基于支持向量机模型的泰坦尼克号幸存者预测

莫梁虹，余荫铠，曾植

2021 年 11 月 27 日

1　模型描述

1.1　模型选择

先说结论，我们最终选择的训练模型是带高斯核的支持向量机（SVM）模型。

要预测泰坦尼克号的乘客是否幸存，预测结果只有生存和死亡两种，这显然是一个分类问题。并且观察我们的原始数据可以发现，每条数据已经标识了是否幸存，如果我们使用机器学习的方法来处理，那么这就是一个有监督学习的分类问题。

对于有监督学习的分类问题，我们所掌握的分析模型有支持向量机、逻辑回归、决策树、随机森林等。我们主要采用支持向量机方法来分析问题。由于我们无法直接判断分类边界是否是线性的，我们采用带有核函数的支持向量机来处理，既可以做线性分类，也可以做非线性分类，是较为通用的。尤其是对于非线性问题，SVM 利用核函数将特征空间映射到高维，能够很好地处理非线性的边界。其次，我们所拿到的原始数据的特征共有 22 个，经过特征工程后，特征数可能会增加或减少，但大体是这个量级，是较少的，而我们的原始数据有 891 条，属于中等，那么这正适合使用带有高斯核或多项式核的支持向量机。SVM 的最终决策函数只由少数的支持向量所确定，计算的复杂性取决于支持向量的数目，一定程度上避免了“维数灾难”。SVM 的不敏感的成本函数“剔除”大量冗余样本，使算法高效且具有较好的鲁棒性。还有，我们的数据量不是太大，即使是采用这种需要大量矩阵运算的算法，也不会消耗太多计算成本。至于我们最终选择了高斯核函数而非其他核函数，这是由网格搜索得到的最优情况。最后，我们在训练模型的过程中使用了交叉验证的训练方法，以提高我们的数据利用率，毕竟这数据量不是太多，同时，该方法还能自动地避免过拟合和欠拟合的问题，方便我们的训练。

1.2　模型解释

支持向量机又称为大间距分类算法，其基本原理就是找到一个分隔超平面，它能把数据正确地分类，并且间距最大。这是一个优化算法。当训练样本线性可分时，通过硬

间距最大化，学习一个线性可分支持向量机。当训练样本近似线性可分时，通过软间距最大化，学习一个线性支持向量机。当训练样本线性不可分时，通过核技巧和软间距最大化，学习一个非线性支持向量机。

对于线性不可分的样品，采用带核技巧的非线性支持向量机的目标函数为

$$L = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} \alpha_i \alpha_j y^{(i)} y^{(j)} K(x^{(i)}, x^{(j)}) \tag{1}$$

预测函数为

$$y = \sum_{i=1}^{m} \alpha_i K(x^{(i)}, x) + b \tag{2}$$

其中，$K(x^{(i)}, x^{(j)})$ 为核函数，用于衡量两个特征向量的相似度。常用的核函数有线性核函数，多项式核函数，高斯核函数。sklearn 的支持向量机程序包中，除了输入我们选择的核函数外，还需要输入两个超参数 c 和 gamma。参数 C 是正数，标识对不符合最大间距规则的样品的惩罚力度。参数 gamma 标识高斯核函数的展宽，当然，对于其他类型的核函数，不需要输入 gamma。

在训练之前，我们有必要对数据进行处理，以保证最佳的学习效果。由于原始数据存在一定的缺失，我们需要进行数据清洗，我们将通过一定的算法和原则缺失的数据，对于数据缺失过多的特征，我们则直接舍去，毕竟如果强行补全，可能会对结果产生人为的影响。我们补全数据的目的是为了得到更多的完整数据来训练，同时尽量使该补全行为不影响预测结果。具体的数据清洗流程和结果，我们会在下文中详述。

清洗完数据之后，我们需要确定模型训练所采用的特征，同时为了构造有效的特征，以提高学习的效率和准确率，我们需要根据实际意义以及相关程度，开展特征工程，构造一些新的特征，拆分或合并一些特征，删去一些特征。具体的分析亦详细展示在下文中。

1.2.1 数据清理

初览数据之前主观分析：Titanic 乘客死亡的原因之一是没有足够的救生艇给乘客和机组人员。虽然幸存下来的运气有一些因素，但一些人比其他人更有可能生存，比如妇女，儿童和上层阶级。

乘客的个人信息包含数值型数据和分类型数据，查看数据后我们发现两类数据均存在缺失值, 对于数值数据存在缺失的年龄，我们采用利用随机森林算法对已有数据进行训练，从而预测出缺失的数据。

在分类型数据中，登船港口 Embarked 和客舱 Cabin 均存在缺失，而由于客舱信息缺失过多，没有填补依据，因此我们舍弃这列数据。

通过分析登船港口的登船人数情况，我们发现 s 港口的登船人数最多，因此我们利

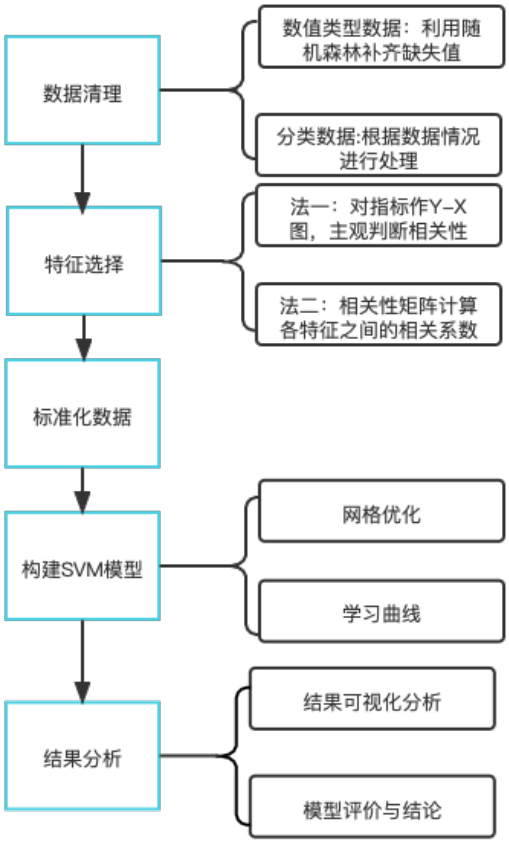

图 1: 流程图

用 s 来填补登船港口缺失信息。

1.2.2 特征工程

我们使用两种方法在 8 个指标中选取特征：

方法一：对指标作百分比堆积条形图，主观判断两者是否有关，以选择特征。

Pclass：从等级 1 到等级 3，死亡人数百分比显著上升，而且是单调递增。显然上层阶级和富豪有更大的生存机会，这与我们对当时社会的认知相符合。

Sex：男性死亡比例很大，而女性恰好相反，这与我们之前的主观分析符合：社会的价值观决定遇到灾难时人们往往更倾向于保护女性，与西方绅士观念相符合。男性的大男子主义也导致男性更多地会牺牲自己。

Age：百分比堆积条形图整体看上去没有体现出年龄与生存情况的关系，但是考虑儿童和老人的情况，可以看出，儿童的生存率明显高于老人。在现实中，这是因为社会往往把妇女儿童都当作需要保护的弱者，她们有更多的机会上救生船，而老人对社会难以再有较大贡献，在生死存亡面前不会受到优待。

Fare：明显地，票价越贵，生存机会越大。理由与 Pclass 指标相同。

表 1: 各特征的意义

PassengerId	乘客 ID	备注
Survived	生存情况	作为纵轴指标 y，不列入特征
Name	姓名（包括头衔）	只提取头衔
Pclass	乘客舱等级	1、2、3 等舱位
Sex	性别	
Age	年龄	有少数缺失，用随机森林预测填充缺失数据
SibSp	堂兄弟/妹个数	
Parch	父母与小孩个数	
Ticket	船票信息	票价已经反映在 Fare 上了，剩余都是编号，同乘客 ID 一样没有意义
Fare	票价	
Cabin	客舱	有大量缺失，没有填充的意义
Embarked	登船港口	

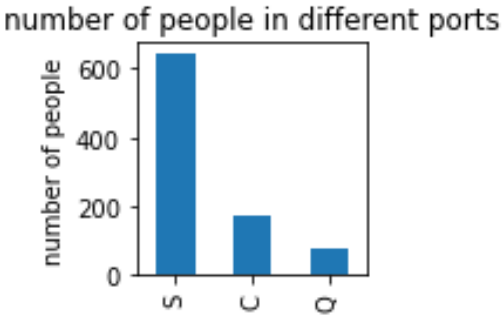

图 2: 登船港口情况和存活情况关系

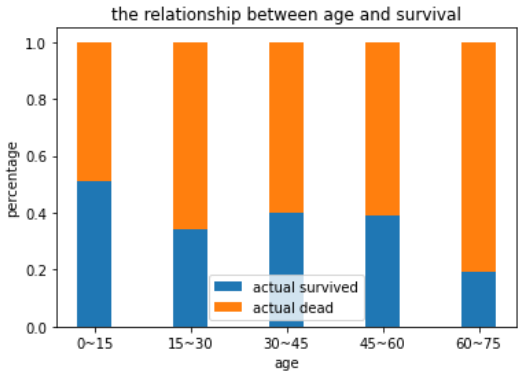

图 3: Age 和存活情况关系

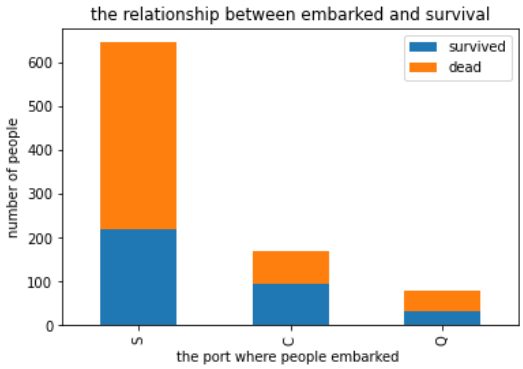

图 4: Embarked 和存活情况关系

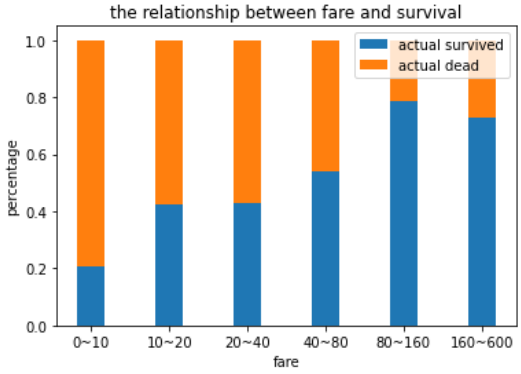

图 5: Fare 和存活情况关系

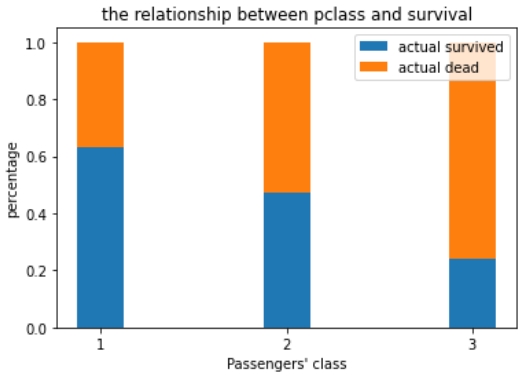

图 6: Pclass 和存活情况关系

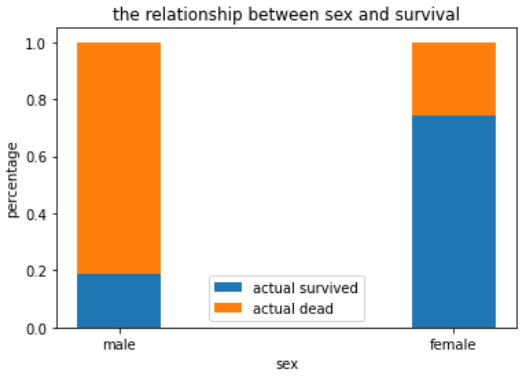

图 7: Sex 和存活情况关系

Embarked：从图像上看登船港口与生存情况没有明显关系，从现实逻辑上也难以判断。

方法一用来寻找显著的、从现实意义上理解能够存在的关系。name,sibsp,parch,ticket 画图显然无法看出与生存情况是否存在关系，故略去。

总结：由图可见，除了 embarked 与存活情况没有显著关系以外，fare,age,sex,pclass 与存活情况都有关系，age 的关系主要体现在数值大小两端。

方法二：对指标进行相应编码，用相关性矩阵计算各特征之间的相关系数，查看各特征与 Survived 的相关系数，选取相关系数绝对值在 0.2 以上的特征。

表 2: 各特征与 Survived 的相关系数

特征名	与 Survived 的相关系数
Mrs	0.341994
Miss	0.332795
Pclass_1	0.285904
Family_Small	0.279855
Fare	0.257307
Embarked_C	0.16824
Pclass_2	0.093349
Master	0.085221
Parch	0.081629
FamilySize	0.016639
Royalty	0.01604
Embarked_Q	0.00365
Officer	-0.031316
SibSp	-0.035322
Age	-0.054519
Family_Large	-0.125147
Embarked_S	-0.149683
Family_Single	-0.203367
Pclass_3	-0.322308
Sex	-0.543351
Mr	-0.549199

方法二用来寻找不直观的关系。且更加客观。（缺点：线性相关，年龄与生存的关系无法体现）

除了基本的特征提取（基本方法是 map 函数映射）外，我们还做了如下特征工程：

(1) 提取头衔。名和姓千差万别，没有规律。叫什么名字显然和是否生存无关，但是 title

比较重要，可以分析不同身份的人的生存率。要提取中间的 title 可以用 split 函数，然后用 strip 函数去除空格。我们建立字典，对不同的头衔进行分类映射，具体操作见附录。

(2) 提取客舱类别。这里可以用 lambda 函数来定义，我们只需要取客舱号的首字母。

(3) 计算家庭人数。从表中可以看出 parch, sibsp 都是家属，我们完全可以把他们相加，放入一个类别 familysize 可以定义为三种 <=1 小家庭 <=3 中等家庭。我们在这看了一下不同分类的个数，单人坐船的有 790 人，中等家庭 394 人，大家庭 125 人。单人坐船占了绝大多数。这个分类基本还是有一定代表性。

2 模型训练

经过数据清洗和特征工程这两步，我们最终得到了 891 个乘客的数据，每个乘客有 16 个信息，我们将其存为一个 891×16 的特征矩阵`X`。

其中，为了保证训练结果不被数据的尺度差异影响，我们在训练模型之前，需要引入`sklearn.preprocessing`模块中的`StandardScaler`对象，对原始的特征矩阵进行标准化处理，`X`是已经标准化后的特征矩阵。

我们采用`sklearn`库中的`SVC`函数进行支持向量机的学习。

`SVC`函数提供了线性核、多项式核和高斯核三种核函数。由于我们原始数据是十六维的，我们无法直观地看出数据是否线性可分，因此我们使用网格优化的思想，对带有三种不同核函数的支持向量机进行训练，取评分最高的模型作为最优模型。在`sklearn`库的网格搜索`GridSearchCV`函数中，需要指定网格化取值的超参数，而由于不同的核函数需要的超参数类型以及数量是不同的，如果我们把核函数类型也作为网格搜索的维度之一，就需要同时列出所有类型核函数需要的超参数，而这会带来相当大量的冗余计算，因为有些超参数在取某些核的情况下不会被用到。因此，尽管我们采用网格搜索的思想检索最优核函数，但是我们在实际操作中，是分别取三种核函数，进行三次网格搜索，再比较三种核函数的最优评分，由此得到最优核函数，这种方法节省了大量计算成本。

最终，我们通过这样的方法发现，最优的核函数为高斯核函数。在初步的网格搜索中，三种核函数分别搜索到的最优评分展示在表3中，可见高斯核函数的评分是较高的。

训练过程中，我们采用`sklearn`库中的`cross_validate`函数进行五折交叉验证，在训练的过程中自动地避免过拟合和欠拟合的问题。在后续的可视化和学习曲线分析中，为了展示我们的训练结果是不存在过拟合和欠拟合的，我们仍利用`train_test_split`函数随机划分了固定的训练集和测试集，而没有继续采用交叉验证来训练。

表 3: 不同核函数的粗搜最优评分

核函数	评分
高斯核	0.833871069
线性核	0.828259369
多项式核	0.827135773

2.1 网格优化

我们选定高斯核函数，对超参数gamma和C进行网格优化。我们的优化本着由粗到细的原则，共进行了三轮网格优化，每轮优化中，在区间内对每个超参数离散化地取 $20-100$ 个值进行遍历，以避免搜索到局部最优。

我们取初始的搜索范围为

```
grid ={
'C': np.linspace(0.001, 20.001, 51),
'gamma': np.linspace(0.001, 5.001, 51)}
```

经过两次细化，最终搜索到的最优超参数及评分为

```
grid search best param:
{'C': 0.621, 'gamma': 0.085}
grid search best score: 0.8338710689849979
```

三轮网格搜索之后，最优评分不再变化，我们认为这就是最优的超参数取值。更精细的搜索是没有意义的，因为原始数据是有限的离散值，我们将决策边界在数据点之间微调而不跨越数据点，得到的评分并不会变化。

在我们后续的学习曲线分析中，我们能够更加清晰的看到我们的参数是最优的，同时也能看到原始数据为有限的离散值对学习过程的影响。

我们将最终得到的超参数结果列在表4中。

表 4: 网格优化结果

超参数	最优取值	最优评分
C	0.621	0.833871069
gamma	0.085	

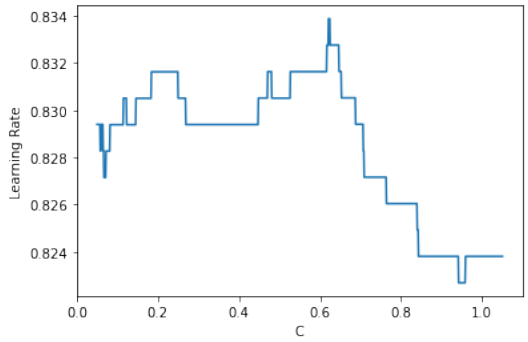

图 8: gamma 学习率曲线 (最优值 0.085)

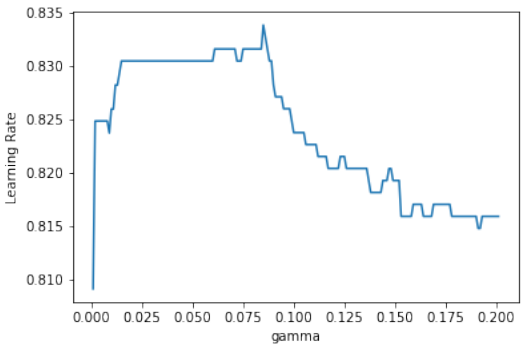

图 9: C 学习曲线 (最优值 0.621)

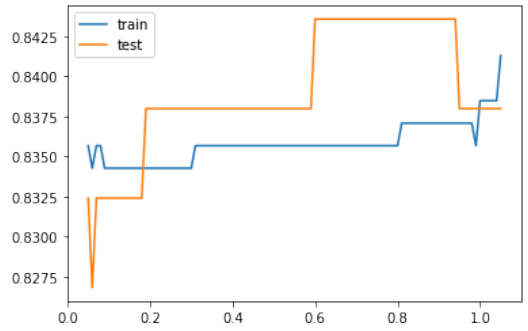

图 10: C 学习曲线 (局部)

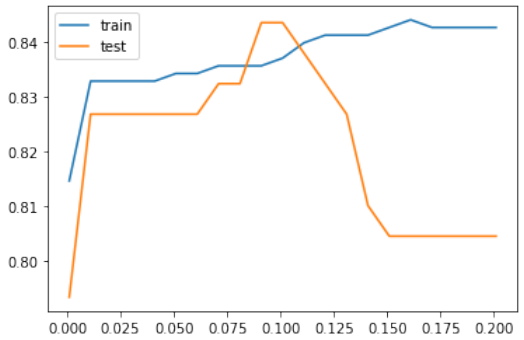

图 11: gamma 学习曲线 (局部)

2.2 学习率曲线

我们在最优超参数取值点附近，取定其中一个超参数，并调节另一个超参数，通过交叉验证得到评分随超参数的变化曲线（图8、图9）。可见，我们给定的超参数确实是该范围内的最优取值，至于更大的调节范围的评分情况，我们暂不给出，因为它们的评分远远低于我们的最优评分，将它们放在一起比较，并不能明显地观察到最优点局部的变化情况。

需注意的是，由于采用交叉验证，我们这里得到的评分综合考虑的不同的训练集和测试集，这里得到的最优已经是避免了过拟合问题的。为了更清晰展示我们没有过拟合，利用`train_test_split`函数随机划分了固定的训练集和测试集，在我们的最优超参数下重新训练模型，得到测试集评分和训练集评分两条学习曲线（图10、图11）。

从图10、图11中我们可以看到，在最优点附近，测试集评分不明显小于训练集评分，没有出现过拟合，而在更远处，有过拟合的趋势。值得注意的是，出现了测试集评分略微大于训练集评分的情况，这是由于数据集划分的随机性导致的，这种情况有一定的概率出现。我们在更大的调节尺度上调节超参数（图12、图13），就不会看到明显的这种情况了。

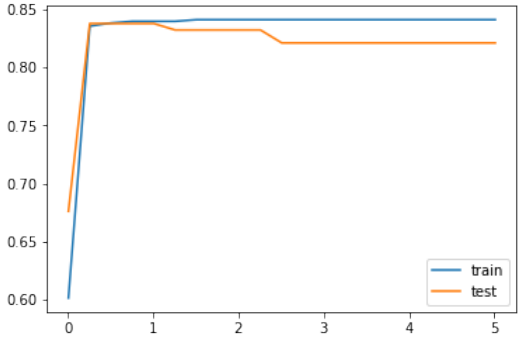

图 12: C 学习曲线

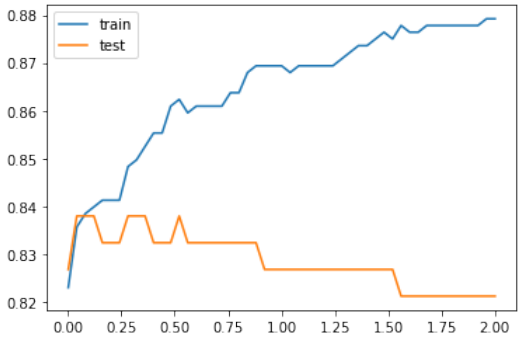

图 13: gamma 学习曲线

2.3 学习曲线

我们调整训练集的样本数，计算测试集和训练集的评分。

根据学习曲线14我们可以发现，在曲线的右端，即我们实际训练采取的样品数下，训练集评分和测试集评分都位于较高水平，我们的结果没有明显的过拟合。

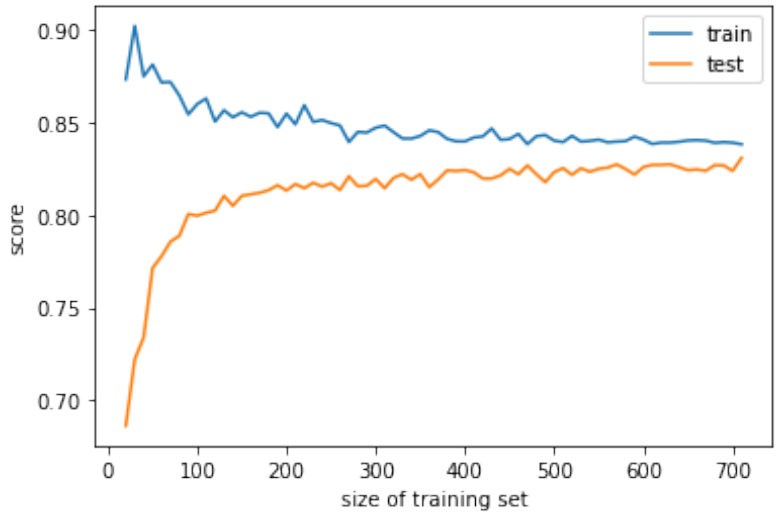

图 14: 学习曲线

3 结果分析

我们采用带有高斯核的支持向量机进行训练，得到的最优超参数和最优评分如表4所示，五折交叉验证得到评分为

$$0.833871068984997 \tag{3}$$

3.1 结果可视化分析

在这一部分，我采用两种可视化方法。

第一种方法是对原始数据进行主成分分析，投影到二维主特征平面，再在十六维特征空间做出该二维截面。在这个过程中，我们需要做做主成分分析的逆变换。即，先在需要做显示的二维主特征平面中画出网格，再将网格乘以主成分分析的两个主成分特征向量构成的子过渡矩阵，从而将原来的二维主特征平面上的网格映射到十六维的特征空间中，接着使用模型对这些格点进行预测，得到决策边界的二维截面，同时我们也将原始数据散点投影在该截面上进行比对（图15）。

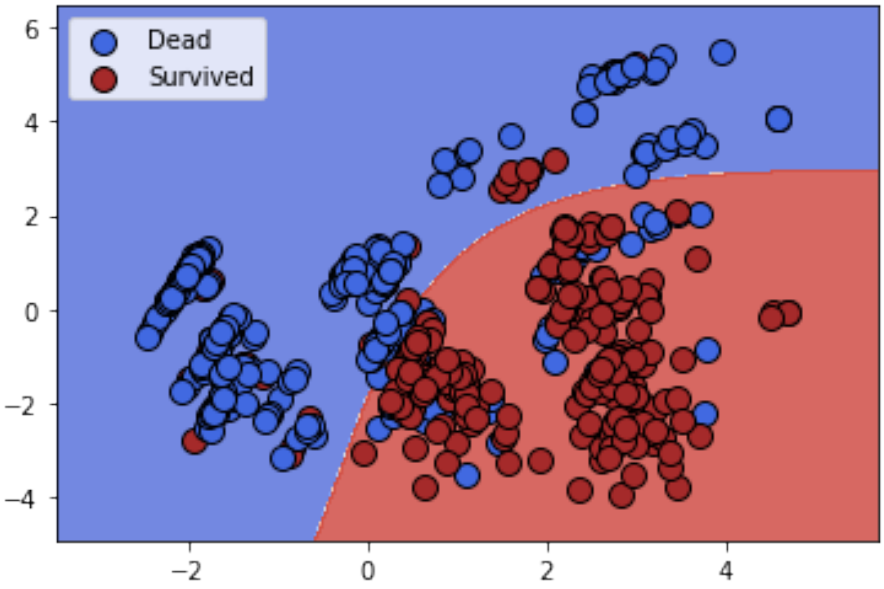

图 15: SVM 训练得到的十六维决策边界的二维主截面

在图15中我们可以看到，我们的决策边界基本上可以将原始数据很好地划分开来。需要指出的是，这些散点只是原始数据在二维主界面上的投影，在其他的截面上，决策边界可能会有所改变，因此，我们这是拿投影和截面做比照，只能定性地观察结果的好坏。

由此我们给出第二种可视化的分析方法。我们以不同的特征为横轴，画出原始数据的生存死亡率直方分布图和预测结果的生存死亡率的直方分布图（图16、图17、图18、图19）。在每个直方种，左边是原始数据，右边是 SVM 预测结果。这种方法可以避免降维带来的信息损失，同时也更具实际应用意义。我们可以看到，我们训练出的 SVM 模型大体上能预测生存率或者死亡率随特征变化的趋势。只有在生存死亡比例的性别分布图（图17）中，我们的模型显得更加敏感，有一定程度的过拟合，这一点没有在学习曲线中显现，可能是因为我们的超参数取值仍是最优值附近的局部最优点，由于网格搜索密度的有限性，这是可能的。

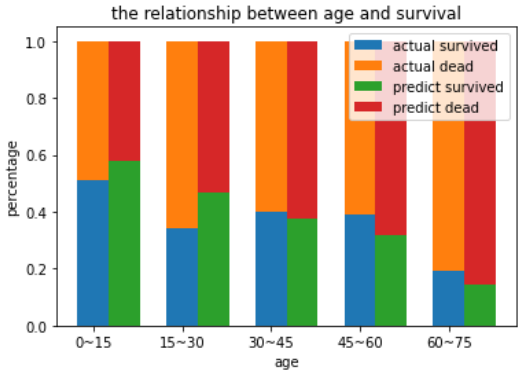

图 16: Age 实际数据与预测数据对比

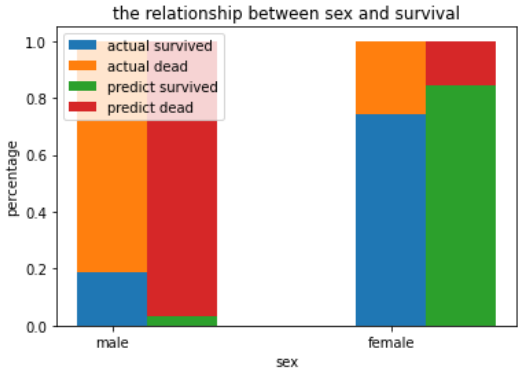

图 17: Sex 实际数据与预测数据对比

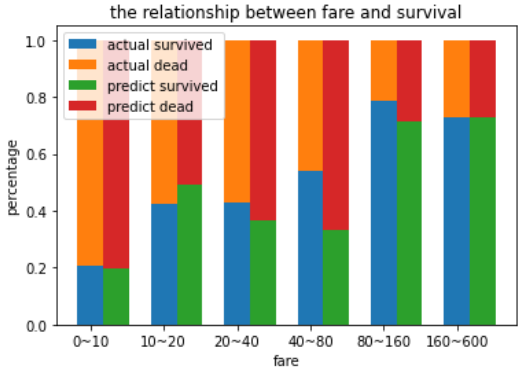

图 18: Fare 实际数据与预测数据对比

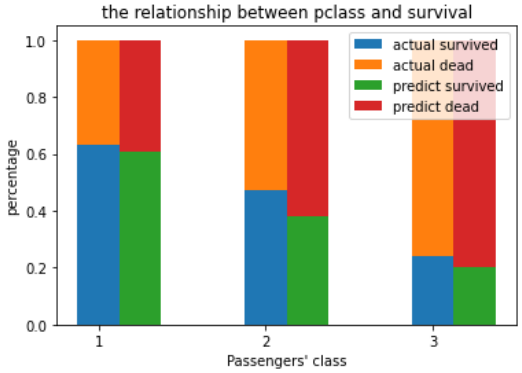

图 19: Pclass 实际数据与预测数据对比

3.2 模型评价

我们利用带有高斯核的支持向量机训练模型，基于泰坦尼克号乘客的信息，预测乘客是否幸存。在五折交叉验证中，我们训练出的模型最优评分为

$$0.833871068984997 \tag{4}$$

我们对比网络上其他对同类问题的模型预测，均没有超过我们的评分。

通过学习曲线的分析，我们发现在我们的搜索维度上，我们的超参数处于最优点，没有出现过拟合和欠拟合的情况。

通过主截面决策边界和项目分布直方图两种可视化方法，我们可以观察得到我们的预测结果基本符合原始数据的分布趋势，在性别特征上稍显敏感。我们的模型具有较强的解释性，具有实际应用的意义。

3.3 结论

根据泰坦尼克号乘客的信息，我们使用带有高斯核的支持向量机进行学习，预测乘客是否幸存，得到了准确度达 83.4% 的预测模型，该模型能够较为准确地预测乘客的幸存情况与乘客的各种信息之间的关系。同时，我们的模型具有较强的解释性，我们可以通过项目分布直方图看出幸存率随各个特征变化的趋势，我们的模型能较为准确地预测这一点。我们的模型有助于分析泰坦尼克号事件的社会意义以及相应时代背景的特征，也有可以联系习近平新时代中国特色社会主义思想均衡发展的理念，指导我们建设平衡发展的社会。

4 附录（训练代码）

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.ensemble import RandomForestRegressor
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
import numpy as np

#数据导入、初步分析、清洗与删除
titanic =pd.read_csv("titanic_data.csv") #可改变
#查看数据每列总数和数据类型
titanic.info()
#可见Embarked, Age和Cabin这3个标签是不全的，Cabin缺失过多，没有填补依据，可忽略
#分析登船港口情况，以便于数据清洗
plt.subplot2grid((2,3),(1,2))
titanic.Embarked.value_counts().plot(kind='bar')
plt.title("number of people in different ports")
plt.ylabel("number of people")
plt.show()
#可知S港口最多，用S来填补缺失值
#数据清洗
#填充缺失的登船港口
titanic["Embarked"] =titanic["Embarked"].fillna('S')

#填补缺失年龄
# way1: titanic["Age"] = titanic["Age"].fillna(titanic["Age"].mean())
# way2: random forest 这里使用随机森林来预测
#把数值的部分都提出来
titanic_data =titanic[['Age', 'Pclass', 'SibSp', 'Parch', 'Fare']]
#把乘客分成两部分：年龄已知和未知
Age_unknown =titanic_data[titanic_data.Age.isnull()].values
Age_known =titanic_data[titanic_data.Age.notnull()].values
#训练集
X = Age_known[:, 1:]
y = Age_known[:, 0]
RDF =RandomForestRegressor(random_state=0, n_estimators=2000, n_jobs=-1)
RDF.fit(X, y)
predictedAges =RDF.predict(Age_unknown[:, 1::])
#填充缺失值
titanic.loc[(titanic.Age.isnull()), 'Age'] =predictedAges
#dummies_Embarked = pd.get_dummies(titanic['Embarked'], prefix= 'Embarked')
#dummies_Sex = pd.get_dummies(titanic['Sex'], prefix= 'Sex')
```

```
#dummies_Pclass = pd.get_dummies(titanic['Pclass'], prefix= 'Pclass')

#tt = pd.concat([titanic, dummies_Embarked, dummies_Sex, dummies_Pclass], axis=1)
#tt.drop(['Pclass', 'Name', 'Sex', 'Ticket', 'Cabin', 'Embarked'], axis=1,
                                                inplace=True)
#print(tt.describe())

#删除经过初步分析后无用的指标
titanic.drop(['PassengerId','Cabin','Ticket'],axis=1,inplace=True)



'''注意在同一个程序中法一和法二顺序不能调换不然无法运行'''
#特征提取————————方法一
#乘客舱的等级的情况
X=titanic['Pclass']
y=titanic["Survived"]
survivedd=np.array([0,0,0])
deadd=np.array([0,0,0])
for i in range(0,891):
       if X[i]==1:
           if y[i]==0:
               deadd[0]+=1
           else:
               survivedd[0]+=1
       elif X[i]==2:
           if y[i]==0:
               deadd[1]+=1
           else:
               survivedd[1]+=1
       else:
           if y[i]==0:
               deadd[2]+=1
           else:
               survivedd[2]+=1
labels =[1, 2, 3]
data_life =[survivedd, deadd]
x =range(len(labels))
width =0.25

#将bottom_y元素都初始化为0
bottom_y =[0] *len(labels)
#计算每组柱子的总和，为计算百分比做准备
```

```
sums =[sum(i) for i in zip(survivedd, deadd)]
for i in data_life:
    y =[a/b for a, b in zip(i, sums)] #计算每个柱子的高度，即百分比
    if i[0]==survivedd[0]:
        plt.bar(x, y, width, bottom=bottom_y,label='actual survived')
    else:
        plt.bar(x, y, width, bottom=bottom_y,label='actual dead')
    bottom_y =[(a+b) for a, b in zip(y, bottom_y)] #计算下一个bottom参数的位置

'''
#把预测值放上去
survived2=np.array([28,14,20])
dead2=np.array([18,23,79])
data_life2 = [survived2, dead2]
X = np.arange(len(labels))
width = 0.25

# 将bottom_y2元素都初始化为0
bottom_y2 = [0] * len(labels)
# 计算每组柱子的总和，为计算百分比做准备
sums2 = [sum(i) for i in zip(survived2, dead2)]
for i in data_life2:    #第一个是survived，bottom为0，所以survived在下面
    Y = [a/b for a, b in zip(i, sums2)] #计算每个柱子的高度，即百分比
    if i[0]==survived2[0]:
        plt.bar(X+width, Y, width, bottom=bottom_y2,label='predict survived')
    else:
        plt.bar(X+width, Y, width, bottom=bottom_y2,label='predict dead')
    bottom_y2 = [(a+b) for a, b in zip(Y, bottom_y2)] #计算bottom参数的位置
'''
plt.legend()
plt.xticks(x, labels)
plt.title("the relationship between pclass and survival")
plt.xlabel("Passengers' class")
plt.ylabel("percentage")
plt.show()



#看看各性别的情况
X=titanic['Sex']
y=titanic["Survived"]
survivedd=np.array([0,0])
```

```
deadd=np.array([0,0])
for i in range(0,891):
       if X[i]=='male':
           if y[i]==0:
               deadd[0]+=1
           else:
               survivedd[0]+=1
       else:
           if y[i]==0:
               deadd[1]+=1
           else:
               survivedd[1]+=1
labels =['male', 'female']
data_life =[survivedd, deadd]
x =range(len(labels))
width =0.25

#将bottom_y元素都初始化为0
bottom_y =[0] *len(labels)
#计算每组柱子的总和，为计算百分比做准备
sums =[sum(i) for i in zip(survivedd, deadd)]
for i in data_life:
    y =[a/b for a, b in zip(i, sums)] #计算每个柱子的高度，即百分比
    if i[0]==survivedd[0]:
        plt.bar(x, y, width, bottom=bottom_y,label='actual survived')
    else:
        plt.bar(x, y, width, bottom=bottom_y,label='actual dead')
    bottom_y =[(a+b) for a, b in zip(y, bottom_y)] #计算下一个bottom参数的位置
'''
#把预测值放上去
survived2=np.array([4,54])
dead2=np.array([110,10])
labels2 = ['male', 'female']
data_life2 = [survived2, dead2]
X = np.arange(len(labels2))#X=[0,1]
width = 0.25

# 将bottom_y2元素都初始化为0
bottom_y2 = [0] * len(labels2)
# 计算每组柱子的总和，为计算百分比做准备
sums2 = [sum(i) for i in zip(survived2, dead2)]
for i in data_life2:    #第一个是survived，bottom为0，所以survived在下面
    Y = [a/b for a, b in zip(i, sums2)] #计算每个柱子的高度，即百分比
```

```
    if i[0]==survived2[0]:
        plt.bar(X+width, Y, width, bottom=bottom_y2,label='predict survived')
    else:
        plt.bar(X+width, Y, width, bottom=bottom_y2,label='predict dead')
    bottom_y2 = [(a+b) for a, b in zip(Y, bottom_y2)] #计算bottom参数的位置
'''
plt.legend()
plt.xticks(x, labels)
plt.title("the relationship between sex and survival")
plt.xlabel("sex")
plt.ylabel("percentage")
plt.show()



#看看各年龄的情况
X=titanic['Age']
y=titanic["Survived"]
n=6#分为n-1组
survivedd=np.linspace(0,0,n-1)#n-1个柱子
deadd=np.linspace(0,0,n-1)
parts=np.linspace(0,75,n)#n个边界值，围成n-1个柱子
for i in range(0,891):
    for t in range(0,n-1):
        if parts[t]<X[i]<parts[t+1]:
            if y[i]==0:
                deadd[t]+=1
            else:
                survivedd[t]+=1
labels =['0~15', '15~30', '30~45', '45~60','60~75']#分别对应n-1个柱子，这个标签自己修改
data_life =[survivedd, deadd]
x =range(len(labels))
width =0.35

bottom_y =[0] *len(labels)
sums =[sum(i) for i in zip(survivedd, deadd)]
for i in data_life:
    y = [a/b for a, b in zip(i, sums)]
    if i[0]==survivedd[0] and i[1]==survivedd[1]:
        plt.bar(x, y, width, bottom=bottom_y,label='actual survived')
    else:
        plt.bar(x, y, width, bottom=bottom_y,label='actual dead')
```

```
    bottom_y =[(a+b) for a, b in zip(y, bottom_y)]
'''
#预测值
survived2=np.array([52,179,99,27,3])
dead2=np.array([38,205,165,58,18])
data_life2 = [survived2, dead2]
X = np.arange(len(labels))

bottom_y2 = [0] * len(labels)
sums2 = [sum(i) for i in zip(survived2, dead2)]
for i in data_life2:
    Y = [a/b for a, b in zip(i, sums2)]
    if i[0]==survived2[0] and i[1]==survived2[1]:
        plt.bar(X+width, Y, width, bottom=bottom_y2,label='predict survived')
    else:
        plt.bar(X+width, Y, width, bottom=bottom_y2,label='predict dead')
    bottom_y2 = [(a+b) for a, b in zip(Y, bottom_y2)]
'''
plt.legend()
plt.xticks(x, labels)
plt.title("the relationship between age and survival")
plt.xlabel("age")
plt.ylabel("percentage")
plt.show()


#看看各票价的情况
X=titanic['Fare']
y=titanic["Survived"]
n=7#分为n-1组
survivedd=np.linspace(0,0,n-1)#n-1个柱子
deadd=np.linspace(0,0,n-1)
parts=np.array([0,10,20,40,80,160,600])
for i in range(0,891):
    for t in range(0,n-1):
        if parts[t]<X[i]<parts[t+1]:
            if y[i]==0:
                deadd[t]+=1
            else:
                survivedd[t]+=1
labels =['0~10', '10~20', '20~40','40~80','80~160','160~600']
#分别对应n-1个柱子，这个标签自己修改
data_life =[survivedd, deadd]
```

```
x =range(len(labels))
width =0.35


bottom_y =[0] *len(labels)
sums =[sum(i) for i in zip(survivedd, deadd)]
for i in data_life:
    y =[a/b for a, b in zip(i, sums)]
    if i[0]==survivedd[0] and i[1]==survivedd[1]:
        plt.bar(x, y, width, bottom=bottom_y,label='actual survived')
    else:
        plt.bar(x, y, width, bottom=bottom_y,label='actual dead')
    bottom_y =[(a+b) for a, b in zip(y, bottom_y)]

'''
#预测值
survived2=np.array([64,88,73,39,37,16])
dead2=np.array([257,91,127,78,15,6])
data_life2 = [survived2, dead2]
X = np.arange(len(labels))

bottom_y2 = [0] * len(labels)
sums2 = [sum(i) for i in zip(survived2, dead2)]
for i in data_life2:
    Y = [a/b for a, b in zip(i, sums2)]
    if i[0]==survived2[0] and i[1]==survived2[1]:
        plt.bar(X+width, Y, width, bottom=bottom_y2,label='predict survived')
    else:
        plt.bar(X+width, Y, width, bottom=bottom_y2,label='predict dead')
    bottom_y2 = [(a+b) for a, b in zip(Y, bottom_y2)]
'''
plt.legend()
plt.xticks(x, labels)
plt.title("the relationship between fare and survival")
plt.xlabel("fare")
plt.ylabel("percentage")
plt.show()



#看看各登录港口的情况
fig =plt.figure()
fig.set(alpha=0.2) #设定图表颜色alpha参数
```

```
Survived_0 =titanic.Embarked[titanic.Survived ==0].value_counts()
Survived_1 =titanic.Embarked[titanic.Survived ==1].value_counts()
df=pd.DataFrame({'survived':Survived_1, 'dead':Survived_0})
df.plot(kind='bar', stacked=True)
plt.title("the relationship between embarked and survival")
plt.xlabel("the port where people embarked ")
plt.ylabel("number of people")
plt.show()
#name,sibsp,parch,ticket画图显然无法看出与生存情况是否存在关系，故略去
'''总结：由图可见，除了embarked与存活情况没有显著关系以外，
fare,age,sex,pclass与存活情况都有关系，
age关系主要体现在数值大小两端'''



#现在进行特征提取————方法二
#数据特征提取（特征工程）
#对直接类别的数据进行处理：Sex, Embarked, Pclass
sex_mapDict={'male':1,'female':0}
titanic['Sex']=titanic['Sex'].map(sex_mapDict)

Embarkeddf=pd.DataFrame()
Embarkeddf=pd.get_dummies(titanic['Embarked'],prefix='Embarked')

#使用get_dummies进行one-hot编码，产生虚拟变量，列名前缀Embarked

Pclassdf=pd.DataFrame()
Pclassdf=pd.get_dummies(titanic['Pclass'],prefix='Pclass')

#需要把one-hot编码产生的虚拟变量添加到titanic中，并删除Embarked和Pclass两列
titanic=pd.concat([titanic,Embarkeddf],axis=1)
titanic.drop('Embarked',axis=1,inplace=True)
titanic=pd.concat([titanic,Pclassdf],axis=1)
titanic.drop('Pclass',axis=1,inplace=True)



#对字符串类型进行处理：Name, Parch和SibSP
def gettitle(name):
    str1=name.split(',')[1]
    str2=str1.split('.')[0]
    str3=str2.strip()
    return str3
```

```
titledf=pd.DataFrame()
titledf['title']=titanic['Name'].map(gettitle)

#姓名中头衔字符串与定义头衔类别的映射关系
title_mapDict={
                "Capt":       "Officer",
                "Col":        "Officer",
                "Major":      "Officer",
                "Jonkheer":   "Royalty",
                "Don":        "Royalty",
                "Dr":         "Officer",
                "Rev":        "Officer",
                "the Countess": "Royalty",
                "Mme":        "Mrs",
                "Mlle":       "Miss",
                "Mr":         "Mr",
                "Mrs":        "Mrs",
                "Miss":       "Miss",
                "Master":     "Master",
                "Lady":       "Royalty"
                }
titledf['title']=titledf['title'].map(title_mapDict)
titledf=pd.get_dummies(titledf['title'])
titanic=pd.concat([titanic,titledf],axis=1)
titanic.drop('Name',axis=1,inplace=True)

'''家庭人数FamilySize = Parch（同代直系亲属数）+SibSP（不同代直系亲属数）+乘客本人;
无家庭Family_Single: FamilySize =1;
小家庭Family_Small: 2< =FamilySize<=4;
大家庭Family_Large: FamilySize>=5'''
'''对家庭大小的定义其实具有较大的主观性'''
familydf=pd.DataFrame()
familydf['FamilySize']=titanic['Parch']+titanic['SibSp']+1
familydf['Family_Single']=familydf['FamilySize'].map(lambda s :1 if s ==1 else 0)
familydf['Family_Small'] =familydf['FamilySize'].map(lambda s :1 if 2 <=s <=4 else 0)
familydf['Family_Large'] =familydf['FamilySize'].map(lambda s :1 if 5 <=s else 0)
titanic=pd.concat([titanic,familydf],axis=1)

'''利用corr计算各特征之间的相关系数，查看各特征与Survived的相关系数，并以降序的顺序排列'''
corrdf=titanic.corr()
print('各特征与Survived的相关系数:\n',corrdf['Survived'].sort_values(ascending=False))

#把法二的绝对值在0.2以下的特征除去，总结法一和法二得到如下特征选择
```

```
titanic_X=pd.concat([titledf,#头衔
                     Pclassdf,#客舱等级
                     familydf,#家庭大小
                     titanic['Fare'],#船票价格
                     titanic['Sex'],#性别
                     titanic['Age'],#年龄
                     ],axis=1)



##模型训练、预测及评分
import numpy as np
import matplotlib.pyplot as plt
from sklearn.svm import SVC
from sklearn.model_selection import cross_validate
from sklearn.model_selection import GridSearchCV
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler

#模型训练
X = StandardScaler().fit_transform(titanic_X)
y = titanic["Survived"]

# 支持向量机，五折交叉验证
SVM =SVC(kernel='rbf', C=1.0, gamma=0.7)
cv_result =cross_validate(SVM, X, y, cv=5)
cv_value_vec =cv_result["test_score"]
cv_mean =np.mean(cv_value_vec)
print("score:", cv_mean)

# 先粗略搜搜

## 先看看选哪个核，为避免重复的计算，我们手动来改变核

### 高斯核
grid ={
    'C': np.linspace(0.001, 20.001, 51),
    'gamma': np.linspace(0.001, 5.001, 51)}
SVM =GridSearchCV(SVC(kernel='rbf'), grid, cv=5)
SVM.fit(X, y)
best_paras =SVM.best_params_
print('RBF grid search best param: {0}'.format(best_paras))
print('RBF grid search best score: {0}'.format(SVM.best_score_))
```

```
### 线性核
grid ={'C': np.linspace(0.001, 20.001, 201)}
SVM =GridSearchCV(SVC(kernel='linear'), grid, cv=5)
SVM.fit(X, y)
best_paras =SVM.best_params_
print('LINEAR grid search best param: {0}'.format(best_paras))
print('LINEAR grid search best score: {0}'.format(SVM.best_score_))

### 多项式核
grid ={'degree': np.arange(1, 20, 1)}
SVM =GridSearchCV(SVC(kernel='poly'), grid, cv=5)
SVM.fit(X, y)
best_paras =SVM.best_params_
print('POLY grid search best param: {0}'.format(best_paras))
print('POLY grid search best score: {0}'.format(SVM.best_score_))

# 再精细搜搜

## 高斯核
grid ={
    'C': np.linspace(0.001, 0.801, 21),
    'gamma': np.linspace(0.001, 0.201, 21)}
SVM =GridSearchCV(SVC(kernel='rbf'), grid, cv=5)
SVM.fit(X, y)
best_paras =SVM.best_params_
print('RBF grid search best param:\n {0}'.format(best_paras))
print('RBF grid search best score: {0}\n'.format(SVM.best_score_))

## 线性核
grid ={
    'C': np.linspace(0.1, 0.5, 501)}
SVM =GridSearchCV(SVC(kernel='linear'), grid, cv=5)
SVM.fit(X, y)
best_paras =SVM.best_params_
print('LINEAR grid search best param:\n {0}'.format(best_paras))
print('LINEAR grid search best score: {0}\n'.format(SVM.best_score_))

# 再精细搜搜
grid ={
    'C': np.linspace(0.6, 0.7, 101),
    'gamma': np.linspace(0.05, 0.10, 51)}
SVM =GridSearchCV(SVC(kernel='rbf'), grid, cv=5)
```

```
SVM.fit(X, y)
best_paras =SVM.best_params_
print('grid search best param:\n {0}'.format(best_paras))
print('grid search best score: {0}\n'.format(SVM.best_score_))

##预测以及评分

SVM =SVC(kernel='rbf', C=0.621, gamma=0.085)
cv_result =cross_validate(SVM, X, y, cv=5)
cv_value_vec =cv_result["test_score"]
cv_mean =np.mean(cv_value_vec)
print("score:", cv_mean)

#学习曲线
# C 学习曲线
C_learn =[]
C_range =np.linspace(0.050, 1.050, 1001)
for C in C_range:
    SVM =SVC(kernel='rbf', C=C, gamma=0.085)
    cv_result =cross_validate(SVM, X, y, cv=5)
    cv_value_vec =cv_result["test_score"]
    cv_mean =np.mean(cv_value_vec)
    C_learn.append(cv_mean)
plt.plot(C_range, C_learn)
plt.show()

#交叉验证方法其实在训练的过程中就已经排除了过拟合和欠拟合的问题，
                                        但是我们为了更加清晰地显示这一点，
                                        #我们还是将训练集和测试集分开，
                                        在学习曲线上直观地观察这一点。

# 不用交叉验证训练模型
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test =train_test_split(X, y, test_size=0.2)

# C 学习曲线
C_learn_train =[]
C_learn_test =[]
C_range =np.linspace(0.050, 1.050, 101)
for C in C_range:
    SVM_split =SVC(kernel='rbf', C=C, gamma=0.085)
    SVM_split.fit(X_train, y_train)
    C_learn_train.append(SVM_split.score(X_train, y_train))
```

```
    C_learn_test.append(SVM_split.score(X_test, y_test))

plt.figure()
plt.plot(C_range, C_learn_train, label='train')
plt.plot(C_range, C_learn_test, label='test')
plt.legend()
plt.show()

# gamma 学习曲线
gamma_learn_train =[]
gamma_learn_test =[]
gamma_range =np.linspace(0.001, 0.201, 21)
for gamma in gamma_range:
    SVM_split =SVC(kernel='rbf', C=0.621, gamma=gamma)
    SVM_split.fit(X_train, y_train)
    gamma_learn_train.append(SVM_split.score(X_train, y_train))
    gamma_learn_test.append(SVM_split.score(X_test, y_test))

plt.figure()
plt.plot(gamma_range, gamma_learn_train, label='train')
plt.plot(gamma_range, gamma_learn_test, label='test')
plt.legend()
plt.show()

# 不用交叉验证训练模型
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test =train_test_split(X, y, test_size=0.2)
X = StandardScaler().fit_transform(titanic_X)
y = titanic["Survived"]

# C 学习曲线
C_learn_train =[]
C_learn_test =[]
C_range =np.linspace(0.010, 5.010, 21)
for C in C_range:
    SVM_split =SVC(kernel='rbf', C=C, gamma=0.085)
    SVM_split.fit(X_train, y_train)
    C_learn_train.append(SVM_split.score(X_train, y_train))
    C_learn_test.append(SVM_split.score(X_test, y_test))

plt.figure()
plt.plot(C_range, C_learn_train, label='train')
plt.plot(C_range, C_learn_test, label='test')
```

```
plt.legend()
plt.show()

# gamma 学习曲线
gamma_learn_train =[]
gamma_learn_test =[]
gamma_range =np.linspace(0.001, 2.001, 51)
for gamma in gamma_range:
    SVM_split =SVC(kernel='rbf', C=0.621, gamma=gamma)
    SVM_split.fit(X_train, y_train)
    gamma_learn_train.append(SVM_split.score(X_train, y_train))
    gamma_learn_test.append(SVM_split.score(X_test, y_test))

plt.figure()
plt.plot(gamma_range, gamma_learn_train, label='train')
plt.plot(gamma_range, gamma_learn_test, label='test')
plt.legend()
plt.show()

SVM_split =SVC(kernel='rbf', C=0.651, gamma=0.085)
SVM_split.fit(X_train, y_train)
SVM_split.score(X_train, y_train)
SVM_split.score(X_test, y_test)


# 学习曲线！！！！！！！
from sklearn.model_selection import train_test_split

X = StandardScaler().fit_transform(titanic_X)
y = titanic["Survived"]

def trainSVM_learn(test_size):
    X_train, X_test, y_train, y_test =train_test_split(X, y, test_size=test_size)
    SVM_split =SVC(kernel='rbf', C=0.621, gamma=0.085)
    SVM_split.fit(X_train, y_train)
    train_score =SVM_split.score(X_train, y_train)
    test_score =SVM_split.score(X_test, y_test)
    return train_score, test_score

train_num_range =np.arange(20, 713, 10)
train_num_learn_train =[]
train_num_learn_test =[]
avg_num =30
```

```
for train_num in train_num_range:
    train_score_set =np.ones(avg_num)
    test_score_set =np.ones(avg_num)
    for i in range(avg_num):
        train_score, test_score =trainSVM_learn(1-train_num/891)
        train_score_set[i] =train_score
        test_score_set[i] =test_score
    train_num_learn_train.append(train_score_set.mean())
    train_num_learn_test.append(test_score_set.mean())

plt.figure()
plt.plot(train_num_range, train_num_learn_train, label='train')
plt.plot(train_num_range, train_num_learn_test, label='test')
plt.legend()
plt.xlabel('size of training set')
plt.ylabel('score')
plt.show()

##可视化分析

Dead_X    = titanic[titanic["Survived"]==0]
Survived_X =titanic[titanic["Survived"]==1]

plt.figure()
plt.scatter(Dead_X['Pclass_3'] , Dead_X['Age'], color='r', alpha=.2, lw=0.2,
                                            label='Dead')
plt.scatter(Survived_X['Pclass_3'], Survived_X['Age'], color='b', alpha=.2, lw=0.2,
                                            label='Survived')
plt.legend()
plt.show()

X_train, X_test, y_train, y_test =train_test_split(X, y, test_size=0.2)
SVM_split =SVC(kernel='rbf', C=0.651, gamma=0.085)
SVM_split.fit(X_train, y_train)
# SVM_split.predict(titanic_X[titanic["Sex"]==1])
# X_test
np.array(titanic_X[titanic["Sex"]==1])


# 主成分分析，提取前两个奇异值，将特征降至两维
# X   = StandardScaler().fit_transform(X)
pca   = PCA(n_components=2)
X_r   = pca.fit_transform(X)
```

```
print("explained variance ratio = ", pca.explained_variance_ratio_)
print(pca.components_)

# 原始数据散点
plt.figure()
plt.scatter(X_r[y==0,0], X_r[y==0,1], color='r', alpha=.8, lw=2, label='Dead')
plt.scatter(X_r[y==1,0], X_r[y==1,1], color='b', alpha=.8, lw=2, label='Survived')
plt.legend()
plt.show()

# 数据网格化
def make_meshgrid(x, y, h=.02):
    x_min, x_max =x.min() -1, x.max() +1
    y_min, y_max =y.min() -1, y.max() +1
    xx, yy =np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))
    return xx, yy

# 画等高线图
def plot_contour(clf, xx, yy, X_train, y_train, **kwargs):
    clf.fit(X_train, y_train)
    Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)
    out =plt.contourf(xx, yy, Z, **kwargs)
    return out


# pca 逆变换

## 取出两个主特征向量
X_pc1 =X_r[:,0]
X_pc2 =X_r[:,1] # 它们都已经标准化了，这非常好，意味着我们后续不需要进行零点对齐

## 在主特征向量张成的二维空间中画格点
xx_pc1, yy_pc2 =make_meshgrid(X_pc1, X_pc2, h=.02)

## 映射成十六维空间中的二维平面
XDelta_pc1 =xx_pc1.ravel().reshape(-1,1) *pca.components_[0]
XDelta_pc2 =yy_pc2.ravel().reshape(-1,1) *pca.components_[1]
X_grid =XDelta_pc1 +XDelta_pc2

## 预测每个格点的结果
SVM =SVC(kernel='rbf', C=0.651, gamma=0.085)
SVM.fit(X, y) # 这里用了全部原始数据来训练，因为不需要划分测试集
```

```
Z = SVM.predict(X_grid)
Z = Z.reshape(xx_pc1.shape)

plt.figure()

# 画预测图
plt.contourf(xx_pc1, yy_pc2, Z, cmap=plt.cm.coolwarm, alpha=0.8)

# 原始数据散点
plt.scatter(X_pc1[y==1], X_pc2[y==1], color='brown', cmap=plt.cm.coolwarm, s=100,
                                        edgecolors='black', label='Survived')
plt.scatter(X_pc1[y==0], X_pc2[y==0], color='royalblue', cmap=plt.cm.coolwarm, s=100,
                                        edgecolors='black', label='Dead')
plt.legend()
plt.show()

musk =SVM.predict(X_pc1.reshape(-1,1) *pca.components_[0] +X_pc2.reshape(-1,1) *
                                        pca.components_[1])
gate =musk*y +(1-musk)*(1-y)
blue1 = np.logical_and(gate==0,y==0)
brown1 =np.logical_and(gate==0,y==1)
blue2 = np.logical_and(gate==1,y==0)
brown2 =np.logical_and(gate==1,y==1)


plt.figure()

# 画预测图
plt.contourf(xx_pc1, yy_pc2, Z, cmap=plt.cm.coolwarm, alpha=0.8)

# 原始数据散点
plt.scatter(X_pc1[blue1], X_pc2[blue1], color='royalblue', cmap=plt.cm.coolwarm,
                                        s=100, edgecolors='black', label='Dead')
plt.scatter(X_pc1[brown1], X_pc2[brown1], color='brown', cmap=plt.cm.coolwarm, s=100,
                                        edgecolors='black', label='Survived')
plt.scatter(X_pc1[blue2], X_pc2[blue2], color='royalblue', cmap=plt.cm.coolwarm,
                                        s=100, edgecolors='black', label='Dead')
plt.scatter(X_pc1[brown2], X_pc2[brown2], color='brown', cmap=plt.cm.coolwarm, s=100,
                                        edgecolors='black', label='Survived')

plt.legend()
plt.show()
```

```
##下面是尝试使用t-SNE方法而非PCA方法来降维，只是一个不成熟的尝试，可以暂且不管。
from sklearn.manifold import TSNE

tsne =TSNE(n_components=2, init='pca', )
X_tsne =tsne.fit_transform(X)
X_tsne_data =np.vstack((X_tsne.T, y)).T
df_tsne =pd.DataFrame(X_tsne_data, columns=['Dim1', 'Dim2', 'Survived'])
df_tsne.head()

tsne =TSNE(n_components=2)
X_r =tsne.fit_transform(X)
plt.figure()
# 原始数据散点
plt.scatter(X_r[y==0,0], X_r[y==0,1], color='r', alpha=.8, lw=2, label='Dead')
plt.scatter(X_r[y==1,0], X_r[y==1,1], color='b', alpha=.8, lw=2, label='Survived')
plt.legend()
plt.show()
```