1、处理数据

- 删去某些特征缺失数据的的数据
- 用随机森林补全那些缺失数据多的数据（图片：处理数据代码.png）

```
]:
# 处理数据
import numpy as np
import pandas as pd

# 读取文件
data_set = pd.read_csv('train.csv',header=None)
# 将？用NULL 来替换
new_data_set = data_set.replace(['?'],[None])
new_data_set.to_csv('new_data_set1.csv')
"""
## 用随机森林补全数据
* 特征数据完整的特征  0  2  6  8  13
* 因为有 7000 多条数据，对于特征缺少比较小的部分的数据进行删除
* * 1  3  4  9  11
* 对于 5  7  10 12   缺失较多的特征用随机森林来补全
"""
# 删数据
new_data_set = pd.read_csv('new_data_set.csv')
new_data_set.dropna(subset=['1','3','4','9','11'],inplace=True)

# 利用随机森林补全  12 列
from sklearn.ensemble import RandomForestRegressor

def set_missing_ages(df):
    # 把已有数值特征取出来丢进Random Forest Regressor（随机森林回归）中
    all_df = df[['0', '1','2' ,'3', '4', '6', '8', '9','11','12','13']]
    know_goal = all_df[all_df['12'].notnull()]
    unknow_goal = all_df[all_df['12'].isnull()]

    # 构建随机森林模型
    y = know_goal['12'].values.tolist()
    know_goal = know_goal.drop('12',1)
    x = know_goal.values

    rfr = RandomForestRegressor(random_state=0,n_estimators=2000,n_jobs=-1)
    rfr.fit(x,y)

    un_x = unknow_goal.drop('12',1).values
    pred = rfr.predict(un_x)

    for i in range(len(pred)):
        pred[i] = int(pred[i])

    df.loc[(df['12'].isnull()),'12'] = pred

    return df
new_data_set = set_missing_ages(new_data_set)
# 10
def set_missing_ages(df):
    # 把已有数值特征取出来丢进Random Forest Regressor（随机森林回归）中
    all_df = df[['0', '1','2' ,'3', '4', '6', '8', '9','10','11','12','13']]
    know_goal = all_df[all_df['10'].notnull()]
    unknow_goal = all_df[all_df['10'].isnull()]

    # 构建随机森林模型
```

```
    y = know_goal['10'].values.tolist()
    know_goal = know_goal.drop('10',1)
    x = know_goal.values

    rfr = RandomForestRegressor(random_state=0,n_estimators=2000,n_jobs=-1)
    rfr.fit(x,y)

    un_x = unknow_goal.drop('10',1).values
    pred = rfr.predict(un_x)

    for i in range(len(pred)):
        pred[i] = int(pred[i])

    df.loc[(df['10'].isnull()),'10'] = pred

    return df
new_data_set = set_missing_ages(new_data_set)

# 7
def set_missing_ages(df):
    # 把已有数值特征取出来丢进Random Forest Regressor（随机森林回归）中
    all_df = df[['0', '1','2' ,'3', '4', '6','7', '8', '9','10','11','12','13']]
    know_goal = all_df[all_df['7'].notnull()]
    unknow_goal = all_df[all_df['7'].isnull()]

    # 构建随机森林模型
    y = know_goal['7'].values.tolist()
    know_goal = know_goal.drop('7',1)
    x = know_goal.values

    rfr = RandomForestRegressor(random_state=0,n_estimators=2000,n_jobs=-1)
    rfr.fit(x,y)

    un_x = unknow_goal.drop('7',1).values
    pred = rfr.predict(un_x)

    for i in range(len(pred)):
        pred[i] = int(pred[i])

    df.loc[(df['7'].isnull()),'7'] = pred

    return df
new_data_set = set_missing_ages(new_data_set)


# 5
def set_missing_ages(df):
    # 把已有数值特征取出来丢进Random Forest Regressor（随机森林回归）中
    all_df = df[['0', '1','2' ,'3', '4','5', '6', '8', '9','10','11','12','13']]
    know_goal = all_df[all_df['5'].notnull()]
    unknow_goal = all_df[all_df['5'].isnull()]

    # 构建随机森林模型
    y = know_goal['5'].values.tolist()
    know_goal = know_goal.drop('5',1)
    x = know_goal.values

    rfr = RandomForestRegressor(random_state=0,n_estimators=2000,n_jobs=-1)
    rfr.fit(x,y)

    un_x = unknow_goal.drop('5',1).values
```

```
117         pred = rfr.predict(un_x)
118
119         for i in range(len(pred)):
120             pred[i] = int(pred[i])
121
122         df.loc[(df['5'].isnull()),'5'] = pred
123
124         return df
125     new_data_set = set_missing_ages(new_data_set)
126     new_data_set.info()
127
128     # 保存处理好的数据
129     new_data_set.to_csv('complete_data_set1.csv')
130
```

executed in 36.8s, finished 11:38:44 2020-10-23

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 6694 entries, 0 to 7192
Data columns (total 15 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   Unnamed: 0   6694 non-null   int64
 1   0            6694 non-null   int64
 2   1            6694 non-null   float64
 3   2            6694 non-null   int64
 4   3            6694 non-null   float64
 5   4            6694 non-null   float64
 6   5            6694 non-null   float64
 7   6            6694 non-null   int64
 8   7            6694 non-null   float64
 9   8            6694 non-null   int64
 10  9            6694 non-null   float64
 11  10           6694 non-null   float64
 12  11           6694 non-null   float64
 13  12           6694 non-null   float64
 14  13           6694 non-null   int64
dtypes: float64(9), int64(6)
memory usage: 836.8 KB
```

2、挑选模型

- 为了保证模型的准确率，所以使用5折交叉验证的方法来选取最佳模型

KNN模型

- 代码：

In [ ]:

```
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
from sklearn import  metrics
# 分层划分数据
from sklearn.model_selection import StratifiedKFold
# 获取数据
datas = pd.read_csv("complete_data_set.csv")
datas = datas.drop('Unnamed: 0',1).values
X = datas[:,:-1]
Y = datas[:,-1]
# 分层划分数据，采用五折家产验证
sfolder = StratifiedKFold(n_splits=5,shuffle=True, random_state=13)

# 利用cross_val_score自动获得结果
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import cross_val_score
from sklearn.preprocessing import MinMaxScaler
min_max = MinMaxScaler()
X = min_max.fit_transform(X)
print(X[0])

max_ = 0.0
string = None

k_list = [i for i in range(1,480,2)]
for k in k_list:
    clf=KNeighborsClassifier(n_neighbors=k)
    scores=cross_val_score(clf, X, Y, cv=sfolder)
    print("k = "+str(k),end="  ")
    print(scores,end="  ")
    avg_s = np.mean(np.array(scores))
    print(avg_s)
    if max_ < avg_s:
                max_ = avg_s
                string = ("{}   {}   {}   ->{}".format(c,depth,feat,avg_s))

print("最好：  "+string)
```

- 运行最好效果参数
-
  - 最好： 47 ->0.33179033948767067

随机森林

- 代码

'''key

```
# 随机森林
# 利用cross_val_score自动获得结果
import numpy  as np
import pandas as pd
from sklearn.model_selection import cross_val_score
# 随机森林
from sklearn.ensemble import RandomForestClassifier

# 获取数据
datas = pd.read_csv("complete_data_set.csv")
datas = datas.drop('Unnamed: 0',1).values
X = datas[:,:-1]
Y = datas[:,-1]
# 分层划分数据，采用五折家产验证
```

```
sfolder = StratifiedKFold(n_splits=5,shuffle=True, random_state=13)

criterion_list = ['gini','entropy']
max_depth_list = [i for i in range(4,16)]
max_features_list = ['auto','sqrt','log2']

max_ = 0.0
string = None
for c in criterion_list:
    for depth in max_depth_list:
        for feat in max_features_list:
            clf =
RandomForestClassifier(criterion=c,max_depth=depth,max_features=feat,n_estimators=
120)
            scores=cross_val_score(clf, X, Y, cv=sfolder)
            avg_s = np.mean(np.array(scores))
            print("{}  {}  {}   ->{}".format(c,depth,feat,avg_s))
            if max_ < avg_s:
                max_ = avg_s
                string = ("{}  {}  {}   ->{}".format(c,depth,feat,avg_s))

print("最好： "+string)
```

'''

- 运行最好效果参数
- - gini 8 log2 ->0.3612178510389142

SVM支持向量机

- 代码

'''

```
import numpy as np
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
from sklearn.model_selection import cross_val_score

# 获取数据
datas = pd.read_csv("/kaggle/input/123345/complete_data_set.csv")
datas = datas.drop('Unnamed: 0',1).values
X = datas[:,:-1]
Y = datas[:,-1]
# 分层划分数据，采用五折家产验证
sfolder = StratifiedKFold(n_splits=5,shuffle=True, random_state=13)
# 对数据进行归一化
from sklearn.preprocessing import MinMaxScaler
min_max = MinMaxScaler()
```

```
X = min_max.fit_transform(X)

C_list = [1e-3, 1e-2, 1e-1, 1, 10, 100, 1000]
kernel_list  = ['poly', 'rbf','sigmoid','precomputed']
gamma_list = ['scale','auto']
class_weight_list = ['balanced']


max_ = 0.0
string = None

for k in kernel_list:
    for gamma in gamma_list:
        for c in C_list:
            for class_weight in class_weight_list:
                svc = SVC(kernel=k,gamma=gamma,C=c,class_weight=class_weight)
                clf = make_pipeline(StandardScaler(),svc)
                scores=cross_val_score(clf, X, Y, cv=sfolder)
                avg_s = np.mean(np.array(scores))
                print("{}  {}  {} {} ->{}".format(k,gamma,c,class_weight,avg_s))
                if max_ < avg_s:
                    max_ = avg_s
                    string = ("{}  {}  {} {} ->
{}".format(k,gamma,c,class_weight,avg_s))

print(string)
```

'''

- 运行最好效果参数
- - rbg auto 1 balanced ->0.3462178510389142(应该是这个参数，当时跑了好几个小时得出来的)

神经网络

- 代码

'''key

```
# 利用cross_val_score自动获得结果
import numpy  as np
import pandas as pd
from sklearn.model_selection import cross_val_score, StratifiedKFold
# 获取数据
from sklearn.neural_network import MLPClassifier
from sklearn.preprocessing import MinMaxScaler
min_max = MinMaxScaler()

import warnings

warnings.filterwarnings('ignore')
```

```
datas = pd.read_csv("/kaggle/input/123345/complete_data_set.csv")
datas = datas.drop('Unnamed: 0', 1).values
X = datas[:, :-1]
X = min_max.fit_transform(X)
Y = datas[:, -1]
# 分层划分数据，采用五折家产验证
sfolder = StratifiedKFold(n_splits=5, shuffle=True, random_state=13)

# 激活函数
activation_list = ['relu']
# 求解器
solver_list = ['lbfgs']
alpha_list = [0.01 * i for i in range(0,1001,20)]

max_ = 0.0
string = None

for activation in activation_list:
    for solver in solver_list:
        for alpha in alpha_list:
            clf =
MLPClassifier(random_state=13,max_iter=200,activation=activation,solver=solver,alp
ha=alpha)
            scores = cross_val_score(clf, X, Y, cv=sfolder)
            avg_s = np.mean(np.array(scores))
            print("{}  {}  {}   ->{}".format(activation, solver, alpha, avg_s))
            if max_ < avg_s:
                max_ = avg_s
                string = ("{}  {}  {}   ->{}".format(activation, solver, alpha,
avg_s))

print("最好： " + string)
```

'''

- 运行最好效果参数
- - relu lbfgs 4.6 ->0.3497148330358309

3、根据交叉验证得到的结果，所以选取最佳模型为随机森林

- 参数为：gini 8 log2

构建最佳模型代码：

'''key

```
# 随机森林
import numpy  as np
```

```
import pandas as pd

# 随机森林
from sklearn.ensemble import RandomForestClassifier

# 获取数据
datas = pd.read_csv("complete_data_set.csv")
datas = datas.drop('Unnamed: 0',1).values
X = datas[:,:-1]
Y = datas[:,-1]


clf =
RandomForestClassifier(criterion='gini',max_depth=8,max_features='log2',n_estimato
rs=120)

# 保存模型
from sklearn.externals import joblib

# 保存模型
joblib.dump(clf,'RandomForest.pickle')
```

'''