基础

**监督模型模板**

model=Model()

model.fit(X,y)

model.predict(Xnew) #输出预测标签

model.predict\_proba(Xnew) 或model.decision\_function(Xnew)#输出预测概率

**保存/加载模型**

import joblib

joblib.dump(model,’model.pkl’) #保存

model=joblib.load(‘model.pkl’) #加载

**网格搜索**

from sklearn.model\_selection import GridSearchCV

grid=[

{参数名1:参数1取值列表, 参数名2:参数2取值列表,…},

{参数名1:参数1取值列表, 参数名2:参数2取值列表,…},

…,

]

grid\_search=GridSearchCV(model,grid,cv=折数,scoring=评分方法,return\_train\_score=True)

grid\_search.fit(X,y, \*validation\_data)

#grid中不同的{}代表了不同的参数搜索组合

**随机搜索**

from sklearn.model\_selection import RandomizedSearchCV

param\_distribution=[

{参数名1:参数1取值范围,参数名2:参数2取值范围,…},

{参数名1:参数1取值范围,参数名2:参数2取值范围,…},

…,

]

rnd\_search=RandomizedSearchCV(model,param\_distribution,n\_iter=取值次数,cv=折数)

rnd\_search.fit(X,y,\*validation\_data)

**查看不同超参数组合的结果**

grid\_search.cv\_results\_

**查看最佳超参数**

grid\_search.best\_params\_

**访问最佳模型**

model=rnd\_search.best\_estimator\_.model

**随机搜索**

RandomizedSearchCV()

**拷贝模型**

from sklearn.base import clone

model\_backup=clone(model)

数据预处理

## 预处理工具

**估算器**：根据数据集对某些参数进行估算的类方法

-执行：fit(data)

**转换器**：根据原始数据对某些数据进行转换的类方法

-拟合(基于数据获取转换参数)：fit(data)

-转换：transform(data)

-拟合并转换（获取转换参数后立刻将转换应用于数据上）：fit\_transform(data)

**预测器**：根据给定的数据进行预测的类方法

-执行：predict(X)

-评估：score(X)

**转换流水线构造（用以同时实现多个转换方法）**

from sklearn.pipeline import Pipeline

pipeline=Pipeline([

(转换器名1,转换器1), (转换器名2,转换器2),…

])

data\_transformed=pipeline.fit\_transform(data)

**分列转换器**

from sklearn.compose import ColumnTransformer

transformer=ColumnTransformer([

(转换器名1,转化器1,列名列表1), (转换器名2,转化器2,列名列表2),…

])

data\_transformed=transformer.fit\_transform(data)

#所运用的转化器可以是构造的转换流水线

## 标准化/数值化

**标准化**

from sklearn.preprocessing import StandardScaler

standardscaler = StandardScaler()

data\_scaled=standardscaler.fit\_transform(data)

data\_recovered= standardscaler. inverse\_transform(data\_scaled)

**！**X\_train和X\_test应使用相同初始化的标准化器

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

**文本标签数值化**

from sklearn.preprocessing import OrdinalEncoder

#当y是dataframe文本列时还需要以下转换

#y\_test=y\_test.values.reshape(-1,1)

ordinal\_encoder=OrdinalEncoder()

labels\_y=ordinal\_encoder.fit\_transform(y)

**查看数值化的标签所对应的原标签**

ordinal\_encoder.categories\_

**文本标签one-hot化**

from sklearn.preprocessing import OneHotEncoder

#当y是dataframe文本列时还需要以下两步

#y\_test=y\_test.values

#y\_test=y\_test.reshape(-1,1)

onhotencoder=OneHotEncoder(handle\_unknown="ignore")

labels\_y=onehotencoder.fit\_transform(y)

**查看数值化的标签所对应的原标签**

labels\_y.categories\_

**labels\_y转化为numpy矩阵**

labels\_y.toarray()

**数值标签one-hot化**

from sklearn.preprocessing import LabelBinarizer

#当y是dataframe文本列时还需要以下两步

#y\_test=y\_test.values

#y\_test=y\_test.reshape(-1,1)

labels\_y = LabelBinarizer().fit\_transform(y)

#y是列表，labels\_y是y正交化后的二维矩阵，用一个向量代表一种类型，如[1,0,0]代表三种类型中的第一种

**one-hot标签转数值标签**

labels\_y=np.argmax(y,axis=1)

**根据样本量计算各标签权重**

from sklearn.utils.class\_weight import compute\_class\_weight

class\_weight=compute\_class\_weight('balanced',np.unique(y),y)

## 缺失值估算

**缺失值估算**

from sklearn.impute import SimpleImputer

imputer=SimpleImputer(Strategy=”median”/”mean”/…)

imputer.fit(data)

**查看估算结果**

imputer.statistics\_

## 数据切分

**数据分堆**

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size=0.k, random\_state=n,stratify=分层抽样用list)

#随机拿0.k的样本做测试集, 函数返回四堆数据：X、Y的训练集和测试集；random\_state是随机数种子

**分层分堆：**

split=StratifiedShuffleSplit(n\_splits=1,test\_size=0.1,random\_state=1)

train\_index, test\_index=list(split.split(X,y))[0]

strat\_X\_train=X.iloc[train\_index,:]

strat\_y\_train=y.iloc[train\_index]

strat\_X\_test=X.iloc[test\_index,:]

strat\_y\_test=y.iloc[test\_index]

**按比例分层抽样的n次可重复数据分堆**

from sklearn.model\_selection import StratifiedShuffleSplit

split=StratifiedShuffleSplit(n\_splits=n,test\_size,random\_state)

for train\_index, test\_index in split.split(X,分层抽样基于的数据可以是X的某列或y):

strat\_X\_train=X.iloc[train\_index,:]

strat\_y\_train=y.iloc[train\_index]

strat\_X\_test=X.iloc[test\_index,:]

strat\_y\_test=y.iloc[test\_index]

**交叉检验：**

1. **自建评估架构**

**from sklearn.model\_selection import StratifiedKFold**

**kf = StratifiedKFold (n\_splits=n,random\_state)** #按照索引值分为n份

**for train\_index, test\_index in kf.split(X,分层抽样基于的列数据):** # 建模、测试各份样本

**X\_train, X\_test = X.iloc[train\_index,:], X.iloc[test\_index,:]** #利用分好份的索引值来调用X

**y\_train, y\_test = Y.iloc [train\_index,:], Y.iloc [test\_index,:]** #利用分好份的索引值来调用Y

… …

1. **直接输出评估**

**输出评估分数：**

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

scores=cross\_val\_score(model,X,y,scoring=”neg\_mean\_squared\_error”/’’accuracy’’/’’recall’’/’f1’,cv=折数)

**输出每折的预测结果：**

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

y\_pred=cross\_val\_predict(model,X,y,cv=折数,method=’predict’/’predict\_proba’)

评估

## 均方误差

from sklearn.metrics import mean\_squared\_error

mse=mean\_squared\_error(y,y\_pred)

## Confusion Matrix

from sklearn.metrics import confusion\_matrix

confusion\_matrix(y,y\_pred,labels=指定标签排列顺序的列表)

#横是真纵是预测

常用热度图来对Confusion Matrix做可视化

**转Dataframe:**

confusion\_matrix=pd.DataFrame(confusion\_matrix(y\_test,y\_pred),index,columns)

## 精度/召回率/F1值/Accuracy

from sklearn.metrics import precision\_score,recall\_score,f1\_score, accuracy\_score

precision\_score(y,y\_pred)

recall\_score(y,y\_pred)

f1\_score(y,y\_pred)

accuracy\_score(y,y\_pred)

## Brier Score(二分类问题的MSE)

from sklearn.metrics import brier\_score\_loss

brier\_score\_loss(y\_true,y\_probs)

## 多标签分类F1值

f1\_score(y,y\_pred,average)

#average=’’，各标签F1值取平均

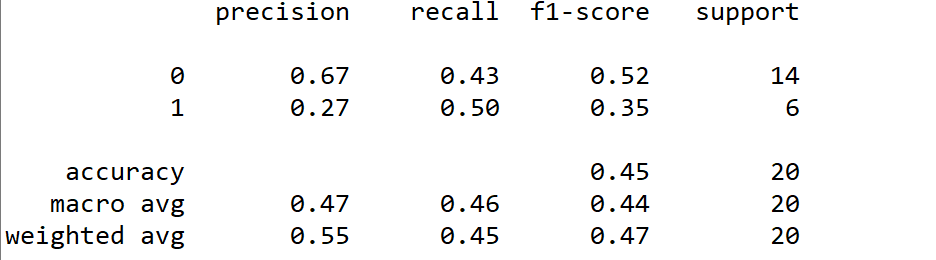
#average=’weighted’，各标签F1值取权重值

## 模型分类报告

from sklearn.metrics import classification\_report

print (classification\_report(ytest, predictions))

输出如下:



## ROC曲线(用于二分类问题)

from sklearn.metrics import roc\_curve, auc

fpr,tpr,thresholds=roc\_curve(y,y\_pred)

plt.plot(fpr,tpr)

plt.title('ROC for Test')

plt.xlabel('FPR')

plt.ylabel('TPR')

plt.show()

#y\_pred在此处为预测的决策分数而非实际的标签

**AUC值(Area under ROC)**

from sklearn.metrics import roc\_auc\_score

print('AUC:',roc\_auc\_score(y\_test,y\_pred))

**用plotly画**

import plotly.express as px

fig=px.line(data\_frame=pd.DataFrame({'FPR':fpr,'TPR':tpr,'Threshold':thresholds}),x='FPR',y='TPR',custom\_data=['Threshold'],title='ROC for Test')

fig.update\_traces(

hovertemplate="<br>".join([

"Threshold: %{customdata[0]}",

"FPR: %{x}",

"TPR: %{y}"

])

)

fig.show()

## 概率校准曲线(用于分类模型)

from sklearn.calibration import calibration\_curve

fraction\_of\_positives,mean\_predicted\_value=calibration\_curve(y\_true,y\_probs,n\_bins,normalize=False)

plt.plot(mean\_predicted\_value, fraction\_of\_positives,’b’,label=’Model Probs’)

plt.plot([0,1],[0,1],label=’Theoretical Probs’)

plt.legend()

plt.show()

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

分类模型

## OneVsRest策略

from sklearn.multiclass import OneVsRestClassifier

model=OneVsRestClassifier(某一分类模型)

model.fit(X,y)

## OneVsOne策略

from sklearn.multiclass import OneVsOneClassifier

model=OneVsOneClassifier(某一分类模型)

model.fit(X,y)

## 朴素贝叶斯

from sklearn.naive\_bayes import CategoricalNB

from sklearn.naive\_bayes import GaussianNB

from sklearn.naive\_bayes import MultinomialNB

## 感知机

from sklearn.linear\_model import Perceptron

## 决策树

from sklearn.tree import DecisionTreeClassifier

model=DecisionTreeClassifier(

max\_depth=最大深度,criterion=’entropy’/’gini’,min\_samples\_split=节点最小样本数,min\_samples\_leaf=叶节点最小样本数,max\_leaf\_nodes=最大叶节点数量,max\_features=用于分类的特征数,random\_state)

mode.fit(X,y)

**决策树可视化**

#首先要安装graphviz-2.38.msi，默认装在C:/Program Files (x86)/Graphviz2.38

from IPython.display import Image

from sklearn import tree

import pydotplus

from six import StringIO

import os

#为graphviz添加环境变量

os.environ["PATH"] += os.pathsep + 'C:/Program Files (x86)/Graphviz2.38/bin/'

dot\_data = StringIO()

tree.export\_graphviz(model, out\_file= dot\_data,

feature\_names=特征名, #对应的特征

class\_names=出现的标签名, #对应类别

filled=True, rounded=True,

special\_characters=True)

graph = pydotplus.graph\_from\_dot\_data(dot\_data.getvalue().replace('helvetica','"Microsoft YaHei"')) #冒号没有打错，中文字符要加双引号

graph.write\_png(r'tree.png') #保存图像至本地

Image(graph.create\_png())

#中文乱码解决方案：<https://www.bilibili.com/read/cv4930018/>

**决策树可视化2**

from sklearn import tree

tree.plot\_tree(dt,feature\_names = features, class\_names=labels,filled = True)

## 随机森林

from sklearn.ensemble import RandomForestClassifier

model=RandomForestClassifier(n\_estimators=树数.max\_leaf\_nodes=最大叶节点数,n\_jobs=-1)

model.fit(X,y)

## 极端随机树

from sklearn.ensemble import ExtraTreesClassifier

model=ExtraTreesClassifier(n\_estimators=树数.max\_leaf\_nodes=最大叶节点数,n\_jobs=-1)

model.fit(X,y)

**查看特征重要性**

model.feature\_importances\_

## SVM

**线性SVM**

from sklearn.svm import SVC

svc=SVC(kernel=’linear’,C=正则值’,probability=True)

svc.fit(X,y)

from sklearn.svm import LinearSVC

svc=LinearSVC(loss=’hinge’,C=正则值,probability=True)

#方法①的运行速度更快

**多项式内核SVM**

from sklearn.svm import SVC

svc=SVC(kernel=’poly’,degree=最高阶数,coef=高阶或低阶项的影响程度,C=正则值,probability=True)

svc.fit(X,y)

**高斯内核SVM**

from sklearn.svm import SVC

svc=SVC(kernel=’rbf’,gamma=’高斯核指数的分母’,C=正则值,probability=True)

svc.fit(X,y)

#高斯内核的分母gamma控制着高斯内核的钟形函数的宽度，减小gamma会使得钟形变宽，每个样本点的影响范围扩大，模型泛化能力扩大但也更易欠拟合

#C正则值越高，正则化程度越低

**sigmoid内核SVM**

from sklearn.svm import SVC

svc=SVC(kernel=’sigmoid’, C=正则值,probability=True)

svc.fit(X,y)

**对每个样本输出每个类别的预测分值**

svc.decision\_function(X)

## KNN

from sklearn.neighbors import KNeighborsClassifier

knn=KNeighborsClassifier()

knn.fit(X,y)

#该KNN分类器可以处理多标签分类问题

## 逻辑回归

from sklearn.linear\_model import LogisticRegression

model=LogisticRegression()

model.fit(X,y)

**输出预测可能性**

model.predict\_proba(X)

## 投票分类器

from sklearn.ensemble import VotingClassifier

voting\_model=VotingClassifier(

estimators=[(‘分类器名1’,分类器1), (‘分类器名2’,分类器2),…],voting=’hard’/’soft’)

voting\_model.fit(X,y)

## Bagging和Pasting方法

from sklearn.ensemble import BaggingClassifier

bagging\_model=BagginClassifier(model,n\_estimators=分类器个数,max\_samples=样本子采样大小,bootstrap=True/False,n\_jobs=-1,obb\_score=是否启用包外评估,bootstrap\_features=是否对特征空间子采样,max\_features=特征子采样时的最大子采样特征数)

bag\_model.fit(X,y)

#bootstrap=True，是Bagging方法；False是Pasting方法

#如果model由predict\_proba()方法，则BaggingClassifier执行软投票

**查看包外评估分数**

bagging\_model.obb\_score\_

**查看包外预测值**

bagging\_model.obb\_decision\_function\_

## AdaBoost方法

from sklearn.ensemble import AdaBoostClassifier

ada\_model=AdaBoostClassifier(\*base\_estimator=model,

n\_estimators=基分类器数,algorithm=’SAMME.R’/’SAMME’,learning\_rate)

ada\_model.fit(X,y)

#基分类器model不指定的话，默认为决策树

#当基分类器有predict\_proba()方法时，用’SAMME.R’通常会更好

## 梯度提升树（GBRT）

from sklearn.ensemble import GradientBoostingClassifier

gbrt\_model = GradientBoostingClassifier()

gbrt\_model.fit(X\_train,y\_train)

## XGBoost

import xgboost

model=xgboost.XGBClassifier()

model.fit(X,y,eval\_set=[(X\_val,y\_val)],early\_stopping\_rounds=触发提前停止的未改善轮数)

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

回归模型

## 线性回归

from sklearn.linear\_model import LinearRegression

model=LinearRegression()

**查看截距和系数**

model.intercept\_

model.coef\_

## 多元线性回归

from sklearn.preprocessing import PolynomialFeatures

from sklearn.linear\_model import LinearRegression

poly\_transformer=PolunomialFeatures(degree=次幂,include\_bias=False)

X\_poly=poly\_transformer(X)

model=LinearRegression()

LinearRegression.fit(X\_poly,y)

**#X\_poly包含了X的原始数据和X的2到n次幂数据**

**！当X有多个特征时，**PolynomialFeatures()还将包含不同特征的不同次幂的乘积组合

如：X含两个特征a,b，则degree=3还会包含特征ab,a2b,ab2

## 岭回归

from sklearn.linear\_model import Ridge

model=Ridge(alpha=惩罚系数)

model.fit(X,y)

## Lasso回归

from sklearn.linear\_model import Lasso

model=Lasso(alpha=惩罚系数)

model.fit(X,y)

## 弹性网络(Lasso与Ridge的线性加和模型)

from sklearn.linear\_model import ElasticNet

model=ElasticNet(alpha=惩罚系数,l1\_ratio=L1正则项权重)

model.fit(X,y)

## PLSR回归

from sklearn.cross\_decomposition import PLSRegression

pls = PLSRegression(n\_components=k,scale=False)

#取k个主成分，scale=True表示将X数据做自标准差标准化

## 决策树

from sklearn.tree import DecisionTreeRegressor

model=DecisionTreeRegressor()

model.fit(X,y)

## 随机森林回归

from sklearn.ensemble import RandomForestRegressor

model=RandomForestRegressor()

model.fit(X,y)

## AdaBoost方法

from sklearn.ensemble import AdaBoostRegressor

ada\_model=AdaBoostClassifier(\*base\_estimator=model,

n\_estimators=基分类器数,learning\_rate,loss={‘linear’, ‘square’, ‘exponential’})

ada\_model.fit(X,y)

#基分类器model不指定的话，默认为决策树

## 极端随机树

from sklearn.ensemble import ExtraTreesRegressor

model=RandomForestClassifier(n\_estimators=树数.max\_leaf\_nodes=最大叶节点数,n\_jobs=-1)

model.fit(X,y)

## 梯度提升树（GBRT）

from sklearn.ensemble import GradientBoostingRegressor

model=GradientBoostingRegressor(max\_depth=单棵树最大深度,n\_estimators=树数,learning\_rate,subsample=训练每棵树时的样本子采样比例, criterion={‘friedman\_mse’, ‘mse’, ‘mae’})

model.fit(X,y)

#learning\_rate控制单棵树的权重，learning\_rate越低，就需要更多树来拟合，但泛化会改善

#criterion是评价节点分裂优良的指标

**输出训练了1至n棵树时模型的预测值(结合metrics可以寻找最佳的树数超参数)**

model.staged\_predict(X)

## XGBoost

import xgboost

model=xgboost.XGBRegressor()

model.fit(X,y,eval\_set=[(X\_val,y\_val)],early\_stopping\_rounds=触发提前停止的未改善轮数)

## SVM

分类SVM模型中加上用于控制间隔大小的超参数epsilon即可，通常取(0,2), epsilon越大，间隔越大

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

降维

## PCA

from sklearn.decomposition import PCA

pca=PCA(n\_components=主成分数或可解释方差比,svd\_solver=’randomized’/‘full’)

Xnew=pca.fit\_transform(X)

#svd\_solver=’full’将使用完全的SVD分解，‘randomized’ SVD算法常用于主成分数远小于原维度数时

**新空间的第i个基**

pca.components\_.T[:,i]

**可解释方差比**

pca.explained\_variance\_ratio\_

**解压缩**

pca.inverse\_transform(Xnew)

**增量PCA**

from sklearn.decomposition import IncrementalPCA

inc\_pca=IncrementalPCA(n\_components)

for batch in batchs:

inc\_pca.partial\_fit(batch)

Xnew=inc\_pca.transform(X)

## LLE

from sklearn.manifold import LocallyLinearEmbedding

lle=LocallyLinearEmbedding(n\_components,n\_neighbors=进行流形化时每个样本点使用的近邻数)

Xnew=lle.fit\_transform(X)

## SVD

from sklearn.decomposition import TruncatedSVD

svd= TruncatedSVD(n\_components=K, random\_state=42)

svd.fit(X)

svd.components\_ #ΣVT

singular\_values\_ #Σ

explained\_variance\_ #各主成分所解释原数据方差值

explained\_variance\_ratio\_ #各主成分所解释原数据方差比例

## NMF (Non-negative Matrix Factorization)

NMF将X近似分解为W和H的积，满足X≈WH，W和H中元素大于零

from sklearn.decomposition import NMF

nmf=NMF(n\_components=K,\*alpha\_W=W矩阵的正则系数, \*alpha\_H=H矩阵的正则系数)

nmf.fit(X)

nmf.n\_components #右矩阵H

## t-SNE

from sklearn.manifold import TSNE

tsne = TSNE(n\_components=降维维度数, verbose=1, perplexity=40, n\_iter=250)

tsne\_results = tsne.fit\_transform(data)

x,y=zip(\*tsne\_results) #降二维后的坐标

聚类

## K-means

from sklearn.cluster import KMeans

model=KMeans(n\_clusters=k,init=人为初始化中心点ndarray,n\_init=随机初始化次数)

y\_pred=model.fit\_predict(X)

**查看聚类中心点**

model.cluster\_centers\_

**获取中心点样本的序号**

np.argmin(model.transform(X))

**查看每个样本点分配的标签**

model.labels\_

**查看每个样本点到各集群中心点的距离**

dist=model.transform(X,axis=0)

**查看当前模型中的集群内方差**

model.inertia\_

#sklean中的KMeans方法默认使用KMeans++初始化

**查看当前模型中各样本点的平均silhouette值**

from sklearn.metrics import silhouette\_score

silhouette\_score(X,model.labels\_)

**轮廓图**

<https://www.scikit-yb.org/en/latest/api/cluster/silhouette.html>

from yellowbrick.cluster import SilhouetteVisualizer

visualizer = SilhouetteVisualizer(model, colors='yellowbrick')

visualizer.fit(data) # Fit the data to the visualizer

visualizer.show() # Finalize and render the figure

**轮廓值** metrics.silhouette\_score(X,y\_pred)

**簇间簇内方差比(Calinski harabaz Score)** metrics.calinski\_harabaz\_score(X,y\_pred)

## 小批量K-means

from sklearn.cluster import MiniBatchKMeans

X=np.memmap(filepath,dtype=’float32’,mode=’readonly’,shape=(m,n)

#memmap类让数据可以分批部分加载到内存

batch\_size=m/batch\_num

model=MiniBatchKMeans(n\_clusters=k,batch\_size)

model.fit(X)

## DBScan

from sklearn.cluster import DBSCAN

model=DBSCAN(eps=核心点判定半径,min\_samples=核心点的判定域内的最少点数)

!没有predict()方法，只有fit\_predict()方法，可通过添加分类算法来实现

**查看核心点index**

model.core\_sample\_indices\_

**查看核心点**

model.components\_

**查看各点标签**

model.labels\_

#标签-1代表异常点

## 高斯混合模型

from sklearn.mixture import GaussianMixture

model=GaussianMixture(n\_components=簇群数,n\_init=重复运行次数,covariance\_type=’full’)

model.fit(X)

#GMM模型随机初始化使得收敛结果不一定是全局最优，故而需要重复运行并保存最优结果

#covariacne\_type=’spherical’:所有生成的簇群都是球形

covariacne\_type=’diag’:所有生成的簇群都的方向平行于坐标轴

covariacne\_type=’tied’:所有生成的簇群具有相同的形状、大小、方向

**查看估计的参数**

model.weights\_ #各高斯分布簇群的权重

model.means\_ #各高斯分布簇群的均值

model\_covariances\_ #各高斯分布簇群的协方差

**获取第i个簇群的样本**

X,y=model.sample(i)

**获取样本处的GMM密度**

density=model.score\_samples(X)

**通过密度筛选异常点**

anomalies=X[density<threshold]

## 阶层聚类

sklearn.cluster import AgglomerativeClustering

model= AgglomerativeClustering(*n\_clusters=*簇群数distance\_threshold=0)

**可视化**

def plot\_dendrogram(model, \*\*kwargs):

from scipy.cluster.hierarchy import dendrogram

# Create linkage matrix and then plot the dendrogram

# create the counts of samples under each node

counts = np.zeros(model.children\_.shape[0])

n\_samples = len(model.labels\_)

for i, merge in enumerate(model.children\_):

current\_count = 0

for child\_idx in merge:

if child\_idx < n\_samples:

current\_count += 1 # leaf node

else:

current\_count += counts[child\_idx - n\_samples]

counts[i] = current\_count

linkage\_matrix = np.column\_stack([model.children\_, model.distances\_,

counts]).astype(float)

# Plot the corresponding dendrogram

dendrogram(linkage\_matrix, \*\*kwargs)

plot\_dendrogram(model, truncate\_mode='level', p=40)

# NLP

## BOW(Bag-of-words)

from sklearn.feature\_extraction.text import CountVectorizer

bow=CountVectorizer(\*encoding=’utf-8’, \*decode\_error, \*lowercase=True, \*stop\_words, \*max\_df,\*min\_df, \*max\_features,\*binary)

X\_bow=bow.fit\_transform(text\_list)

#decode\_error指示在给定encoding下遇到无法解码字符时的操作，{'strict', 'ignore', 'replace'}

#stop\_words指示在进行tokenize前需要去除的停用词，默认使用自带的英语停用词库，也可以自定义赋一个stop\_words\_list

#max\_df和min\_df指示过滤掉在文本中频率大于或小于给定阈值的词，取值0-1

# max\_features指示仅考虑top\_n的词

#binary=True, 变为单词出现矩阵，含有某词为1，不含为0

## BOW with TF-IDF

from sklearn.feature\_extraction.text import TfidfVectorizer

tfidf\_bow = TfidfVectorizer(\*encoding=’utf-8’, \*decode\_error, \*lowercase=True, \*stop\_words, \*max\_df,\*min\_df, \*max\_features,\*binary,\*ngram\_range)

tfidf\_bow.fit(text\_list)

#ngram\_range输入一个元组(a,b)，表示构建a-gram到b-gram的n-gram

## LDA(LatentDirichletAllocation)

X=WH

from sklearn.decomposition import LatentDirichletAllocation

lda=LatentDirichletAllocation(n\_components=K)

lda.fit(X\_bow)

lda.components\_ #topic-word H矩阵