 5.377e+03 CG 1  
cg reaches trust region boundary  
iter 28 act 9.298e+00 pre 9.223e+00 delta 4.300e-01 f 3.185e+02 |g| 2.770e+02 CG 12  
iter 29 act 2.898e-02 pre 2.878e-02 delta 4.300e-01 f 3.092e+02 |g| 3.736e+04 CG 2  
iter 30 act 1.198e+00 pre 1.177e+00 delta 4.300e-01 f 3.092e+02 |g| 8.193e+02 CG 6  
iter 31 act 1.835e-03 pre 1.832e-03 delta 4.300e-01 f 3.080e+02 |g| 9.437e+03 CG 2  
cg reaches trust region boundary  
iter 32 act 7.216e+00 pre 7.313e+00 delta 4.477e-01 f 3.080e+02 |g| 9.929e+01 CG 12  
iter 33 act 6.677e-02 pre 6.597e-02 delta 4.477e-01 f 3.008e+02 |g| 5.300e+04 CG 2  
iter 34 act 3.256e+00 pre 3.259e+00 delta 4.477e-01 f 3.007e+02 |g| 8.364e+02 CG 8  
iter 35 act 3.303e-02 pre 3.265e-02 delta 4.477e-01 f 2.974e+02 |g| 3.015e+04 CG 2  
iter 36 act 3.761e-01 pre 3.697e-01 delta 4.477e-01 f 2.974e+02 |g| 3.958e+02 CG 6  
iter 37 act 2.707e-04 pre 2.705e-04 delta 4.477e-01 f 2.970e+02 |g| 3.163e+03 CG 2
```

user_id	Attrition
442	0
1091	0
981	0
785	0
1332	1
501	0
1058	1
1253	0
751	0
122	0
268	0
940	0
1125	0
540	0
1034	0
432	0
794	0
666	0
942	0
1114	0
853	0
1330	0
692	0
548	0
54	1
319	0
1147	0
1235	0
1436	1

SVM工具

SVM工具:

- sklearn中支持向量分类主要有三种方法: SVC、NuSVC、LinearSVC
- `sklearn.svm.SVC(C=1.0, kernel='rbf', degree=3, gamma='auto', coef0=0.0, shrinking=True, probability=False, tol=0.001, cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_function_shape='ovr', random_state=None)`
- `sklearn.svm.NuSVC(nu=0.5, kernel='rbf', degree=3, gamma='auto', coef0=0.0, shrinking=True, probability=False, tol=0.001, cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_function_shape='ovr', random_state=None)`
- `sklearn.svm.LinearSVC(penalty='l2', loss='squared_hinge', dual=True, tol=0.0001, C=1.0, multi_class='ovr', fit_intercept=True, intercept_scaling=1, class_weight=None, verbose=0, random_state=None, max_iter=1000)`

常用参数:

- C, 惩罚系数, 类似于LR中的正则化系数, C越大惩罚越大
- nu, 代表训练集训练的误差率的上限 (用于NuSVC)
- kernel, 核函数类型, RBF, Linear, Poly, Sigmoid, precomputed, 默认为RBF径向基核 (高斯核函数)
- gamma, 核函数系数, 默认为auto
- degree, 当指定kernel为'poly'时, 表示选择的多项式的最高次数, 默认为三次多项式
- probability, 是否使用概率估计
- shrinking, 是否进行启发式, SVM只用少量训练样本进行计算
- penalty, 正则化参数, L1和L2两种参数可选, 仅LinearSVC有
- loss, 损失函数, 有 'hinge' 和 'squared_hinge' 两种可选, 前者又称L1损失, 后者称为L2损失
- tol: 残差收敛条件, 默认是0.0001, 与LR中的一致

SVM工具

SVM工具:

- SVC, Support Vector Classification, 支持向量机用于分类

libsvm中自带了四种核函数: 线性核、多项式核、RBF以及sigmoid核

- SVR, Support Vector Regression, 支持向量机用于回归

Kernel 核的选择技巧的:

- sklearn中支持向量分类主要有三种方法: SVC、NuSVC、LinearSVC

- 如果样本数量 < 特征数:

- 基于libsvm工具包实现, 台湾大学林智仁教授在2001年开发的一个简单易用的SVM工具包

方法1: 简单的使用线性核就可以, 不用选择非线性核

方法2: 可以先对数据进行降维, 然后使用非线性核

- SVC, C-Support Vector Classification, 支持向量分类

- 如果样本数量 >= 特征数

- NuSVC, Nu-Support Vector Classification, 核支持向量分类, 和SVC类似, 不同的是可以使用参数来控制支持向量的个数

可以使用非线性核, 将样本映射到更高维度, 可以得到比较好的结果

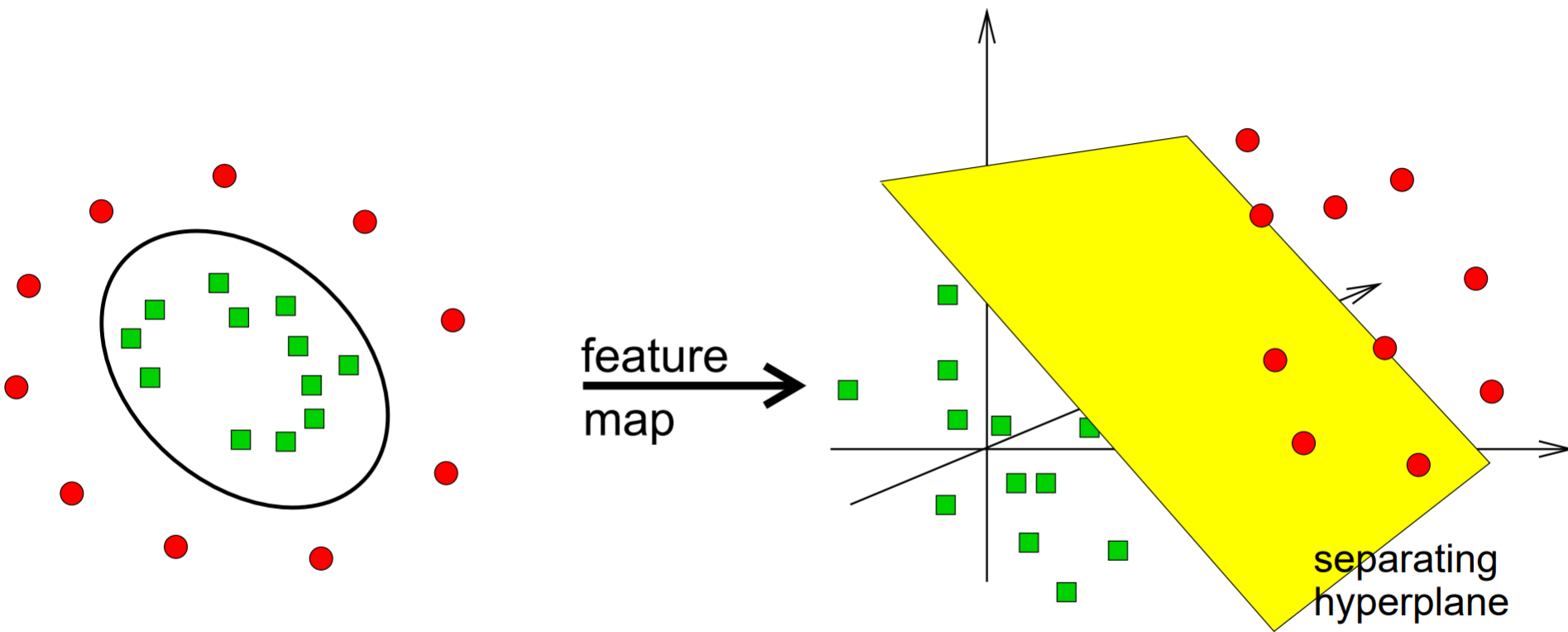
- LinearSVC, Linear Support Vector Classification

线性支持向量分类, 使用的核函数是linear

SVM工具

SVM思想：一些线性不可分的问题可能是非线性可分的，也就是在高维空间中存在分离超平面（separating hyperplane）

使用非线性函数从原始的特征空间映射至更高维的空间，转化为线性可分问题



complex in low dimensions

simple in higher dimensions

SVM工具

Step3, 模型参数配置

```
model = LinearSVC(max_iter=1000,  
                  random_state=33,  
                  verbose=True,  
                  )  
model.fit(X_train, y_train)  
predict = model.predict(test)  
print(predict)  
test1['Attrition']=predict  
test1[['Attrition']].to_csv('submit_svc.csv')
```



```
.....*.....**.  
optimization finished, #iter = 500  
Objective value = -359.033106  
nSV = 614  
[LibLinear][0 0 0 0 1 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0  
0 0 0 0  
0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0  
0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0  
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0  
0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  
1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0  
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0]
```

user_id	Attrition
442	0
1091	0
981	0
785	0
1332	1
501	0
1058	1
1253	0
751	0
122	0
268	0
940	0
1125	0
540	1
1034	0
432	0
794	0
666	0
942	0
1114	0
853	0
1330	0
692	0
548	0
54	1
319	0
1147	0
1235	0
1436	1

打卡：员工离职预测分析



针对员工离职数据集进行分析

训练集： train.csv

测试集： test.csv

- 使用不同的分类模型： LR, SVM, XGBoost, LightGBM, CatBoost
- 员工画像分析

哪些人容易离职

哪些人不容易离职

- 后续决策建议

Project B: 男女声音识别

男女声音识别

- 数据集：3168个录制的声音样本（来自男性和女性演讲者），采集的频率范围是0hz-280hz，已经对数据进行了预处理
- 一共有21个属性值，请判断该声音是男还是女？
- 使用Accuracy作为评价标准



Project B: 男女声音识别

数据表字段:

字段+	定义
meanfreq	平均频率（单位kHz）
sd	频率标准差
median	中位频率（单位kHz）
Q25	频率第一个四分位数（单位kHz）
Q75	频率第三个四分位数（单位kHz）
IQR	四分位数间距
skew	歪斜
kurt	峰度
sfm	频谱平坦度
mode	波模频率
centroid	质心频率

字段	定义
peakf	峰值频率
meanfun	测量声信号的基频平均值
minfun	测量声信号的最小基频
maxfun	测量声波信号的最大基频
meandom	通过声学信号测量的主导频率的平均值
mindom	通过声学信号测量的最小频率
maxdom	通过声学信号测量的最大频率
dfrange	声学信号测量的主频范围
modindx	调制指数（计算基频相邻测量值之间的累积绝对差除以频率范围）
label	男 or 女

Project B: 男女声音识别

男女声音识别

- Step1, 数据加载
- Step2, 数据预处理

分离特征X 和Target y

使用标签编码, male -> 1, female -> 0

将特征X矩阵进行规范化

#标准差标准化, 处理后的数据符合标准正态分布

```
scaler = StandardScaler()
```

- Step3, 数据集切分, train_test_split

- Step4, 模型训练

SVM, Linear SVM

- Step5, 模型预测

常用预测模型

- 数据预处理
- 分类算法：LR , SVM， KNN
- 树模型： GBDT, XGBoost, LightGBM, CatBoost， NGBost

Summary

每种模型都有适用的场景

LR优点:

- 实现简单，广泛的应用于工业问题上；
- 分类时计算量非常小，速度很快，使用资源低；
- 方便观测样本概率分数；

LR缺点:

- 当特征空间很大时，LR的性能不是很好；
- 容易欠拟合，准确度不太高；
- 不能很好地处理大量多类特征或变量；
- 通常只处理二分类问题，多分类需要使用softmax（LR在多分类的推广），且必须线性可分；
- 对于非线性特征，需要进行转换；

SVM优点:

- 可以解决高维问题，即大型特征空间；
- 能够处理非线性特征的相互作用；
- 需要先对数据进行归一化，因为计算是基于距离的模型，所以SVM和LR都需要对数据进行归一化处理

SVM缺点:

- 当样本很多时，效率并不是很高；
- 对非线性问题没有通用解决方案，可能会很难找到合适核函数
- 对缺失数据敏感；

SVM核的选择是有技巧的，样本数量<特征数，线性核，大于特征数使用非线性核

Summary

每种模型都有适用的场景

- 可以使用LR模型作为预测的Baseline
- FM衍生模型在推荐系统，尤其是CTR预估中有广泛应用，弥补了LR模型的不足（需要人工组合特征，耗费大量时间和人力）
- Attention机制，对于Diversity多样性的情况，Attention机制可以提升效率，并且得出更好的结果
- Tree Ensemble模型，比如GBDT，使用广泛，因为训练模型更可控

对于 LR 模型，如果欠拟合，需要增加feature，才能提高准确率。而对于 tree-ensemble 来说，解决方法是训练更多的决策树 tree。Kaggle比赛中使用很多

常用预测模型：

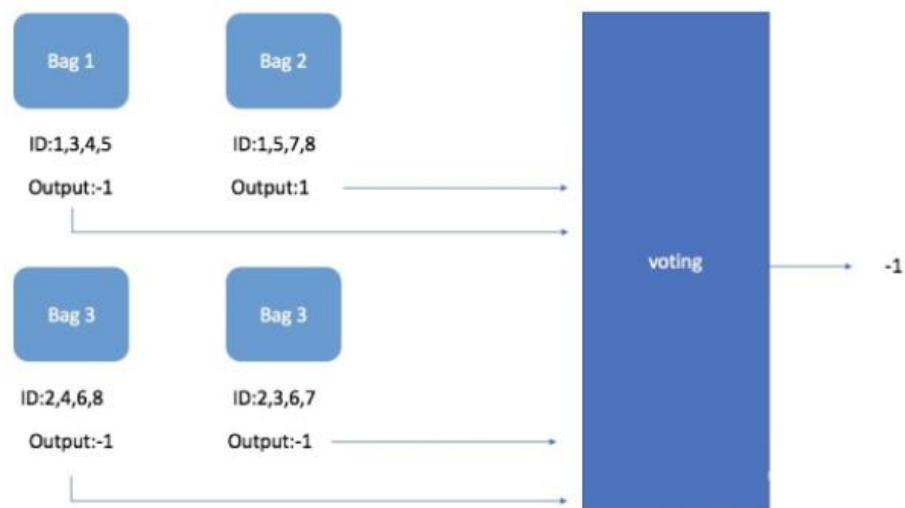
- 分类算法： LR , SVM， KNN
- 矩阵分解： FunkSVD， BiasSVD， SVD++
- FM模型： FM, FFM, DeepFM, NFM， AFM
- 树模型： GBDT, XGBoost, LightGBM, CatBoost， NGBoost
- Attention模型： DIN, DIEN, DSIN

2/2 机器学习神器

什么是集成学习

集成学习：

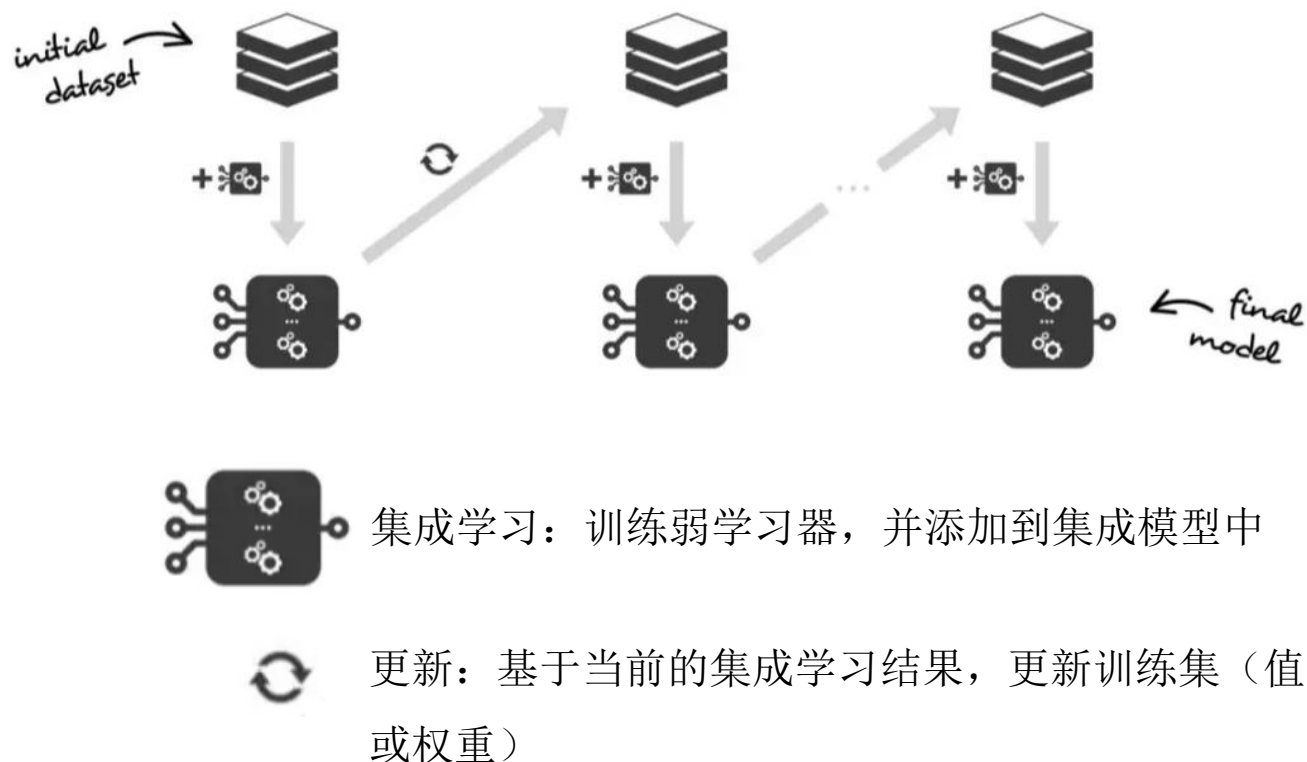
- 思想，将多个弱分类器按照某种方式组合起来，形成一个强分类器（三个臭皮匠赛过诸葛亮）
- Bagging**，把数据集通过有放回的抽样方式，划分为多个数据集，分别训练多个模型。针对分类问题，按照少数服从多数原则进行投票，针对回归问题，求多个测试结果的平均值
- Stacking**，通常是不同的模型，而且每个分类都用了全部训练数据，得到预测结果 y_1, y_2, \dots, y_k ，然后再训练一个分类器 **Meta Classifier**，将这些预测结果作为输入，得到最终的预测结果



什么是集成学习

集成学习：

- Boosting，与Bagging一样，使用的相同的弱学习器，不过是以自适应的方法顺序地学习这些弱学习器，即每个新学习器都依赖于前面的模型，并按照某种确定性的策略将它们组合起来
- 两个重要的 Boosting 算法：AdaBoost（自适应提升）和 Gradient Boosting（梯度提升）
- AdaBoost，使用前面的学习器用简单的模型去适配数据，然后分析错误。然后会给予错误预测的数据更高权重，然后用后面的学习器去修复
- Boosting通过把一些列的弱学习器串起来，组成一个强学习器



什么是集成学习

Boosting与Bagging:

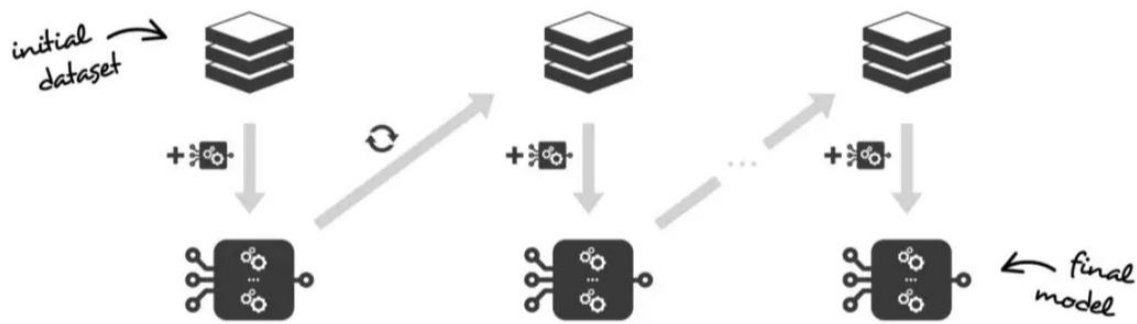
- 结构上, Bagging是基分类器并行处理, 而Boosting是串行处理
- 训练集, Bagging的基分类器训练是独立的, 而Boosting的训练集是依赖于之前的模型
- 作用, Bagging的作用是减少variance, 而Boosting在于减少bias

对于Bagging, 对样本进行重采样, 通过重采样得到的子样本集训练模型, 最后取平均。因为子样本集的相似性, 而且使用相同的弱学习器, 因此每个学习器有近似相等的bias和variance, 因为每个学习器相互独立, 所以可以显著降低variance, 但是无法降低bias

对于Boosting, 采用顺序的方式最小化损失函数, 所以bias自然是逐步下降, 子模型之和不能显著降低variance



Bagging学习方式

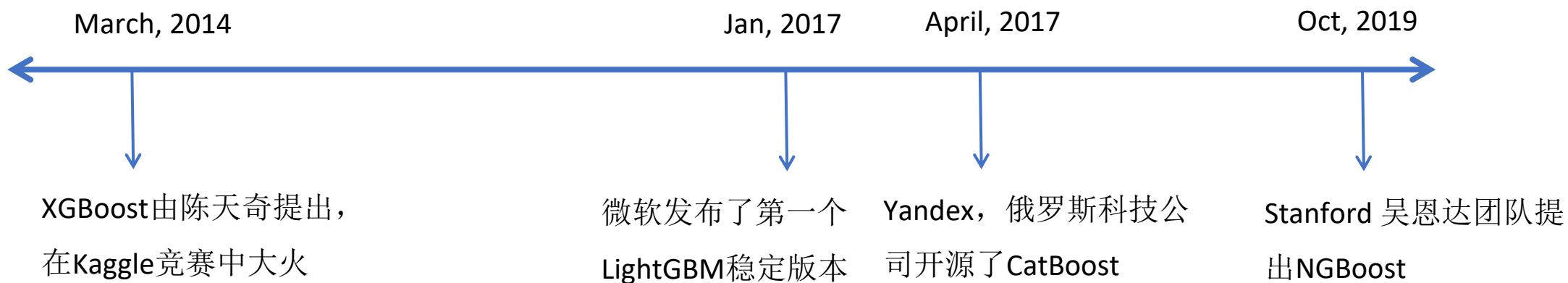


Boosting学习方式

机器学习神器

Gradient Boosting集成学习:

- XGBoost, LightGBM, CatBoost, NGBoost实际上是对GBDT方法的不同实现, 针对同一目标、做了不同的优化处理



XGBoost

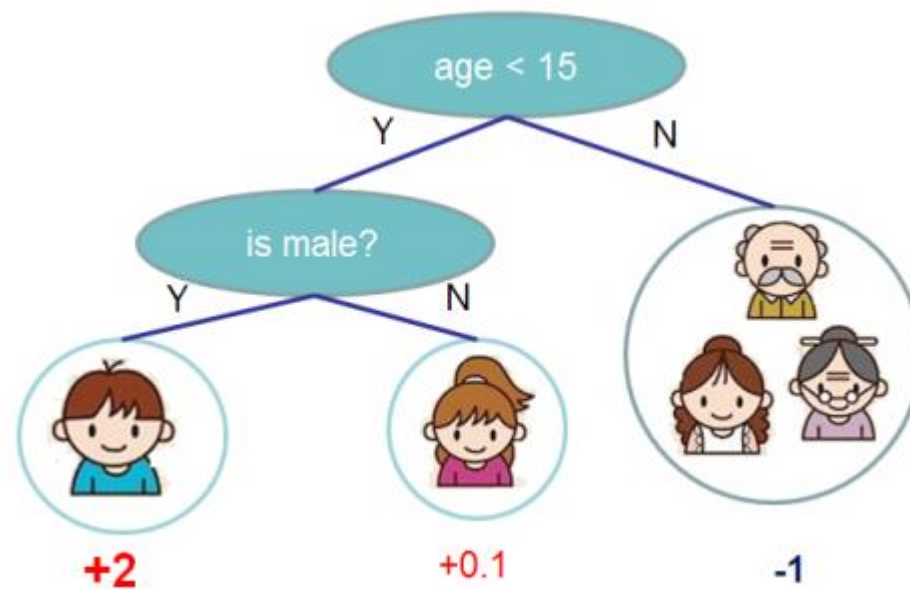
XGBoost:

- <https://arxiv.org/abs/1603.02754>
- 对于一个问题，INPUT X: age, gender, occupation,

Target y: How does the person like computer games?



基学习器，采用CART回归树



每个叶子节点对应预测的分数

XGBoost

XGBoost:

- 目标函数=损失函数 + 正则化项

$$Obj(\Theta) = L(\Theta) + \Omega(\Theta)$$

损失函数: 拟合数据

正则化项: 惩罚复杂模型

- 误差函数尽量拟合训练数据, 正则化项鼓励简单的模型
- $\Omega(f_t)$ 用于控制树的复杂度, 防止过拟合, 使得模型更简化, 也使得最终的模型的预测结果更稳定

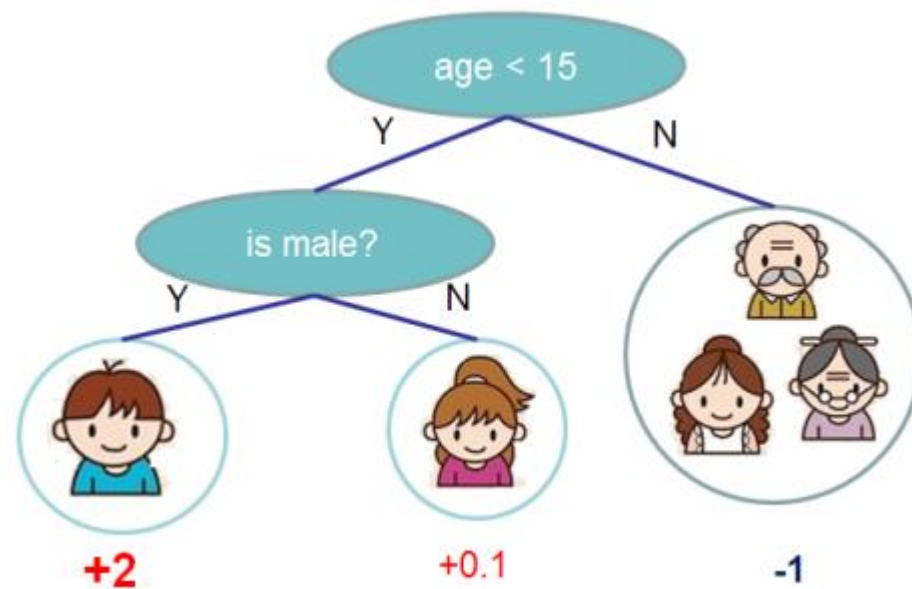
$$\Omega(f_t) = \gamma T + \frac{1}{2} \lambda \sum_{j=1}^T w_j^2$$

T : 叶子数量

w_j : 叶子分数的L2正则项

γ : 加入新叶子节点引入的复杂度代价

$f_t(x) = w_{q(x)}$ w 代表叶子向量, q 表示树的结构



$$\Omega = \gamma \times 3 + \frac{1}{2} \lambda (4 + 0.01 + 1)$$

XGBoost

Tree Ensemble 集成学习：

- 单个CART回归树过于简单，可以通过多个CART回归树组成一个强学习器
- 预测函数，样本的预测结果=每棵树预测分数之和

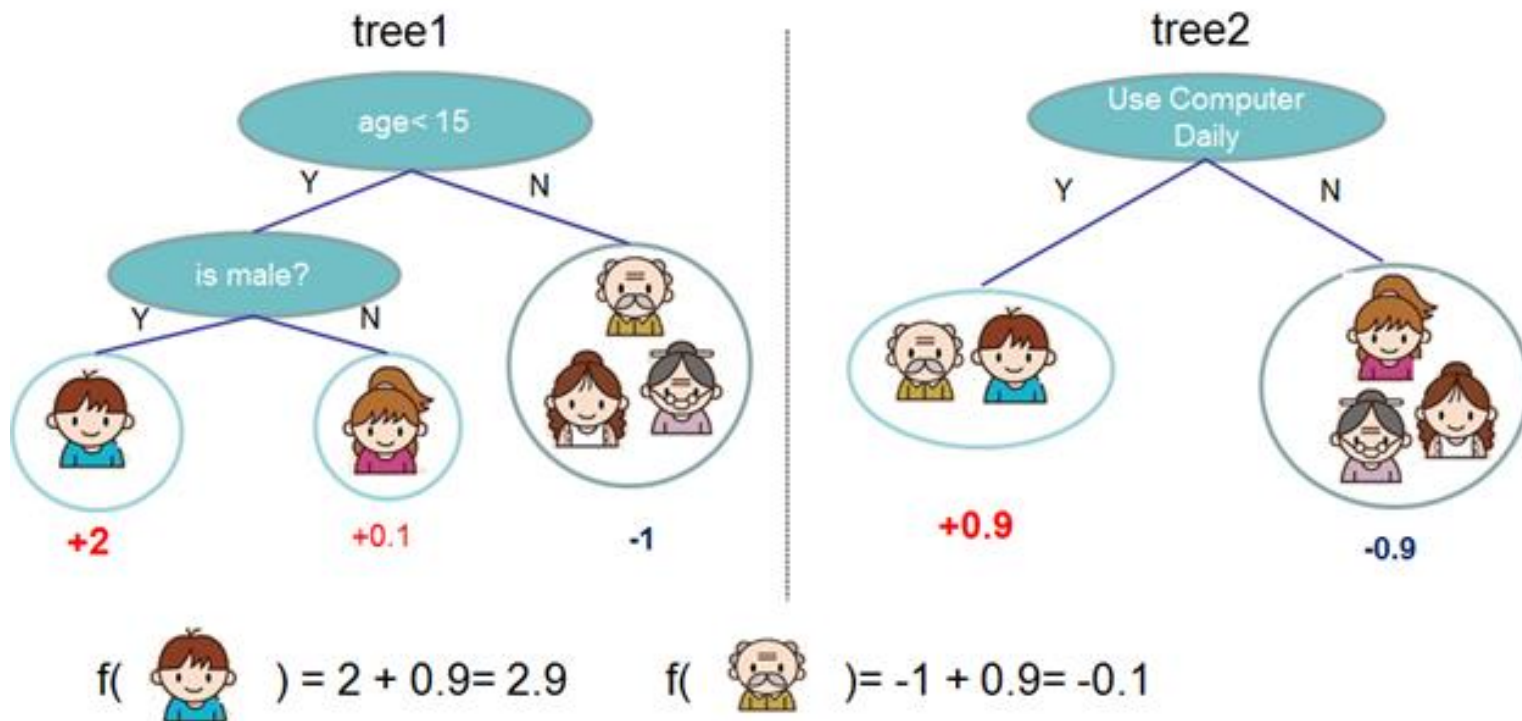
$$\hat{y}_i = \sum_{k=1}^K f_k(x_i)$$

- 目标函数优化

$$Obj(\Theta) = \sum_i l(y_i, \hat{y}_i) + \sum_k \Omega(f_k)$$

$$\Omega(f) = \gamma T + \frac{1}{2} \lambda \|w\|^2$$

正则项是由叶子结点的数量和叶子结点权重的平方和决定



XGBoost

XGBoost的目标函数：

$$Obj^t = \sum_{i=1}^n l(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)) + \Omega(f_t) + \text{constant}$$

- 对目标函数改进，进行二阶泰勒展开：


$$f(x + \Delta x) \approx f(x) + f'(x)\Delta x + \frac{1}{2} f''(x)\Delta x^2$$

- 定义

$$g_i = \partial_{\hat{y}^{(t-1)}} l(y_i, \hat{y}^{(t-1)})$$

$$h_i = \partial_{\hat{y}^{(t-1)}}^2 l(y_i, \hat{y}^{(t-1)})$$

$$Obj^t \approx \sum_{i=1}^n \left[l(y_i, \hat{y}_i^{(t-1)}) + g_i \boxed{f_t(x_i)} + \frac{1}{2} h_i f_t^2(x_i) \right] + \Omega(f_t) + \text{constant}$$



第t轮的模型预测

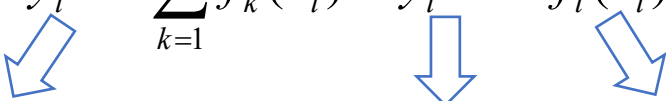
$$\hat{y}_i^{(0)} = 0$$

$$\hat{y}_i^{(1)} = f_1(x_i) = \hat{y}_i^{(0)} + f_1(x_i)$$

$$\hat{y}_i^{(2)} = f_1(x_i) + f_2(x_i) = \hat{y}_i^{(1)} + f_2(x_i)$$

...

$$\hat{y}_i^{(t)} = \sum_{k=1}^t f_k(x_i) = \hat{y}_i^{(t-1)} + f_t(x_i)$$



保留前t-1轮的模型预测

加入新的预测函数

如何选择每一轮的预测函数f？

选取一个f来使得目标函数尽量降低，即加入f后的预测结果与实际结果误差减少

XGBoost

XGBoost 的目标函数:

$$\begin{aligned} Obj^t &\approx \sum_{i=1}^n \left[g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i) \right] + \Omega(f_t) \\ &= \sum_{i=1}^n \left[g_i w_{q(x_i)} + \frac{1}{2} h_i w_{q(x_i)}^2 \right] + \gamma T + \lambda \frac{1}{2} \sum_{j=1}^T w_j^2 \\ &= \sum_{j=1}^T \left[\left(\sum_{i \in I_j} g_i \right) w_j + \frac{1}{2} \left(\sum_{i \in I_j} h_i + \lambda \right) w_j^2 \right] + \gamma T \end{aligned}$$

• T为叶子节点数量

I_j 定义为每个叶子节点里面的样本集合 $I_j = \{i \mid q(x_i) = j\}$

$f_t(x_i) = w_{q(x_i)}$ 即每个样本所在叶子节点索引的分数 (叶子权重w)

G_j, H_j 分别表示每个叶子节点的一阶梯度的和, 与二阶梯度的和:

$$G_j = \sum_{i \in I_j} g_i$$

$$H_j = \sum_{i \in I_j} h_i$$

目标函数改写为:

$$\begin{aligned} Obj^t &= \sum_{j=1}^T \left[\left(\sum_{i \in I_j} g_i \right) w_j + \frac{1}{2} \left(\sum_{i \in I_j} h_i + \lambda \right) w_j^2 \right] + \gamma T \\ &= \sum_{j=1}^T \left[G_j w_j + \frac{1}{2} (H_j + \lambda) w_j^2 \right] + \gamma T \end{aligned}$$

求偏导得到: $\frac{\partial Obj}{\partial w_j} = G_j + (H_j + \lambda) w_j = 0$

求解得 $w_j = -\frac{G_j}{H_j + \lambda}$ $Obj = -\frac{1}{2} \sum_{j=1}^T \frac{G_j^2}{H_j + \lambda} + \gamma T$

XGBoost

XGBoost 的目标函数:

- Obj 目标函数也称为结构分数（打分函数），代表当指定一个树的结构的时候，我们在目标上最多可以减少多少

样本编号 梯度计算

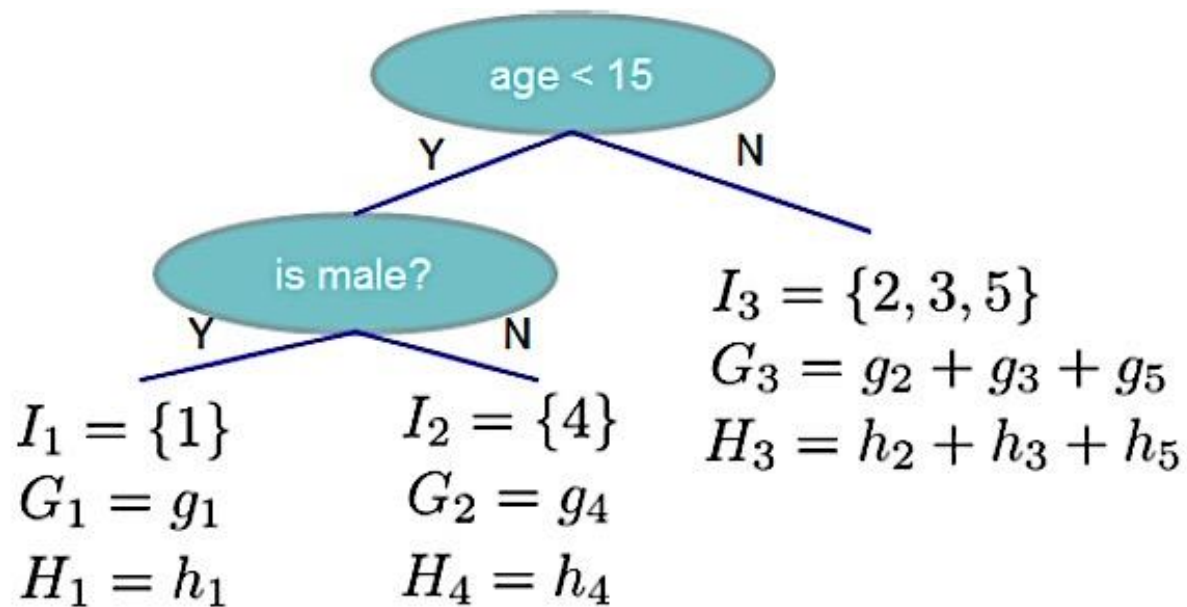
1  g1, h1

2  g2, h2

3  g3, h3

4  g4, h4

5  g5, h5



$$Obj = -\frac{1}{2} \sum_j \frac{G_j^2}{H_j + \lambda} + 3\gamma$$

分数越小，代表树的结构越好

XGBoost

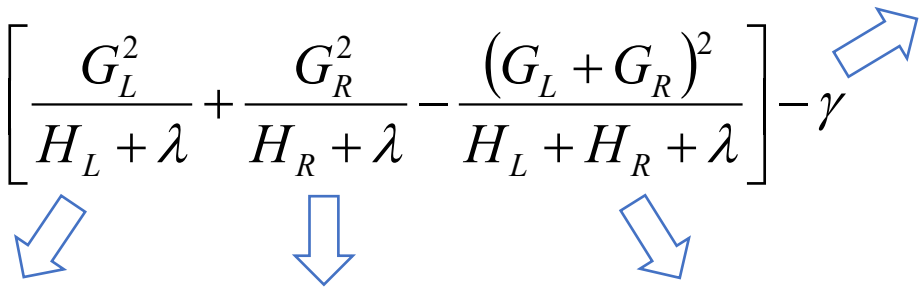
XGBoost的目标函数：

- 求Obj分数最小的树结构，可以穷举所有可能，但计算量太大

$$Obj = -\frac{1}{2} \sum_{j=1}^T \frac{G_j^2}{H_j + \lambda} + \gamma T$$

- 使用贪心法，即利用打分函数（计算增益）

以Gain作为是否分割的条件，Gain看作是未分割前的Obj减去分割后的左右Obj，
加入新叶子节点引入的复杂度代价

$$Gain = \frac{1}{2} \left[\frac{G_L^2}{H_L + \lambda} + \frac{G_R^2}{H_R + \lambda} - \frac{(G_L + G_R)^2}{H_L + H_R + \lambda} \right] - \gamma$$


左子树分数

右子树分数

不分割可以拿到的分数

如果Gain<0，则此叶节点不做分割，分割方案个数很多，计算量依然很大

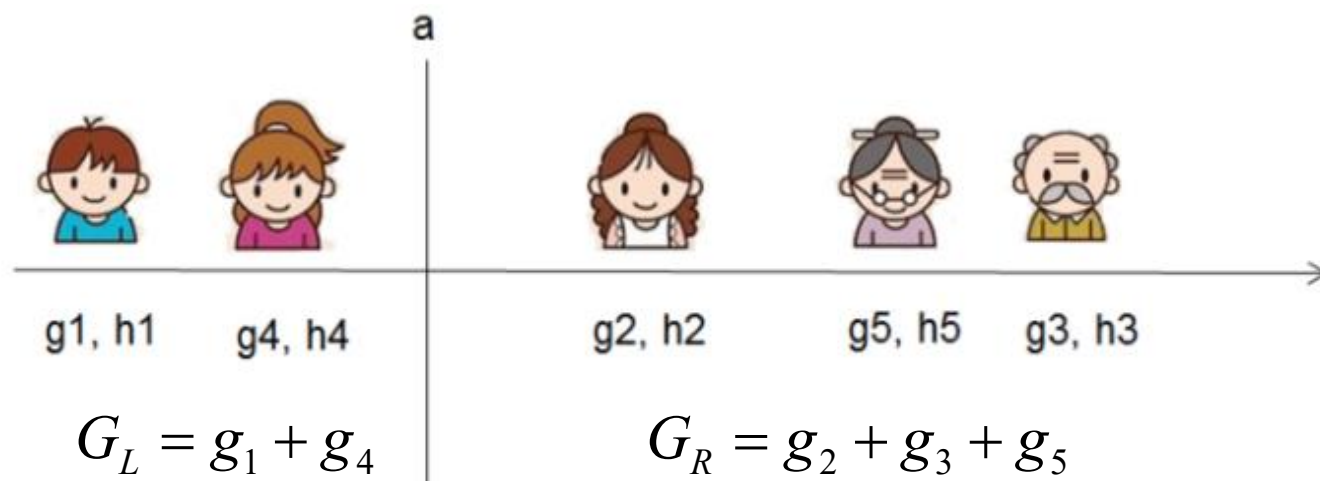
XGBoost

XGBoost的分裂节点算法:

- 贪心方法, 获取最优分割节点 (split point)

将所有样本按照 g_i 从小到大排序, 通过遍历, 查看每个节点是否需要分割

- 对于特征值的个数为 n 时, 总共有 $n-1$ 种划分
- Step1, 对样本扫描一遍, 得出 G_L , G_R
- Step2, 根据Gain的分数进行分割
- 通过贪心法, 计算效率得到大幅提升, XGBoost重新定义划分属性, 即Gain, 而Gain的计算是由目标损失函数obj决定的



Algorithm 1: Exact Greedy Algorithm for Split Finding

Input: I , instance set of current node

Input: d , feature dimension

$gain \leftarrow 0$

$G \leftarrow \sum_{i \in I} g_i$, $H \leftarrow \sum_{i \in I} h_i$

for $k = 1$ to m do

$G_L \leftarrow 0$, $H_L \leftarrow 0$

 for j in sorted(I , by x_{jk}) do

$G_L \leftarrow G_L + g_j$, $H_L \leftarrow H_L + h_j$

$G_R \leftarrow G - G_L$, $H_R \leftarrow H - H_L$

$score \leftarrow \max(score, \frac{G_L^2}{H_L + \lambda} + \frac{G_R^2}{H_R + \lambda} - \frac{G^2}{H + \lambda})$

 end

end

Output: Split with max score

XGBoost

XGBoost的分裂节点算法（近似算法，Histogram 2016 paper）：

- 对于连续型特征值，样本数量非常大，该特征取值过多时，遍历所有取值会花费很多时间，且容易过拟合
- 方法，在寻找split节点的时候，不会枚举所有的特征值，而会对特征值进行聚合统计，然后形成若干个bucket(桶)，只将bucket边界上的特征值作为split节点的候选，从而获得性能提升
- 从算法伪代码中该流程还可以分为两种，全局的近似是在新生成一棵树之前就对各个特征计算分位点并划分样本，之后在每次分裂过程中都采用近似划分，而局部近似就是在具体的某一次分裂节点的过程中采用近似算法

Algorithm 2: Approximate Algorithm for Split Finding

```
for  $k = 1$  to  $m$  do
    | Propose  $S_k = \{s_{k1}, s_{k2}, \dots, s_{kl}\}$  by percentiles on feature  $k$ .
    | Proposal can be done per tree (global), or per split(local).
end
for  $k = 1$  to  $m$  do
    |  $G_{kv} \leftarrow \sum_{j \in \{j | s_{k,v} \geq x_{jk} > s_{k,v-1}\}} g_j$ 
    |  $H_{kv} \leftarrow \sum_{j \in \{j | s_{k,v} \geq x_{jk} > s_{k,v-1}\}} h_j$ 
end
Follow same step as in previous section to find max
score only among proposed splits.
```

Summary(XGBoost)

XGBoost算法特点:

- XGBoost将树模型的复杂度加入到正则项中，从而避免过拟合，泛化性能好
- 损失函数是用泰勒展开式展开的，用到了一阶导和二阶导，可以加快优化速度
- 在寻找最佳分割点时，采用近似贪心算法，用来加速计算
- 不仅支持CART作为基分类器，还支持线性分类器，在使用线性分类器的时候可以使用L1，L2正则化
- 支持并行计算，XGBoost的并行是基于特征计算的并行，将特征列排序后以block的形式存储在内存中，在后面的迭代中重复使用这个结构。在进行节点分裂时，计算每个特征的增益，选择增益最大的特征作为分割节点，各个特征的增益计算可以使用多线程并行
- 优点：速度快、效果好、能处理大规模数据、支持自定义损失函数等
- 缺点：算法参数过多，调参复杂，不适合处理超高维特征数据

XGBoost工具

XGBoost工具:

- <https://github.com/dmlc/xgboost>

参数分为:

- 通用参数: 对系统进行控制
- **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

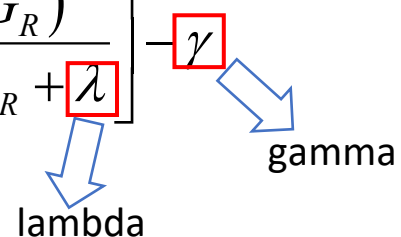
通用参数:

- **booster**, 模型选择, **gbtree**或者**gblinear**。**gbtree**使用基于树的模型进行提升计算, **gblinear**使用线性模型进行提升计算。
[default=gbtree]
- **silent**, 缄默方式, 0表示打印运行时, 1表示以缄默方式运行, 不打印运行时信息。[default=0]
- **nthread**, XGBoost运行时的线程数, [default=缺省值是当前系统可以获得的最大线程数]
- **num_feature**, boosting过程中用到的特征个数, XGBoost会自动设置

XGBoost工具

Booster参数:

- eta [default=0.3], 为了防止过拟合, 更新过程中用到的收缩步长。在每次提升计算之后, 算法会直接获得新特征的权重。 eta通过缩减特征的权重使提升计算过程更加保守, 取值范围为[0,1]
- gamma [default=0], 分裂节点时, 损失函数减小值只有大于等于gamma节点才分裂, gamma值越大, 算法越保守, 越不容易过拟合, 但性能就不一定能保证, 需要trade off, 取值范围 [0,∞]
- max_depth [default=6], 树的最大深度, 取值范围为[1,∞], 典型值为3-10
- min_child_weight [default=1], 一个子集的所有观察值的最小权重和。如果新分裂的节点的样本权重和小于min_child_weight则停止分裂。这个可以用来减少过拟合, 但是也不能太高, 会导致欠拟合, 取值范围为[0,∞]

$$Gain = \frac{1}{2} \left[\frac{G_L^2}{H_L + \lambda} + \frac{G_R^2}{H_R + \lambda} - \frac{(G_L + G_R)^2}{H_L + H_R + \lambda} \right] - \gamma$$


XGBoost工具

Booster参数:

- subsample [default=1], 构建每棵树对样本的采样率, 如果设置成0.5, XGBoost会随机选择50%的样本作为训练集
- colsample_bytree [default=1], 列采样率, 也就是特征采样率
- lambda [default=1, alias: reg_lambda], L2正则化, 用来控制XGBoost的正则化部分
- alpha [default=0, alias: reg_alpha], L1正则化, 增加该值会让模型更加收敛
- scale_pos_weight [default=1], 在类别高度不平衡的情况下, 将参数设置大于0, 可以加快收敛

XGBoost工具

学习目标参数:

- `objective [default=reg:linear]`, 定义学习目标, `reg:linear`, `reg:logistic`, `binary:logistic`, `binary:logitraw`, `count:poisson`, `multi:softmax`, `multi:softprob`, `rank:pairwise`
- `eval_metric`, 评价指标, 包括`rmse`, `logloss`, `error`, `merror`, `mlogloss`, `auc`, `ndcg`, `map`等
- `seed[default=0]`, 随机数的种子
- `dtrain`, 训练的数据
- `num_boost_round`, 提升迭代的次数, 也就是生成多少基模型
- `early_stopping_rounds`, 早停法迭代次数
- `evals`: 这是一个列表, 用于对训练过程中进行评估列表中的元素。形式是`evals = [(dtrain,'train'),(dval,'val')]`或者是`evals = [(dtrain,'train')]`, 对于第一种情况, 它使得我们可以在训练过程中观察验证集的效果
- `verbose_eval`, 如果为`True`, 则对`evals`中元素的评估输出在结果中; 如果输入数字, 比如5, 则每隔5个迭代输出一次
- `learning_rates`: 每一次提升的学习率的列表

XGBoost工具

天猫用户复购预测（XGBoost使用示意）

```
X_train, X_valid, y_train, y_valid = train_test_split(train_X, train_y, test_size=.2)
```

使用XGBoost

```
model = xgb.XGBClassifier(
```

```
    max_depth=8, #树的最大深度
```

```
    n_estimators=1000, #提升迭代的次数，也就是生成多少基模型
```

```
    min_child_weight=300, #一个子集的所有观察值的最小权重和
```

```
    colsample_bytree=0.8, #列采样率，也就是特征采样率
```

```
    subsample=0.8, #构建每棵树对样本的采样率
```

```
    eta=0.3, # eta通过缩减特征的权重使提升计算过程更加保守，防止过拟合
```

```
    seed=42 #随机数种子
```

```
)
```

```
model.fit(
```

```
    X_train, y_train,
```

```
    eval_metric='auc', eval_set=[(X_train, y_train), (X_valid, y_valid)],
```

```
    verbose=True,
```

```
    #早停法，如果auc在10epoch没有进步就stop
```

```
    early_stopping_rounds=10
```

```
)
```

```
model.fit(X_train, y_train)
```

```
prob = model.predict_proba(test_data)
```

XGBoost工具

Step3, 模型参数配置

```
param = {'boosting_type':'gbdt',  
        'objective' : 'binary:logistic', #任务目标  
        'eval_metric' : 'auc', #评估指标  
        'eta' : 0.01, #学习率  
        'max_depth' : 15, #树最大深度  
        'colsample_bytree':0.8, #设置在每次迭代中使用特征的比例  
        'subsample': 0.9, #样本采样比例  
        'subsample_freq': 8, #bagging的次数  
        'alpha': 0.6, #L1正则  
        'lambda': 0, #L2正则  
}
```

XGBoost工具

Step4, 模型训练, 得出预测结果

```
X_train, X_valid, y_train, y_valid = train_test_split(train.drop('Attrition',axis=1), train['Attrition'],
test_size=0.2, random_state=42)
```

```
train_data = xgb.DMatrix(X_train, label=y_train)
```

```
valid_data = xgb.DMatrix(X_valid, label=y_valid)
```

```
test_data = xgb.DMatrix(test)
```

```
model = xgb.train(param, train_data, evals=[(train_data, 'train'), (valid_data, 'valid')],
num_boost_round = 10000, early_stopping_rounds=200, verbose_eval=25)
```

```
predict = model.predict(test_data)
```

```
test['Attrition']=predict
```

```
# 转化为二分类输出
```

```
test['Attrition']=test['Attrition'].map(lambda x:1 if x>=0.5 else 0)
```

```
test[['Attrition']].to_csv('submit_lgb.csv')
```



user_id	Attrition
442	0
1091	0
981	0
785	0
1332	1
501	0
1058	0
1253	0
751	0
122	0
268	0
940	0
1125	0
540	1
1034	0
432	0
794	0
666	0
942	0
1114	0
853	1
1330	0
692	0
548	0
54	0
319	0
1147	0
1235	0
1436	1

LightGBM

LightGBM:

- 2017年经微软推出，XGBoost的升级版
- Kaggle竞赛使用最多的模型之一，必备机器学习神器
- Light => 在大规模数据集上运行效率更高
- GBM => Gradient Boosting Machine

LightGBM

Motivation:

- 常用的机器学习算法，例如神经网络等算法，都可以以mini-batch的方式训练，训练数据的大小不会受到内存限制
- GBDT在每一次迭代的时候，都需要遍历整个训练数据多次。如果把整个训练数据装进内存则会限制训练数据的大小；如果不装进内存，反复地读写训练数据又会消耗非常大的时间。对于工业级海量的数据，普通的GBDT算法是不能满足其需求的
- LightGBM的提出是为了解决GBDT在海量数据遇到的问题，让GBDT可以更好更快地用于工业场景

数据集	XGBoost	XGBoost_approx	LightGBM
Higgs	4.853GB	4.875GB	0.822GB
Yahoo LTR	1.907GB	2.221GB	0.831GB
MS LTR	5.469GB	5.600GB	0.745GB
Expo	1.553GB	1.560GB	0.450GB

内存消耗

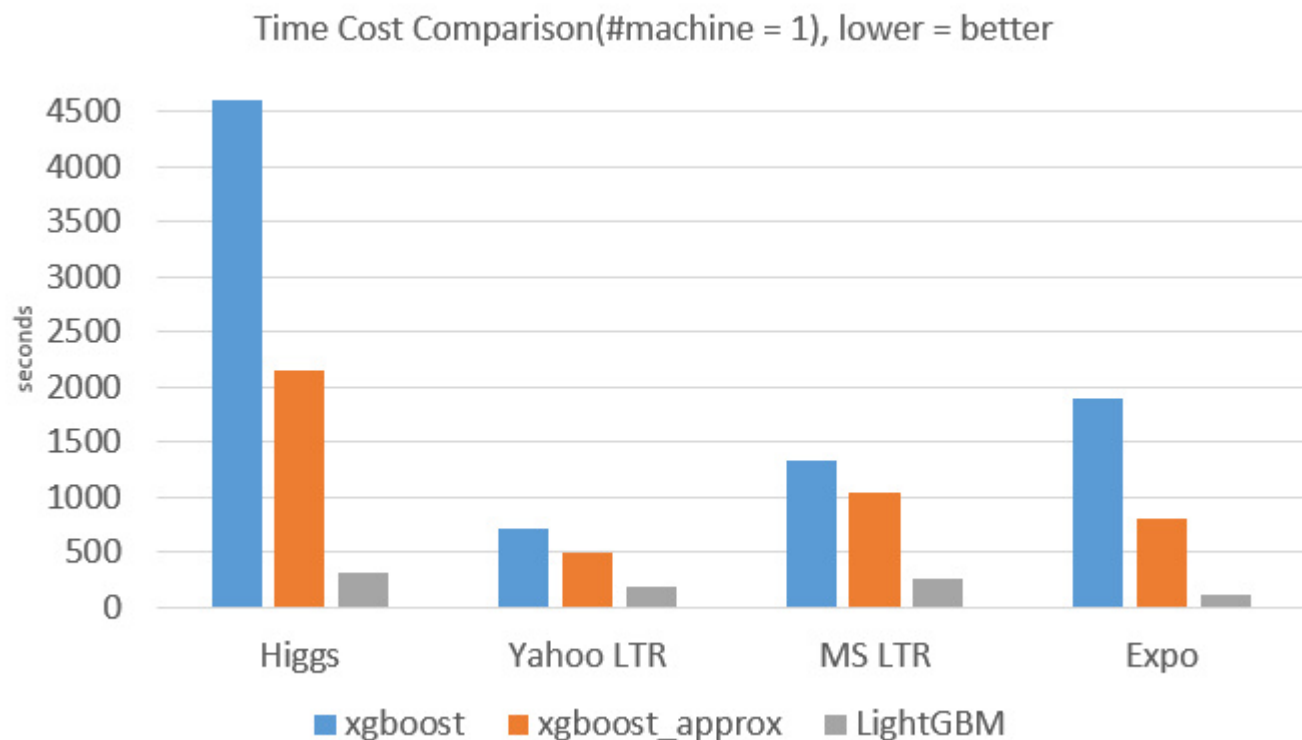
数据集	指标	XGBoost	XGBoost_approx	LightGBM
Higgs	AUC	0.839528	0.840533	0.845123
Yahoo LTR	NDCG@5	0.740316	0.739363	0.756369
MS LTR	NDCG@5	0.476245	0.474441	0.510153
Expo	AUC	0.75548	0.757071	0.781061

评估指标（AUC，NDCG）

LightGBM

LightGBM与XGBoost:

- 模型精度: 两个模型相当
- 训练速度: LightGBM训练速度更快 => 1/10
- 内存消耗: LightGBM占用内存更小 => 1/6
- 特征缺失值: 两个模型都可以自动处理特征缺失值
- 分类特征: XGBoost不支持类别特征, 需要对其进行OneHot编码, 而LightGBM支持分类特征



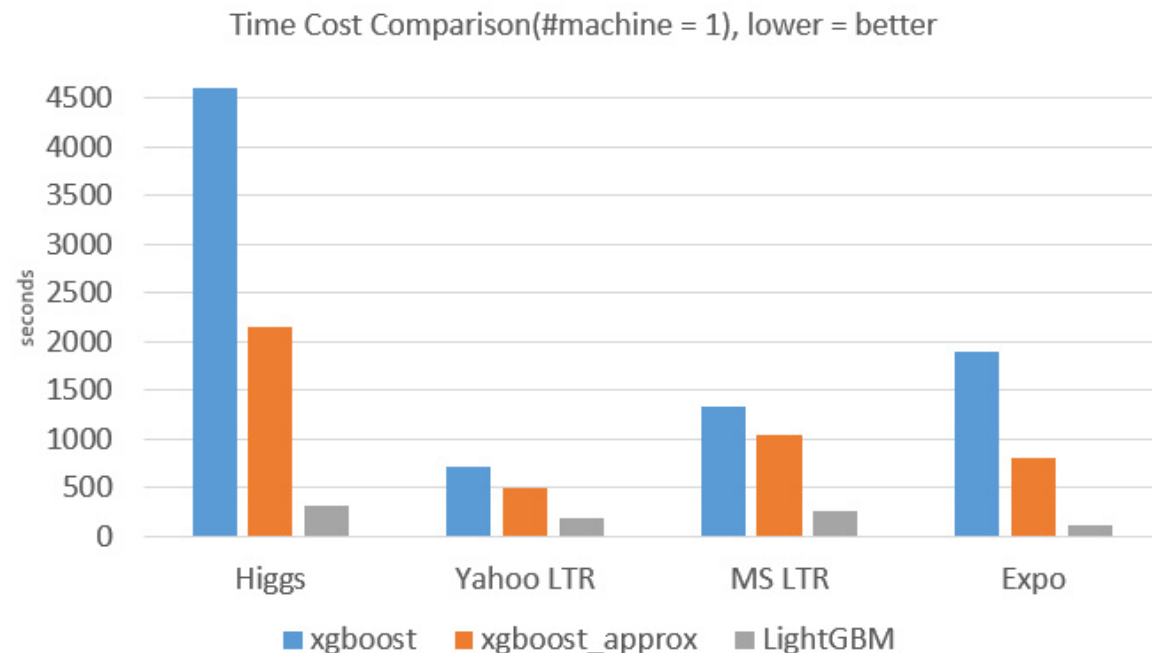
LightGBM

XGBoost模型的复杂度:

- 模型复杂度 = 树的棵数 \times 每棵树的叶子数量 \times 每片叶子生成复杂度
- 每片叶子生成复杂度 = 特征数量 \times 候选分裂点数量 \times 样本的数量

针对XGBoost的优化:

- Histogram算法, 直方图算法 => 减少候选分裂点数量
- GOSS算法, 基于梯度的单边采样算法 => 减少样本的数量
- EFB算法, 互斥特征捆绑算法 => 减少特征的数量
- LightGBM = XGBoost + Histogram + GOSS + EFB



LightGBM

XGBoost的预排序（pre-sorted）算法：

- 将样本按照特征取值排序，然后从全部特征取值中找到最优的分裂点位
- 预排序算法的候选分裂点数量=样本特征不同取值个数减1

i	1	2	3	4	5	6	7	8
x_i	0.1	2.1	2.5	3.0	3.0	4.0	4.5	5.0
g_i	0.01	0.03	0.06	0.05	0.04	0.7	0.6	0.07
h_i	0.2	0.04	0.05	0.02	0.08	0.02	0.03	0.03

$$Gain = \frac{1}{2} \left[\frac{G_L^2}{H_L + \lambda} + \frac{G_R^2}{H_R + \lambda} - \frac{(G_L + G_R)^2}{H_L + H_R + \lambda} \right] - \gamma$$

LightGBM

LightGBM的Histogram算法:

- 替代XGBoost的预排序算法
- 思想是先连续的浮点特征值离散化成k个整数，同时构造一个宽度为k的直方图，即将连续特征值离散化到k个bins上（比如k=255）
- 当遍历一次数据后，直方图累积了需要的统计量，然后根据直方图的离散值，遍历寻找最优的分割点
- XGBoost需要遍历所有离散化的值，LightGBM只要遍历k个直方图的值
- 候选分裂点数量 = k-1

	bin1			bin2			bin3	
i	1	2	3	4	5	6	7	8
x_i	0.1	2.1	2.5	3.0	3.0	4.0	4.5	5.0
g_i	0.01	0.03	0.06	0.05	0.04	0.7	0.6	0.07
h_i	0.2	0.04	0.05	0.02	0.08	0.02	0.03	0.03



$bin(i)$	bin1	bin2	bin3
N_i	3	3	2
G_i	0.1	0.79	0.67
H_i	0.29	0.12	0.06

LightGBM

GOSS算法:

- Gradient-based One-Side Sampling, 基于梯度的单边采样算法
- 思想是通过样本采样, 减少目标函数增益Gain的计算复杂度
- 单边采样, 只对梯度绝对值较小的样本按照一定比例进行采样, 而保留了梯度绝对值较大的样本
- 因为目标函数增益主要来自于梯度绝对值较大的样本 => GOSS算法在性能和精度之间进行了很好的trade off

i	1	2	3	4	5	6	7	8
x_i	0.1	2.1	2.5	3.0	3.0	4.0	4.5	5.0
g_i	0.01	0.03	0.06	0.05	0.04	0.7	0.6	0.07
h_i	0.2	0.04	0.05	0.02	0.08	0.02	0.03	0.03

gi: 梯度绝对值, 假设阈值为0.1
对gi<0.1的样本 => 1/3概率进行采样
对gi>=0.1的样本 => 全部保留

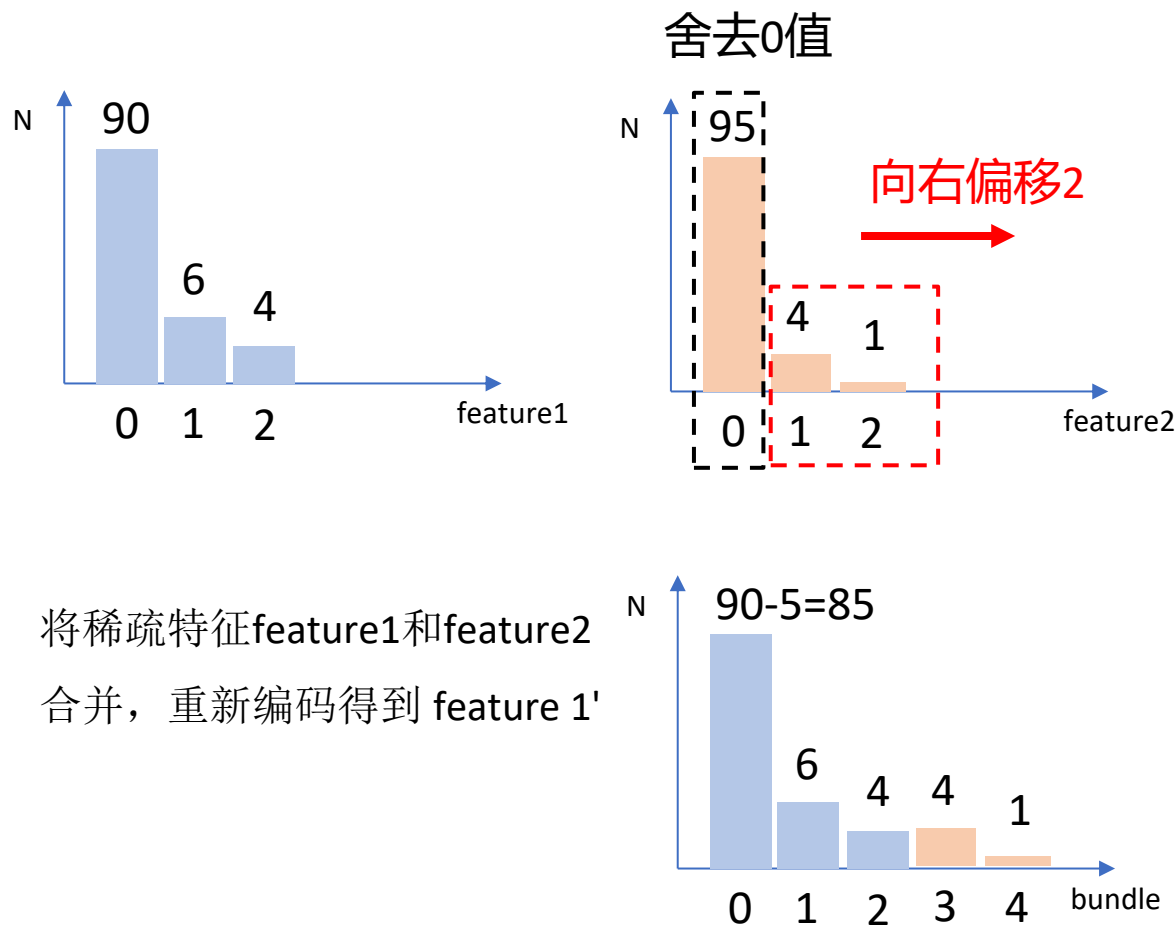


$bin(i)$	bin1	bin2	bin3
N_i	3=0+1*3	4=1+1*3	1=1+0*3
G_i	0.09	0.85	0.6
H_i	0.12	0.08	0.03

LightGBM

EFB算法:

- Exclusive Feature Bundling, 互斥特征绑定算法
- 思想是特征中包含大量稀疏特征的时候, 减少构建直方图的特征数量, 从而降低计算复杂度
- 数据集中通常会有大量的稀疏特征 (大部分为0, 少量为非0)
我们认为这些稀疏特征是互斥的, 即不会同时取非零值
- EFB算法可以通过对某些特征的取值重新编码, 将多个这样互斥的特征绑定为一个新的特征
- 类别特征可以转换成onehot编码, 这些多个特征的onehot编码是互斥的, 可以使用EFB将他们绑定为一个特征
- 在LightGBM中, 可以直接将每个类别取值和一个bin关联, 从而自动地处理它们, 也就无需预处理成onehot编码



LightGBM工具

LightGBM工具:

- `import lightgbm as lgb`
- 官方文档: <http://lightgbm.readthedocs.io/en/latest/Python-Intro.html>

参数:

- `boosting_type`, 训练方式, `gbdt`
- `objective`, 目标函数, 可以是`binary`, `regression`
- `metric`, 评估指标, 可以选择`auc`, `mae`, `mse`, `binary_logloss`, `multi_logloss`
- `max_depth`, 树的最大深度, 当模型过拟合时, 可以降低`max_depth`
- `min_data_in_leaf`, 叶子节点最小记录数, 默认20

Bagging参数: `bagging_fraction`+`bagging_freq` (需要同时设置)

- `bagging_fraction`, 每次迭代时用的数据比例, 用于加快训练速度和减小过拟合
- `bagging_freq`: `bagging`的次数。默认为0, 表示禁用`bagging`, 非零值表示执行k次`bagging`, 可以设置为3-5
- `feature_fraction`, 设置在每次迭代中使用特征的比例, 例如为0.8时, 意味着在每次迭代中随机选择80%的参数来建树
- `early_stopping_round`, 如果一次验证数据的一个度量在最近的round中没有提高, 模型将停止训练

LightGBM工具

参数:

- `lambda`, 正则化项, 范围为0~1
- `min_gain_to_split`, 描述分裂的最小 `gain`, 控制树的有用的分裂
- `max_cat_group`, 在 `group` 边界上找到分割点, 当类别数量很多时, 找分割点很容易过拟合时
- `num_boost_round`, 迭代次数, 通常 100+
- `num_leaves`, 默认 31
- `device`, 指定cpu 或者 gpu
- `max_bin`, 表示 feature 将存入的 bin 的最大数量
- `categorical_feature`, 如果 `categorical_features = 0,1,2`, 则列 0, 1, 2是 categorical 变量
- `ignore_column`, 与 `categorical_features` 类似, 只不过不是将特定的列视为categorical, 而是完全忽略

LightGBM工具

Step3, 模型参数配置

```
param = {'boosting_type':'gbdt',  
         'objective' : 'binary', #任务类型  
         'metric' : 'auc', #评估指标  
         'learning_rate' : 0.01, #学习率  
         'max_depth' : 15, #树的最大深度  
         'feature_fraction':0.8, #设置在每次迭代中使用特征的比例  
         'bagging_fraction': 0.9, #样本采样比例  
         'bagging_freq': 8, #bagging的次数  
         'lambda_l1': 0.6, #L1正则  
         'lambda_l2': 0, #L2正则  
}
```

LightGBM工具

Step4, 模型训练, 得出预测结果

```
X_train, X_valid, y_train, y_valid = train_test_split(train.drop('Attrition',axis=1), train['Attrition'],
test_size=0.2, random_state=42)
```

```
trn_data = lgb.Dataset(X_train, label=y_train)
```

```
val_data = lgb.Dataset(X_valid, label=y_valid)
```

```
model = lgb.train(param,train_data,valid_sets=[train_data,valid_data],num_boost_round =
10000 ,early_stopping_rounds=200,verbose_eval=25, categorical_feature=attr)
```

```
predict=model.predict(test)
```

```
test['Attrition']=predict
```

```
# 转化为二分类输出
```

```
test['Attrition']=test['Attrition'].map(lambda x:1 if x>=0.5 else 0)
```

```
test[['Attrition']].to_csv('submit_lgb.csv')
```



user_id	Attrition
442	0
1091	0
981	0
785	0
1332	1
501	0
1058	0
1253	0
751	0
122	0
268	0
940	0
1125	0
540	1
1034	0
432	0
794	0
666	0
942	0
1114	0
853	1
1330	0
692	0
548	0
54	0
319	0
1147	0
1235	0
1436	1

参数对比

xgb	lgb	XGBClassifier (xgb.sklearn)	LGBMClassifier (lgb.sklearn)
booster='gbtree'	boosting='gbdt'	booster='gbtree'	boosting_type='gbdt'
objective='binary:logistic'	application='binary'	objective='binary:logistic'	objective='binary'
max_depth=7	num_leaves=2**7	max_depth=7	num_leaves=2**7
eta=0.1	learning_rate=0.1	learning_rate=0.1	learning_rate=0.1
num_boost_round=10	num_boost_round=10	n_estimators=10	n_estimators=10
gamma=0	min_split_gain=0.0	gamma=0	min_split_gain=0.0
min_child_weight=5	min_child_weight=5	min_child_weight=5	min_child_weight=5
subsample=1	bagging_fraction=1	subsample=1.0	subsample=1.0
colsample_bytree=1.0	feature_fraction=1	colsample_bytree=1.0	colsample_bytree=1.0
alpha=0	lambda_l1=0	reg_alpha=0.0	reg_alpha=0.0
lambda=1	lambda_l2=0	reg_lambda=1	reg_lambda=0.0
scale_pos_weight=1	scale_pos_weight=1	scale_pos_weight=1	scale_pos_weight=1
seed	bagging_seed		
feature_fraction_seed	random_state=888	random_state=888	
nthread	num_threads	n_jobs=4	n_jobs=4
evals	valid_sets	eval_set	eval_set
eval_metric	metric	eval_metric	eval_metric
early_stopping_rounds	early_stopping_rounds	early_stopping_rounds	early_stopping_rounds
verbose_eval	verbose_eval	verbose	verbose

祖传参数 (LightGBM)

LGBMClassifier经验参数

```
clf = lgb.LGBMClassifier(
```

```
    num_leaves=2**5-1, reg_alpha=0.25, reg_lambda=0.25,
```

```
    objective='binary',
```

```
    max_depth=-1, learning_rate=0.005, min_child_samples=3,
```

```
    random_state=2021,
```

```
    n_estimators=2000, subsample=1, colsample_bytree=1,
```

```
)
```

num_leaves=2**5-1 #树的最大叶子数, 对比XGBoost一般为
2^(max_depth)

reg_alpha, L1正则化系数

reg_lambda, L2正则化系数

max_depth, 最大树的深度

n_estimators, 树的个数, 相当于训练的轮数

subsample, 训练样本采样率 (行采样)

colsample_bytree, 训练特征采样率 (列采样)

祖传参数 (XGBoost)

XGBoost VS LightGBM

XGBoost效果相对LightGBM可能会好一些

```
xgb = xgb.XGBClassifier(  
  
    max_depth=6, learning_rate=0.05, n_estimators=2000,  
  
    objective='binary:logistic', tree_method='gpu_hist',  
  
    subsample=0.8, colsample_bytree=0.8,  
  
    min_child_samples=3, eval_metric='auc', reg_lambda=0.5  
)
```

max_depth , 树的最大深度

learning_rate, 学习率

reg_lambda, L2正则化系数

n_estimators, 树的个数, 相当于训练的轮数

objective, 目标函数, binary:logistic 用于二分类任务

tree_method, 使用功能的树的构建方法, hist代表使用直方图优化的近似贪婪算法

subsample, 训练样本采样率 (行采样)

colsample_bytree, 训练特征采样率 (列采样)



subsample, colsample_bytree是个值得调参的参数, 典型的取值为0.5-0.9 (取0.7效果可能更好)

CatBoost

CatBoost算法:

- 俄罗斯科技公司Yandex开源的机器学习库（2017年）
- <https://arxiv.org/pdf/1706.09516.pdf>
- CatBoost = Catgorical + Boost
- 高效的处理分类特征（**categorical features**），首先对分类特征做统计，计算某个分类特征（**category**）出现的频率，然后加上超参数，生成新的数值型特征（**numerical features**）
- 同时使用组合类别特征，丰富了特征维度
- 采用的基模型是对称决策树，算法的参数少、支持分类变量，通过可以防止过拟合



CatBoost

CatBoost, LightGBM, XGBoost对比:

- 2015 年航班延误数据, 包含分类和数值变量
- <https://www.kaggle.com/usdot/flight-delays/data>
- 一共有约 500 条记录, 使用10%的数据, 即50条记录
- CatBoost 过拟合程度最小, 在测试集上准确度最高 0.816, 同时预测用时最短, 但这个表现仅仅在有分类特征, 而且调节了one-hot最大量时才会出现
- 如果不利用 CatBoost 算法在这些特征上的优势, 表现效果就会变成最差, AUC 0.752
- 使用CatBoost需要数据中包含分类变量, 同时适当地调节这些变量时, 才会表现不错

模型	XGBoost	LightGBM		CatBoost	
使用参数	max_depth:50 learning_rate:0.16 min_child_weight:1 n_estimators:200	max_depth:50 learning_rate:0.1 num_leaves:900 n_estimators:300		max_depth:10 learning_rate:0.15 l2_leaf_reg=9 iteration:500 one_hot_max_size=50	
训练集 AUC	0.999	没有使用分类特征索引	使用分类特征索引	没有使用分类特征索引	使用分类特征索引
		0.992	0.999	0.842	0.887
测试集 AUC	0.789	0.785	0.772	0.752	0.816
训练用时	970秒	153秒	326秒	180秒	390秒
预测用时	184秒	40秒	156秒	2秒	14秒

三种模型在flight-delays预测中的训练速度和准确度

CatBoost

CatBoost, LightGBM, XGBoost对比:

- 处理特征为分类的神器
- 支持即用的分类特征，因此我们不需要对分类特征进行预处理（比如使用LabelEncoding 或 OneHotEncoding）
- CatBoost 设计了一种算法验证改进，避免了过拟合。因此处理分类数据比LightGBM 和 XGBoost 强
- 准确性比 XGBoost 更高，同时训练时间更短
- 支持 GPU 训练
- 可以处理缺失的值

	Baseline XGBoost	Baseline LightGBM	Baseline CatBoost	Tuned XGBoost	Tuned LightGBM	Tuned CatBoost
Training Time	14m 08s	2m 33s	4m 39s	1h 2m 34s	32m 25s	23m 29s
Prediction Time	0.84s	0.4s	0.9s	1.2s	1.1s	0.9s
Accuracy Score	0.8484	0.8509	0.7851	0.8717	0.8716	0.8197

	Baseline XGBoost	Baseline LightGBM	Baseline CatBoost	Tuned XGBoost	Tuned LightGBM	Tuned CatBoost
Training Time	5.94s	1.71s	0.55s	9.84s	2.23s	4.46s
Prediction Time	0.012s	0.012s	0.048s	0.0155s	0.018s	0.030s
R2 Score	2.89	2.97	3.27	3.45	3.67	3.98

	Baseline XGBoost	Baseline LightGBM	Baseline CatBoost
Training Time	43s	17s	37s
Prediction Time	0.003s	0.007s	0.004s
R2 Score	12.43	12.06	7.87

实验1, 分类模型 MNIST识别 (6万数据, 784特征)
实验2, 回归模型, 预测纽约出租车票价 (6万数据, 7个特征)
实验3, 回归模型, 预测纽约出租车票价 (200万数据, 7个特征)

CatBoost工具

CatBoost工具:

- <https://github.com/dmlc/xgboost>
- https://catboost.ai/docs/concepts/python-reference_catboostclassifier.html

构造参数:

- `learning_rate`, 学习率
- `depth`, 树的深度
- `l2_leaf_reg`, L2正则化系数
- `n_estimators`, 树的最大数量, 即迭代次数
- `one_hot_max_size`, one-hot编码最大规模, 默认值根据数据和训练环境的不同而不同
- `loss_function`, 损失函数, 包括Logloss, RMSE, MAE, CrossEntropy, 回归任务默认RMSE, 分类任务默认Logloss
- `eval_metric`, 优化目标, 包括RMSE, Logloss, MAE, CrossEntropy, Recall, Precision, F1, Accuracy, AUC, R2

CatBoost工具

fit函数参数:

- X, 输入数据数据类型可以是: list; pandas.DataFrame; pandas.Series
- y=None
- cat_features=None, 用于处理分类特征
- sample_weight=None, 输入数据的样本权重
- logging_level=None, 控制是否输出日志信息, 或者其他信息
- plot=False, 训练过程中, 绘制, 度量值, 所用时间等
- eval_set=None, 验证集合, 数据类型list(X, y)tuples
- baseline=None
- use_best_model=None
- verbose=None

CatBoost工具

model = CatBoostClassifier(iterations=1000, #最大树数, 即迭代次数

depth = 6, #树的深度

learning_rate = 0.03, #学习率

custom_loss='AUC', #训练过程中, 用户自定义的损失函数

eval_metric='AUC', #过拟合检验（设置True）的评估指标, 用于优化

bagging_temperature=0.83, #贝叶斯bootstrap强度设置

rsm = 0.78, #随机子空间

od_type='Iter', #过拟合检查类型

od_wait=150, #使用Iter时, 表示达到指定次数后, 停止训练

metric_period = 400, #计算优化评估值的频率

l2_leaf_reg = 5, #l2正则参数

thread_count = 20, #并行线程数量

random_seed = 967 #随机数种子

)

CatBoost工具

Step3, 模型参数配置

```
model = cb.CatBoostClassifier(iterations=1000,  
                               depth=7,  
                               learning_rate=0.01,  
                               loss_function='Logloss',  
                               eval_metric='AUC',  
                               logging_level='Verbose',  
                               metric_period=50)
```

得到分类特征的列号

```
categorical_features_indices = []
```

```
for i in range(len(X_train.columns)):
```

```
    if X_train.columns.values[i] in attr:
```

```
        categorical_features_indices.append(i)
```

```
print(categorical_features_indices)
```

```
attr=['Age','BusinessTravel','Department','Education','EducationField','Gender','JobRole','MaritalStatus','Over18','OverTime']
```



```
[0, 1, 3, 5, 6, 9, 13, 15, 19, 20]
```

CatBoost工具

Step4, 模型训练, 得出预测结果

```
model.fit(X_train, y_train, eval_set=(X_valid, y_valid),  
cat_features=categorical_features_indices)
```

```
predict = model.predict(test)
```

```
test['Attrition']=predict
```

```
test[['Attrition']].to_csv('submit_cb.csv')
```



```
0:  test: 0.6390374 best: 0.6390374 (0) total: 80.6ms  remaining: 1m 20s  
50:  test: 0.7998472 best: 0.7998472 (50) total: 805ms  remaining: 15s  
100:  test: 0.8054131 best: 0.8054131 (100) total: 1.54s  remaining: 13.8s  
150:  test: 0.8053039 best: 0.8054131 (100) total: 2.51s  remaining: 14.1s  
200:  test: 0.8075958 best: 0.8075958 (200) total: 3.55s  remaining: 14.1s  
250:  test: 0.8062862 best: 0.8075958 (200) total: 4.55s  remaining: 13.6s  
300:  test: 0.8045400 best: 0.8075958 (200) total: 5.51s  remaining: 12.8s  
350:  test: 0.8059587 best: 0.8075958 (200) total: 6.45s  remaining: 11.9s  
400:  test: 0.8065044 best: 0.8075958 (200) total: 7.41s  remaining: 11.1s  
450:  test: 0.8065044 best: 0.8075958 (200) total: 8.44s  remaining: 10.3s  
500:  test: 0.8077049 best: 0.8077049 (500) total: 9.46s  remaining: 9.43s  
550:  test: 0.8090145 best: 0.8090145 (550) total: 10.6s  remaining: 8.6s  
600:  test: 0.8106515 best: 0.8106515 (600) total: 11.5s  remaining: 7.62s  
650:  test: 0.8113063 best: 0.8113063 (650) total: 12.4s  remaining: 6.67s  
700:  test: 0.8125068 best: 0.8125068 (700) total: 13.4s  remaining: 5.71s  
750:  test: 0.8126160 best: 0.8126160 (750) total: 14.4s  remaining: 4.77s  
800:  test: 0.8116337 best: 0.8126160 (750) total: 15.5s  remaining: 3.84s  
850:  test: 0.8121794 best: 0.8126160 (750) total: 16.5s  remaining: 2.89s  
900:  test: 0.8116337 best: 0.8126160 (750) total: 17.6s  remaining: 1.94s  
950:  test: 0.8111972 best: 0.8126160 (750) total: 18.7s  remaining: 965ms  
999:  test: 0.8108698 best: 0.8126160 (750) total: 19.8s  remaining: 0us  
  
bestTest = 0.8126159555  
bestIteration = 750  
  
Shrink model to first 751 iterations.
```

user_id	Attrition
442	0
1091	0
981	0
785	0
1332	1
501	0
1058	0
1253	0
751	0
122	0
268	0
940	0
1125	0
540	1
1034	0
432	0
794	0
666	0
942	0
1114	0
853	0
1330	0
692	0
548	0
54	0
319	0
1147	0
1235	0
1436	1

Summary

- LighGBM效率高，在Kaggle比赛中应用多
- CatBoost对于分类特征多的数据，可以高效的处理，过拟合程度小，效果好
- XGBoost, LightGBM, CatBoost参数较多，调参需要花大量时间
- Boosting集成学习包括AdaBoosting和Gradient Boosting
- Boosting只是集成学习中的一种（Bagging, Stacking）

打卡：二手车价格预测



针对AI大赛：二手车价格预测，编写AI算法，进行预测，挑战分数 < 500

<https://tianchi.aliyun.com/competition/entrance/231784/information>

训练集：used_car_train_20200313.csv

测试集：used_car_testB_20200421.csv

- XGBoost模型
- LightGBM模型
- CatBoost模型

你觉得哪个模型的模型效果好？

如何防止模型过拟合

Thinking: 如何防止模型过拟合?

- 1、模型层面优化
- 2、数据层面优化
- 3、业务逻辑约束

```
[19995] validation_0-mae:493.30894
[19996] validation_0-mae:493.30957
[19997] validation_0-mae:493.30842
[19998] validation_0-mae:493.30761
[19999] validation_0-mae:493.30777
模型 3 验证集 MAE: 493.31
```

7.2 集成模型预测

集成模型验证集 MAE: 492.49

8. 评估模型性能

学习赛第二季: 466/780.2130

学习赛第二季

- 日期: 2025-06-06 00:09:57
分数: 818.1239
- 日期: 2025-06-05 12:55:31
分数: 826.2079
- 日期: 2025-06-05 12:28:42
分数: 826.2079
- 日期: 2025-06-05 12:08:09
分数: 780.2130

如何防止模型过拟合

Thinking: 如何在模型层面进行优化？

- 正则化技术:

树模型: 调整`max_depth`(建议5-8)、`min_samples_leaf`(建议10-20, 代表每个叶子节点至少包含的样本数)

神经网络: 添加`dropout`层(0.2-0.5) + L2正则化

线性模型: 增大L1/L2正则化系数

- 早停机制:

监控验证集loss, 设置`patience`=10-20个epoch, `patience`代表允许验证集损失validation loss连续不改善的轮次(epochs)数量

- 集成方法:

使用Blending (用70%训练基模型, 30%训练元模型)

如何防止模型过拟合（树模型）

XGBoost中的超参数

```
params = {  
    'objective': 'reg:squarederror', # 目标函数，回归问题用平方误差  
    'eval_metric': 'mae',          # 评估指标，平均绝对误差  
    'learning_rate': 0.01,         # 学习率，控制每棵树对最终结果的影响，越小越保守  
    'max_depth': 6,                # 树的最大深度，防止过拟合  
    'subsample': 0.8,              # 每棵树随机采样的样本比例，防止过拟合  
    'colsample_bytree': 0.8,       # 每棵树随机采样的特征比例，防止过拟合  
    'seed': 42,                   # 随机种子，保证结果可复现  
    'nthread': -1                 # 使用全部CPU线程加速训练  
}
```

如何防止模型过拟合（树模型）

在sklearn决策树中，可以使用 `min_samples_leaf` 参数

```
from sklearn.ensemble import RandomForestRegressor
# 尝试不同值观察效果
model = RandomForestRegressor(
    min_samples_leaf=10, # 初始建议值
    n_estimators=100,
    random_state=42
)
model.fit(X_train, y_train)
```

XGBoost 没有 `min_samples_leaf` 这个参数，在 XGBoost 中，类似的参数是：`min_child_weight`，即一个叶子节点上最小的样本权重和（对于回归问题就是样本个数的和）。如果一个叶子节点的样本权重和小于这个值，则不会再分裂。

如何防止模型过拟合（神经网络）

如果使用神经网络，可以增加 `layers.Dropout` 层

```
from tensorflow import keras
from tensorflow.keras import layers
model = keras.Sequential([
    layers.Dense(128, activation='relu', input_shape=(X_train.shape[1],)),
    layers.Dropout(0.2),
    layers.Dense(64, activation='relu'),
    layers.Dropout(0.1),
    layers.Dense(1)
])
```

神经网络是一种受人脑神经元结构启发的机器学习模型。核心思想是通过大量“神经元”节点的层层连接和非线性变换，自动学习输入特征与输出目标之间的复杂映射关系。

如何防止模型过拟合（线性模型）

如果使用线性模型，比如逻辑回归，可以使用L1或L2正则化系数

```
model = LogisticRegression(  
    max_iter=100,  
    verbose=True,  
    random_state=42,  
    tol=1e-4,  
    penalty='l2', # 或 'l1'  
    C=1.0,      # 正则化强度倒数，越小正则化越强  
)
```

L1 正则化项是模型权重的绝对值之和：

$$L1 = \lambda \sum_{i=1}^n |w_i|$$

L2 正则化项是模型权重的平方和

$$L2 = \lambda \sum_{i=1}^n w_i^2$$

如何防止模型过拟合（早停机制）

使用早停机制，可以让模型初始化训练次数更多一些

```
model = xgb.train(  
    params,  
    dtrain,  
    num_boost_round=2000,  
    evals=evals,  
    early_stopping_rounds=20,  
    verbose_eval=100  
)
```

```
early_stop = keras.callbacks.EarlyStopping(monitor='val_loss',  
    patience=20, restore_best_weights=True)  
history = model.fit(  
    X_train_scaled, y_train,  
    validation_data=(X_val_scaled, y_val),  
    epochs=200,  
    batch_size=64,  
    callbacks=[early_stop],  
    verbose=2  
)
```

早停是一种正则化技术，用于防止模型在训练过程中过拟合。核心思想是：在验证集性能不再提升时提前终止训练，而不是一直训练到收敛。

Blending集成学习方法

Thinking: Blending 是什么？

Blending 是一种分层集成学习技术，通过以下两步组合多个模型：

- 基模型（Base Models）：用训练集的70%训练多个不同模型（如随机森林、XGBoost、神经网络等）。
- 元模型（Meta Model）：用剩下的30%数据生成基模型的预测结果作为新特征，训练一个次级模型（通常为线性模型）进行最终预测。

神经网络是一种受人脑神经元结构启发的机器学习模型。核心思想是通过大量“神经元”节点的层层连接和非线性变换，自动学习输入特征与输出目标之间的复杂映射关系。

nn_pred.py

Blending集成学习方法

假设数据集有 15,000条 二手车记录，特征包括车龄、里程、品牌等，目标为价格。

```
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
from sklearn.linear_model import LinearRegression

# 原始数据
X, y = load_car_data() # 假设已加载数据
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.3, random_state=42)

# 进一步划分训练集： 70%基模型 / 30%元模型
X_base, X_meta, y_base, y_meta = train_test_split(
    X_train, y_train, test_size=0.3, random_state=42
)
```

Blending集成学习方法

Step1: 训练基模型

```
# 定义3个基模型
model_rf = RandomForestRegressor(n_estimators=100, random_state=42)
model_gb = GradientBoostingRegressor(n_estimators=100, random_state=42)
model_nn = make_neural_network() # 自定义的神经网络

# 在70%数据上训练基模型
model_rf.fit(X_base, y_base)
model_gb.fit(X_base, y_base)
model_nn.fit(X_base, y_base)
```

Blending集成学习方法

Step2: 生成元特征

用基模型预测剩余30%数据，生成新特征

```
# 获取基模型对元数据集的预测
meta_features = np.column_stack([
    model_rf.predict(X_meta),
    model_gb.predict(X_meta),
    model_nn.predict(X_meta)
])
```

```
# 元特征示例（每条样本的3个基模型预测值）
print(meta_features[:3])
```

输出类似：

[[12.5, 13.1, 11.8],

[8.2, 7.9, 8.5],

[20.1, 19.7, 21.3]]

Blending集成学习方法

Step3: 训练元模型

```
# 用基模型的预测结果作为输入，真实价格作为目标
meta_model = LinearRegression()
meta_model.fit(meta_features, y_meta)
```

通过Blending学习，可以得到三个模型（rf, gb, nn）的线性回归系数，比如为：

权重系数 = [0.4, 0.5, 0.1], 偏置 = 0.2

最终预测 = $w_1 \times \text{模型1预测} + w_2 \times \text{模型2预测} + w_3 \times \text{模型3预测} + b$

Step4: 预测新数据

```
# 对验证集生成基模型预测
val_meta_features = np.column_stack([
    model_rf.predict(X_val),
    model_gb.predict(X_val),
    model_nn.predict(X_val)
])
# 用元模型做最终预测
final_predictions = meta_model.predict(val_meta_features)

# 计算MAE
print("Blending MAE:", mean_absolute_error(y_val, final_predictions))
```

Summary

过拟合（**Overfitting**）是导致模型在实际应用中表现糟糕的主要原因之一。

在二手车价格预测这类数据噪声大、特征复杂的任务中，过拟合会直接影响模型的商业价值和可靠性。

训练MAE = 400（模型似乎很准），测试MAE = 800（实际预测误差翻倍）=> 典型的过拟合

方法	作用
早停（Early Stopping）	监控验证集MAE， patience=15轮无改善则停止训练
Dropout	在神经网络中随机丢弃20%神经元
L1/L2正则化	线性回归添加 L1或L2正则化
集成方法	使用Blending组合随机森林、XGBoost和神经网络



Thank You
Using data to solve problems

