### dataframe, series, list, array

* **dataframe**

row index+ column name + value

多个series组成，接受dict类型数据，如果不指定，dict中的key默认为column name。

* **ndarray**

多维数组,数据类型相同，和list差不多，list内的数据类型可不同

* **series**

是一个字典，row index+value

#list to series,或直接接受dict

myseries = Series(mylist, index = ['one', 'two'])

# list to dataframe或直接接受dict

mydataframe = DataFrame(mylist, index = ['one', 'two'], columns = ['year', 'state', 'pop'])

# dataframe to array

ndarray = mydataframe.values #移除row index and column names

# list to array

ndarray = np.array(mylist)

# dataframe to list, 或直接接受dict

# dataframe to array

arr = mydataframe.values

### Import

import numpy as np

import pandas as pd

import itertools #迭代循环

import cPickle #使用pickle模块你可以把Python对象直接保存到文件，而不需要把他们转化为字符串

|  |
| --- |
| **For plotting** |
| import seaborn as sns # need install, pip install seaborn  import matplotlib.pyplot as plt  plt.style.use("ggplot") #无需自定义图中网格等，自己会美化 |

|  |
| --- |
| **For AB test** |
| from scipy.stats import ttest\_ind |

|  |
| --- |
| **For algorithm** |
| import lightgbm # pip install lightgbm  from sklearn import tree  from sklearn.tree import DecisionTreeClassifier  import graphviz # decision tree visualization  from sklearn.cluster import KMeans  from sklearn.metrics import silhouette\_score #评价kmeans性能，越接近1，聚类越好  from sklearn.decomposition import PCA |

|  |
| --- |
| **For data splitting** |
| from **sklearn**.model\_selection import train\_test\_split |

|  |
| --- |
| **For evaluation metrics** |
| from **sklearn**.metrics import auc, roc\_curve, classification\_report, precision\_score, recall\_score,  accuracy\_score, roc\_auc\_score |

|  |
| --- |
| **For feature engineering** |
| from sklearn.preprocessing import LabelEncoder #feature encoding  from sklearn.preprocessing import normalize # feature normalization  from sklearn.preprocessing import StandardScaler # feature normalization  from sklearn.feature\_selection import chi2,f\_classif #计算特征间的线性相似度 |

|  |
| --- |
| **For tuning parameters** |
| from sklearn.model\_selection import GridSearchCV  from sklearn.model\_selection import RandomizedSearchCV |

### Data loading

#### 3.1从csv读入数据

将c1列和c2列解析为时间数据，如将20200102解析为2020-01-02，返回所有原始数据和解析后的列替换原来的列

data = pd.read\_csv(r'C:\Users\juanchen\Desktop\120 jifu zhao\data\loan\_table.csv', parse\_dates=['c1', 'c2'])

data=

#### 3.2从a.txt读入数据，数据间，逗号

如果输入文档名称为a.txt,文档内容如下

1, 2

3, 6

data=pd.read\_csv('a.txt',sep=',')

data.columns=['feature1', 'target']

X=data[['feature1']] # data[['feature1']].values ?

y=data[['target']]

### Data overview

#### 3.1 list top 10 rows of a data frame

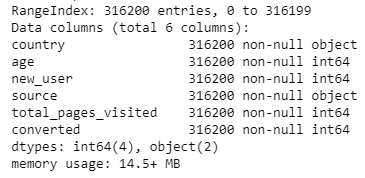
data.head(10)

#### 3.1 list last 10 rows of a data frame

data.tail(10)

#### 3.2 Views the number of rows and columns, the memory usage, the data type of each column.

data.info()



#### 3.3 View the count, mean, max, min, 25% percentile, 50% percentile, std etc. of each column

def f1(mylist): #mylist=[1,2,5,11,232]

mylist \_25 = np.percentile(mylist, 25)

mylist \_50 = np.percentile(mylist, 50)

mylist \_75 = np.percentile(mylist, 75)

mylist \_max = np.max(mylist)

mylist\_mean = np.mean(mylist)

mylist \_min = np.min(mylist)

return pd.Series([mylist \_mean, mylist\_min, mylist\_25, mylist\_50, mylist\_75, mylist\_max],

index=['mean', 'min', '25%', '50%', '75%', 'max']) #rename

stats = data.groupby('c1').apply(f1).reset\_index()

stats.head()

#### 3.4 Merge two tables

按照列c1进行连接，左连接，连接两张表：data和data1返回merge后的表,如果data和data1合并时，有两列重名，data该列名字加后缀\_l, data1该列名字加后缀\_r。

pd.merge(left=data, right=data1, how='left', on='c1', suffixes=['\_l', '\_r'])

### Data cleaning

* View part of data ‘age’<100

data[data['age'] <100]

#### 4.1 missing data

* **Find missing data**
* 统计每列null 值的数量

返回两列，第一列为索引列，第二列为统计的有空值的特征的空值的数量。第一列的值为有空值的列名。

data.isnull().sum()

增加索引列0,1,2…第一列列名为index,不再是索引，给第二列重命名为c1

pd.DataFrame(data.isnull().sum()).rename(columns={0: 'c1'} .reset\_index()

* 选出c1列，条件为c1列值为空

data['c1'].isnull()

* 选出行，条件为c1列值为空

data[data['c1'].isnull()]

* 选出c1列，条件为c1列值非空

~data['c1'].isnull()

* 选出行，条件为c1列值非空

data[~data['c1'].isnull()]

* 计算非空c1列的值占总数的比率

1 – data['c1'].isnull().sum() / len(data)

* **Fill missing values**
* fill missing values with -1

data = data.fillna({'c1': -1, 'c2': -1})

* fill missing values with median

data = data.fillna({'c2': data['c2'].median()})

#### 4.1 Delete

* select rows based on some conditions

data=data[data['c1'] <100] # select rows, c1列的值<100的

data[~data['c1'].isnull()] #select rows，c1列的值not null的

* select columns

data['c1'] #选出c1列

* select columns

data[data['c2'] <100]['c1'] #选出行，c2列中c1列，其中c1列中的c2

#### duplication， 唯一值，unique

查看c1列的值是否有重复，没有重复，返回True

len(data['c1'].unique()) == len(data)

返回c1列的值，只返回唯一的值

data['c1'].unique()

c1列唯一值的数量

len(data['c1'].unique())

#### Sort the data in ascending order

返回根据列c1值降序排序的

data.sort\_values(by='c1', ascending=False)

选出c1列，并对c1升序排序

sorted(data['c1'])

#### select rows and columns选择行，列

选择列c1,c2

data1 = data[['c1', 'c2']] #两个]]

选择c1列的值，选择dataframe里，c1列第1行的值，没有.values[0]就只是返回一个索引

tmp['c1'].values[0]

选择行

data['c1'] == 'Mobile'

#### 选择某些行select rows，多张表

返回选择的行,注意这里用到了2张表

data1[(data['c1'] <= 100) & (data['c3'] >= 20)]

#### Create data, 用已有的列的值，构造新的data

用已经有列数据c11,c22插入data中，data中包括2列，c1,c2

data = pd.DataFrame({'c1': c11, 'c2': c22}, columns=['c1', 'c2'])

c11=data[['home', 'search', 'payment']]

c22=data[[0.5, 0.1, 0.07]]

#### Add new rows/columns新增

dada中新增一列c2，用已有的列值c3赋值

data['c2'] = c3

c3 = data.groupby('c1')['c2'].count().reset\_index()

c3 = c3.rename(columns={'c2': 'c2\_count'})

#### date to week, month, hour, seconds

返回新的一列，c1列的每个日期对应的月

data[‘month’]=data['c1'].apply(lambda x: x.month)

返回新的一列，c1列的每个时间对应的小时

data['hour'] = data['c1'].apply(lambda x: x.hour)

返回新的一列，c1列的每个日期对应‘一年中的第几周（1,2....）

data['c1'].apply(lambda x: x.week)

返回新的一列两个日期之间（c3和c2）间隔多少天

data[cn]=(data['c3'] - data['c2']).apply(lambda x: x.days)

data[cn]=(data['c3'] - data['c2']).apply(lambda x: x.dt.days) 返回int值

返回新的一列两个日期之间（c3和c2）间隔多少秒

np.array(list(map(lambda x: x.seconds, data['c3'] - data['c2'])))

生成dataframe,每个元素是一个日期，每两个元素间隔1天，元素介于'2011-01-24'和'2015-12-13'之间，包括'2011-01-24', 和'2015-12-13'

pd.date\_range(start='2011-01-24', end='2015-12-13', freq='D')

#返回新的一列，c1列的每个日期对应的day of the week，0~6

data[‘dayofweek’]=data['c1'].apply(lambda x: x.dayofweek)

解析时间 parse time

data['c1'] = pd.to\_datetime(data['c1'])

将c1列和c2列解析为时间数据，如将20200102解析为2020-01-02，返回所有原始数据和解析后的列替换原来的列

pd.read\_csv('./juanchen/raw\_data.csv', parse\_dates=['c1', 'c2'])

取出c1中最早的一天

data['c1'].min( )

#### 新增列， add a new column

* **每行的和 sum**

仅保留原索引列，并增加一列，重命名为count,该列是原来每一行数据之和（数据之和不包括索引列的值），axis=1表示按行加，新增索引列0,1,2,原索引列为普通数据列

data1 = data.sum(axis=1).reset\_index().rename(columns={0: 'count'})

data1['is\_exceed'] = (data['c1'] > data ['c2']).astype(int)

#### 新增列，表示另一张表在一张表中是否存在

连接两张表data 和data1，在data中新增列c2,值为1表示该row在两张表都存在。

data1['c2'] = 1 #data1中新增列c2,值都为1

data = pd.merge(left=data, right=data1, how='left', on='c1')

data = data.fillna(value=0) #给所有NA填充0，因为data中有，data1中没有的值在c2列都是NA

data['c2'] = data['c2'].astype(int)

#### axis=1, axis=0

重点在于方向，而不是行和列

当axis=0时，方向从上到下，从上到下求平均，上下纵向拼接，drop表示纵向发生变化，即行的减少

df.dropna(axis=0,how='any', inplace=True)本列是否有NaN，若有，删去该NaN的行,inplace表示语句立即执行生效

当axis=1时，方向从左到右，从左到右横向求平均，左右横向拼接，drop表示横向发生变化，即删除列，这个太难理解了

#### 删除c1列

data = data.drop(labels='c1', axis=1)

#### 4.2Groupby

Groupyby之后是没有index的，例如根据c1的值分组（相同c1为一组），计算每组的c2的均值。返回2列，第一列为c1也是索引列，第二列为该组c2的均值，返回的是series, reset\_index()给返回的数据增加索引：0,1,2…使得返回的是dataframe。c1变为非索引的普通列

series类似于1维数组，由索引+数值组成，dataframe的某一列，返回的就是series

data1=data.groupby('c1')['c2'].mean().reset\_index()

data1 = data1.rename(columns={'c2': 'c2\_mean'}) #一般groupby之后都要重命名

data中新增列c2\_mean，值为每组的数量/均值

data = pd.merge(left=data, right=data1, on='c1')

根据c1的值分组（相同c1为一组），计算每组的c2的唯一数量。 unique\_count是一个函数，输入为该组的c2的值，求不同值的数量

data.groupby('c1')['c2'].apply(unique\_count).reset\_index()

def unique\_count(x):

return len(np.unique(x))

#### 分组后返回两列或多列，经常自定义函数并apply于组内groupby

根据c1分组，分组后返回多列c1,c2,c3，找到组内c2值最小的那一行对应的c2,c3值

def f1(df):

index = df['c2'].argmin() #该组内c2值最小的行的索引

return df.loc[index, ['c2', 'c3']] #该索引对应行的，c2和c3的值

data1=data.groupby('c1').apply(f1).reset\_index()

data1.rename(columns={'c2': 'c2\_min', 'c3': 'c3\_min'})

another example

当我们需要分组，并且得到多列，每列或者为统计，或者为其他值…

groupy+function

Take the feature ‘device’ for example

def run\_ttest(df):

c2 = df['c5'].values

c3 = c2.mean()

return pd.Series({'c2': len(test\_data), 'c3': test\_mean,})

tests.groupby('c1').apply(data). .reset\_index()

another example

def f1(df):

a = df['c3'].sum()

b = df['c4'].values[0] #组内，c3列的值是相同的，随便取一个值

return pd.Series([a, b], index=['c33', 'c44']) 返回两列，列名为c22和c33，值为a,b

data1 = data.groupby(['c1', 'c2']).apply(f1).reset\_index()

#### 最小值,最大值的行的索引index

c1最小值的那一行的索引

index = data['c1'].argmin()

该索引对应的行的c2和c3的值

data.loc[index, ['c2', 'c3']]

#### 4.2 rename 重命名

给第二列重命名为c1，注意是第0列，第一列为index即每个列的列名

conv\_ratio = 1 - pd.DataFrame(data.isnull().sum()).rename(columns={0: 'c1'}) / len(data)

给c1列重命名为c11，返回所有列，其中c1列更名为c11。

data.rename(columns={'c1': 'c11'})

#### 4.2 行、列取值

data.loc[‘r1’, ’c1’] #行列名，第r1行，第c1列

data.iloc[0, 3] #索引第1行，第4列

#### Read the row index, column index, column name 列名

DataFrame：列索引+行索引+行列数据

row index一般就是0,1,2...

>>>data.index

column index就是column name

>>>data.column

#### 计算某个矩阵特征的相似性similarity

根据c1和c2对data进行分组，计算组内c3的数量，索引为c1和c2，值为c3\_count。应用unstack后，c1变为行索引，c2变为列索引

data1 = data.groupby(['c1', 'c2'])['c3'].count().unstack(fill\_value=0) #缺失值用0填充

计算相似性similarity

data2 = normalize(data1, axis=1) # normalize the matrix

data3 = np.dot(data2, data2.T) # calculate the similarity matrix 点积即A\*A^T

similarity\_df = pd.DataFrame(data3, index=data1.index, columns=data1.index) 重新确定索引和列名称

def find\_topk('A', similarity\_df, k=10): #find the top 10

df = similarity\_df.loc['A'].sort\_values(ascending=False)[1:k + 1].reset\_index() #similarity\_df中，列‘A’中最大的10个值取出，保留原索引，并增加新索引0,1,2...

df = df.rename(columns={'c1': 'c11', 'A': 'c22'}) #c1是原来的索引列

return df

#### 求每一天比前一天的变化率，如增长率等，一段时间的trend

#df是一个dataframe，按照date升序排序后，取出c1列的值即为一个array

#计算c1\_array中每一天相对于昨天的变化率即 今天的值/昨天的值，c1\_array[1:]表示取出除了第一个值

的所有值，c1\_array[:-1]表示取出除了最后一个值外的所有值

c1\_array = df.sort\_values(by='date')['c1'].values

ratio = c1\_array[1:] / c1\_array[:-1]

### Feature engineering

There are about 20 features, delete 8

#### 5.1 Convert text or categorical values into numerical values.

* **Through label encoding**

features = subset[['c2', 'c3']]

# c1不变，新增c2列为c1的numerical values

le = LabelEncoder()

data['c2'] = le.fit\_transform(data['c1'])

* **Through one hot encoding**

（1）对data中所有categorical进行 one hot encoding

data = pandas.get\_dummies(data, drop\_first=True) #data中的所有categorical---numerical,其余特征不变

（2）保留c1列，新增encoding的若干列，并对若干列重命名，对c1列进行encoding后存入data1

data1 = pd.get\_dummies(data, columns=["c1"], prefix=["Type\_is"] ) #pd是pandas的缩写

data = data.join(data1)

* **Transform from categorical type into int type**

encoder = LabelEncoder()

data[target] = encoder.fit\_transform(data[target])

#### 5.1 Transform text values into numerical values

c1列有两个值’short’和’long’，分别转化为2和4变为c2列，删除c1，新增c2

data['c2'] = np.where(data.c1 == 'short',2,4)

del data['c1']

c1列有两个值’true’和’false’，分别转化为2和4变为c2列，删除c1，新增c2

data['c2'] = (data.c1 =='true').astype(np.int)

del data['c1']

weekday2index = {"Monday":1,"Tuesday":2,"Wednesday":3,"Thursday":4,"Friday":5,"Saturday":6,"Sunday":7}

emails["weekday"] = emails.weekday.map(weekday2index)

# rename long column names to shorter names, make it easier to read

emails.rename(columns={'user\_past\_purchases':'purchases','user\_country':'country'},inplace=True)

#### 5.2 Delete redundant features

* **plot the feature-feature pair correlation**

#df.dropna(axis=0,how='any')本列是否有NaN，若有，删去该NaN的行

fig, ax = plt.subplots(figsize=(12, 10))

sns.heatmap(data.dropna(axis=0, how='any').corr(), ax=ax)

plt.show()

#### 5.3 Delete features which are not related to the target variable

#将c1列和target列label encoding后，再计算每个feature和target间的相似性

#Print: feature name + chi-squared score+Fscore

resp\_lb\_encoder = LabelEncoder()

cnty\_lb\_encoder = LabelEncoder()

target = resp\_lb\_encoder.fit\_transform(target)

features['c1'] = cnty\_lb\_encoder.fit\_transform(features.c1)

chi2scores,\_ = chi2(features, target) #计算每个特征和target的chi-squared score

fscores, pvalues = f\_classif(features, target) #计算每个特征和target的Fscore

feat\_scores = pd.DataFrame({'chi2scores': chi2scores, 'fscores': fscores, 'chi2\_pvalue':pvalues },index= names) # names是列名，可以通过获取data.column，然后names=['c1', 'c2', 'c3']

feat\_scores.sort\_values(by='chi2scores',ascending=False)

#### 5.4 Create new features that may be related to the target variable

Sum, difference, product, division

### Modelling

#### Check imbalanced classes

print(data[data[‘target’] ==1][ ‘target’] .count())

print(data[data[‘target’] ==0][ ‘target’] .count())

#### 6.1 Logistic Regression

|  |
| --- |
| **Logistic Regression** |
| Logistic Regression LR  import numpy as np  import pandas as pd  import seaborn as sns  import matplotlib.pyplot as plt  from sklearn.preprocessing import LabelEncoder  from sklearn.preprocessing import StandardScaler  from sklearn.model\_selection import train\_test\_split  from sklearn.linear\_model import LogisticRegression  from sklearn.linear\_model import LogisticRegressionCV #crossvalidation+ LR,之后其他都一样  #不适用imbalanced classes严重的，features是int,float，不能是object,categorical  ss = StandardScaler() #对训练和测试数据的特征归一化  X\_train = ss.fit\_transform(X\_train)  X\_test = ss.transform(X\_test)  #LR = LogisticRegressionCV(cv=5, random\_state=0)  LR = LogisticRegression()  lrmodel=LR.fit(X\_train,y\_train)  lrpred=lrmodel.predict(X\_test) #predicted laber on test dataset  lrpred\_prob=lrmodel.predict\_proba(X\_test)[:,1] #predicted probability on test dataset，因为会产生对label0和label1的预测概率，只取label1的概率即可  LR主要看每个特征的p\_value and coefficients, p\_values越小，该特征对于target越重要，其次看系数，系数绝对值越大越好。系数为正表明和特征正相关。负值绝对值大也好  print(lrmodel.coef\_) #coefficients  feature\_selection库的SelectKBest类查看每个特征的pvalue |

#### 6.2 Linear Regression

|  |
| --- |
| **Linear Regression** |
| 用statsmodels来做linear regression  每一个linear regression will output cooefficient+p\_value  import statsmodels.api as sm  data['intercept'] = 1    #build the linear regression  lr = sm.OLS(data['target'],data(‘feature’)).fit()  coefficient=lr.params[0]  p\_value=lr.pvalues[0]  下面是用sklearn来做linear regression  import numpy as np  import pandas as pd  import sklearn  from sklearn.preprocessing import StandardScaler  from sklearn.linear\_model import LinearRegression  from sklearn.model\_selection import train\_test\_split  import matplotlib as mpl  import matplotlib.pyplot as plt  plt.style.use('ggplot')  import seaborn as sns  import warnings  warnings.filterwarnings('ignore')  #data statistics  data.head()  data.info()  data.describe()  data cleaning  #make a bar plot 看the total number of missing data for each input variable  import missingno as mis  mis.bar(data,labels=True)  data visualization  1. plot and show the relationship between the input variable and the target variable.  可以看出，第3幅图的特征和target variable relationship不大  sns.pairplot(data,x\_vars=['TV','Radio','Newspaper'],y\_vars='Sales',size=7,aspect=0.8)    Data splitting, feature scaling  x\_train,x\_test,y\_train,y\_test = train\_test\_split(x,y,test\_size=0.3, random\_state=2)  用statsmodel来build the model  import statsmodels.api as sm  X\_train=sm.add\_constant(X\_train)  model=sm.OLS(y\_train,X\_train).fit()  #调用summary，查看模型的各项指标值  print(model.summary())  summary可以看到如下信息  1.R-squared, (0, 1)之间，越接近1，model越好  2.F-statistic和F\_theory， F-statistic>>F\_theory,好  3.coef: 回归系数  4.对每一个input variable, p\_value越小越好，如果>=0.05，则说明该input variable和target variable关系不大  5.Durbin-Watson，用于No autocorrelation of residuals检验，如果Durbin-Watson约为2，则residuals之间不太相关，满足。  LR存在一些假设，构建模型后测试这些假设是否符合  1.The residuals should be normally distributed.  画图看residuals distribution，是否是正态分布。  model=sm.OLS(y\_train,X\_train.iloc[:,0:-1]).fit()  import scipy.stats as stats  sns.distplot(model.resid,bins=10,fit=stats.norm,norm\_hist=True,  hist\_kws={'color':'steelblue','edgecolor':'black'},  kde\_kws={'color':'black','linestyle':'--'},  fit\_kws={'color':'red','linestyle':'--'})  plt.xlabel('residual',fontsize=14)  核密度曲线与正态密度曲线的趋势比较吻合，故直观上可认为误差项服从正态分布。    也可以用Q-Q Plot里，散点会近似的落在一条直线上，如下图    2.No perfect multicollinearity input variables之间不存在线性关系。  VIF<3即符合  from statsmodels.stats.outliers\_influence import variance\_inflation\_factor  X=sm.add\_constant(data.loc[:,['特征1','特征2']])  vif=pd.DataFrame()  vif['features']=X.columns  vif['VIF Factor']=[variance\_inflation\_factor(X.values,i) for i in range(X.shape[1])]  vif  3. Residuals are independent of each other. 彼此不相关 This is applicable especially for time series data. When the residuals are autocorrelated, it means that the current value is dependent of the previous (historic) values。  如果Durbin-Watson约为2，如2.03等，则residuals之间不太相关，满足。  4.方差齐性是指要求模型残差项的方差  plot residual vs 特征1, residual vs 特征2  plt.figure(figsize=(30,6))  plt.subplot(121)  plt.scatter(X\_train.loc[:,'特征1'],(model.resid-model.resid.mean())/model.resid.std())  plt.hlines(y=0,xmin=X\_train.loc[:,'特征1'].min(),xmax=X\_train.loc[:,'特征1'].max())  plt.xlabel('特征1',fontsize=24)  plt.ylabel('std\_residual',fontsize=24)  plt.subplot(122)  plt.scatter(X\_train.loc[:,'特征2'],(model.resid-model.resid.mean())/model.resid.std())  plt.hlines(y=0,xmin=X\_train.loc[:,'特征2'].min(),xmax=X\_train.loc[:,'特征2'].max())  plt.xlabel(特征2',fontsize=24)  plt.ylabel('std\_residual',fontsize=24)  由图可知，残差几乎均匀地分布在参考线y=0的附近，满足假设。    满足所有假设条件后，可以用trained model , make predictions  X\_test=sm.add\_constant(X\_test)  y\_pred=model.predict(X\_test.iloc[:,0:-1])  plt.figure(figsize=(10,6))  plt.scatter(y\_test,y\_pred)  plt.plot([y\_test.min(),y\_test.max()],[y\_pred.min(),y\_pred.max()],color='blue',linestyle='--')  plt.xlabel('y\_true',fontsize=14)  plt.ylabel('y\_predict',fontsize=14)  plt.show()  Feature scaling, 用scikit learn 来build the model.  ss = StandardScaler()  x\_train\_s = ss.fit\_transform(x\_train)  x\_test\_s = ss.transform(x\_test)    lrmodel = LinearRegression().fit(x\_train\_s,y\_train)  y\_predict=lrmodel.predict(x\_test\_s)  print( lrmodel.score(x\_test\_s,y\_test) ) # 拟合程度，1.0最高  print( lrmodel.coef\_ ) #系数  print( lrmodel.intercept\_ ) #截距  #plotting two curve, ‘actual target value’ vs ‘predicted target value’  t=np.arange(len(x\_test\_s))  plt.figure(facecolor='w')  plt.plot(t, y\_test, 'r-', linewidth=2, label='actual target value')  plt.plot(t, y\_predict, 'g-', linewidth=1, label='predicted target value')  plt.legend(loc = 'upper left')  plt.title("actual target value vs predicted target value", fontsize=20)  plt.grid(b=True)  plt.show() |

#### 6.2 Linear Regression （加强版，包括）

#### 6.3 LightGBM

|  |
| --- |
| **LightGBM** |
| **Windows里jupyter安装LightGBM模块，进入anaconda prompt pip install lightgbm**  Data splitting  features和target是从data中取出的特征和label  features\_train, features\_test, target\_train, target\_test = train\_test\_split(featuers, target, test\_size=0.25, stratify=y, random\_state=42)  # create LightGBM dataset,注意categorical\_feature是data中的categorical 类型的所有features，target不能是categorical类型的，如果是，需要通过labelencoding转化  tree\_train=lgb.Dataset(data=features\_train, label=target\_train, categorical\_feature=categorical\_feature, free\_raw\_data=False)  Cross validation and find optimal parameters  parameters= {'learning\_rate': 0.01,  'boosting\_type': 'gbdt',  'objective': 'binary',  'metric': ['binary\_logloss', 'auc'],  'sub\_feature':0.5,  'num\_leaves': 31,  'min\_data': 50,  'max\_depth': 30,  'is\_unbalance': True}  history = lgb.cv(parameters, train\_set=tree\_train, num\_boost\_round=1000, nfold=5,  early\_stopping\_rounds=20, seed=42, verbose\_eval=False)  print('Best rounds:\t', len(history['auc-mean']))  Best rounds: 767 #an optimal parameter: run 767 rounds    Re-train the model and make predictions on test dataset  clf = lgb.train(parameters, train\_set=d\_train, num\_boost\_round=767)  pred = clf.predict(features\_test)  Plotting feature importance  features = clf.feature\_name()  importance = clf.feature\_importance()  fig, ax = plt.subplots(figsize=(10, 8))  lgb.plot\_importance(clf, ax=ax, height=0.5)  plt.show()  画图threshold vs myaccuracy,找到an optimal threshold  #given 100 thresholds between (0, 1)  thresholds = list(np.linspace(0, 1, 100)) ☺  Calculate predicted labels, given a threshold  pred\_label =(pred > threshold).astype(int) #pre是根据算法预测出来的label的概率  accuracy=根据pred\_label计算预测的精确度（如果不是已有的accuracy,auc等）  对于每一个threshold都预测一个精确度，所有精确度存入myaccuracy  myaccuracy.append(accuracy)  fig, ax = plt.subplots(figsize=(8, 6))  ax.plot(thresholds, pred\_label, label='New Model')  ax.set\_xlabel('Threshold', fontsize=12)  ax.set\_ylabel('Profit', fontsize=12)  ax.legend(fontsize=12)  plt.tight\_layout()  plt.show() |

#### 6.4 Random Forest

|  |
| --- |
| **Random Forest** |
| **Windows里jupyter安装seaborn模块绘图，进入anaconda prompt**  **pip install seaborn**  Compare 3 models   * Base line RF * Baseline RF with PCA (降维) * RF with PCA (降维)+ parameter selection (RandomizedSearch+gridsearch)   #data splitting  #stratify = y 以确保训练集和测试集与原始数据集的 0 和 1 的比例一致。  #设定random\_state后，每次拆分的训练集和测试集也是相同的。  #data 包括features 和target  train\_features, test\_features, train\_target, test\_target = train\_test\_split(features, target, test\_size=0.3, random\_state = 2020, stratify=y)  #data normalization  from sklearn.preprocessing import StandardScaler  ss = StandardScaler()  train\_features\_scaled = ss.fit\_transform(train\_features)  test\_features\_scaled = ss.transform(test\_features)  train\_target = train\_target.values  #modelling and display the recall\_score of the model  from sklearn.ensemble import RandomForestClassifier  from sklearn.metrics import recall\_score    rf = RandomForestClassifier()  rfmodel =rf.fit(train\_features\_scaled, train\_target)      rfpred=rfmodel.predict(test\_features\_scaled) #predicted laber on test dataset  rfpred\_prob=rfmodel.predict\_proba(test\_features\_scaled)[:,1] #predicted probability on test dataset    print(accuracy\_score(test\_target,rfpred)) #default threshold 0.5  print(precision\_score(test\_target,rfpred))  print(roc\_auc\_score(test\_target,rfpred\_prob))  #feature importance using chart and table 服务于降维，之后再用PCA double check  feats = {}  for feature, importance in zip(data.columns, rf.feature\_importances\_):  feats[feature] = importance  #第一列为索引，第二列重命名为gini-importance  importances = pd.DataFrame.from\_dict(feats, orient='index').rename(columns={0: 'Gini-Importance'})  #按照gini-importance降序排序  importances = importances.sort\_values(by='Gini-Importance', ascending=False)  #新增索引列，原来的索引列为第一列  importances = importances.reset\_index()  #原来的索引列重命名为features  importances = importances.rename(columns={'index': 'Features'})  #画,3列，第一列索引，第二列特征名称，第三列特征重要性值  sns.set(font\_scale = 5)  sns.set(style="whitegrid", color\_codes=True, font\_scale = 1.7)  fig, ax = plt.subplots()  fig.set\_size\_inches(30,15)  sns.barplot(x=importances['Gini-Importance'], y=importances['Features'], data=importances, color='skyblue')  plt.xlabel('Importance', fontsize=25, weight = 'bold')  plt.ylabel('Features', fontsize=25, weight = 'bold')  plt.title('Feature Importance', fontsize=25, weight = 'bold')  display(plt.show())  print(importances)  #find the number of components for PCA. find the most important compents 降到多少维即the number of components，性能不再显著提升即cvr不再明显增长  import matplotlib.pyplot as plt  import seaborn as sns  from sklearn.decomposition import PCA  pca\_test = PCA(n\_components=30)  pca\_test.fit(train\_features\_scaled)  #plot a curve, no. of features vs cvr  sns.set(style='whitegrid')  plt.plot(np.cumsum(pca\_test.explained\_variance\_ratio\_))  plt.xlabel('number of features')  plt.ylabel('cumulative explained variance')  plt.axvline(linewidth=4, color='r', linestyle = '--', x=10, ymin=0, ymax=1)  display(plt.show())  #visualize the no. of features and the cvr, 大概多少个features之后，cvr不再明显增长，则不再多选feature  cvr = np.cumsum(pca\_test.explained\_variance\_ratio\_)  pca\_df = pd.DataFrame()  pca\_df['Cumulative Variance Ratio'] = cvr  display(pca\_df.head(10))  #apply PCA on训练数据和测试数据，降到10维，retrain using RF  pca = PCA(n\_components=10)  pca.fit(train\_features\_scaled)  train\_features\_scaled\_pca = pca.transform(train\_features\_scaled)  test\_features\_scaled\_pca = pca.transform(test\_features\_scaled)  rf2= RandomForestClassifier()  rf2.fit(train\_features\_scaled\_pca, train\_target)  print(rf2.score(train\_features\_scaled\_pca, train\_target)) # recall\_score  the third model  parameter selection using RandomSearchCV  from sklearn.model\_selection import RandomizedSearchCV  n\_estimators = [int(x) for x in np.linspace(start = 100, stop = 1000, num = 10)]  max\_features = ['log2', 'sqrt']  max\_depth = [int(x) for x in np.linspace(start = 1, stop = 15, num = 15)]  min\_samples\_split = [int(x) for x in np.linspace(start = 2, stop = 50, num = 10)]  min\_samples\_leaf = [int(x) for x in np.linspace(start = 2, stop = 50, num = 10)]  bootstrap = [True, False]  param\_dist = {'n\_estimators': n\_estimators,  'max\_features': max\_features,  'max\_depth': max\_depth,  'min\_samples\_split': min\_samples\_split,  'min\_samples\_leaf': min\_samples\_leaf,  'bootstrap': bootstrap}  rf3= RandomForestClassifier()  #迭代100次，测试100组参数组合，3folds crossvalidation,构造3\*100=300个森林，n\_jobs=-1使用所有CPU  rs = RandomizedSearchCV(rf3, param\_dist, n\_iter = 100, cv = 3, verbose = 1,  n\_jobs=-1, random\_state=0)  rs.fit(train\_features\_scaled\_pca, train\_target) #用最好的参数在训练集上重新训练  rs.best\_params\_ #print best parameters      ————————————————————————————————————————————  # {'n\_estimators': 700,  # 'min\_samples\_split': 2,  # 'min\_samples\_leaf': 2,  # 'max\_features': 'log2',  # 'max\_depth': 11,  找到每个参数取值范围，使得score值最大。对每个参数，画一幅图。mean\_test\_score vs 参数。  fig, axs = plt.subplots(ncols=3, nrows=2)  sns.set(style="whitegrid", color\_codes=True, font\_scale = 2)  fig.set\_size\_inches(30,25)sns.barplot(x='param\_n\_estimators', y='mean\_test\_score', data=rs\_df, ax=axs[0,0], color='lightgrey')  axs[0,0].set\_ylim([.83,.93])axs[0,0].set\_title(label = 'n\_estimators', size=30, weight='bold')sns.barplot(x='param\_min\_samples\_split', y='mean\_test\_score', data=rs\_df, ax=axs[0,1], color='coral')  axs[0,1].set\_ylim([.85,.93])axs[0,1].set\_title(label = 'min\_samples\_split', size=30, weight='bold')sns.barplot(x='param\_min\_samples\_leaf', y='mean\_test\_score', data=rs\_df, ax=axs[0,2], color='lightgreen')  axs[0,2].set\_ylim([.80,.93])axs[0,2].set\_title(label = 'min\_samples\_leaf', size=30, weight='bold')sns.barplot(x='param\_max\_features', y='mean\_test\_score', data=rs\_df, ax=axs[1,0], color='wheat')  axs[1,0].set\_ylim([.88,.92])axs[1,0].set\_title(label = 'max\_features', size=30, weight='bold')sns.barplot(x='param\_max\_depth', y='mean\_test\_score', data=rs\_df, ax=axs[1,1], color='lightpink')  axs[1,1].set\_ylim([.80,.93])axs[1,1].set\_title(label = 'max\_depth', size=30, weight='bold')sns.barplot(x='param\_bootstrap',y='mean\_test\_score', data=rs\_df, ax=axs[1,2], color='skyblue')  axs[1,2].set\_ylim([.88,.92])  通过上面的图，了解每个参数取值对score的影响。  n\_estimators：300、500、700 的平均分数几乎最高；  min\_samples\_split：较小的值（如 2 和 7）得分较高。23 处得分也很高。我们可以尝试一些大于 2 的值，以及 23 附近的值；   min\_samples\_leaf：较小的值可能得到更高的分，我们可以尝试使用 2–7 之间的值；   max\_features：「sqrt」具有最高平均分；   max\_depth：没有明确的结果，但是 2、3、7、11、15 的效果很好；  bootstrap：「False」具有最高平均分。  第 2 轮超参数调整：GridSearchCV  使用 RandomSearchCV 之后，对于相同的参数，缩小参数值的搜索范围，使用 GridSearchCV 进行更精确的搜索。  6 个参数搜索 10 个不同的参数值， 3 折交叉验证，拟合模型 3,000,000 次！  parameter selection using GridSearchCV  from sklearn.model\_selection import GridSearchCV  n\_estimators = [300,500,700]  max\_features = ['sqrt']  max\_depth = [2,3,7,11,15]  min\_samples\_split = [2,3,4,22,23,24]  min\_samples\_leaf = [2,3,4,5,6,7]  bootstrap = [False]  param\_grid = {'n\_estimators': n\_estimators,  'max\_features': max\_features,  'max\_depth': max\_depth,  'min\_samples\_split': min\_samples\_split,  'min\_samples\_leaf': min\_samples\_leaf,  'bootstrap': bootstrap}  rf4= RandomForestClassifier()  #这里作者没有使用scoring参数，在一些情况下，sklearn中没有现成的scoring函数，需要自己定义  gs = GridSearchCV(rf4, param\_grid, cv = 3, verbose = 1, n\_jobs=-1)  gs.fit(train\_features\_scaled\_pca, train\_target)  rf4 = gs.best\_estimator\_ #最好的model  gs.best\_params\_ #最好的参数组合，返回最优的精度gs.best\_score\_, gs.v\_results\_ 返回结果如参数等  #打印3个model的recall score,分别对positive, negative的number of correctly predicted observations  pred = rf.predict(test\_features\_scaled)  pred2= rf2.predict(test\_features\_scaled \_pca)  pred3 = gs.best\_estimator\_.predict(test\_features\_scaled\_pca)  from sklearn.metrics import confusion\_matrix  conf\_matrix\_baseline = pd.DataFrame(confusion\_matrix(y\_test, y\_pred), index = ['actual 0', 'actual 1'], columns = ['predicted 0', 'predicted 1'])  conf\_matrix\_baseline\_pca = pd.DataFrame(confusion\_matrix(y\_test, y\_pred\_pca), index = ['actual 0', 'actual 1'], columns = ['predicted 0', 'predicted 1'])  conf\_matrix\_tuned\_pca = pd.DataFrame(confusion\_matrix(y\_test, y\_pred\_gs), index = ['actual 0', 'actual 1'], columns = ['predicted 0', 'predicted 1'])  display(conf\_matrix\_baseline)  display('Baseline Random Forest recall score', recall\_score(test\_target, pred))  display(conf\_matrix\_baseline\_pca)  display('Baseline Random Forest With PCA recall score', recall\_score(test\_target, pred2))  display(conf\_matrix\_tuned\_pca)  display('Hyperparameter Tuned Random Forest With PCA Reduced Dimensionality recall score', recall\_score(test\_target, pred3)) |

#### 6.5 Decision Tree

|  |
| --- |
| **Decision Tree** |
| clf = DecisionTreeClassifier(max\_depth=3, min\_samples\_leaf=30, random\_state=42)  clf = clf.fit(X=train\_features, y=train\_target)    # Visualization 画图看生成的DT整个过程  features = list(train\_features.columns)  #target 的两个值0/1，如果想在树叶子节点显示的两个值为'Not quit'和 'Quit'，则 targets = ['Not quit', 'Quit']  targets = decision\_tree\_classifier.classes\_  dot\_data = tree.export\_graphviz(clf, out\_file=None, feature\_names=features, class\_names=targets,  filled=True, rounded=True, special\_characters=True, )  graph = graphviz.Source(dot\_data)  graph  #View feature importance in a descending order default  importance = sorted(zip(features, clf.feature\_importances\_), key=lambda x:x[1], reverse=True)  for feature, importance\_val in importance:  print('{0:10s} | {1:.5f}'.format(feature, importance\_val)) |

#### 6.6 PCA

|  |
| --- |
| **PCA** |
| Reduce dimensionality  # normalize the data  scaler = StandardScaler()  norm\_features = scaler.fit\_transform(features)  # apply PCA降维2维，feature为没有label的数据  pca = PCA(n\_components=2, random\_state=42)  new\_features= pca.fit\_transform(norm\_features)  visualize the new data after implementing PCA  fig, ax = plt.subplots(figsize=(10, 8))  ax.plot(new\_features[:, 0], new\_features[:, 1], '.', markersize=1)  ax.set\_xlabel('PCA Component 1', fontsize=12)  ax.set\_ylabel('PCA Component 2', fontsize=12)  plt.show() |

#### 6.7 K-Means

|  |
| --- |
| **K-Means** |
| import warnings  warnings.simplefilter('ignore')  import numpy as np  import pandas as pd  import seaborn as sns  import matplotlib.pyplot as plt  from sklearn.preprocessing import StandardScaler  from sklearn.decomposition import PCA  from sklearn.cluster import KMeans  from sklearn.metrics import silhouette\_score  找最佳k值  画图看no. of clusters vs silhouette\_score，找到silhouette\_score越接近1的k值  #feature为没有label的数据，最好data normalize一下  silhouettes = []  for k in range(2, 30): #尝试k=2~30  kmodel = KMeans(n\_clusters=k, init='k-means++', random\_state=42, n\_jobs=-1) .fit(feature)  label = kmodel.predict(feature) //计算silhouette非常非常慢，跑一晚上没跑出来    silhouettes.append(silhouette\_score(feature, label))  #开始画图  fig, ax = plt.subplots(nrows=1, ncols=1, figsize=(18, 6))  ax[1].plot(range(2, 30), silhouettes, 'o-', label='Silhouette Coefficient')  ax[1].grid(True)  plt.legend(fontsize=12)  plt.tight\_layout()  plt.show()  用最佳的k值，聚类并预测  kmeans = KMeans(n\_clusters=10, init='k-means++', random\_state=42, n\_jobs=-1) #分为10个类  kmeans = kmeans.fit(feature)  label = kmeans.predict(feature)  画图，散点图，不同的类不同颜色  #5个类，label值为0,1,2…4。每个类的点颜色分别为r,b,y,g,c,k(black)  color = ('r', 'b','y','g','c','k')  label\_color = np.array(color)[label5]  plt.scatter(pca\_x[:, 0], pca\_x[:, 1], c=label\_color)  plt.show()  print(len(pca\_x[label5==0])) #每一类点的数量  print(len(pca\_x[label5==1] ))  print(len(pca\_x[label5==2] ))  print(len(pca\_x[label5==3] ))  print(len(pca\_x[label5==4] )) |

#### 6.7 IsolationForest

|  |
| --- |
| **IsolationForest** |
| import warnings  warnings.simplefilter('ignore')  import numpy as np  import pandas as pd  import seaborn as sns  import matplotlib.pyplot as plt  from sklearn.preprocessing import StandardScaler  from sklearn.decomposition import PCA  from sklearn.ensemble import IsolationForest  IsolationForest is an ensemble method. It is linear 具有线性时间复杂度。是ensemble的方法，所以可以用在含有海量数据的数据集上面。  iForest不适用于特别高维的数据。所以可以用PCA来降维后使用。‘fraudulant transaction identification’项目中，十几分钟就跑完了。但是DBscan和kmeans时间都很长。  #norm\_feature是归一化后的特征  forest = IsolationForest(n\_estimators=100, max\_samples='auto', contamination=0.1,  max\_features=1.0, bootstrap=False, n\_jobs=-1, random\_state=42).fit(norm\_feature)    score = forest.decision\_function(norm\_feature)  label = forest.predict(norm\_feature) # label：+1 表示正常样本， -1表示异常样本。就是一个一维数据，然而kmeans是0,1,2…没有负值  #画图用不同颜色显示two classes。  #需要用到PCA降到2维的数据pca\_x（归一化后再降维），第一维和第二维分别为pca\_x[:, 0]和pca\_x[:, 1]，label会对应到pac\_x中  plt.scatter(pca\_x[:, 0], pca\_x[:, 1], c=label)  plt.show()  print(len(pca\_x[label5==-1])) #每一类点的数量  print(len(pca\_x[label5==1] ))  降维  pca = PCA(n\_components=2, random\_state=42)  pca\_x = pca.fit\_transform(norm\_feature) |

#### 6.8 Time series prediction

|  |
| --- |
| **Time series prediction Prophet ‘profit’** |
| Windows系统下安装Python版本的Prophet  import matplotlib.dates as mdates  from matplotlib import rcParams    #预测data1中2016-12-15的c1的值，目前c1的值肯定在2016-12-15前  data1= data[['date', 'clicks']]  interval= (pandas.to\_datetime("2016-12-15")-data1['date'].max()).days #间隔多少天  ts = fbprophet.Prophet() #modelling  ts.fit(data1)  future\_data = ts.make\_future\_dataframe(periods = interval)  pred = ts.predict(future\_data) #the predicted the no. of clicks for the interval    #get the predicted value for the last day 2016-12-15  pred\_lastday=pred['yhat'].iat[-1]  #plot date vs 'clicks' for previous period+interval  ts.plot(pred)  plt.show() |

#### 6.9 XGB

|  |
| --- |
| **XGB** |
| # split for training and testing  train\_features,test\_features,train\_target,test\_target = train\_test\_split(features,taraget,test\_size=0.33333,random\_state = seed)  train\_matrix = xgb.DMatrix(train\_features,train\_target)  test\_matrix = xgb.DMatrix(test\_features)  params = {}  params['objective'] = 'binary:logistic' # output probabilities  params['eval\_metric'] = 'auc'  params["num\_rounds"] = 300  params["early\_stopping\_rounds"] = 30  # params['min\_child\_weight'] = 2  params['max\_depth'] = 6  params['eta'] = 0.1  params["subsample"] = 0.8  params["colsample\_bytree"] = 0.8  cv\_results = xgb.cv(params,train\_matrix,  num\_boost\_round = params["num\_rounds"],  nfold = params.get('nfold',5),  metrics = params['eval\_metric'],  early\_stopping\_rounds = params["early\_stopping\_rounds"],  verbose\_eval = True,  seed = seed)  n\_best\_trees = cv\_results.shape[0]  print "best number of trees: {}".format(n\_best\_trees) # output 53  watchlist = [(train\_matrix, 'train')]  gbt = xgb.train(params, train\_matrix, n\_best\_trees,watchlist)  # plot feature importances  xgb.plot\_importance(gbt)  Plotting ROC curve  def validation\_roc():  train\_features1,valid\_features,train\_target1,valid\_target = train\_test\_split(train\_features,train\_target,test\_size=0.2,random\_state=seed)    train\_data = xgb.DMatrix(train\_features1,train\_target1)  valid\_data = xgb.DMatrix(valid\_features)    # retrain on training set  xgb\_train = xgb.train(params, train\_data, n\_best\_trees)    # predict on validation set  valid\_probas = xgb\_train.predict(valid\_data, ntree\_limit=n\_best\_trees)    d = {}  d['FPR'],d['TPR'],d['Threshold'] = roc\_curve(valid\_target,valid\_probas)  return pd.DataFrame(d)  roc\_results = validation\_roc()  \_ = plt.figure()  plt.plot(roc\_results.FPR,roc\_results.TPR)  plt.xlabel("FPR")  plt.ylabel('TPR')  roc\_results.loc[(roc\_results.TPR > 0.6) & (roc\_results.TPR < 0.65),:] |

#### 6.10 evaluation metrics

from **sklearn**.metrics import auc, roc\_curve, classification\_report, precision\_score, recall\_score,

accuracy\_score, roc\_auc\_score

**accuracy：**

accuracy = accuracy\_score(actual\_target,predict\_target) #真实的label，预测的label

**calculate recall**： minimize FN

recall\_score(actual\_target,predict\_target) #真实的label，预测的label

**calculate precision:** minimize FP

precision\_score(actual\_target,predict\_target) #真实的label，预测的label

**calculate auc：**

auc=roc\_auc\_score(actual\_target,predict\_target\_prob) #真实的label，预测的label概率

#### 6.11 plot roc curve and find the best threshold

#the intersection between the line y=-x+1 and the roc curve

from sklearn.metrics import roc\_auc\_score,roc\_curve

from matplotlib import pyplot

#label\_proba是LR，RF预测的label的概率，如lrpred\_prob=lrmodel.predict\_proba(X\_test)[:,1]

#test\_target为test dataset的target

fpr, tpr, thresholds = roc\_curve(test\_target, label\_proba)

pyplot.plot(fpr, tpr, marker='.', label='Logistic')

pyplot.xlabel('False Positive Rate')

pyplot.ylabel('True Positive Rate')

pyplot.legend()

pyplot.show()

#找到最佳cut-off 即thresholds[i]

for i in range(len(fpr)):

if fpr[i] + tpr[i] >= 1: #最靠近left top的那个threshold

i = i -1

break

#the best threshold is (fpr[i], tpr[i])的thresholds[i]

print(thresholds[i])

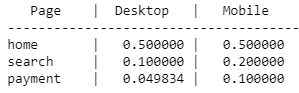
### Print

print('{0:^10s} | {1:^10s} | {2:^10s}'.format('Page', 'Desktop', 'Mobile'))#打印page | desktop

print('-' \* 40) #打印--------

for name, desk\_rate, mobile\_rate in zip(names, desk\_rates, mobile\_rates): #逐行打印

print('{0:10s} | {1:10.6f} | {2:10.6f}'.format(name, desk\_rate, mobile\_rate))



### Visualization

#### 8.1Feature distribution + feature vs target, feature是device

* **一行两幅图，feature是categorical，类似age这种也能用，就是bar比较密集**

1. (feature distribution)

fig, ax = plt.subplots(nrows=1, ncols=2, figsize=(16, 6))

sns.countplot(x='device', data=data, ax=ax[0])

ax[0].set\_xlabel('device', fontsize=12)

ax[0].set\_ylabel('Count', fontsize=12)

ax[0].set\_title('different devices distribution', fontsize=16)

1. (feature vs target)

sns.barplot(x='device', y='target', data=data, ax=ax[1])

ax[1].set\_xlabel('device', fontsize=12)

ax[1].set\_ylabel('target', fontsize=12)

ax[1].set\_title('device vs. target', fontsize=16)

plt.tight\_layout()

plt.show()

* **一行两幅图，feature是continuous**

1. (feature distribution) bar+curve

#sns.distplot绘制a histogram with a line on it

hist\_kws={'histtype': 'bar', 'edgecolor':'black', 'alpha': 0.2}

warnings.filterwarnings('ignore')

#feature c1 distribution feature c1在target取0和取1时的两个分布，每个分布都带有一条拟合曲线

fig, ax = plt.subplots(nrows=1, ncols=2, figsize=(16, 6))

sns.distplot(data[data['target'] == 0]['c1'], label='taraget = 0', ax=ax[0], hist\_kws=hist\_kws)

sns.distplot(data[data['target'] == 1]['c1'], label='target = 1', ax=ax[0], hist\_kws=hist\_kws)

ax[0].set\_title('Histogram of c1', fontsize=16)

ax[0].legend()

* (feature distribution) **三行，每行一幅图，同横坐标，便于对比**

#每幅图40个bar，横坐标是1~40，纵坐标是c1的值，c1有40个值。本来是c1值的数量，但是因为指明40个bar，显然每个横坐标对应1个值，这样统计总数量没有意义

hist\_kws={'histtype': 'bar', 'edgecolor':'black', 'alpha': 0.2}

fig, ax = plt.subplots(nrows=3, ncols=1, figsize=(12, 10), sharex=True)

sns.distplot(data['c1'], bins=40, ax=ax[0], label='25%', hist\_kws=hist\_kws)

ax[0].legend(fontsize=12)

sns.distplot(data['c2'], bins=40, ax=ax[1], label='50%', hist\_kws=hist\_kws)

ax[1].legend(fontsize=12)

sns.distplot(data['c3'], bins=40, ax=ax[2], label='75%', hist\_kws=hist\_kws)

ax[2].legend(fontsize=12)

plt.tight\_layout()

plt.show()

(2) (feature vs target) curve .-

#取出两列c1和target，根据c1分组，求target的均值，如果target是0/1，则求的是target为1的百分率。返回两列c1和target的百分率（但列名依然为'target'），新增索引0,1,2...

data1 = data[['c1', 'target']].groupby('c1').mean().reset\_index()

ax[1].plot(data1['c1'], data1['target'], '.-') #绘制一条线

ax[1].set\_title('c1 vs. Mean target', fontsize=16)

ax[1].set\_xlabel('c1')

ax[1].set\_ylabel('Mean target')

ax[1].grid(True)

plt.show()

#### target vs feature ----4 features 4 charts 2\*2

4幅图，每行2幅，2行。一幅图：特征vs target

plt.style.use(‘fivethirtyeight‘)

fig, ((ax1, ax2), (ax3, ax4)) = plt.subplots(ncols=2, nrows=2, figsize=(12, 12))

#对于日期自动旋转60度，比如x-axis是日期类型，则如果横轴密密麻麻，日期重叠，则旋转，比如c1是日期

fig.autofmt\_xdate(rotation=60)

ax1.plot(data[‘c1’], data[‘target ‘], linewidth=4)

ax1.set\_xlabel(‘‘), ax1.set\_ylabel(‘\*\*\* ‘), ax1.set\_title(‘\*\*\*‘)

ax2.plot(data[‘c2’], data[‘target ‘], linewidth=4)

ax2.set\_xlabel(‘‘), ax1.set\_ylabel(‘\*\*\* ‘), ax1.set\_title(‘\*\*\* ‘)

ax3.plot(data[‘c3’], data[‘target ‘], linewidth=4)

ax3.set\_xlabel(‘‘), ax1.set\_ylabel(‘\*\*\* ‘), ax1.set\_title(‘\*\*\* ‘)

ax4.plot(data[‘c4’], data[‘target ‘], linewidth=4)

ax4.set\_xlabel(‘‘), ax1.set\_ylabel(‘\*\*\*‘), ax1.set\_title(‘\*\*\* ‘)

plt.show()