流程速览：

1. 逛一下Kaggle网站 （10min）

- 2min : 逛主界面
- 4min ：浏览今天题目：Home credit default risk
- 4min： -----**讨论**-----

2. A quick simple solution （10min）

- 3min : 看代码
- 5min：主持人统一过一遍

- 2min： -----**讨论**-----

3. 从逻辑回归→回归树→boost家族（10min）

- 2min： 逻辑回归的 ***极大似然估计***
- 2min： 用决策树来处理***似然函数***
- 4min： 基于加法模型的Boost系列
- 2min： -----**讨论**-----

4. 特征工程与featuretools （10min）

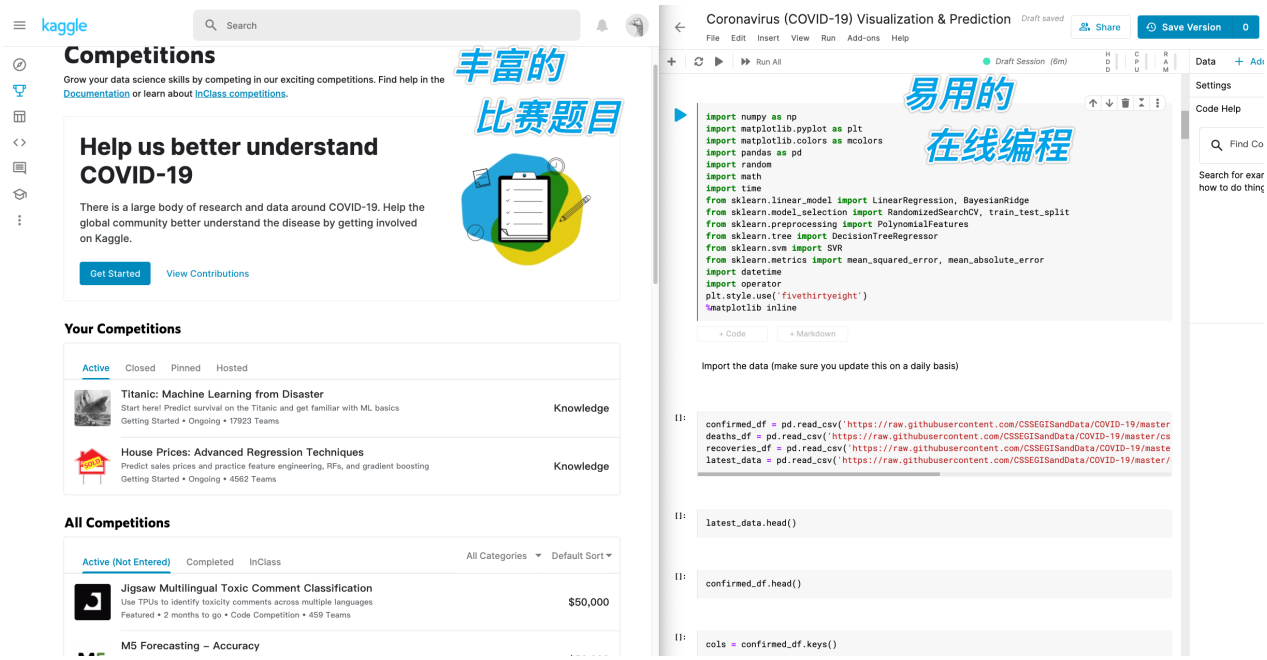
- 2min： Featuretools
- 3min： 自编码器
- 2min： “业务含义”造变量
- 3min： -----**讨论**-----

5. “剑走偏锋”还是“旁门左道”（10min）

- 4min： 主流高分方案： **stacking** & **blending**
- 6min： “**剑走偏锋**”高分方案

6. 讨论

一、Kaggle是个啥?



1.1 Kaggle：

有着丰富的**数据挖掘**比赛题目，大量公开的**代码方案**

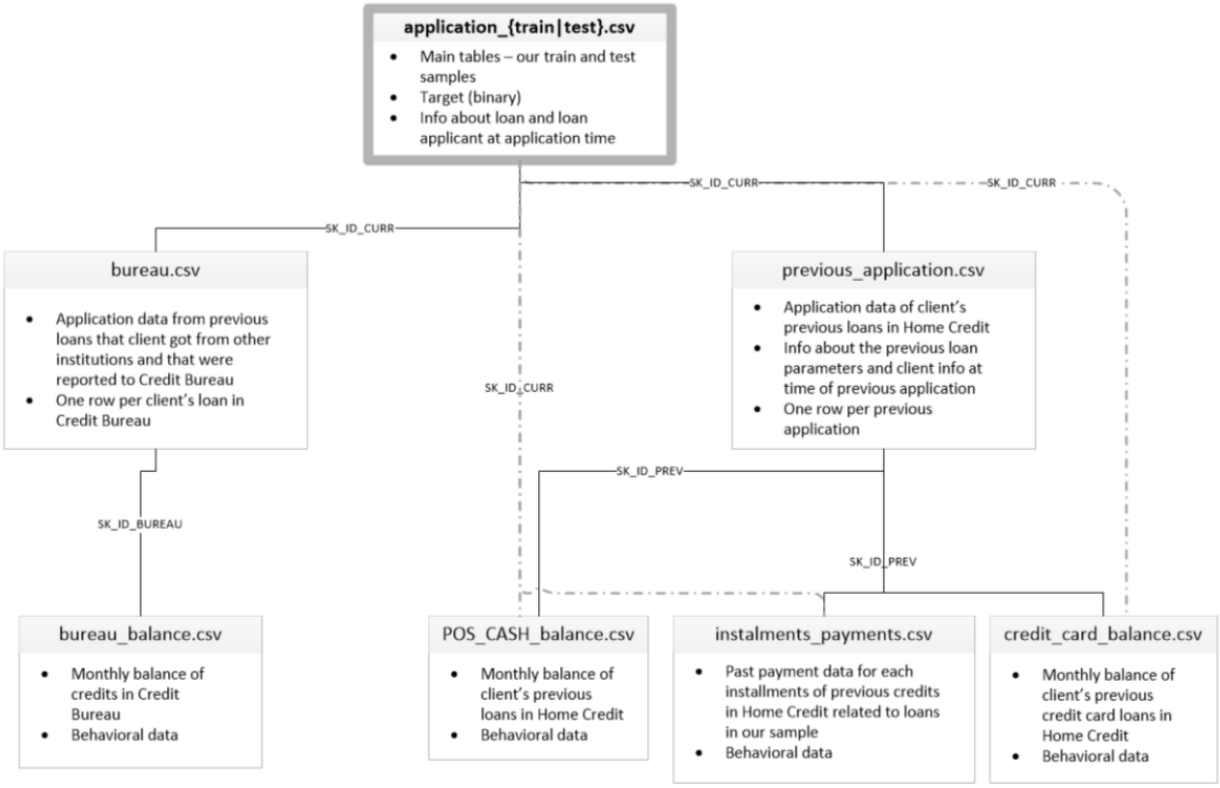
1.2 今天的topic：Home Credit Default Risk

(1) 业务背景（目标是什么）

希望能通过数据挖掘和机器学习算法来估计客户的贷款违约概率PD

(2) 数据（哪些信息摸得到）

- **客户申请表** application_train/test
- **客户信用记录历史**(月数据) bureau/bureau_balance
- **客户账户余额&贷款历史**(月数据) POS_CASH_balance
- **客户信用卡信息表** (月数据)credit_card_balance
- **客户历史申请表**previous_application
- **客户先前信用卡还款记录** installments_payments



(3) 结果评判标准

测试数据集上的**AUC**

二、快速上手：A simple & quick solution

我们来看看：一个最简单的**逻辑回归**建模，要经历哪些**典型步骤**

2.1 利用pandas.read_csv()读取数据

```
app_train = pd.read_csv('../input/application_train.csv')
app_test = pd.read_csv('../input/application_test.csv')
bureau = pd.read_csv('../input/bureau.csv')
bureau_balance = pd.read_csv('../input/bureau_balance.csv')
pos_cash_balance = pd.read_csv('../input/POS_CASH_balance.csv')
previous_app = pd.read_csv('../input/previous_application.csv')
installments_payments = pd.read_csv('../input/installments_payments.csv')
credit_card_balance = pd.read_csv('../input/credit_card_balance.csv')
```

2.2 数据处理

(1) 衍生变量：统计客户的历史贷款次数

```
previous_loan_counts = bureau.groupby('SK_ID_CURR', as_index=False)
['SK_ID_BUREAU'].count().rename(columns = {'SK_ID_BUREAU':
'previous_loan_counts'})
```

客户编号 SK_ID_CURR	Previous_loan_counts
1000001	7
1000002	8
1000003	4

(2) 继续对多张表按照客户编号SK_ID_CURR进行groupby

每个客户对应一行特征，这样依据*客户编号sk_id_curr*做横向merge简单好理解。

```
# Grouping data  so  that we can merge all the files in 1 dataset
data_bureau_agg=bureau.groupby(by='SK_ID_CURR').mean()
data_credit_card_balance_agg=credit_card_balance.groupby(by='SK_ID_CURR').mean()
data_previous_application_agg=previous_app.groupby(by='SK_ID_CURR').mean()
data_installments_payments_agg=installments_payments.groupby(by='SK_ID_CURR').mean()
data_POS_CASH_balance_agg=pos_cash_balance.groupby(by='SK_ID_CURR').mean()

data_bureau_agg.head()
```

Out[7]:

	SK_ID_BUREAU	DAYS_CREDIT	CREDIT_DAY_OVERDUE	DAYS_CREDIT_ENDDATE	DAYS_ENDDATE_FACT	AMT_CRE
SK_ID_CURR						
100001	5896633.000	-735.000000	0.0	82.428571	-825.500000	NaN
100002	6153272.125	-874.000000	0.0	-349.000000	-697.500000	1681.02
100003	5885878.500	-1400.750000	0.0	-544.500000	-1097.333333	0.000
100004	6829133.500	-867.000000	0.0	-488.500000	-532.500000	0.000
100005	6735201.000	-190.666667	0.0	439.333333	-123.000000	0.000

(3) 做left join

```
def merge(df):
    df = df.join(data_bureau_agg, how='left', on='SK_ID_CURR', lsuffix='1', rsuffix='2')
    df = df.join(bureau_counts, on = 'SK_ID_CURR', how = 'left')
    df = df.join(previous_loan_counts, on = 'SK_ID_CURR', how = 'left')
    df = df.join(data_credit_card_balance_agg, how='left', on='SK_ID_CURR', lsuffix='1', rsuffix='2')
    df = df.join(data_previous_application_agg, how='left', on='SK_ID_CURR', lsuffix='1', rsuffix='2')   #这里的suffix是当两张表出现同样的列名时，对left table、right table增加不同的后缀
    df = df.join(data_installments_payments_agg, how='left', on='SK_ID_CURR', lsuffix='1', rsuffix='2')
    return df

train = merge(app_train)
test = merge(app_test)
display(train.head())
```

	SK_ID_CURR	TARGET	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN
0	100002	1	Cash loans	M	N	Y	0
1	100003	0	Cash loans	F	N	N	0
2	100004	0	Revolving loans	M	Y	Y	0
3	100006	0	Cash loans	F	N	Y	0
4	100007	0	Cash loans	M	N	Y	0

(4) 变量取值预处理

负的天数调整为**正数**，缺失值**fillna()**

```
# Now we will convert days employed and days registration and days id publish to a positive no.
def correct_birth(df): #负号并转换成年份
    df['DAYS_BIRTH'] = round((df['DAYS_BIRTH'] * (-1))/365)
    return df
def convert_abs(df): #取绝对值
    df['DAYS_EMPLOYED'] = abs(df['DAYS_EMPLOYED'])
    df['DAYS_REGISTRATION'] = abs(df['DAYS_REGISTRATION'])
    df['DAYS_ID_PUBLISH'] = abs(df['DAYS_ID_PUBLISH'])
    df['DAYS_LAST_PHONE_CHANGE'] = abs(df['DAYS_LAST_PHONE_CHANGE'])
    return df
# Now we will fill misisng values in OWN_CAR_AGE.
#Most probably there will be missing values if the person does not own a car. So we will fill with 0
def missing(df):  #填充缺失值
```

```
    features = ['previous_loan_counts','NONLIVINGAPARTMENTS_MEDI',
'NONLIVINGAPARTMENTS_AVG','NONLIVINGAREA_MEDI','OWN_CAR_AGE']
    for f in features:
        df[f] = df[f].fillna(0 )
    return df
def transform_app(df):
    df = correct_birth(df)
    df = convert_abs(df)
    df = missing(df)
    return df



all_data = transform_app(all_data)
```

数值变量进行值域放缩**MinMaxScaler()**

```
from sklearn.preprocessing import MinMaxScaler
def encoder(df):
    scaler = MinMaxScaler()
    numerical = all_data.select_dtypes(exclude = ["object"]).columns
    features_transform = pd.DataFrame(data= df)
    features_transform[numerical] = scaler.fit_transform(df[numerical])
    display(features_transform.head(n = 5))
    return df

all_data = encoder(all_data)
```

类别型变量进行**LabelEncoder()**、**get_dummies()**

```
from sklearn.preprocessing import LabelEncoder
all_data[col] = LabelEncoder().fit_transform(all_data[col])
all_data = pd.get_dummies(all_data)
```

2.3 训练模型：逻辑回归

逻辑回归：LogisticRegression()

```
from sklearn.linear_model import LogisticRegression
logreg = LogisticRegression(random_state=0, class_weight='balanced', C=100)
logreg.fit(X_train, Y_train)
Y_pred = logreg.predict_proba(X_test)[:,1]
```

这是一个验证集AUC=0.75的方案，过程比较基础，请问：

`LogisticRegression(random_state=0, class_weight='balanced', C=100)`里面的参数设置起到什么作用呢?

三、从逻辑回归 → XGBoost

3.1 逻辑回归Logistic Regression

$\mathrm{In}\frac{PD}{1-PD} = w^T * x + b$

$\mathrm{In}\frac{p(y=1|x)}{p(y=0|x)} = w^T * x + b$

那么*w*和*b*是怎么求呢，和**OLS**一样吗?

- 直接用MSE作为损失函数是非凸的!
- 引入**极大似然估计**：***已知结果，去反推最大概率导致该结果的参数***

$l(w,b) = \prod_{i=1}^{m} p(y_i|x_i;w,b)$ ---------- **每个样本的取值概率相乘**

LogLoss（对数损失函数）：

$LogLoss = L(w,b) = \mathrm{In}\ l(w,b) = \sum_{i=1}^{m} \mathrm{In}\ p(y_i|x_i;w,b)$ ------- **对数运算In：连乘$\prod$→累加$\sum$**

$p(y_i|x_i;w,b) = y_i \frac{e^{w^T x+b}}{1+e^{w^T x+b}} + (1-y_i)\frac{1}{1+e^{w^T x+b}}$

找出$L(w,\ b)$似然函数最大时所对应的w, b。

基本上都是基于**一阶梯度、二阶梯度**迭代寻找w, b；

回到`sklearn.linear_model.LogisticRegression()`的参数上：

> **solver**：优化算法选择
>
> - *liblinear*：使用了开源的liblinear库实现，内部使用了坐标轴下降法来迭代优化损失函数。
> - *lbfgs*：拟牛顿法的一种，利用损失函数二阶导数矩阵即海森矩阵来迭代优化损失函数。
> - *newton-cg*：也是牛顿法家族的一种，利用损失函数二阶导数矩阵即海森矩阵来迭代优化损失函数。只用于L2
> - *sag*：即随机平均梯度下降，是梯度下降法的变种，和普通梯度下降法的区别是每次迭代仅仅用一部分的样本来计算梯度，适合于样本数据多的时候。只用于L2
> - *saga*：线性收敛的随机优化算法的的变重。只用于L2
>
> **penalty**：**惩罚项**
> str类型，默认为l2。newton-cg、sag和lbfgs求解算法只支持L2规范,L2假设的模型参数满足高斯分布。
> l1:L1G规范假设的是模型的参数满足拉普拉斯分布.
> **tol**：**停止求解的标准**，float类型，默认为1e-4。就是求解到多少的时候，停止，认为已经求出最优解。
> **c**：**正则化系数λ的倒数**，float类型，默认为1.0。必须是正浮点型数。像SVM一样，越小的数值表示越强的正则化。
> **fit_intercept**：是否存在截距或偏差，bool类型，默认为True。
> **class_ weight**：

> 用于标示分类模型中**各种类型的权重**，可以是一个字典或者'balanced'字符串，默认为不输入，也就是不考 虑权重，即为None。
> 如果选择输入的话，可以选择balanced让类库自己计算类型权重，或者自己输入各个类型的权重。
>
> **random_state**：随机数种子，int类型，可选参数，默认为无，仅在正则化优化算法为sag,liblinear时有用。
>
> **max_iter**：算法收敛最大迭代次数，int类型，默认为10。仅在正则化优化算法为newton-cg, sag和lbfgs才 有用，算法收敛的最大迭代次数。
> **multi_class**：分类方式选择参数，str类型，可选参数为ovr和multinomial，默认为ovr。ovr即前面提到的 one-vs-rest(OvR)，而multinomial即前面提到的many-vs-many(MvM)，区别主要在多元逻辑回归上。

3.2 Boosting策略

我们总是能听到Catboost、XGBoost、LightGBM、GBDT这类：

以**决策树**模型为基学习器，还带着个**Boost**为形容词的高大上算法，我们快速了解一下吧！

(1) CART (Classification & Regression Tree)：一个能回归也能分类的树模型

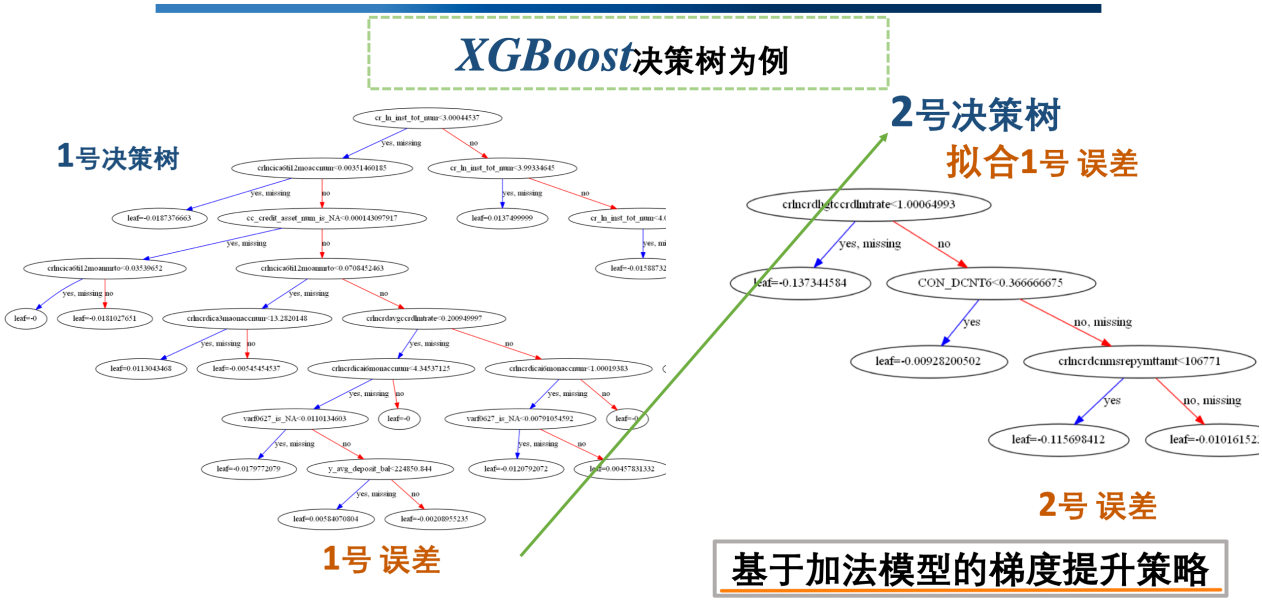
CART作为Boosting策略使用最广泛的基学习器：

- 用于**回归** *regression*：最小化MSE（均方误差）
- 用于**分类** *classification*：最小化GINI系数

(2) 作用在梯度上的Boosting （Gradient Boosting）

后一个基学习器 → *拟合*→ 前一个基学习器的残差

这里的残差在工程上一般定义为：损失函数的负梯度



（3）瞅一瞅XGBoost

1. 目标函数：***LogLoss***(对数损失函数) + ***Regularization***(正则项)

$$obj(\theta) = \sum [y_i \mathrm{In}(1 + e^{-\hat{y}_i}) + (1 - y_i)\mathrm{In}(1 + e^{\hat{y}_i})] + \sum \Omega(f_k)$$

2. 部分参数一览

```
import xgboost as xgb
xgb.XGBRegressor({'n_estimators': 500,'eta': 0.3,  'gamma': 0,
'max_depth': 6,\
                  'reg_lambda': 1, 'reg_alpha': 0,'seed': 33})
#采样、叶子节点细节的相关参数就暂时忽略吧：
#'min_child_weight': 1, 'colsample_bytree': 1, 'colsample_bylevel': 1,
'subsample': 1
```

n_estimators：弱学习器的数量

max_depth：默认是6，树的最大深度，值越大，越容易过拟合；[0，∞]

eta ： 默认是0.3，别名是 **leanring_rate**，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；[0,1]

seed：随机数种子，相同的种子可以复现随机结果，用于调参！

----------以下是**正则项**参数----------

为大家贴心的准备了**正则项**公式：

$$\Omega(f) = \sum[\gamma T + \frac{1}{2}\lambda\|w\|_2 + \alpha\|w\|_1]$$

gamma：默认是0，别名是 min_split_loss，gamma值越大，算法越保守（越不容易过拟合）；[0，∞]

lambda：默认是1，别名是reg_lambda，L2 正则化项的权重系数，越大模型越保守；

alpha：默认是0，别名是reg_alpha，L1 正则化项的权重系数，越大模型越保守；

----------**以下可以不看**----------

min_child_weight：默认是1，决定最小叶子节点样本权重和，加权和低于这个值时，就不再分裂产生新的叶子节点。当它的值较大时，可以避免模型学习到局部的特殊样本。但如果这个值过高，会导致欠拟合。[0，∞]
max_delta_step：默认是0，这参数限制每颗树权重改变的最大步长。如果是 0 意味着没有约束。如果是正值那么这个算法会更保守，通常不需要设置。[0，∞]
subsample：默认是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。但是这个值设置的过小，它可能会导致欠拟合。 (0,1]
colsample_bytree：默认是1，用来控制每颗树随机采样的列数的占比； (0,1]
colsample_bylevel：默认是1，用来控制的每一级的每一次分裂，对列数的采样的占比； (0,1]