机器学习模型的笔记

1数据处理的管道

建立处理节点，处理流程，方便替换节点和组织流程。

1.1管道的四大组织构件：

1.1.1.Pipeline串联管道

from sklearn.pipeline import Pipeline

管道类生成对象。pipe=Pipeline(steps=[('pca',PCA()),('svc',SVC())])。setps参数是列表，嵌套元组，元组内第一个是自命名，第二个是生成数据处理类的实例，生成的新管道函数的对象，同时具有fit 和 transform的方法。前n-1个step中的类都必须有transform函数，最后一步可有可无，一般最后一步为模型。

1.1.2.ColumnTransformer

from sklearn.compose import ColumnTransformer按不同列应用不同的估计转化器，最后列合并。不同于管道函数按照先后顺序。对于dataframe不同的列是包含列名的列表，对于array可以是列序号的列表，均输出合并后特征向量的二位数组。

num\_attribs = list(housing\_num)

cat\_attribs = ["ocean\_proximity"]

full\_pipeline = ColumnTransformer([

("num", num\_pipeline, num\_attribs),

("cat", OneHotEncoder(), cat\_attribs),

])

housing\_prepared = full\_pipeline.fit\_transform(housing)

1.1.3.FeatureUnion 并联管道

把若干节点并联成一个新的结点。这个并联的各个结点同时处理同一个输入数组，生成各自的数组结果后，列拼接成为一个更大的特征向量数组。

from sklearn.pipeline import FeatureUnion

from sklearn.decomposition import PCA

from sklearn.decomposition import KernelPCA

estimators = [('linear\_pca', PCA()), ('kernel\_pca', KernelPCA())]

combined = FeatureUnion(estimators)

# **三者结合很有用,但是ColumnTransformer 可以实现 FeatureUnion 的功能**

1.1.4.FunctionTransformer

可以将任何构造函数包装成一个类,统一 fit 、transform方法，放进pipeline使用非常方便。但是记住构造函数设计用来接受数组，输出数组。构造函数的参数，通过FunctionTransformer创造对象的时候除了传入函数，另外指定一个参数kw\_args={"add\_bedrooms\_per\_room": False}来定义

from sklearn.preprocessing import FunctionTransformer

def add\_extra\_features(X, add\_bedrooms\_per\_room=True):

rooms\_per\_household = X[:, rooms\_ix] / X[:, household\_ix]

population\_per\_household = X[:, population\_ix] / X[:, household\_ix]

if add\_bedrooms\_per\_room:

bedrooms\_per\_room = X[:, bedrooms\_ix] / X[:, rooms\_ix]

return np.c\_[X, rooms\_per\_household, population\_per\_household,

bedrooms\_per\_room]

else:

return np.c\_[X, rooms\_per\_household, population\_per\_household]

attr\_adder = FunctionTransformer(add\_extra\_features, validate=False,

kw\_args={"add\_bedrooms\_per\_room": False})

attr\_adder.fit\_transform(housing.values)

1.2 Pipline总结：

Pipleline中最后一个之外的所有estimators（估计器）外都必须同时是转换器（transformers），最后一个estimator可以是任意类型（classifier，regresser），没有模型也没关系当成转换器用,如果最后一个estimator是个分类器，则整个pipeline就可以作为分类器使用，如果最后一个estimator是个聚类器，则整个pipeline就可以作为聚类器使用。

组成管道对象后，管道对象的方法以最后一个节点的方法为准，运用fit方法，除了最后一步，前面步骤都会用fit\_transform方法，如果管道对象最后一步是模型，用predict方法，前面处理步骤自动会运用transform方法。

注意：如果最后一步是模型的管道对象fit()会传入两个参数，只对前面那个X用前面数据处理，y不会被处理。但有的估计器需要y参数。

pipline.get\_params()获得所有节点超参数。

pipline.set\_params(lr\_\_fit\_intercept = False)修改某一结点的超参数

Pipline[‘节点名’]直接得到这个结点对象。

pipline.score(X\_test, Y\_test) 缺省方法评分。可以用sklearn中都在sklearn.metrics类下更多的评分办法。

from sklearn.model\_selection import cross\_val\_score

scores = cross\_val\_score(dtc,X,Y,cv=5,scoring='f1') 用训练集进行交叉验证。模型评估效果更客观。

管道对象除第一个节点的输入其他都采用数组。

1.3管道构造过程

1.3.1在进入管道前先用from sklearn.model\_selection import train\_test\_split 随机划分或者按重要属性分层划分出测试训练集集后 ，单对训练集进行处理 ，因为划分函数不是估计转换器不能进入管道操作。时间序列如何来划分？时间序列如何交叉验证。

训练集首先进行数据探索，寻找相关关系。确定模型 确定是否需要增加计算指标

1.3.2正式开始处理数据

拆分出输入变量和目标变量，，只看输入变量（输出变量也需要处理？）

检查缺失值，变量多可以选择扔掉缺失值多的变量，样本多可以选择仍掉有缺失值的样本，

训练集中的输入变量中的数值变量和类别变量分开，采用不同的处理方法，用ColumnTransformer对象函数来处理

一般采用from sklearn.impute import SimpleImputer 打头 因为输入df，输出数组。优先进行缺失值填补。数值型采用平均数，类别值采用众数

类别变量进行一个编码转化from sklearn.preprocessing import OrdinalEncoder 或者OneHotEncoder(sparse=False),接受的Series、df 、数组。输出数组

数值型变量的计算转化，用 FunctionTransformer包装转化函数，转化函数考虑输入的是数组，因为在管道内。输出全部数值型数据。

管道估计器填补缺失值，计算指标，标准化。标准化类、编码转化。管道内的节点全部拥有除了最后一个其他必须拥有fit transform 方法。

2.管道模型的评分测试。

2.1人为验证数据集。指定的训练集fit，再用指定的验证集preidct，后用评分函数。训练集中的80%用来fit，剩下的20%作为验证集来评估，此方式不推荐。

评分函数在sklearn中都在sklearn.metrics类下，务必记住哪些评分适合分类，那些适合回归，不能混着用。

分类模型：

accuracy\_score(y\_test,y\_pre)准确率得分，是模型分类正确的数据除以样本总数 ，model.score(x\_test,y\_test)，效果一样。

classification\_report(y\_test,y\_log\_pre)。其中的各项得分的avg/total 是每一分类占总数的比例加权算出来的

print(classification\_report(y\_test,y\_log\_pre))

precision recall f1-score support

0 0.87 0.94 0.90 105

1 0.91 0.79 0.85 73

avg / total 0.88 0.88 0.88 178

confusion\_matrix(y\_true, y\_pred)，混淆矩阵，用来评估分类的准确性。有的分类问题，实际样本中1000个A，10个B，如果最后分类大多数B都被预测错误了，但依据其他评估方法，得分反而很高(因为A的数目相对太多导致的)，召回率和精确率的区别。

1. **网格搜索与随机搜索**

3.1对于计算量不大或者计算能力强的可以使用sklearn的网格搜索

from sklearn.model\_selection import GridSearchCV

from sklearn.ensemble import RandomForestRegressor

param\_grid = [

# try 12 (3×4) combinations of hyperparameters

{'n\_estimators': [3, 10, 30], 'max\_features': [2, 4, 6, 8]},

# then try 6 (2×3) combinations with bootstrap set as False

{'bootstrap': [False], 'n\_estimators': [3, 10], 'max\_features': [2, 3, 4]},

]

forest\_reg = RandomForestRegressor(random\_state=42)

# train across 5 folds, that's a total of (12+6)\*5=90 rounds of training

grid\_search = GridSearchCV(forest\_reg, param\_grid, cv=5,

scoring='neg\_mean\_squared\_error', return\_train\_score=True)

#模型默认超参数+超参数搜索网格+交叉验证+选择标准 共同组成一个不带数据集的交叉网格估计器，可以应用fit方法对已有训练集进行超参数搜索

grid\_search.fit(housing\_prepared, housing\_labels)

grid\_search.best\_params\_ #最佳超参数

grid\_search.best\_estimator\_#获得forest\_reg的最佳节点设置

grid\_search.cv\_results\_

for mean\_score, params in zip(cvres["mean\_test\_score"], cvres["params"]):

print(np.sqrt(-mean\_score), params)#查看所有搜索结果

pd.DataFrame(grid\_search.cv\_results\_)

3.2计算量太大可以用随机搜索。

from sklearn.model\_selection import RandomizedSearchCV

from scipy.stats import randint

param\_distribs = {

'n\_estimators': randint(low=1, high=200),

'max\_features': randint(low=1, high=8),

}

forest\_reg = RandomForestRegressor(random\_state=42)

rnd\_search = RandomizedSearchCV(forest\_reg, param\_distributions=param\_distribs,

n\_iter=10,cv=5, scoring='neg\_mean\_squared\_error', random\_state=42)

rnd\_search.fit(housing\_prepared, housing\_labels)

查看随机搜索结果

cvres = rnd\_search.cv\_results\_

for mean\_score, params in zip(cvres["mean\_test\_score"], cvres["params"]):

print(np.sqrt(-mean\_score), params)

**回归模型**

mean\_absolute\_error（MAE、平均绝对误差）

mean\_squared\_error（MSE、均方误差）

r2\_score（R^2、可决系数）

2.2训练集自动划分验证集，交叉验证 传入全管道全量训练集的x数组 和 y目标值。交叉验证函数自动输出每折的评分。样本条件所限的话，取消测试集评估，直接上线预测。

from sklearn.model\_selection import cross\_val\_score

scores = cross\_val\_score(tree\_reg, housing\_prepared, housing\_labels,

scoring="neg\_mean\_squared\_error", cv=10)

因为评分函数结果都一致为越大越小，对于越小越好的mse，评分函数给出负值。

lin\_rmse\_scores = np.sqrt(-lin\_scores).mean()

交叉验证cross\_val\_score的scoring参数：

分类：accuracy(准确率)、f1、f1\_micro、f1\_macro（这两个用于多分类的f1\_score）、precision(精确度)、recall(召回率)、roc\_auc

回归：neg\_mean\_squared\_error（MSE、均方误差）、r2

聚类：adjusted\_rand\_score、completeness\_score等 【这一块我没怎么用过】

3.在交叉验证后确定了管道中的所有估计器和预测器后，超参数的调优.

3.1网格搜索

再确定了管道流程特备是模型后，需要对模型调优，进行超参数搜索。已选择最佳超参数。

param\_grid = dict(features\_\_pca\_\_n\_components=[1, 2, 3],

#注意估计器名称后两个下划线

features\_\_univ\_select\_\_k=[1, 2],

svm\_\_C=[0.1, 1, 10])

grid\_search = GridSearchCV(pipeline, param\_grid=param\_grid, verbose=10)

grid\_search.fit(X, y)

print(grid\_search.best\_estimator\_)

1. **常用模型**

**空值、判断**

#判断数据框里面所有不是有限数据的数，可以排除null，np.nan,np.inf

sup\_water[~(np.isfinite(sup\_water).all(axis=1))]

特征工程：扩增特征变量，或者浓缩特征变量，或者对特征变量进行选择。将文字便来给你转化为编码。具体编码转换方式见数据处理笔记。

from sklearn.preprocessing import PolynomialFeatures

poly\_features = PolynomialFeatures(degree=3,include\_bias=1)

X\_poly = poly\_features.fit\_transform(sup\_water[['最低温度','最高温度','平均温度']])

**2.1回归任务**

回归任务一般都要预先判断各特征与输出结果的相关关系，考虑到有些指标为曲线相关可以利用sklearn进行多项式变化，然后再做相关矩阵。

#将poly数组转化程dataframe画相关矩阵，np也有相关矩阵函数，但带列名更加清楚。

sup\_water\_poly= pd.DataFrame(data=X\_poly,columns=poly\_features.get\_feature\_names())

sup\_water\_poly['水厂供水总量'] = sup\_water['水厂供水总量'].values

sup\_water\_poly.corrwith(sup\_water\_poly['水厂供水总量']).sort\_values(ascending=False)

pd还提供做相关矩阵和相关性图的函数，但因为指标太多，因此省去。

sup\_water.corr()

pd.plotting.scatter\_matrix(sup\_water,figsize=(12,8),alpha=0.2)

* + 1. **时间序列**

**2.1.1.1 时间序列独特的离群值判断：**

am = sup\_water['最低温度'].rolling(window=3,min\_periods=0,center=True).mean()

diff = sup\_water['最低温度'].values-am

sup\_water.loc[diff.abs()>diff.std(ddof=0)\*3,]

sup\_water['上限']=am+diff.std(ddof=0)\*3

sup\_water['下限']=am-diff.std(ddof=0)\*3

**2.1.1.2时间序列的交叉验证原理：**

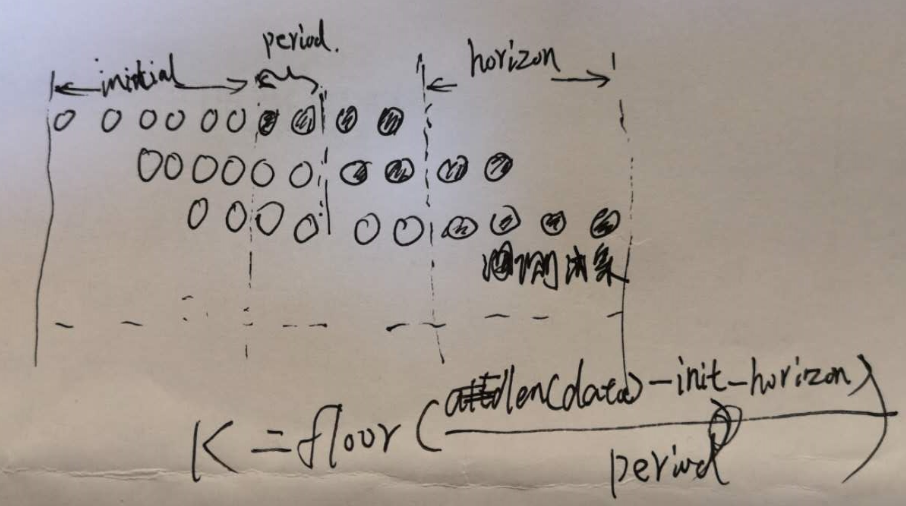
因为时间序列的样本之间是无法交换的，所以没办法像KFold交叉验证一样把数据集切分成若干份训练集和测试集（每份训练集的模型超参数是一样的）。一个比较好的思路是按照时间顺序设置。这里有三个参数：

horizon: 模型预测的范围，如从cutoff点开始数未来30天

period: 每两个 cutoff 点之间的间隔，也就是间隔多少期开始新的一份训练集。cutoff为horizon的开始点。

initial: 用于训练的日期范围，如730天

K计算方法如下图



计算举例：

#initial = 730,horizon = 365,K=10 ,period=7

Len(data)=dt.date(2020,3,12)-dt.date(2017,1,1)=1167

K=(1167-730-365)/7=10

计算所需数据长度 initial+period\*（k-1）+horizon+1 如果不用最后留一段test数据的话就用K-1.

可以自己写一个函数将数据集分多对训练集和测试集

horizon,initial,period=30,1500,30

df\_train\_list,df\_test\_list=ts\_model\_selection.train\_test\_split(df,

horizon='{}days'.format(horizon),

initial='{}days'.format(initial),

period='{}days'.format(period))

for k in range(len(df\_train\_list)):

#print(k)

df\_train,df\_test=df\_train\_list[k],df\_test\_list[k]

……对每一个df\_train训练并做预测。然后和df\_test的y进行对比计算mape或者rmse，收集在一个list里面。可以计算平均值。

**2.1.1.3 Fbprophet**

m = Prophet(growth='linear',#'logistic'

changepoints=None, #List of dates,不指定系统自动选择

n\_changepoints=25, #25只是变点的最上限。具体多少还要看range和scale。

changepoint\_range=0.8,

yearly\_seasonality='auto',#设定是否添加年度季节性，可以为‘auto’，True，False。

weekly\_seasonality='auto', #设定是否添加周季节性

daily\_seasonality=False, #设定是否添加日内的周期性

holidays=holidays, #不指定为none，holidays可以有一列可选列prior\_scale指明每个节假日的现有规模。

seasonality\_mode='additive', #'multiplicative'forecast['yhat']=forecast['trend']\*(1+forecast['multiplicative\_terms'])。季节性波动跟随trend变化。

seasonality\_prior\_scale=10.0, #季节先验规模，可以在单独指定

holidays\_prior\_scale=10.0, #节假日成分模型的强度，数字越大假期影响越大，除非在节假日输入中被重写。

changepoint\_prior\_scale=0.05, #变点先验规模，调整变点的识别严格程度。较大的值会产生较多的改变点，较小的值则产生较少的转折点。

#mcmc\_samples=0, #提供采样数量可以显示季节周期的不确定性。一般不设置。

#interval\_width=0.8, #预测的置信区间。

#uncertainty\_samples=1000,

stan\_backend=None

)

m.add\_seasonality(name='weekly', period=7, fourier\_order=3, prior\_scale=5)#可以通过指定mode='addiative'或'multiplicative'作为参数来覆盖之前的设定

m.add\_seasonality(name='yearly',period=365.25,fourier\_order=7, prior\_scale=2.6)#月用‘monthly’、季度用'quarterly'

m.add\_regressor('最高温度',prior\_scale=10)#prior\_scale不提供按照节假日数据，默认standardize='auto'（if notbinary)。True,or False.

m.fit(sup\_water\_train)

1.数据的选择和预处理

np.log()或者归一化处理

初始周期应该足够长，以捕获模型的所有特性，特别是季节性和额外的回归变量:对年的季节性至少保证一年，对周的季节性至少保证一周，等等。

2.y异常值处理。

异常值最好的方法是移除它们，而 Prophet 是能够处理缺失数据的。如果在历史数据中某行的值为空（ NA ），但是在待预测日期数据框 future 中仍保留这个日期，那么 Prophet 依旧可以给出该行的预测值。过大的异常值会破坏季节性效应的估计，最好移除。

3.如果是长期预测有上限的指标可以设置参数Prophet(growth=’logistic’)增加cap或者floor进行限制增长极限。Cap/floor在future数据框里也要具备。

4.增加回归量比如天气温度可以在网上爬取，作为新列添加到train和futrue数据框里。

Train是爬取历史天气，futrue是爬取未来的预测天气。

5.假期或者特殊事件 是单独的dataframe。包括测试数据框和futrue数据框都不用再对holiday进行设置。涉及以农历节日的假期，需要用以下代码或者阳历日期

import sxtwl

lunar = sxtwl.Lunar()

solar\_day = lunar.getDayByLunar(2019,12,30) #2020年春节

solar\_date = dt.date(solar\_day.y, solar\_day.m, solar\_day.d)

'lower\_window': -14, 'upper\_window': 14，分别设置每日节日时间点向前或者向后延续的影响的天数。

对于新冠疫情采用分阶段预测。

6.季节性周期值，调整傅里叶级数让周期平滑，或者过拟合。

如果训练集的周期有规律的不完整，比如每天只有0-6点的数据，那么天的季节性在一天剩下的时间里是不受约束的，估计也不准确。周期性影响量的判断不完整，未来dataframe里面的时间窗和历史数据也应该保持一致。比如每周只有工作日的数据，未来也应只有工作日。

同样的历史数据集使用的是月数据。只有每个月第一天的数据，对于其他天的周期效应是不可测的。因此当可以通过在 make\_future\_dataframe 中传入频率参数的频率必须和历史训练数据集保持一致。

Prophet 只会返回趋势中的不确定性和观测值噪声的影响。你必须使用贝叶斯取样的方法来得到季节效应的不确定性，可通过设置 mcmc.samples 参数（默认下取 0 ）来实现。

7、改变突变点

变化速率灵活性更大时（通过增大参数 changepoint\_prior\_scale 的值），预测的不确定性也会随之增大。原因在于如果将历史数据中更多的变化速率加入了模型，也就代表我们认为未来也会变化得更多，就会使得预测区间成为反映过拟合的标志。

增加回归因素

8、评估预测效果

有自动交叉验证函数

#划分数据

from fbprophet.diagnostics import cross\_validation

df\_cv = cross\_validation(m, initial='730 days', period='7 days', horizon = '365 days')

rmse\_ave = df\_cv.groupby('cutoff').apply(fun\_rmse).mean()

df1.loc[i-1] = [param,rmse\_ave]

把每个超参数以及 rmse\_ave 记录进入数据框以供选择

df1 = df1.sort\_values(['rmse']).head(10)

for param,rmse in zip(df1['param'],df1['rmse']):

print(param,rmse)

查看从预测未来1days到365days的rmse，mape，coverage等参数

from fbprophet.diagnostics import performance\_metrics

df\_p = performance\_metrics(df\_cv)

可以指定某个参数随着预测时期的增长的每个数值和平均值画出来

from fbprophet.plot import plot\_cross\_validation\_metric

fig = plot\_cross\_validation\_metric(df\_cv, metric='mape')

9.生成未来预测数据框

m.make\_future\_dataframe ( period,freq =’D’,include\_history = True)

m:Prophet对象

Periods:未来要预测时间范围，整型

Freq:‘day’，‘week’，‘month’，‘quarter’，‘year’，1 (1 sec)，60 ( 1 minute ) or 3600 ( 1 hour )

include\_history:布尔值，是否包含历史日期

将cap，regresssor添加进数据框。如果需要历史数据的预测，可以和剔除y的历史数据框合并成新的future

10.利用交叉验证确定超参数后预测

m.predict (df = futrue, …)

**SARIMAX系列模型**

前提时间序列是稳定的

1.常量的均值 弹簧的轴不歪，原因趋势（trend）-数据随着时间变化。比如说升高或者降低。解决办法：差分

2.常量的方差 弹簧的横向均匀，不扩大。原因数据在特定的时间段内变动。比如说节假日，或者活动。解决办法：对数变换，或者平滑法具体分为移动平均法和指数平均法。

3.与时间独立的自协方差 弹簧没有被暴力拉扯。原因周期变了。

可以通过观察，计算累计方差，累计平均数画线观察。也可以通过teststationarity(ts)检验。

更成熟，灵活的方法。将时间序列按照不同因素分解不同序列相乘或者相加。一般分成三种序列，趋势序列，周期序列，残差序列。运用sm.tsa.seasonal\_decompose（）函数。可以分别对趋势、周期、残差进行预测。再相乘或者相加。

from statsmodels.tsa.arima\_model import ARIMA 使用前 先用移动平均剥离周期性，输出的预测值是差分值+残差，。真实数据需要去反差分，再反移动平均，甚至np.exp（）反log去还原。

缺点，特殊假期无法模拟，外生变量影响没有体现，不能捕捉非线性，，用于规则变化周期性，线性增长残差平稳的。

Sarimax 模型不用反差分，不用反移动平均。更方便。缺点参数太多不好确定，使用网格搜索。

Ar，ma ，都要求平稳序列。一切源于稳定的时间序列的均值开始，如果差分均值就是差值，ar 是均值加上前p期数值乘以各自系数的回归方程。ma是前q期的均值，加上每期实际数与预测数的误差项目乘以系数的回归方程，得到本期的预测值。

Holt

Expotion

是否就是其他模型的变种。

2.1.2线性回归

sklearn.linear\_model.LinearRegression()

.intercept\_[0]看截距，.coef\_[0][0]看系数。

2.1.3分类回归树

随机森林

神经网络

支持向量机

分类任务

聚类任务