

导入包、模块

```
1 # 基础
2 import os
3 import zipfile
4 import numpy as np
5 import pandas as pd
6 # 画图
7 import seaborn as sns
8 import matplotlib.pyplot as plt
9 from matplotlib import font_manager as fm
10 from matplotlib import cm
11 % matplotlib inline
12 plt.style.use('ggplot')
13 # 中文图输出
14 from pylab import mpl
15 mpl.rcParams['font.sans-serif'] = ['STZhongsong'] # 指定默认字体：解决plot不能显示中文问题
16 mpl.rcParams['axes.unicode_minus'] = False # 解决保存图像是负号 '-' 显示为方块的问题
17 # 数据集归一化
18 from sklearn import datasets
19 from sklearn import preprocessing
20 # 切割训练数据和样本数据
21 from sklearn.model_selection import train_test_split
22 from sklearn.model_selection import cross_val_score
23 from sklearn.model_selection import StratifiedKFold, cross_val_score
24 # 模型
25 from sklearn.neighbors import KNeighborsClassifier
26 from sklearn.neural_network import MLPClassifier
27 from sklearn.ensemble import RandomForestClassifier
28 from sklearn.svm import SVC
29 from sklearn.tree import DecisionTreeClassifier
30 from sklearn.linear_model import LogisticRegression
31 # from sklearn.metrics import mean_squared_error
32 from sklearn.metrics import *
33 # 导出决策树
34 import graphviz
35 import pydotplus
36 from sklearn.tree import export_graphviz
37 from sklearn.externals.six import StringIO
```

定义全局函数

```
1 # 定义一个路径引用的函数
2 def file_path(dir_path, dir_name):
3     con_path =
4     "D:\\onedrive\\02_work\\01_ScienceResearch\\01_undergraduate_thesis\\01_data\\"
5     path = os.path.join(con_path, dir_path, dir_name)
6     return path
```

导入原始数据

对2010-2016年经济指标文件解压

```
1 # 对2010-2016年经济指标文件解压
2 def unzip_file(path,zip_name):
3     for file in os.listdir(path):
4         file_path=os.path.join(path,file)
5         if os.path.splitext(file_path)[1]==zip_name:
6             fz=zipfile.ZipFile(file_path,'r')
7             for zip_file in fz.namelist():
8                 fz.extract(zip_file,path)
9 unzip_file("D:\\onedrive\\02_work\\01_ScienceResearch\\01_undergraduate_thesis\\01_data",".zip")
```

读入数据

```
1 # 去掉各个变量的标签
2 pd.read_csv(file_path("01_rawdata","ACS_10_5YR_DP02_with_ann.csv"))
   [1:].to_csv(file_path("02_output","ACS_10_5YR_DP02_with_ann.csv"),encoding="
   utf-8-sig")
3 pd.read_csv(file_path("01_rawdata","ACS_11_5YR_DP02_with_ann.csv"))
   [1:].to_csv(file_path("02_output","ACS_11_5YR_DP02_with_ann.csv"),encoding="
   utf-8-sig")
4 pd.read_csv(file_path("01_rawdata","ACS_12_5YR_DP02_with_ann.csv"))
   [1:].to_csv(file_path("02_output","ACS_12_5YR_DP02_with_ann.csv"),encoding="
   utf-8-sig")
5 pd.read_csv(file_path("01_rawdata","ACS_13_5YR_DP02_with_ann.csv"))
   [1:].to_csv(file_path("02_output","ACS_13_5YR_DP02_with_ann.csv"),encoding="
   utf-8-sig")
6 pd.read_csv(file_path("01_rawdata","ACS_14_5YR_DP02_with_ann.csv"))
   [1:].to_csv(file_path("02_output","ACS_14_5YR_DP02_with_ann.csv"),encoding="
   utf-8-sig")
7 pd.read_csv(file_path("01_rawdata","ACS_15_5YR_DP02_with_ann.csv"))
   [1:].to_csv(file_path("02_output","ACS_15_5YR_DP02_with_ann.csv"),encoding="
   utf-8-sig")
8 pd.read_csv(file_path("01_rawdata","ACS_16_5YR_DP02_with_ann.csv"))
   [1:].to_csv(file_path("02_output","ACS_16_5YR_DP02_with_ann.csv"),encoding="
   utf-8-sig")
9 # 读入2010-2016年经济指标数据
10 ACS_10_5YR_DP02_with_ann =
   pd.read_csv(file_path("02_output","ACS_10_5YR_DP02_with_ann.csv"),na_values=
   ["(X)","*****","***","**","-","+","N"])
11 ACS_11_5YR_DP02_with_ann =
   pd.read_csv(file_path("02_output","ACS_11_5YR_DP02_with_ann.csv"),na_values=
   ["(X)","*****","***","**","-","+","N"])
12 ACS_12_5YR_DP02_with_ann =
   pd.read_csv(file_path("02_output","ACS_12_5YR_DP02_with_ann.csv"),na_values=
   ["(X)","*****","***","**","-","+","N"])
13 ACS_13_5YR_DP02_with_ann =
   pd.read_csv(file_path("02_output","ACS_13_5YR_DP02_with_ann.csv"),na_values=
   ["(X)","*****","***","**","-","+","N"])
14 ACS_14_5YR_DP02_with_ann =
   pd.read_csv(file_path("02_output","ACS_14_5YR_DP02_with_ann.csv"),na_values=
   ["(X)","*****","***","**","-","+","N"])
```

```

15 ACS_15_5YR_DP02_with_ann =
pd.read_csv(file_path("02_output", "ACS_15_5YR_DP02_with_ann.csv"), na_values=
["(x)", "*****", "****", "***", "-", "+", "N"])
16 ACS_16_5YR_DP02_with_ann =
pd.read_csv(file_path("02_output", "ACS_16_5YR_DP02_with_ann.csv"), na_values=
["(x)", "*****", "****", "***", "-", "+", "N"])
17 ## 读入各个地区阿片类使用量数据
18 MCM_NFLIS_Data=pd.read_excel(file_path("01_rawdata", "MCM_NFLIS_Data.xlsx"), s
heet_name=1)
19 ## 读入药物具体分类数据
20 MCM_NFLIS_Medication=pd.read_csv(file_path("01_rawdata", "class_medication.cs
v"))
21 # 读入变量标签数据
22 ACS_10_5YR_DP02_metadata =
pd.read_csv(file_path("01_rawdata", "ACS_10_5YR_DP02_metadata.csv"), header=No
ne)
23 ACS_11_5YR_DP02_metadata =
pd.read_csv(file_path("01_rawdata", "ACS_11_5YR_DP02_metadata.csv"), header=No
ne)
24 ACS_12_5YR_DP02_metadata =
pd.read_csv(file_path("01_rawdata", "ACS_12_5YR_DP02_metadata.csv"), header=No
ne)
25 ACS_13_5YR_DP02_metadata =
pd.read_csv(file_path("01_rawdata", "ACS_13_5YR_DP02_metadata.csv"), header=No
ne)
26 ACS_14_5YR_DP02_metadata =
pd.read_csv(file_path("01_rawdata", "ACS_14_5YR_DP02_metadata.csv"), header=No
ne)
27 ACS_15_5YR_DP02_metadata =
pd.read_csv(file_path("01_rawdata", "ACS_15_5YR_DP02_metadata.csv"), header=No
ne)
28 ACS_16_5YR_DP02_metadata =
pd.read_csv(file_path("01_rawdata", "ACS_16_5YR_DP02_metadata.csv"), header=No
ne)

```

数据处理

整理ACS_ALL_5YR_DP02数据

```

1 ## 处理无效数据
2 # 2010
3 # 删除类型异常的变量 (NaN、(x))
4 typedata = ACS_10_5YR_DP02_with_ann.dtypes.reset_index()
5 nonnormal_var = typedata.loc[typedata.ix[:,1] == "object"]["index"]
[2:].tolist()
6 ACS_10_5YR_DP02_DropNorm =
ACS_10_5YR_DP02_with_ann.drop(nonnormal_var,axis=1)
7 # 删除全为空的变量 (列)
8 ACS_10_5YR_DP02_DropColumn=ACS_10_5YR_DP02_DropNorm.dropna(axis=1,how="all")
9 # 用列均值填补缺失数据
10 for column in
list(ACS_10_5YR_DP02_DropColumn.columns[ACS_10_5YR_DP02_DropColumn.isnull().
sum() > 0]):
11     mean_val = ACS_10_5YR_DP02_DropColumn[column].mean()
12     ACS_10_5YR_DP02_DropColumn[column].fillna(mean_val, inplace=True)
13

```

```

14 # 2011
15 # 删除类型异常的变量 (NaN、(x))
16 typedata = ACS_11_5YR_DP02_with_ann.dtypes.reset_index()
17 nonnormal_var = typedata.loc[typedata.ix[:,1] == "object"]["index"]
18 ACS_11_5YR_DP02_DropNorm =
19 ACS_11_5YR_DP02_with_ann.drop(nonnormal_var,axis=1)
20 # 删除全为空的变量 (列)
21 ACS_11_5YR_DP02_DropColumn=ACS_11_5YR_DP02_DropNorm.dropna(axis=1,how="all")
22 # 用列均值填补缺失数据
23 for column in
24 list(ACS_11_5YR_DP02_DropColumn.columns[ACS_11_5YR_DP02_DropColumn.isnull().
25 sum() > 0]):
26     mean_val = ACS_11_5YR_DP02_DropColumn[column].mean()
27     ACS_11_5YR_DP02_DropColumn[column].fillna(mean_val, inplace=True)
28
29 # 2012
30 # 删除类型异常的变量 (NaN、(x))
31 typedata = ACS_12_5YR_DP02_with_ann.dtypes.reset_index()
32 nonnormal_var = typedata.loc[typedata.ix[:,1] == "object"]["index"]
33 ACS_12_5YR_DP02_DropNorm =
34 ACS_12_5YR_DP02_with_ann.drop(nonnormal_var,axis=1)
35 # 删除全为空的变量 (列)
36 ACS_12_5YR_DP02_DropColumn=ACS_12_5YR_DP02_DropNorm.dropna(axis=1,how="all")
37 # 用列均值填补缺失数据
38 for column in
39 list(ACS_12_5YR_DP02_DropColumn.columns[ACS_12_5YR_DP02_DropColumn.isnull().
40 sum() > 0]):
41     mean_val = ACS_12_5YR_DP02_DropColumn[column].mean()
42     ACS_12_5YR_DP02_DropColumn[column].fillna(mean_val, inplace=True)
43
44 # 2013
45 # 删除类型异常的变量 (NaN、(x))
46 typedata = ACS_13_5YR_DP02_with_ann.dtypes.reset_index()
47 nonnormal_var = typedata.loc[typedata.ix[:,1] == "object"]["index"]
48 ACS_13_5YR_DP02_DropNorm =
49 ACS_13_5YR_DP02_with_ann.drop(nonnormal_var,axis=1)
50 # 删除全为空的变量 (列)
51 ACS_13_5YR_DP02_DropColumn=ACS_13_5YR_DP02_DropNorm.dropna(axis=1,how="all")
52 # 用列均值填补缺失数据
53 for column in
54 list(ACS_13_5YR_DP02_DropColumn.columns[ACS_13_5YR_DP02_DropColumn.isnull().
55 sum() > 0]):
56     mean_val = ACS_13_5YR_DP02_DropColumn[column].mean()
57     ACS_13_5YR_DP02_DropColumn[column].fillna(mean_val, inplace=True)
58
59 # 2013
60 # 删除类型异常的变量 (NaN、(x))
61 typedata = ACS_14_5YR_DP02_with_ann.dtypes.reset_index()
62 nonnormal_var = typedata.loc[typedata.ix[:,1] == "object"]["index"]
63 ACS_14_5YR_DP02_DropNorm =
64 ACS_14_5YR_DP02_with_ann.drop(nonnormal_var,axis=1)
65 # 删除全为空的变量 (列)
66 ACS_14_5YR_DP02_DropColumn=ACS_14_5YR_DP02_DropNorm.dropna(axis=1,how="all")
67 # 用列均值填补缺失数据

```

```

58 for column in
list(ACS_14_5YR_DP02_DropColumn.columns[ACS_14_5YR_DP02_DropColumn.isnull().
sum() > 0]):
59     mean_val = ACS_14_5YR_DP02_DropColumn[column].mean()
60     ACS_14_5YR_DP02_DropColumn[column].fillna(mean_val, inplace=True)
61
62 # 2015
63 # 删除类型异常的变量 (NaN、(x))
64 typedata = ACS_15_5YR_DP02_with_ann.dtypes.reset_index()
65 nonnormal_var = typedata.loc[typedata.ix[:,1] == "object"]["index"]
[2:].tolist()
66 ACS_15_5YR_DP02_DropNorm =
ACS_15_5YR_DP02_with_ann.drop(nonnormal_var,axis=1)
67 # 删除全为空的变量 (列)
68 ACS_15_5YR_DP02_DropColumn=ACS_15_5YR_DP02_DropNorm.dropna(axis=1,how="all")
69 # 用列均值填补缺失数据
70 for column in
list(ACS_15_5YR_DP02_DropColumn.columns[ACS_15_5YR_DP02_DropColumn.isnull().
sum() > 0]):
71     mean_val = ACS_15_5YR_DP02_DropColumn[column].mean()
72     ACS_15_5YR_DP02_DropColumn[column].fillna(mean_val, inplace=True)
73
74 # 2016
75 # 删除类型异常的变量 (NaN、(x))
76 typedata = ACS_16_5YR_DP02_with_ann.dtypes.reset_index()
77 nonnormal_var = typedata.loc[typedata.ix[:,1] == "object"]["index"]
[2:].tolist()
78 ACS_16_5YR_DP02_DropNorm =
ACS_16_5YR_DP02_with_ann.drop(nonnormal_var,axis=1)
79 # 删除全为空的变量 (列)
80 ACS_16_5YR_DP02_DropColumn=ACS_16_5YR_DP02_DropNorm.dropna(axis=1,how="all")
81 # 用列均值填补缺失数据
82 for column in
list(ACS_16_5YR_DP02_DropColumn.columns[ACS_16_5YR_DP02_DropColumn.isnull().
sum() > 0]):
83     mean_val = ACS_16_5YR_DP02_DropColumn[column].mean()
84     ACS_16_5YR_DP02_DropColumn[column].fillna(mean_val, inplace=True)
85
86 # 纵向合并2010-2016年的数据到一个数据框中、# 删除第一行数据 (变量标签)
87 ACS_ALL_5YR_DP02=pd.concat([ACS_10_5YR_DP02_DropColumn,
88                             ACS_11_5YR_DP02_DropColumn,
89                             ACS_12_5YR_DP02_DropColumn,
90                             ACS_13_5YR_DP02_DropColumn,
91                             ACS_14_5YR_DP02_DropColumn,
92                             ACS_15_5YR_DP02_DropColumn,
93
ACS_16_5YR_DP02_DropColumn],axis=0,join="outer",keys=
[2010,2011,2012,2013,2014,2015,2016]).reset_index().convert_objects(convert_
numeric=True)
94 # 用列均值填补缺失数据 (合并各年份数据之后)
95 for column in list(ACS_ALL_5YR_DP02.columns[ACS_ALL_5YR_DP02.isnull().sum()
> 0]):
96     mean_val = ACS_ALL_5YR_DP02[column].mean()
97     ACS_ALL_5YR_DP02[column].fillna(mean_val, inplace=True)
98 # 删除无效的变量 (索引, 中间产生变量、地理位置), 重命名年份变量
99 ACS_ALL_5YR_DP02_Clear = ACS_ALL_5YR_DP02.ix[:,-2].drop(["GEO.display-
label","level_1","GEO.id"],axis=1).rename(columns={"level_0":"YYYY"})

```

整理MCM_NFLIS_Data数据

```
1 # 对阿片类药物使用情况数据键值重命名
2 MCM_NFLIS_Data_Rename=MCM_NFLIS_Data.rename(columns=
  {"FIPS_Combined":"GEO.id2"})
3 # 删除2017相关数据
4 MCM_NFLIS_Data_Drop17=MCM_NFLIS_Data_Rename.loc[MCM_NFLIS_Data_Rename["YYYY"
  ] != 2017]
5 # 匹配药物分类数据
6 MCM_NFLIS_Class=pd.merge(MCM_NFLIS_Data_Drop17,MCM_NFLIS_Medication,how="lef
  t",on=["SubstanceName","YYYY"])
7 # 删除一些无效变量
8 MCM_NFLIS_Class_Clear_Drop=MCM_NFLIS_Class.drop(["FIPS_State","FIPS_County",
  "SubstanceName","code"],axis=1)
9 # 按照中文名药物分类求和
10 MCM_NFLIS_Class_Clear =
  MCM_NFLIS_Class_Clear_Drop.groupby(["YYYY","GEO.id2","State","COUNTY","Subst
  anceClass",
11                                     "SubstanceName_c"])[
  ["DrugReports"].sum().reset_index()
```

整理ACS_All_5YR_DP02_metadata数据

```
1 # 纵向合并2010-2016年的数据到一个数据框中、# 删除第一行数据（变量标签）
2 ACS_All_5YR_DP02_metadata=pd.concat([ACS_10_5YR_DP02_metadata,
3                                     ACS_11_5YR_DP02_metadata,
4                                     ACS_12_5YR_DP02_metadata,
5                                     ACS_13_5YR_DP02_metadata,
6                                     ACS_14_5YR_DP02_metadata,
7                                     ACS_15_5YR_DP02_metadata,
8                                     ACS_16_5YR_DP02_metadata],axis=0,join="outer",)
9 # 删除重复值
10 ACS_All_5YR_DP02_metadata_Dup =
  ACS_All_5YR_DP02_metadata.drop_duplicates(list(ACS_All_5YR_DP02_metadata.co1
  umns)[0],keep="first")
11 ACS_All_5YR_DP02_metadata_Dup.columns = ["Var","Var_label"]
```

匹配阿片类药物使用情况

```
1 # 合并阿片类使用情况与相关经济指标
2 NFLIS_and_ACS_ALL=pd.merge(ACS_ALL_5YR_DP02_Clear,MCM_NFLIS_Class_Clear,how="
  right",on=["YYYY","GEO.id2"])
```

按照三类药物数据透视

```
1 # 分类计数
2 NFLIS_and_ACS_ALL_ClassSum =
  NFLIS_and_ACS_ALL.groupby(["GEO.id2","State","COUNTY",
3
4   "SubstanceClass","YYYY"])[["DrugReports"].sum().reset_index()
5 # 数据透视表
6 NFLIS_and_ACS_ALL_Pivot = pd.pivot_table(data=NFLIS_and_ACS_ALL_ClassSum,
  index=
  ["GEO.id2","State","SubstanceClass","COUNTY"],
```

```

7         columns=["YYYY"], values=
["DrugReports"])
8 # 缺失值填补、转置
9 NFLIS_and_ACS_ALL_Clear =
NFLIS_and_ACS_ALL_Pivot[2:].fillna(0).stack().reset_index()
10 # 合并
11 NFLIS_and_ACS_ALL_Out = pd.merge(NFLIS_and_ACS_ALL_Clear,
12                                  ACS_ALL_5YR_DP02_Clear,
13                                  on=["GEO.id2", "YYYY"], how="left")
14 # 根据药物量分层
15 NFLIS_and_ACS_ALL_Out["DrugReportsclass"] =
np.where(NFLIS_and_ACS_ALL_Out["DrugReports"] >= 5000, "7、5000人以上",
16
np.where(NFLIS_and_ACS_ALL_Out["DrugReports"] >= 1000, "6、1000-4999人",
17
np.where(NFLIS_and_ACS_ALL_Out["DrugReports"] >= 500, "5、500-999人",
18
np.where(NFLIS_and_ACS_ALL_Out["DrugReports"] >= 100, "4、100-499人",
19
np.where(NFLIS_and_ACS_ALL_Out["DrugReports"] >= 10, "3、10-99人",
20
np.where(NFLIS_and_ACS_ALL_Out["DrugReports"] >= 1, "2、1-9人", "1、0
人")))))))

```

统计描述

图1：所有类阿片类药物构成饼图

整理数据为直接可用

```

1 # 提取画图数据"YYYY", "SubstanceName", "DrugReports", "State"
2 NFLIS_Figure1_Data = MCM_NFLIS_Class_Clear.groupby(["SubstanceName_c"])\
["DrugReports"].sum().reset_index().sort_values(by="DrugReports", ascending=T
rue)
3 # 添加列：每种药物的百分占比
4 NFLIS_Figure1_Data["Percent"] =
NFLIS_Figure1_Data["DrugReports"]/(NFLIS_Figure1_Data["DrugReports"].sum())
5 NFLIS_Figure1_Data["Label"] = NFLIS_Figure1_Data["SubstanceName_c"] + \
6     ' ' + \
7     NFLIS_Figure1_Data["Percent"].apply(lambda
x: format(x, '.2%'))
8 # 画图所用的数据
9 Figure1_labels = NFLIS_Figure1_Data["Label"]
10 Figure1_sizes = NFLIS_Figure1_Data["DrugReports"]

```

饼图

```

1 # 设置画布和子图
2 Figure1, axes = plt.subplots(figsize=(20,15), ncols=2)
3 Figure1_ax1, Figure1_ax2 = axes.ravel()
4 # 设置参数：颜色盘-colormap; 间隙-与labels一一对应，数值越大离中心区越远
5 explode = [x * 0.00325 for x in range(len(NFLIS_Figure1_Data))]
6 colors=cm.rainbow(np.arange(len(Figure1_sizes))/len(Figure1_sizes))
7 # 画饼图：类别太多取消标签labels；每个类别离中心的距离；

```

```

8 patches, texts =
  Figure1_ax1.pie(Figure1_sizes, labels=None, shadow=False, explode=explode, start
    angle=0, colors=colors)
9 # 子图: ax1-饼图、ax2-图例
10 Figure1_ax1.axis('equal')
11 Figure1_ax2.axis('off')
12 Figure1_ax2.legend(patches, Figure1_labels, loc="center left", fontsize="xx-
  large")
13 # 调整大小、读取图片
14 plt.tight_layout()
15 Figure1 = plt.gcf()

```

图2：所有类阿片类药物数量条图

整理数据为直接可用

```

1 # 提取画图数据"YYYY", "SubstanceName", "DrugReports", "State"; 排序;
2 NFLIS_Figure2_Data = MCM_NFLIS_Class_Clear.groupby(["SubstanceName_c"])
  ["DrugReports"].sum().reset_index().sort_values(by="DrugReports", ascending=Tr
    ue)
3

```

条图

```

1 # 设置画布
2 plt.figure(figsize=(16,10))
3 # 设置参数: 颜色盘-colormap
4 color=cm.rainbow(np.arange(len(NFLIS_Figure2_Data))/len(NFLIS_Figure2_Data))
5 # 从高到低排列, 改变y轴刻度的排列顺序
6 plt.yticks(np.arange(len(NFLIS_Figure2_Data['SubstanceName_c'])),
  NFLIS_Figure2_Data['SubstanceName_c'])
7 # 水平条图
8 plt.barh(np.arange(len(NFLIS_Figure2_Data['SubstanceName_c'])),
  NFLIS_Figure2_Data['DrugReports'], color=color)
9 # 坐标轴标签
10 plt.ylabel("阿片类药物名")
11 plt.xlabel("报告量")
12 # 格式整理导出
13 plt.tight_layout()
14 Figure2 = plt.gcf()

```

图3：五个州阿片类药物数量热力图

整理数据为直接可用

```

1 # 提取画图数据"YYYY", "SubstanceName", "DrugReports", "State"
2 NFLIS_Figure3_Clear1 =
  MCM_NFLIS_Class_Clear.groupby(["State", "YYYY", "SubstanceName_c"])
  ["DrugReports"].sum().reset_index()
3 # 提取各个州的数据
4 NFLIS_Figure3_KY = NFLIS_Figure3_Clear1.loc[(NFLIS_Figure3_Clear1["State"]
  == "KY")]
5 NFLIS_Figure3_OH = NFLIS_Figure3_Clear1.loc[(NFLIS_Figure3_Clear1["State"]
  == "OH")]

```



```

6 NFLIS_Figure3_PA = NFLIS_Figure3_Clear1.loc[(NFLIS_Figure3_Clear1["State"]
  == "PA")]
7 NFLIS_Figure3_VA = NFLIS_Figure3_Clear1.loc[(NFLIS_Figure3_Clear1["State"]
  == "VA")]
8 NFLIS_Figure3_WV = NFLIS_Figure3_Clear1.loc[(NFLIS_Figure3_Clear1["State"]
  == "WV")]
9 # 匹配每种药物（解决某年可能没有某种药）
10 NFLIS_Figure3_KY_merge = pd.merge(NFLIS_Figure3_KY,MCM_NFLIS_Medication,how
  = "right",on = ["YYYY","SubstanceName_c"])
11 NFLIS_Figure3_OH_merge = pd.merge(NFLIS_Figure3_OH,MCM_NFLIS_Medication,how
  = "right",on = ["YYYY","SubstanceName_c"])
12 NFLIS_Figure3_PA_merge = pd.merge(NFLIS_Figure3_PA,MCM_NFLIS_Medication,how
  = "right",on = ["YYYY","SubstanceName_c"])
13 NFLIS_Figure3_VA_merge = pd.merge(NFLIS_Figure3_VA,MCM_NFLIS_Medication,how
  = "right",on = ["YYYY","SubstanceName_c"])
14 NFLIS_Figure3_WV_merge = pd.merge(NFLIS_Figure3_WV,MCM_NFLIS_Medication,how
  = "right",on = ["YYYY","SubstanceName_c"])
15 # 将数据转置为dataframe矩阵
16 NFLIS_Figure3_pivot_KY = NFLIS_Figure3_KY_merge.pivot_table(index =
  "SubstanceName_c",columns = "YYYY",values = "DrugReports")
17 NFLIS_Figure3_pivot_OH = NFLIS_Figure3_OH_merge.pivot_table(index =
  "SubstanceName_c",columns = "YYYY",values = "DrugReports")
18 NFLIS_Figure3_pivot_PA = NFLIS_Figure3_PA_merge.pivot_table(index =
  "SubstanceName_c",columns = "YYYY",values = "DrugReports")
19 NFLIS_Figure3_pivot_VA = NFLIS_Figure3_VA_merge.pivot_table(index =
  "SubstanceName_c",columns = "YYYY",values = "DrugReports")
20 NFLIS_Figure3_pivot_WV = NFLIS_Figure3_WV_merge.pivot_table(index =
  "SubstanceName_c",columns = "YYYY",values = "DrugReports")

```

热力图

```

1 # 设置画布大小
2 f,(Figure3_ax1,Figure3_ax2,Figure3_ax3,Figure3_ax4,Figure3_ax5) =
  plt.subplots(ncols=5,figsize=(30,10))
3 # 设置连续调色板cubehelix_palette,as_camp传入matplotlib
4 cmap=sns.cubehelix_palette(start=1,rot=3,gamma=0.8,as_cmap=True)
5 # KY州
6 sns.heatmap(NFLIS_Figure3_pivot_KY,cmap=cmap,linewidths=0.05,ax=Figure3_ax1,
  cbar=False)
7 Figure3_ax1.set_title("肯塔基州",fontsize=30)
8 Figure3_ax1.set_xlabel('')
9 Figure3_ax1.set_ylabel('阿片类药物名',fontsize=35)
10 # OH州
11 sns.heatmap(NFLIS_Figure3_pivot_OH,cmap=cmap,linewidths=0.05,ax=Figure3_ax2,
  cbar=False)
12 Figure3_ax2.set_title("俄亥俄州",fontsize=30)
13 Figure3_ax2.set_xlabel('')
14 Figure3_ax2.set_ylabel(' ')
15 Figure3_ax2.set_yticklabels([])
16 # PA州
17 sns.heatmap(NFLIS_Figure3_pivot_PA,cmap=cmap,linewidths=0.05,ax=Figure3_ax3,
  cbar=False)
18 Figure3_ax3.set_title("宾夕法尼亚州",fontsize=30)
19 Figure3_ax3.set_xlabel('年份',fontsize=35)
20 Figure3_ax3.set_ylabel('')
21 Figure3_ax3.set_yticklabels([])
22 # VA州

```

```

23 sns.heatmap(NFLIS_Figure3_pivot_VA,cmap=cmap,linewidths=0.05,ax=Figure3_ax4,
   cbar=False)
24 Figure3_ax4.set_title("弗吉尼亚州",fontsize=30)
25 Figure3_ax4.set_xlabel('')
26 Figure3_ax4.set_ylabel('')
27 Figure3_ax4.set_yticklabels([])
28 # WV州
29 sns.heatmap(NFLIS_Figure3_pivot_WV,cmap=cmap,linewidths=0.05,ax=Figure3_ax5,
   cbar=True)
30 Figure3_ax5.set_title("西弗吉尼亚州",fontsize=30)
31 Figure3_ax5.set_xlabel('')
32 Figure3_ax5.set_ylabel('')
33 Figure3_ax5.set_yticklabels([])
34
35 plt.tight_layout()
36 Figure3 = plt.gcf()

```

图4：五个州三类阿片类药物量折线图

整理数据为直接可用

```

1 # 五个州的总量情况分组
2 NFLIS_Fugure3_Clear1 =
   MCM_NFLIS_Class_Clear.groupby(["YYYY","SubstanceClass"])
   ["DrugReports"].sum().reset_index()
3 NFLIS_Fugure3_Class1_all =
   NFLIS_Fugure3_Clear1.loc[(NFLIS_Fugure3_Clear1["SubstanceClass"] == "半合成阿
   片类药物")]
4 NFLIS_Fugure3_Class2_all =
   NFLIS_Fugure3_Clear1.loc[(NFLIS_Fugure3_Clear1["SubstanceClass"] == "合成阿片
   类药物")]
5 NFLIS_Fugure3_Class3_all =
   NFLIS_Fugure3_Clear1.loc[(NFLIS_Fugure3_Clear1["SubstanceClass"] == "非合成阿
   片类药物")]
6 # 五个州的分别情况分组
7 NFLIS_Fugure3_Class =
   MCM_NFLIS_Class_Clear.groupby(["YYYY","State","SubstanceClass"])
   ["DrugReports"].sum().reset_index()
8 NFLIS_Fugure3_Class1 =
   NFLIS_Fugure3_Class.loc[(NFLIS_Fugure3_Class["SubstanceClass"] == "半合成阿片
   类药物")]
9 NFLIS_Fugure3_Class2 =
   NFLIS_Fugure3_Class.loc[(NFLIS_Fugure3_Class["SubstanceClass"] == "合成阿片类
   药物")]
10 NFLIS_Fugure3_Class3 =
   NFLIS_Fugure3_Class.loc[(NFLIS_Fugure3_Class["SubstanceClass"] == "非合成阿片
   类药物")]
11 # 对每个州进行汇合
12 NFLIS_Figure2_Data_Class1 =
   NFLIS_Fugure3_Class1.pivot_table(index="YYYY",columns="State",values="DrugRe
   ports").reset_index()
13 NFLIS_Figure2_Data_Class2 =
   NFLIS_Fugure3_Class2.pivot_table(index="YYYY",columns="State",values="DrugRe
   ports").reset_index()
14 NFLIS_Figure2_Data_Class3 =
   NFLIS_Fugure3_Class3.pivot_table(index="YYYY",columns="State",values="DrugRe
   ports").reset_index()

```

折线图

```
1  # 创建画布、6个子图
2  plt.figure(figsize=(15,10))
3  f4 = plt.figure(figsize=(20,15))
4  Figure_ax1 = f4.add_subplot(2, 3, 1)
5  Figure_ax2 = f4.add_subplot(2, 3, 2)
6  Figure_ax3 = f4.add_subplot(2, 3, 3)
7  Figure_ax4 = f4.add_subplot(2, 3, 4)
8  Figure_ax5 = f4.add_subplot(2, 3, 5)
9  Figure_ax6 = f4.add_subplot(2, 3, 6)
10
11 # KY州不同类型药物的折线图
12 Figure_ax1.plot(NFLIS_Figure2_Data_Class1["YYYY"],NFLIS_Figure2_Data_Class1
13 ["KY"],label="半合成阿片类药物",linewidth=2)
14 Figure_ax1.plot(NFLIS_Figure2_Data_Class2["YYYY"],NFLIS_Figure2_Data_Class2
15 ["KY"],label="合成阿片类药物",linewidth=2)
16 Figure_ax1.plot(NFLIS_Figure2_Data_Class3["YYYY"],NFLIS_Figure2_Data_Class3
17 ["KY"],label="非合成阿片类药物",linewidth=2)
18 Figure_ax1.set_title("肯塔基州")
19 Figure_ax1.legend(loc=2)
20 Figure_ax1.grid(axis='x')
21 #设置数字标签
22 for a,b in
23 zip(NFLIS_Figure2_Data_Class1["YYYY"],NFLIS_Figure2_Data_Class1["KY"]):
24     Figure_ax1.text(a, b+0.001, '%s' % b, ha='center', va=
25 'bottom',fontsize=11)
26 for a,b in
27 zip(NFLIS_Figure2_Data_Class2["YYYY"],NFLIS_Figure2_Data_Class2["KY"]):
28     Figure_ax1.text(a, b+0.001, '%s' % b, ha='center', va=
29 'bottom',fontsize=11)
30 for a,b in
31 zip(NFLIS_Figure2_Data_Class3["YYYY"],NFLIS_Figure2_Data_Class3["KY"]):
32     Figure_ax1.text(a, b+0.001, '%s' % b, ha='center', va=
33 'bottom',fontsize=11)
34
35 # OH州不同类型药物的折线图
36 Figure_ax2.plot(NFLIS_Figure2_Data_Class1["YYYY"],NFLIS_Figure2_Data_Class1
37 ["OH"],label="半合成阿片类药物",linewidth=2)
38 Figure_ax2.plot(NFLIS_Figure2_Data_Class2["YYYY"],NFLIS_Figure2_Data_Class2
39 ["OH"],label="合成阿片类药物",linewidth=2)
40 Figure_ax2.plot(NFLIS_Figure2_Data_Class3["YYYY"],NFLIS_Figure2_Data_Class3
41 ["OH"],label="非合成阿片类药物",linewidth=2)
42 Figure_ax2.set_title("俄亥俄州")
43 Figure_ax2.legend(loc=2)
44 Figure_ax2.grid(axis='x')
45 #设置数字标签**
46 for a,b in
47 zip(NFLIS_Figure2_Data_Class1["YYYY"],NFLIS_Figure2_Data_Class1["OH"]):
48     Figure_ax2.text(a, b+0.001, '%s' % b, ha='center', va=
49 'bottom',fontsize=11)
50 for a,b in
51 zip(NFLIS_Figure2_Data_Class2["YYYY"],NFLIS_Figure2_Data_Class2["OH"]):
52     Figure_ax2.text(a, b+0.001, '%s' % b, ha='center', va=
53 'bottom',fontsize=11)
```

```

38 for a,b in
zip(NFLIS_Figure2_Data_Class3["YYYY"],NFLIS_Figure2_Data_Class3["OH"]):
39     Figure_ax2.text(a, b+0.001, '%s' % b, ha='center', va=
'bottom',fontsize=11)
40
41 # PA州不同类型药物的折线图
42 Figure_ax3.plot(NFLIS_Figure2_Data_Class1["YYYY"],NFLIS_Figure2_Data_Class1
["PA"],label="半合成阿片类药物",linewidth=2)
43 Figure_ax3.plot(NFLIS_Figure2_Data_Class2["YYYY"],NFLIS_Figure2_Data_Class2
["PA"],label="合成阿片类药物",linewidth=2)
44 Figure_ax3.plot(NFLIS_Figure2_Data_Class3["YYYY"],NFLIS_Figure2_Data_Class3
["PA"],label="非合成阿片类药物",linewidth=2)
45 Figure_ax3.set_title("宾夕法尼亚州")
46 Figure_ax3.legend(loc=2)
47 Figure_ax3.grid(axis='x')
48 #设置数字标签**
49 for a,b in
zip(NFLIS_Figure2_Data_Class1["YYYY"],NFLIS_Figure2_Data_Class1["PA"]):
50     Figure_ax3.text(a, b+0.001, '%s' % b, ha='center', va=
'bottom',fontsize=11)
51 for a,b in
zip(NFLIS_Figure2_Data_Class2["YYYY"],NFLIS_Figure2_Data_Class2["PA"]):
52     Figure_ax3.text(a, b+0.001, '%s' % b, ha='center', va=
'bottom',fontsize=11)
53 for a,b in
zip(NFLIS_Figure2_Data_Class3["YYYY"],NFLIS_Figure2_Data_Class3["PA"]):
54     Figure_ax3.text(a, b+0.001, '%s' % b, ha='center', va=
'bottom',fontsize=11)
55
56 # VA州不同类型药物的折线图
57 Figure_ax4.plot(NFLIS_Figure2_Data_Class1["YYYY"],NFLIS_Figure2_Data_Class1
["VA"],label="半合成阿片类药物",linewidth=2)
58 Figure_ax4.plot(NFLIS_Figure2_Data_Class2["YYYY"],NFLIS_Figure2_Data_Class2
["VA"],label="合成阿片类药物",linewidth=2)
59 Figure_ax4.plot(NFLIS_Figure2_Data_Class3["YYYY"],NFLIS_Figure2_Data_Class3
["VA"],label="非合成阿片类药物",linewidth=2)
60 Figure_ax4.set_title("弗吉尼亚州")
61 Figure_ax4.grid(axis="x")
62 Figure_ax4.legend(loc=2)
63 for a,b in
zip(NFLIS_Figure2_Data_Class1["YYYY"],NFLIS_Figure2_Data_Class1["VA"]):
64     Figure_ax4.text(a, b+0.001, '%s' % b, ha='center', va=
'bottom',fontsize=11)
65 for a,b in
zip(NFLIS_Figure2_Data_Class2["YYYY"],NFLIS_Figure2_Data_Class2["VA"]):
66     Figure_ax4.text(a, b+0.001, '%s' % b, ha='center', va=
'bottom',fontsize=11)
67 for a,b in
zip(NFLIS_Figure2_Data_Class3["YYYY"],NFLIS_Figure2_Data_Class3["VA"]):
68     Figure_ax4.text(a, b+0.001, '%s' % b, ha='center', va=
'bottom',fontsize=11)
69
70 # WV州不同类型药物的折线图
71 Figure_ax5.plot(NFLIS_Figure2_Data_Class1["YYYY"],NFLIS_Figure2_Data_Class1
["WV"],label="半合成阿片类药物",linewidth=2)
72 Figure_ax5.plot(NFLIS_Figure2_Data_Class2["YYYY"],NFLIS_Figure2_Data_Class2
["WV"],label="合成阿片类药物",linewidth=2)

```

```

73 Figure_ax5.plot(NFLIS_Figure2_Data_Class3["YYYY"],NFLIS_Figure2_Data_Class3
["WV"],label="非合成阿片类药物",linewidth=2)
74 Figure_ax5.set_title("西弗吉尼亚州")
75 Figure_ax5.legend(loc=2)
76 Figure_ax5.grid(axis='x')
77 #设置数字标签**
78 for a,b in
zip(NFLIS_Figure2_Data_Class1["YYYY"],NFLIS_Figure2_Data_Class1["WV"]):
79     Figure_ax5.text(a, b+0.001, '%s' % b, ha='center', va=
'bottom',fontsize=11)
80 for a,b in
zip(NFLIS_Figure2_Data_Class2["YYYY"],NFLIS_Figure2_Data_Class2["WV"]):
81     Figure_ax5.text(a, b+0.001, '%s' % b, ha='center', va=
'bottom',fontsize=11)
82 for a,b in
zip(NFLIS_Figure2_Data_Class3["YYYY"],NFLIS_Figure2_Data_Class3["WV"]):
83     Figure_ax5.text(a, b+0.001, '%s' % b, ha='center', va=
'bottom',fontsize=11)
84
85 # 5个州总的不同类型药物的折线图
86 Figure_ax6.plot(NFLIS_Fugure3_Class1_all["YYYY"],NFLIS_Fugure3_Class1_all["
DrugReports"],label="半合成阿片类药物",linewidth=2)
87 Figure_ax6.plot(NFLIS_Fugure3_Class2_all["YYYY"],NFLIS_Fugure3_Class2_all["
DrugReports"],label="合成阿片类药物",linewidth=2)
88 Figure_ax6.plot(NFLIS_Fugure3_Class3_all["YYYY"],NFLIS_Fugure3_Class3_all["
DrugReports"],label="非合成阿片类药物",linewidth=2)
89 Figure_ax6.set_title("总量")
90 Figure_ax6.legend(loc=2)
91 Figure_ax6.grid(axis='x')
92 for a,b in
zip(NFLIS_Fugure3_Class1_all["YYYY"],NFLIS_Fugure3_Class1_all["DrugReports"
]):
93     Figure_ax6.text(a, b+0.001, '%s' % b, ha='center', va=
'bottom',fontsize=11)
94 for a,b in
zip(NFLIS_Fugure3_Class2_all["YYYY"],NFLIS_Fugure3_Class2_all["DrugReports"
]):
95     Figure_ax6.text(a, b+0.001, '%s' % b, ha='center', va=
'bottom',fontsize=11)
96 for a,b in
zip(NFLIS_Fugure3_Class3_all["YYYY"],NFLIS_Fugure3_Class3_all["DrugReports"
]):
97     Figure_ax6.text(a, b+0.001, '%s' % b, ha='center', va=
'bottom',fontsize=11)
98
99 plt.tight_layout()
100 Figure4 = plt.gcf()

```

变量选择

相关系数计算

计算各个年份相关系数

```
1 # 计算2010年相关系数
2 df_corr_2010 =
  NFLIS_and_ACS_ALL_Out.loc[(NFLIS_and_ACS_ALL_Out["YYYY"]==2010)].corr().reset_index()
3 df_corr_ext_2010 =
  df_corr_2010.loc[(df_corr_2010["index"].str.contains("HC"))]
4 df_corr_ext_2010_part =
  df_corr_ext_2010[["index", "DrugReports"]].rename(columns=
    {"DrugReports": "2010年相关系数", "index": "变量名"})
5 # 计算2011年相关系数
6 df_corr_2011 =
  NFLIS_and_ACS_ALL_Out.loc[(NFLIS_and_ACS_ALL_Out["YYYY"]==2011)].corr().reset_index()
7 df_corr_ext_2011 =
  df_corr_2011.loc[(df_corr_2011["index"].str.contains("HC"))]
8 df_corr_ext_2011_part =
  df_corr_ext_2011[["index", "DrugReports"]].rename(columns=
    {"DrugReports": "2011年相关系数", "index": "变量名"})
9 # 计算2012年相关系数
10 df_corr_2012 =
  NFLIS_and_ACS_ALL_Out.loc[(NFLIS_and_ACS_ALL_Out["YYYY"]==2012)].corr().reset_index()
11 df_corr_ext_2012 =
  df_corr_2012.loc[(df_corr_2012["index"].str.contains("HC"))]
12 df_corr_ext_2012_part =
  df_corr_ext_2012[["index", "DrugReports"]].rename(columns=
    {"DrugReports": "2012年相关系数", "index": "变量名"})
13 # 计算2013年相关系数
14 df_corr_2013 =
  NFLIS_and_ACS_ALL_Out.loc[(NFLIS_and_ACS_ALL_Out["YYYY"]==2013)].corr().reset_index()
15 df_corr_ext_2013 =
  df_corr_2013.loc[(df_corr_2013["index"].str.contains("HC"))]
16 df_corr_ext_2013_part =
  df_corr_ext_2013[["index", "DrugReports"]].rename(columns=
    {"DrugReports": "2013年相关系数", "index": "变量名"})
17 # 计算2014年相关系数
18 df_corr_2014 =
  NFLIS_and_ACS_ALL_Out.loc[(NFLIS_and_ACS_ALL_Out["YYYY"]==2014)].corr().reset_index()
19 df_corr_ext_2014 =
  df_corr_2014.loc[(df_corr_2014["index"].str.contains("HC"))]
20 df_corr_ext_2014_part =
  df_corr_ext_2014[["index", "DrugReports"]].rename(columns=
    {"DrugReports": "2014年相关系数", "index": "变量名"})
21 # 计算2015年相关系数
22 df_corr_2015 =
  NFLIS_and_ACS_ALL_Out.loc[(NFLIS_and_ACS_ALL_Out["YYYY"]==2015)].corr().reset_index()
23 df_corr_ext_2015 =
  df_corr_2015.loc[(df_corr_2015["index"].str.contains("HC"))]
24 df_corr_ext_2015_part =
  df_corr_ext_2015[["index", "DrugReports"]].rename(columns=
    {"DrugReports": "2015年相关系数", "index": "变量名"})
25 # 计算2016年相关系数
```

```

26 df_corr_2016 =
  NFLIS_and_ACS_ALL_Out.loc[(NFLIS_and_ACS_ALL_Out["YYYY"]==2016)].corr().reset_index()
27 df_corr_ext_2016 =
  df_corr_2016.loc[(df_corr_2016["index"].str.contains("HC"))]
28 df_corr_ext_2016_part =
  df_corr_ext_2016[["index", "DrugReports"]].rename(columns=
    {"DrugReports": "2016年相关系数", "index": "变量名"})
29 # 计算全部数据的相关系数
30 df_corr_all = NFLIS_and_ACS_ALL_Out.corr().reset_index()
31 df_corr_ext_all = df_corr_all.loc[(df_corr_all["index"].str.contains("HC"))]
32 df_corr_ext_all_part =
  df_corr_ext_all[["index", "DrugReports"]].rename(columns={"DrugReports": "合计
  相关系数", "index": "变量名"})

```

合并各个年份的相关系数

```

1 # 合并各个年份的相关系数
2 df_corr_merge_10_11 =
  pd.merge(df_corr_ext_2010_part, df_corr_ext_2011_part, on="变量名", how="outer")
3 df_corr_merge_11_12 =
  pd.merge(df_corr_merge_10_11, df_corr_ext_2012_part, on="变量名", how="outer")
4 df_corr_merge_12_13 =
  pd.merge(df_corr_merge_11_12, df_corr_ext_2013_part, on="变量名", how="outer")
5 df_corr_merge_13_14 =
  pd.merge(df_corr_merge_12_13, df_corr_ext_2014_part, on="变量名", how="outer")
6 df_corr_merge_14_15 =
  pd.merge(df_corr_merge_13_14, df_corr_ext_2015_part, on="变量名", how="outer")
7 df_corr_merge_15_16 =
  pd.merge(df_corr_merge_14_15, df_corr_ext_2016_part, on="变量名", how="outer")
8 df_corr_merge_all = pd.merge(df_corr_merge_15_16, df_corr_ext_all_part, on="变
  量名", how="outer")
9 # 计算平均数
10 df_corr_merge_all["均值"] = df_corr_merge_all[["2010年相关系数", "2011年相关系
  数", "2012年相关系数",
  "2013年相关系数", "2014年相关系
  数", "2015年相关系数", "2016年相关系数"]].mean(axis=1)
11
12 # 排序：倒序
13 All_Corr = df_corr_merge_all.sort_values(by=["均
  值"], ascending=False).round(4)

```

选择相关系数大于0.5的变量

```

1 All_Corr_Condi = All_Corr[(abs(All_Corr["2010年相关系数"]) >= 0.5)
2                               & (abs(All_Corr["2011年相关系数"]) >=
3                               & (abs(All_Corr["2012年相关系数"]) >=
4                               & (abs(All_Corr["2013年相关系数"]) >=
5                               & (abs(All_Corr["2014年相关系数"]) >=
6                               & (abs(All_Corr["2015年相关系数"]) >=
7                               & (abs(All_Corr["2016年相关系数"]) >=
  0.5)

```

```

8         & (abs(All_Corr["合计相关系数"]) >=
0.5)
9         & (abs(All_Corr["均值"]) >= 0.5)]
10 connames = []
11 for conval in NFLIS_and_ACS_ALL_Out.columns.tolist():
12     if "HC" not in conval:
13         connames.append(conval)
14 NFLIS_and_ACS_All_Corr_Condi =
NFLIS_and_ACS_ALL_Out.ix[:,list(NFLIS_and_ACS_ALL_Out[connames])+list(All_Corr_Condi["变量名"])].dropna()

```

统计推断

归一化

```

1 data=NFLIS_and_ACS_All_Corr_Condi.ix[:,list(All_Corr_Condi["变量名"])]
2 NFLIS_and_ACS_All_Condi_Normal_CH = (data - data.mean())/data.std()
3 # 合并
4 NFLIS_and_ACS_All_Condi_Normal =
pd.concat([NFLIS_and_ACS_All_Corr_Condi.ix[:,list(NFLIS_and_ACS_All_Corr_Condi[connames])],
5
NFLIS_and_ACS_All_Condi_Normal_CH],axis=1)

```

训练集与测试集

```

1 Complex =
NFLIS_and_ACS_All_Condi_Normal.ix[NFLIS_and_ACS_All_Condi_Normal["SubstanceClass"] == "合成阿片类药物"]
2 Non_Complex =
NFLIS_and_ACS_All_Condi_Normal.ix[NFLIS_and_ACS_All_Condi_Normal["SubstanceClass"] == "非合成阿片类药物"]
3 Semi_Complex =
NFLIS_and_ACS_All_Condi_Normal.ix[NFLIS_and_ACS_All_Condi_Normal["SubstanceClass"] == "半合成阿片类药物"]
4
5 Complex_x_train,Complex_x_test,Complex_y_train,Complex_y_test =
train_test_split(Complex.ix[:,list(All_Corr_Condi["变量名"])],
6
Complex.ix[:, "DrugReportsclass"],
7
test_size=0.3,
8
random_state=1234 )
9 Non_Complex_x_train,Non_Complex_x_test,Non_Complex_y_train,Non_Complex_y_test = train_test_split(Non_Complex.ix[:,list(All_Corr_Condi["变量名"])],
10
Non_Complex.ix[:, "DrugReportsclass"],
11
test_size=0.3,
12
random_state=1234 )
13 Semi_Complex_x_train,Semi_Complex_x_test,Semi_Complex_y_train,Semi_Complex_y_test = train_test_split(Semi_Complex.ix[:,list(All_Corr_Condi["变量名"])],

```



```

14         Semi_Complex.ix[:, "DrugReportsclass"],
15
16         test_size=0.3,
17
18         random_state=1234 )

```

KNN

```

1  Complex_KNN = KNeighborsClassifier()
2  Complex_KNN.fit(Complex_x_train,Complex_y_train)
3  Complex_KNN_Y_Predict = Complex_KNN.predict(Complex_x_test)
4  Complex_KNN_train_score = Complex_KNN.score(Complex_x_train,
5  Complex_y_train)
6  Complex_KNN_test_score = Complex_KNN.score(Complex_x_test, Complex_y_test)
7  Non_Complex_KNN = KNeighborsClassifier()
8  Non_Complex_KNN.fit(Non_Complex_x_train,Non_Complex_y_train)
9  Non_Complex_KNN_Y_Predict = Non_Complex_KNN.predict(Non_Complex_x_test)
10 Non_Complex_KNN_train_score = Non_Complex_KNN.score(Non_Complex_x_train,
11 Non_Complex_y_train)
12 Non_Complex_KNN_test_score = Complex_KNN.score(Non_Complex_x_test,
13 Non_Complex_y_test)
14 Semi_Complex_KNN = KNeighborsClassifier()
15 Semi_Complex_KNN.fit(Semi_Complex_x_train,Semi_Complex_y_train)
16 Semi_Complex_KNN_Y_Predict = Semi_Complex_KNN.predict(Semi_Complex_x_test)
17 Semi_Complex_KNN_train_score = Semi_Complex_KNN.score(Semi_Complex_x_train,
18 Semi_Complex_y_train)
19 Semi_Complex_KNN_test_score = Semi_Complex_KNN.score(Semi_Complex_x_test,
20 Semi_Complex_y_test)

```

决策树

```

1  # 决策树
2  Complex_Decision = DecisionTreeClassifier()
3  Complex_Decision.fit(Complex_x_train,Complex_y_train)
4  Complex_Decision_Y_Predict = Complex_Decision.predict(Complex_x_test)
5  Complex_Decision_train_score = Complex_Decision.score(Complex_x_train,
6  Complex_y_train)
7  Complex_Decision_test_score = Complex_Decision.score(Complex_x_test,
8  Complex_y_test)
9  Non_Complex_Decision = DecisionTreeClassifier()
10 Non_Complex_Decision.fit(Non_Complex_x_train,Non_Complex_y_train)
11 Non_Complex_Decision_Y_Predict =
12 Non_Complex_Decision.predict(Non_Complex_x_test)
13 Non_Complex_Decision_train_score =
14 Non_Complex_Decision.score(Non_Complex_x_train, Non_Complex_y_train)
15 Non_Complex_Decision_test_score = Complex_Decision.score(Non_Complex_x_test,
16 Non_Complex_y_test)
17 Semi_Complex_Decision = DecisionTreeClassifier()
18 Semi_Complex_Decision.fit(Semi_Complex_x_train,Semi_Complex_y_train)
19 Semi_Complex_Decision_Y_Predict =
20 Semi_Complex_Decision.predict(Semi_Complex_x_test)
21 Semi_Complex_Decision_train_score =
22 Semi_Complex_Decision.score(Semi_Complex_x_train, Semi_Complex_y_train)

```

```
16 Semi_Complex_Decision_test_score =  
    Semi_Complex_Decision.score(Semi_Complex_x_test, Semi_Complex_y_test)
```

随机森林

```
1  # 随机森林  
2  Complex_RFC = RandomForestClassifier()  
3  Complex_RFC.fit(Complex_x_train,Complex_y_train)  
4  Complex_RFC_Y_Predict = Complex_RFC.predict(Complex_x_test)  
5  Complex_RFC_train_score = Complex_RFC.score(Complex_x_train,  
    Complex_y_train)  
6  Complex_RFC_test_score = Complex_RFC.score(Complex_x_test, Complex_y_test)  
7  Non_Complex_RFC = RandomForestClassifier()  
8  Non_Complex_RFC.fit(Non_Complex_x_train,Non_Complex_y_train)  
9  Non_Complex_RFC_Y_Predict = Non_Complex_RFC.predict(Non_Complex_x_test)  
10 Non_Complex_RFC_train_score = Non_Complex_RFC.score(Non_Complex_x_train,  
    Non_Complex_y_train)  
11 Non_Complex_RFC_test_score = Complex_RFC.score(Non_Complex_x_test,  
    Non_Complex_y_test)  
12 Semi_Complex_RFC = RandomForestClassifier()  
13 Semi_Complex_RFC.fit(Semi_Complex_x_train,Semi_Complex_y_train)  
14 Semi_Complex_RFC_Y_Predict = Semi_Complex_RFC.predict(Semi_Complex_x_test)  
15 Semi_Complex_RFC_train_score = Semi_Complex_RFC.score(Semi_Complex_x_train,  
    Semi_Complex_y_train)  
16 Semi_Complex_RFC_test_score = Semi_Complex_RFC.score(Semi_Complex_x_test,  
    Semi_Complex_y_test)
```

支持向量机

```
1  # SVM  
2  Complex_SVM = SVC()  
3  Complex_SVM.fit(Complex_x_train,Complex_y_train)  
4  Complex_SVM_Y_Predict = Complex_SVM.predict(Complex_x_test)  
5  Complex_SVM_train_score = Complex_SVM.score(Complex_x_train,  
    Complex_y_train)  
6  Complex_SVM_test_score = Complex_SVM.score(Complex_x_test, Complex_y_test)  
7  Non_Complex_SVM = SVC()  
8  Non_Complex_SVM.fit(Non_Complex_x_train,Non_Complex_y_train)  
9  Non_Complex_SVM_Y_Predict = Non_Complex_SVM.predict(Non_Complex_x_test)  
10 Non_Complex_SVM_train_score = Non_Complex_SVM.score(Non_Complex_x_train,  
    Non_Complex_y_train)  
11 Non_Complex_SVM_test_score = Complex_SVM.score(Non_Complex_x_test,  
    Non_Complex_y_test)  
12 Semi_Complex_SVM = SVC()  
13 Semi_Complex_SVM.fit(Semi_Complex_x_train,Semi_Complex_y_train)  
14 Semi_Complex_SVM_Y_Predict = Semi_Complex_SVM.predict(Semi_Complex_x_test)  
15 Semi_Complex_SVM_train_score = Semi_Complex_SVM.score(Semi_Complex_x_train,  
    Semi_Complex_y_train)  
16 Semi_Complex_SVM_test_score = Semi_Complex_SVM.score(Semi_Complex_x_test,  
    Semi_Complex_y_test)
```

神经网络

```
1 # 神经网络
2 Complex_MLP = MLPClassifier(solver='lbfgs', alpha=1e-5,hidden_layer_sizes=(5, 5), random_state=1)
3 Complex_MLP.fit(Complex_x_train,Complex_y_train)
4 Complex_MLP_Y_Predict = Complex_MLP.predict(Complex_x_test)
5 Complex_MLP_train_score = Complex_MLP.score(Complex_x_train,Complex_y_train)
6 Complex_MLP_test_score = Complex_MLP.score(Complex_x_test, Complex_y_test)
7 Non_Complex_MLP = MLPClassifier(solver='lbfgs', alpha=1e-5,hidden_layer_sizes=(5, 5), random_state=1)
8 Non_Complex_MLP.fit(Non_Complex_x_train,Non_Complex_y_train)
9 Non_Complex_MLP_Y_Predict = Non_Complex_MLP.predict(Non_Complex_x_test)
10 Non_Complex_MLP_train_score = Non_Complex_MLP.score(Non_Complex_x_train,Non_Complex_y_train)
11 Non_Complex_MLP_test_score = Complex_MLP.score(Non_Complex_x_test,Non_Complex_y_test)
12 Semi_Complex_MLP = MLPClassifier(solver='lbfgs', alpha=1e-5,hidden_layer_sizes=(5, 5), random_state=1)
13 Semi_Complex_MLP.fit(Semi_Complex_x_train,Semi_Complex_y_train)
14 Semi_Complex_MLP_Y_Predict = Semi_Complex_MLP.predict(Semi_Complex_x_test)
15 Semi_Complex_MLP_train_score = Semi_Complex_MLP.score(Semi_Complex_x_train,Semi_Complex_y_train)
16 Semi_Complex_MLP_test_score = Semi_Complex_MLP.score(Semi_Complex_x_test,Semi_Complex_y_test)
```

线性回归

```
1 # 线性回归
2 Complex_LR = LogisticRegression()
3 Complex_LR.fit(Complex_x_train,Complex_y_train)
4 Complex_LR_Y_Predict = Complex_LR.predict(Complex_x_test)
5 Complex_LR_train_score = Complex_LR.score(Complex_x_train, Complex_y_train)
6 Complex_LR_test_score = Complex_LR.score(Complex_x_test, Complex_y_test)
7 Non_Complex_LR = LogisticRegression()
8 Non_Complex_LR.fit(Non_Complex_x_train,Non_Complex_y_train)
9 Non_Complex_LR_Y_Predict = Non_Complex_LR.predict(Non_Complex_x_test)
10 Non_Complex_LR_train_score = Non_Complex_LR.score(Non_Complex_x_train,Non_Complex_y_train)
11 Non_Complex_LR_test_score = Complex_LR.score(Non_Complex_x_test,Non_Complex_y_test)
12 Semi_Complex_LR = LogisticRegression()
13 Semi_Complex_LR.fit(Semi_Complex_x_train,Semi_Complex_y_train)
14 Semi_Complex_LR_Y_Predict = Semi_Complex_LR.predict(Semi_Complex_x_test)
15 Semi_Complex_LR_train_score = Semi_Complex_LR.score(Semi_Complex_x_train,Semi_Complex_y_train)
16 Semi_Complex_LR_test_score = Semi_Complex_LR.score(Semi_Complex_x_test,Semi_Complex_y_test)
```

模型评估

特征重要性

特征重要性计算

```
1 # 非合成类
2 Non_Complex_Imp = 100.0*(Non_Complex_RFC.feature_importances_/
3                       max(Non_Complex_RFC.feature_importances_))
4 Non_Complex_Importance =
5 pd.DataFrame(np.array([Non_Complex_x_test.columns,Non_Complex_Imp]).T,
6               columns=["Var","非合成类重要度"])
7 Non_Complex_Importance["非合成类重要度"].astype("float")
8 Non_Complex_Importance_Sort= Non_Complex_Importance.sort_values(by="非合成类重
9 要度",ascending=False)
10 # 合成类
11 Complex_Imp = 100.0*(Complex_RFC.feature_importances_/
12                  max(Complex_RFC.feature_importances_))
13 Complex_Importance =
14 pd.DataFrame(np.array([Complex_x_test.columns,Complex_Imp]).T,
15               columns=["Var","合成类重要度"])
16 Complex_Importance["合成类重要度"].astype("float")
17 Complex_Importance_Sort= Complex_Importance.sort_values(by="合成类重要
18 度",ascending=False)
19 # 半合成类
20 Semi_Complex_Imp = 100.0*(Semi_Complex_RFC.feature_importances_/
21                           max(Semi_Complex_RFC.feature_importances_))
22 Semi_Complex_Importance =
23 pd.DataFrame(np.array([Semi_Complex_x_test.columns,Semi_Complex_Imp]).T,
24               columns=["Var","半合成类重要度"])
25 Semi_Complex_Importance["半合成类重要度"].astype("float")
26 Semi_Complex_Importance_Sort= Semi_Complex_Importance.sort_values(by="半合成
27 类重要度",ascending=False)
```

变量名匹配

```
1 Complex_Importance_Rename = pd.merge(Complex_Importance_Sort,
2                                     ACS_All_5YR_DP02_metadata_Dup,
3                                     on="Var",how="left")
4 Non_Complex_Importance_Rename = pd.merge(Non_Complex_Importance_Sort,
5                                     ACS_All_5YR_DP02_metadata_Dup,
6                                     on="Var",how="left")
7 Semi_Complex_Importance_Rename = pd.merge(Semi_Complex_Importance_Sort,
8                                     ACS_All_5YR_DP02_metadata_Dup,
9                                     on="Var",how="left")
10 All_Importance_Rename = pd.concat([Complex_Importance_Rename,
11                                   Non_Complex_Importance_Rename,
12                                   Semi_Complex_Importance_Rename],
13                                   axis=1,join="outer")
```

KFold验证

非合成类

```
1 strKFold = StratifiedKFold(n_splits=10,shuffle=False,random_state=1234)
2 Non_Complex_KNN_Kfold = cross_val_score(Non_Complex_KNN,
3     Non_Complex.ix[:,list(All_Corr_Condi["变量名"])],
4     Non_Complex.ix[:, "DrugReportsclass"],
5     scoring='accuracy',
6     cv=strKFold)
7 Non_Complex_Decision_Kfold = cross_val_score(Non_Complex_Decision,
8     Non_Complex.ix[:,list(All_Corr_Condi["变量名"])],
9     Non_Complex.ix[:, "DrugReportsclass"],
10    scoring='accuracy',
11    cv=strKFold)
12 Non_Complex_RFC_Kfold = cross_val_score(Non_Complex_RFC,
13     Non_Complex.ix[:,list(All_Corr_Condi["变量名"])],
14     Non_Complex.ix[:, "DrugReportsclass"],
15     scoring='accuracy',
16     cv=strKFold)
17 Non_Complex_SVM_Kfold = cross_val_score(Non_Complex_SVM,
18     Non_Complex.ix[:,list(All_Corr_Condi["变量名"])],
19     Non_Complex.ix[:, "DrugReportsclass"],
20     scoring='accuracy',
21     cv=strKFold)
22 Non_Complex_MLP_Kfold = cross_val_score(Non_Complex_MLP,
23     Non_Complex.ix[:,list(All_Corr_Condi["变量名"])],
24     Non_Complex.ix[:, "DrugReportsclass"],
25     scoring='accuracy',
26     cv=strKFold)
27 Non_Complex_LR_Kfold = cross_val_score(Non_Complex_LR,
28     Non_Complex.ix[:,list(All_Corr_Condi["变量名"])],
29     Non_Complex.ix[:, "DrugReportsclass"],
30     scoring='accuracy',
31     cv=strKFold)
```

合成类

```
1 strKFold = StratifiedKFold(n_splits=10,shuffle=False,random_state=1234)
2 Complex_KNN_Kfold = cross_val_score(Complex_KNN,
3     Complex.ix[:,list(All_Corr_Condi["变量名"])],
4     Complex.ix[:, "DrugReportsclass"],
5     scoring='accuracy',
6     cv=strKFold)
7 Complex_Decision_Kfold = cross_val_score(Complex_Decision,
8     Complex.ix[:,list(All_Corr_Condi["变量名"])],
9     Complex.ix[:, "DrugReportsclass"],
10    scoring='accuracy',
11    cv=strKFold)
12 Complex_RFC_Kfold = cross_val_score(Complex_RFC,
13     Complex.ix[:,list(All_Corr_Condi["变量名"])],
14     Complex.ix[:, "DrugReportsclass"],
15     scoring='accuracy',
16     cv=strKFold)
17 Complex_SVM_Kfold = cross_val_score(Complex_SVM,
18     Complex.ix[:,list(All_Corr_Condi["变量名"])],
19     Complex.ix[:, "DrugReportsclass"],
20     scoring='accuracy',
```

```

21 cv=strKFold)
22 Complex_MLP_Kfold = cross_val_score(Complex_MLP,
23                                     Complex.ix[:,list(All_Corr_Condi["变量名"])],
24                                     Complex.ix[:, "DrugReportsclass"],
25                                     scoring='accuracy',
26                                     cv=strKFold)
27 Complex_LR_Kfold = cross_val_score(Complex_LR,
28                                     Complex.ix[:,list(All_Corr_Condi["变量名"])],
29                                     Complex.ix[:, "DrugReportsclass"],
30                                     scoring='accuracy',
31                                     cv=strKFold)

```

半合成类

[illegible]

Kfold结果值

非合成类

```
1 Non_Complex_Kfold_Outdata = pd.DataFrame(np.array([Non_Complex_KNN_Kfold,
2
3 Non_Complex_Decision_Kfold,
4
5 Non_Complex_RFC_Kfold,
6 Non_Complex_SVM_Kfold,
7 Non_Complex_MLP_Kfold,
8 Non_Complex_LR_Kfold])).T.round(3),
9 columns=["KNN", "决策树", "随机森林", "支持向量机", "神经网络", "线性回归"])
10 Non_Complex_Kfold_Box = Non_Complex_Kfold_Outdata.stack().reset_index()
11 Non_Complex_Kfold_Box = Non_Complex_Kfold_Box.rename(columns={"level_1": "各类机器学习算法", "0": "kfold值"})
```

合成类

```
1 Complex_Kfold_Outdata = pd.DataFrame(np.array([Complex_KNN_Kfold,
2
3 Complex_Decision_Kfold,
4 Complex_RFC_Kfold,
5 Complex_SVM_Kfold,
6 Complex_MLP_Kfold,
7 Complex_LR_Kfold])).T.round(3),
8 columns=["KNN", "决策树", "随机森林", "支持向量机", "神经网络", "线性回归"])
9 Complex_Kfold_Box = Complex_Kfold_Outdata.stack().reset_index()
10 Complex_Kfold_Box = Complex_Kfold_Box.rename(columns={"level_1": "各类机器学习算法", "0": "kfold值"})
```

半合成类

```
1 Semi_Complex_Kfold_Outdata = pd.DataFrame(np.array([Semi_Complex_KNN_Kfold,
2
3 Semi_Complex_Decision_Kfold,
4
5 Semi_Complex_RFC_Kfold,
6 Semi_Complex_SVM_Kfold,
7 Semi_Complex_MLP_Kfold,
8 Semi_Complex_LR_Kfold])).T.round(3),
9 columns=["KNN", "决策树", "随机森林", "支持向量机", "神经网络", "线性回归"])
10 Semi_Complex_Kfold_Box = Semi_Complex_Kfold_Outdata.stack().reset_index()
11 Semi_Complex_Kfold_Box = Semi_Complex_Kfold_Box.rename(columns={"level_1": "各类机器学习算法", "0": "kfold值"})
```

Kfold箱式图

```
1 f, (Complex_Box1, Non_Complex_Box2, Semi_Complex_Box3) =  
  plt.subplots(nrows=3, figsize=(15, 15))  
2  
3 sns.boxplot(x = "各类机器学习算法", y = Complex_Kfold_Box.ix[:, -1],  
4             data=Complex_Kfold_Box, ax=Complex_Box1)  
5 Complex_Box1.set_xlabel('')  
6 Complex_Box1.set_ylabel('合成类')  
7  
8 sns.boxplot(x = "各类机器学习算法", y = Non_Complex_Kfold_Box.ix[:, -1],  
9             data=Non_Complex_Kfold_Box, ax=Non_Complex_Box2)  
10 Non_Complex_Box2.set_xlabel('')  
11 Non_Complex_Box2.set_ylabel('非合成类')  
12  
13 sns.boxplot(x = "各类机器学习算法", y = Semi_Complex_Kfold_Box.ix[:, -1],  
14             data=Semi_Complex_Kfold_Box, ax=Semi_Complex_Box3)  
15 Semi_Complex_Box3.set_xlabel('')  
16 Semi_Complex_Box3.set_ylabel('半合成类')  
17  
18 plt.tight_layout()  
19 All_Box = plt.gcf()
```

导出结果

数据清洗结果

```
1 # 整理后的ACS_ALL  
2 ACS_ALL_5YR_DP02.to_csv(file_path("02_output", "ACS_ALL_5YR_DP02.csv"), encoding="utf-8-sig")  
3 # 整理后的MCM_NFLIS  
4 MCM_NFLIS_Class_Clear.to_csv(file_path("02_output", "MCM_NFLIS_Class_Clear.csv"), encoding="utf-8-sig")  
5 # 整理后的ACS_All_5YR_DP02_metadata  
6 ACS_All_5YR_DP02_metadata_Dup.to_csv(file_path("02_output", "ACS_All_5YR_DP02_metadata_Dup.csv"), encoding="utf-8-sig")  
7 # 按照三类药物数据合并  
8 NFLIS_and_ACS_ALL_Out.to_csv(file_path("02_output", "NFLIS_and_ACS_ALL_Out.csv"), encoding="utf-8-sig")  
9 # 相关系数大于0.5的变量  
10 All_Corr_Condi.to_csv(file_path("02_output", "All_Corr_Condi.csv"), encoding="utf-8-sig")  
11 # 相关系数大于0.5的变量的社会经济数据表  
12 NFLIS_and_ACS_All_Corr_Condi.to_csv(file_path("02_output", "NFLIS_and_ACS_All_Corr_Condi.csv"), encoding="utf-8-sig")  
13 # 归一化后相关系数大于0.5的变量的社会经济数据表  
14 NFLIS_and_ACS_All_Condi_Normal.to_csv(file_path("02_output", "NFLIS_and_ACS_All_Condi_Normal.csv"), encoding="utf-8-sig")
```


统计描述结果

```
1 NFLIS_Figure2_Data_Class1.to_csv(file_path("02_output","NFLIS_Figure2_Data_Class1.csv"),encoding="utf-8-sig")
2 NFLIS_Figure2_Data_Class2.to_csv(file_path("02_output","NFLIS_Figure2_Data_Class2.csv"),encoding="utf-8-sig")
3 NFLIS_Figure2_Data_Class3.to_csv(file_path("02_output","NFLIS_Figure2_Data_Class3.csv"),encoding="utf-8-sig")
```

```
1 Figure1.savefig(file_path("02_output","Figure1_Pie.jpg"),dpi=500)
2 Figure2.savefig(file_path("02_output","Figure2_Bar.jpg"),dpi=500)
3 Figure3.savefig(file_path("02_output","Figure3_HeatMap.jpg"),dpi=500)
4 Figure4.savefig(file_path("02_output","Figure4_Plot.jpg"),dpi=500)
5
```

模型评估结果

```
1 # kfold箱式图
2 All_Box.savefig(file_path("02_output","All_Box.jpg"),dpi=500)
3 # 三类药物特征重要度
4 All_Importance_Rename.to_csv(file_path("02_output","All_Importance_Rename.csv"),encoding="utf-8-sig")
5 # K折验证
6 Complex_Kfold_Outdata.to_csv(file_path("02_output","Complex_Kfold_Outdata.csv"),encoding="utf-8-sig")
7 Semi_Complex_Kfold_Outdata.to_csv(file_path("02_output","Semi_Complex_Kfold_Outdata.csv"),encoding="utf-8-sig")
8 Non_Complex_Kfold_Outdata.to_csv(file_path("02_output","Non_Complex_Kfold_Outdata.csv"),encoding="utf-8-sig")
```

模型的混淆矩阵

```
1
2 print('KNN合成类混淆矩阵为: ', confusion_matrix(Complex_y_test, Complex_KNN_Y_Predict), sep='\n')
3 print('KNN半合成类混淆矩阵为: ', confusion_matrix(Semi_Complex_y_test, Semi_Complex_KNN_Y_Predict), sep='\n')
4 print('KNN非合成类混淆矩阵为: ', confusion_matrix(Non_Complex_y_test, Non_Complex_KNN_Y_Predict), sep='\n')
5
6 print('Decision合成类混淆矩阵为: ', confusion_matrix(Complex_y_test, Complex_Ddecision_Y_Predict), sep='\n')
7 print('Decision半合成类混淆矩阵为: ', confusion_matrix(Semi_Complex_y_test, Semi_Complex_Ddecision_Y_Predict), sep='\n')
8 print('Decision非合成类混淆矩阵为: ', confusion_matrix(Non_Complex_y_test, Non_Complex_Ddecision_Y_Predict), sep='\n')
9
10 print('RFC合成类混淆矩阵为: ', confusion_matrix(Complex_y_test, Complex_RFC_Y_Predict), sep='\n')
11 print('RFC半合成类混淆矩阵为: ', confusion_matrix(Semi_Complex_y_test, Semi_Complex_RFC_Y_Predict), sep='\n')
12 print('RFC非合成类混淆矩阵为: ', confusion_matrix(Non_Complex_y_test, Non_Complex_RFC_Y_Predict), sep='\n')
13
```

```

14 print('SVM合成类混淆矩阵为: ', confusion_matrix(Complex_y_test,
    Complex_SVM_Y_Predict), sep='\n')
15 print('SVM半合成类混淆矩阵为: ', confusion_matrix(Semi_Complex_y_test,
    Semi_Complex_SVM_Y_Predict), sep='\n')
16 print('SVM非合成类混淆矩阵为: ', confusion_matrix(Non_Complex_y_test,
    Non_Complex_SVM_Y_Predict), sep='\n')
17
18 print('MLP合成类混淆矩阵为: ', confusion_matrix(Complex_y_test,
    Complex_MLP_Y_Predict), sep='\n')
19 print('MLP半合成类混淆矩阵为: ', confusion_matrix(Semi_Complex_y_test,
    Semi_Complex_MLP_Y_Predict), sep='\n')
20 print('MLP非合成类混淆矩阵为: ', confusion_matrix(Non_Complex_y_test,
    Non_Complex_MLP_Y_Predict), sep='\n')
21
22 print('LR合成类混淆矩阵为: ', confusion_matrix(Complex_y_test,
    Complex_LR_Y_Predict), sep='\n')
23 print('LR半合成类混淆矩阵为: ', confusion_matrix(Semi_Complex_y_test,
    Semi_Complex_LR_Y_Predict), sep='\n')
24 print('LR非合成类混淆矩阵为: ', confusion_matrix(Non_Complex_y_test,
    Non_Complex_LR_Y_Predict), sep='\n')
25

```

模型的评估报告

```

1 print("最近邻法合成阿片类: ")
2 print(classification_report(Complex_KNN_Y_Predict,Complex_y_test))
3 print("最近邻法非合成阿片类: ")
4 print(classification_report(Non_Complex_KNN_Y_Predict,Non_Complex_y_test))
5 print("最近邻法半合成阿片类: ")
6 print(classification_report(Semi_Complex_KNN_Y_Predict,Semi_Complex_y_test))
7
8 print("决策树合成阿片类: ")
9 print(classification_report(Complex_Decision_Y_Predict,Complex_y_test))
10 print("决策树非合成阿片类: ")
11 print(classification_report(Non_Complex_Decision_Y_Predict,Non_Complex_y_test))
12 print("决策树半合成阿片类: ")
13 print(classification_report(Semi_Complex_Decision_Y_Predict,Semi_Complex_y_test))
14
15 print("随机森林合成阿片类: ")
16 print(classification_report(Complex_RFC_Y_Predict,Complex_y_test))
17 print("随机森林非合成阿片类: ")
18 print(classification_report(Non_Complex_RFC_Y_Predict,Non_Complex_y_test))
19 print("随机森林半合成阿片类: ")
20 print(classification_report(Semi_Complex_RFC_Y_Predict,Semi_Complex_y_test))
21
22 print("支持向量机合成阿片类: ")
23 print(classification_report(Complex_SVM_Y_Predict,Complex_y_test))
24 print("支持向量机非合成阿片类: ")
25 print(classification_report(Non_Complex_SVM_Y_Predict,Non_Complex_y_test))
26 print("支持向量机半合成阿片类: ")
27 print(classification_report(Semi_Complex_SVM_Y_Predict,Semi_Complex_y_test))
28
29 print("神经网络合成阿片类: ")
30 print(classification_report(Complex_MLP_Y_Predict,Complex_y_test))
31 print("神经网络非合成阿片类: ")

```

```
32 print(classification_report(Non_Complex_MLP_Y_Predict,Non_Complex_y_test))
33 print("神经网络半合成阿片类: ")
34 print(classification_report(Semi_Complex_MLP_Y_Predict,Semi_Complex_y_test))
35
36 print("线性回归合成阿片类: ")
37 print(classification_report(Complex_LR_Y_Predict,Complex_y_test))
38 print("线性回归非合成阿片类: ")
39 print(classification_report(Non_Complex_LR_Y_Predict,Non_Complex_y_test))
40 print("线性回归半合成阿片类: ")
41 print(classification_report(Semi_Complex_LR_Y_Predict,Semi_Complex_y_test))
42
```