第十一章

1.实践案例

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

import warnings

warnings.filterwarnings('ignore')

import seaborn as sns

from sklearn.tree import DecisionTreeClassifier

from sklearn.svm import LinearSVC

import scipy.stats as stats

#数据预处理：特征提取，归一化，划分数据集

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

from sklearn.model\_selection import cross\_val\_score #交叉验证k-fold

from sklearn.ensemble import AdaBoostClassifier #集成模型

from sklearn.model\_selection import GridSearchCV #超参数搜索

from sklearn.metrics import roc\_curve,precision\_recall\_curve,auc

from sklearn.metrics import accuracy\_score, precision\_score, recall\_score, f1\_score, log\_loss, confusion\_matrix

data = pd.read\_csv(r"C:\Users\11389\Desktop\heart.csv")

data.columns = ['Age', 'Sex', '胸痛类型', '血压', '胆固醇', '血糖异常',

'心电图', '最大心率', '心绞痛', 'Oldpeak', 'ST\_Slope',

'心脏病']

display(data.head())

data.info()

#1.41分类变量

#判断所有分类变量与y都有显著相关关系，所以全部采用热编码方法处理

#右边的序列为分类变量序列，最后一个为y

def trans01(x):

if (x==1) :

return 'T'

else:

return 'F'

data['血糖异常'] = data['血糖异常'].map(trans01)

temp = data.columns[[1,2,5,6,8,10,11]]

n = len(temp)

cols = int(np.ceil(np.sqrt(n-1)))

rows = int(np.ceil((n-1) / cols))

plt.rcParams['font.sans-serif'] = ['Microsoft YaHei'] # 你可以选择其他中文字体，如 SimHei

plt.rcParams['axes.unicode\_minus'] = False # 解决负号显示问题

fig, axes = plt.subplots(rows, cols, figsize=(cols \* 4, rows \* 4))

axes = axes.flatten()

# 绘制每个子图

for i in range(n-1):

sns.barplot(x=temp[i], y=temp[-1], data=data,hue=temp[i], ax=axes[i])

axes[i].set\_title(f'图 {i + 1}{temp[i]}') # 设置中文标题

axes[i].set\_xlabel(temp[i]) # 设置中文 x 轴标签

axes[i].set\_ylabel(temp[-1]) # 设置中文 y 轴标签

contingency\_table = pd.crosstab(data[temp[i]], data[temp[-1]])

# 显示列联表

#print(contingency\_table)

# 进行卡方检验(相关性)

chi2, p\_value, dof, expected = stats.chi2\_contingency(contingency\_table)

#本身的峰度偏度

#skew = stats.skew(data[temp[i]].value\_counts())

#kurtosis = stats.kurtosis(data[temp[i]].value\_counts())

#显示在右上角

axes[i].text(0.95, 0.95, 'chi2：%.2f'%chi2+'\n卡方检验p：%.3f'%p\_value,

ha='center', va='center',

transform=axes[i].transAxes, fontsize=10, color='black', weight='bold')

for j in range(len(axes[i].patches)):

p = axes[i].patches[j]

height = p.get\_height()

counts = data[temp[i]].value\_counts()[axes[i].get\_xticklabels()[j].get\_text()]

axes[i].text(1/(len(axes[i].patches))\*(j+1/2), height,

'\n %.3f' % height+'\n counts:%d' % counts,

ha='center', va='center',transform=axes[i].transAxes,

fontsize=10, color='black', weight='bold')

# 如果有多余的子图，隐藏它们

for i in range(n-1, len(axes)):

axes[i].axis('off')

# 调整布局

plt.tight\_layout()

plt.show()

x = data.drop(['心脏病'],axis=1)

y = data['心脏病']

x = pd.get\_dummies(x)

x\_train,x\_test,y\_train,y\_test = train\_test\_split(x,y,train\_size=0.7,random\_state=1)

clf0 = AdaBoostClassifier(random\_state=66)

clf0.fit(x\_train,y\_train)

print('训练集score:%.7f' % clf0.score(x\_train,y\_train))

print('测试集score:%.7f' % clf0.score(x\_test,y\_test))

print('交叉验证score:%.7f' % np.mean(cross\_val\_score(clf0,x,y,cv=5)))

clf1 = AdaBoostClassifier(n\_estimators=100,learning\_rate=0.1,random\_state=66)

clf1.fit(x\_train,y\_train)

print('训练集score:%.7f' % clf1.score(x\_train,y\_train))

print('测试集score:%.7f' % clf1.score(x\_test,y\_test))

print('交叉验证score:%.7f' % np.mean(cross\_val\_score(clf1,x,y,cv=10)))

param\_full = {

'estimator': [DecisionTreeClassifier(max\_depth=1),DecisionTreeClassifier(max\_depth=2),LinearSVC()],

'n\_estimators': range(50, 300, 10),

'learning\_rate': np.arange(0, 1.1, 0.1),}

Gsearch = GridSearchCV(

estimator= AdaBoostClassifier(),

param\_grid=param\_full, scoring='accuracy',

cv=5,n\_jobs=-1)

Gsearch.fit(x\_train, y\_train)

print('最优组合:', Gsearch.best\_params\_)

print('test最佳得分：', Gsearch.best\_score\_)

\_,learning\_rate,n\_estimators = list(Gsearch.best\_params\_.values())

clf2=AdaBoostClassifier(estimator=DecisionTreeClassifier(max\_depth=2),random\_state=66,n\_estimators=n\_estimators,learning\_rate=learning\_rate)

clf2.fit(x\_train,y\_train)

y\_pred\_prob = clf2.predict\_proba(x\_test)[:, 1]

y\_pred = clf2.predict(x\_test)

print('测试集score:%.7f' % clf2.score(x\_train,y\_train))

print('测试集score:%.7f' % clf2.score(x\_test,y\_test))

print('交叉验证score:%.7f' % np.mean(cross\_val\_score(clf2,x,y,cv=10))

print('accuracy:%.7f' % accuracy\_score(y\_test, y\_pred))

print('precision:%.7f' % precision\_score(y\_test, y\_pred))

print('f1\_score:%.7f' % f1\_score(y\_test, y\_pred))

print('log\_loss:%.7f' % log\_loss(y\_test, y\_pred))

cm = confusion\_matrix(y\_test, y\_pred)

print('confusion\_matrix:\n',cm)

print('confusion\_matrix:\n',cm)

sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')

plt.xlabel('Predicted')

plt.ylabel('Actual')

plt.title('Confusion Matrix')

plt.show()

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

roc\_auc = auc(fpr, tpr)

plt.plot(fpr, tpr, color='blue', label=f'ROC curve (AUC = {roc\_auc:.2f})')

plt.plot([0, 1], [0, 1], color='gray', linestyle='--')

plt.title('Receiver Operating Characteristic (ROC) Curve')

plt.xlabel('False Positive Rate (FPR)')

plt.ylabel('True Positive Rate (TPR)')

plt.legend(loc='lower right')

plt.show()

precision, recall, thresholds = precision\_recall\_curve(y\_test, y\_pred\_prob)

pr\_auc = auc(recall, precision)

plt.plot(recall, precision, color='blue', label=f'PR curve (AUC = {pr\_auc:.2f})')

plt.plot([0, 1], [1, 0], color='gray', linestyle='--') # 随机分类器的基准线

plt.title('Precision-Recall Curve')

plt.xlabel('Recall')

plt.ylabel('Precision')

plt.legend(loc='lower left')

plt.show()

2.实验一

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

import warnings

warnings.filterwarnings('ignore')

import seaborn as sns

import scipy.stats as stats

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

from sklearn.model\_selection import cross\_val\_score #交叉验证k-fold

from sklearn.ensemble import RandomForestClassifier #集成模型

from sklearn.model\_selection import GridSearchCV #超参数搜索

from sklearn.metrics import roc\_curve,precision\_recall\_curve,auc #ROC,PR,AUC

from sklearn.metrics import accuracy\_score,precision\_score,recall\_score,f1\_score,log\_loss,confusion\_matrix

train = pd.read\_excel(r"C:\Users\11389\Desktop\cleaned\_train.xlsx")

test = pd.read\_excel(r"C:\Users\11389\Desktop\cleaned\_test.xlsx")

passengerId = test['PassengerId']

x\_train = train.drop(['Survived'],axis=1)

y\_train = train['Survived']

x\_test = test.drop(['Survived','PassengerId'],axis=1)

display(x\_train,y\_train)

RandomForestClassifier?

rfc = RandomForestClassifier(oob\_score=True, random\_state=66,n\_jobs=-1)

rfc.fit(x\_train,y\_train)

print('样本内score:%.7f' % rfc.score(x\_train,y\_train))

print('oob\_score:%.7f' % rfc.oob\_score\_)

print('交叉验证score:%.7f' % np.mean(cross\_val\_score(rfc,x\_train,y\_train,cv =5)))

rfc = RandomForestClassifier(n\_estimators=500, oob\_score=True, n\_jobs=-1,

max\_depth = 7,random\_state=66,

min\_samples\_split = 3,max\_features =3,

min\_samples\_leaf = 2,)

rfc.fit(x\_train,y\_train)

print('样本内score:%.7f' % rfc.score(x\_train,y\_train))

print('oob\_score:%.7f' % rfc.oob\_score\_)

print('交叉验证score:%.7f' % np.mean(cross\_val\_score(rfc,x\_train,y\_train,cv =5,n\_jobs=-1)))

y\_pred = rfc.predict(x\_test)

y\_pred= y\_pred.astype(int)

pred\_df = pd.DataFrame({'PassengerId' : passengerId, 'Survived' : y\_pred})

pred\_df.to\_csv(r"C:\Users\11389\Desktop\titanic\_pred1.csv", index = False)

param\_full = {'n\_estimators': range(0, 300, 10), 'max\_depth': range(1, 20),

'min\_samples\_split': range(1, 20),

'min\_samples\_leaf': range(1, 20),

'max\_features': range(1, 20),}

Gsearch = GridSearchCV(estimator=RandomForestClassifier(random\_state=50),

param\_grid=param\_full, scoring='accuracy',

cv=10,n\_jobs=-1)

Gsearch.fit(x\_train, y\_train)

tempstr = 'accuracy'

param\_test1 = {'n\_estimators': range(0, 300, 10)}

gsearch1 = GridSearchCV(estimator=RandomForestClassifier(random\_state=50),

param\_grid=param\_test1, scoring=tempstr,

cv=10,n\_jobs=-1)

gsearch1.fit(x\_train, y\_train)

print('test1平均分数:', gsearch1.cv\_results\_['mean\_test\_score'])

print('最优n\_estimators:', gsearch1.best\_params\_)

print('test1最佳得分：', gsearch1.best\_score\_)

n\_estimators = list(gsearch1.best\_params\_.values())[0]

param\_test2 = {'max\_depth': range(1, 20)}

gsearch2 = GridSearchCV(estimator=RandomForestClassifier(n\_estimators=n\_estimators,

random\_state=66,n\_jobs=-1),

param\_grid=param\_test2,scoring=tempstr, cv=5,n\_jobs=-1)

gsearch2.fit(x\_train, y\_train)

print('test2平均分数:', gsearch2.cv\_results\_['mean\_test\_score'])

print('最优max\_deepth:', gsearch2.best\_params\_)

print('test2最佳得分：', gsearch2.best\_score\_)

max\_depth = list(gsearch2.best\_params\_.values())[0]

param\_test3 = {'min\_samples\_split': range(1, 20)}

gsearch3 = GridSearchCV(estimator=RandomForestClassifier(

n\_estimators=n\_estimators,

max\_depth=max\_depth,

random\_state=66,n\_jobs=-1),

param\_grid=param\_test3,

scoring=tempstr, cv=5,n\_jobs=-1)

gsearch3.fit(x\_train, y\_train)

print('test3平均分数:', gsearch3.cv\_results\_['mean\_test\_score'])

print('最优min\_samples\_split:', gsearch3.best\_params\_)

print('test3最佳得分：', gsearch3.best\_score\_)

min\_samples\_split = list(gsearch3.best\_params\_.values())[0]

param\_test4 = {'min\_samples\_leaf': range(1, 20)}

gsearch4 = GridSearchCV(estimator=RandomForestClassifier(n\_estimators=n\_estimators,

max\_depth=max\_depth,

min\_samples\_split=min\_samples\_split,

random\_state=66,n\_jobs=-1),

param\_grid=param\_test4, scoring=tempstr, cv=5,n\_jobs=-1)

gsearch4.fit(x\_train, y\_train)

print('test4平均分数:', gsearch4.cv\_results\_['mean\_test\_score'])

print('最优min\_samples\_leaf:', gsearch4.best\_params\_)

print('test4最佳得分：', gsearch4.best\_score\_)

min\_samples\_leaf = list(gsearch4.best\_params\_.values())[0]

param\_test5 = {'max\_features': range(1, 20)}

gsearch5 = GridSearchCV(estimator=RandomForestClassifier(n\_estimators=n\_estimators,

max\_depth=max\_depth,

min\_samples\_split=min\_samples\_split,

min\_samples\_leaf=min\_samples\_leaf,

random\_state=66,n\_jobs=-1),

param\_grid=param\_test5, scoring=tempstr, cv=5,n\_jobs=-1)

gsearch5.fit(x\_train, y\_train)

print('test5平均分数:', gsearch5.cv\_results\_['mean\_test\_score'])

print('最优max\_features:', gsearch5.best\_params\_)

print('test5最佳得分：', gsearch5.best\_score\_)

max\_features = list(gsearch5.best\_params\_.values())[0]

clf = RandomForestClassifier(n\_estimators=n\_estimators,

max\_features=max\_features,

max\_depth= max\_depth,

min\_samples\_leaf=min\_samples\_leaf,

min\_samples\_split=min\_samples\_split,

oob\_score=True, random\_state=66,n\_jobs=-1)

clf.fit(x\_train,y\_train)

y\_pred\_prob = clf.predict\_proba(x\_test)[:, 1]

y\_pred = clf.predict(x\_test)

# 整理提交的表

clf.fit(x\_train,y\_train)

y\_pred = clf.predict(x\_test)

y\_pred = y\_pred.astype(int)

pred\_df = pd.DataFrame({'PassengerId' : passengerId, 'Survived' : y\_pred})

pred\_df.to\_csv(r"C:\Users\11389\Desktop\pred\_accuracy.csv", index = False)

x\_train1,x\_test1,y\_train1,y\_test1 = train\_test\_split(x\_train,y\_train,train\_size=0.8,random\_state=66)

clf = RandomForestClassifier(n\_estimators=n\_estimators,

max\_features=max\_features,

max\_depth= max\_depth,

min\_samples\_leaf=min\_samples\_leaf,

min\_samples\_split=min\_samples\_split,

oob\_score=True, random\_state=66,n\_jobs=-1)

clf.fit(x\_train1,y\_train1)

y\_pred\_prob1 = clf.predict\_proba(x\_test1)[:, 1]

y\_pred1 = clf.predict(x\_test1)

print('训练集score:%.7f' % clf.score(x\_train1,y\_train1))

print('测试集score:%.7f' % clf.score(x\_test1,y\_test1))

print('oob\_score:%.7f' % clf.oob\_score\_)

print('accuracy:%.7f' % accuracy\_score(y\_test1, y\_pred1))

print('precision:%.7f' % precision\_score(y\_test1, y\_pred1))

print('f1\_score:%.7f' % f1\_score(y\_test1, y\_pred1))

print('log\_loss:%.7f' % log\_loss(y\_test1, y\_pred1))

cm1 = confusion\_matrix(y\_test1, y\_pred1)

print('confusion\_matrix:\n',cm1)

sns.heatmap(cm1, annot=True, fmt='d', cmap='Blues')

plt.xlabel('Predicted')

plt.ylabel('Actual')

plt.title('Confusion Matrix')

plt.show()

fpr, tpr, thresholds = roc\_curve(y\_test1, y\_pred\_prob1)

roc\_auc = auc(fpr, tpr)

plt.plot(fpr, tpr, color='blue', label=f'ROC curve (AUC = {roc\_auc:.2f})')

plt.plot([0, 1], [0, 1], color='gray', linestyle='--')

plt.title('Receiver Operating Characteristic (ROC) Curve')

plt.xlabel('False Positive Rate (FPR)')

plt.ylabel('True Positive Rate (TPR)')

plt.legend(loc='lower right')

plt.show()

precision, recall, thresholds = precision\_recall\_curve(y\_test1, y\_pred\_prob1)

pr\_auc = auc(recall, precision)

plt.plot(recall, precision, color='blue', label=f'PR curve (AUC = {pr\_auc:.2f})')

plt.plot([0, 1], [1, 0], color='gray', linestyle='--') # 随机分类器的基准线

plt.title('Precision-Recall Curve')

plt.xlabel('Recall')

plt.ylabel('Precision')

plt.legend(loc='lower left')

plt.show()

print('cv\_precisiony:%.7f' % np.mean(cross\_val\_score(clf,x\_train,y\_train,cv=5,n\_jobs=-1,scoring='accuracy')))

print('cv\_precisiony:%.7f' % np.mean(cross\_val\_score(clf,x\_train,y\_train,cv=5,n\_jobs=-1,scoring='precision')))

print('cv\_recall:%.7f' % np.mean(cross\_val\_score(clf,x\_train,y\_train,cv=5,n\_jobs=-1,scoring='recall')))

print('cv\_F1:%.7f' % np.mean(cross\_val\_score(clf,x\_train,y\_train,cv=5,n\_jobs=-1,scoring='f1')))

clf.fit(x\_train,y\_train)

reg\_imp = clf.feature\_importances\_

feature\_importance = pd.DataFrame({'Feature': x.columns, 'Importance': reg\_imp})

# 按重要性得分降序排序

feature\_importance\_sorted=feature\_importance.sort\_values('Importance',

ascending=False)

print(feature\_importance\_sorted)

# 重要性排序画图

plt.figure(figsize=(12, 10))

plt.rcParams['font.size'] = 17 # 设置字体大小

plt.rcParams['font.sans-serif'] = ['SimSun'] # 设置中文字体为宋体

plt.rcParams['font.serif'] = ['Times New Roman'] # 设置英文字体为新罗马

plt.rcParams['axes.unicode\_minus'] = False

ax = sns.barplot(x='Importance', y='Feature', data=feature\_importance\_sorted)

plt.title('Feature Importance', fontsize=28) # 修改标题字体和大小

plt.xlabel('Importance', fontsize=24) # 修改横轴标签字体和大小

plt.ylabel('Feature', fontsize=24) # 修改纵轴标签字体和大小

#标注数值

for p in ax.patches:

ax.annotate("%.3f" % p.get\_width(), (p.get\_width(), p.get\_y() + \

p.get\_height() / 2.),

xytext=(5, 0), textcoords='offset points', ha='left', va='center',

fontsize=16)

plt.show()