一、特征工程和数据清洗

当我们得到一个具有特征的数据集时，是不是所有的特性都很重要？可能有许多冗余的特征应该被消除，我们还可以通过观察或从其他特征中提取信息来获得或添加新特性。

1. 年龄特征

离散化

年龄是连续的特征，在机器学习模型中存在连续变量的问题。

如果按年龄分组，有30个人，可能有30个年龄值。在机器学习中很难处理。我们需要对连续值进行离散化来分组。

何为离散化？离散化，就是把无限空间中有限的个体映射到有限的空间中去，以此提高算法的时空效率。

比如有n个数字：32434234，32434234，43324556，8384733，98998988...

去重后给予其对应的索引：0,0,1,2,3分别对应每个数，就可以简化很多操作，减少了很多不必要的资源开销。

年龄的离散化

那么对于年龄数据，乘客的最大年龄是80岁。所以我们将范围从0-80分成5段。所以80/5 = 16。

```
#年龄离散化处理
data['Age_band']=0
data.loc[data['Age']<=16,'Age_band']=0
data.loc[(data['Age']>16)&(data['Age']<=32),'Age_band']=1
data.loc[(data['Age']>32)&(data['Age']<=48),'Age_band']=2
data.loc[(data['Age']>48)&(data['Age']<=64),'Age_band']=3
data.loc[data['Age']>64,'Age_band']=4
data.head(2)
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked	Initial	Age_band
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S	Mr	1
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C	Mrs	2

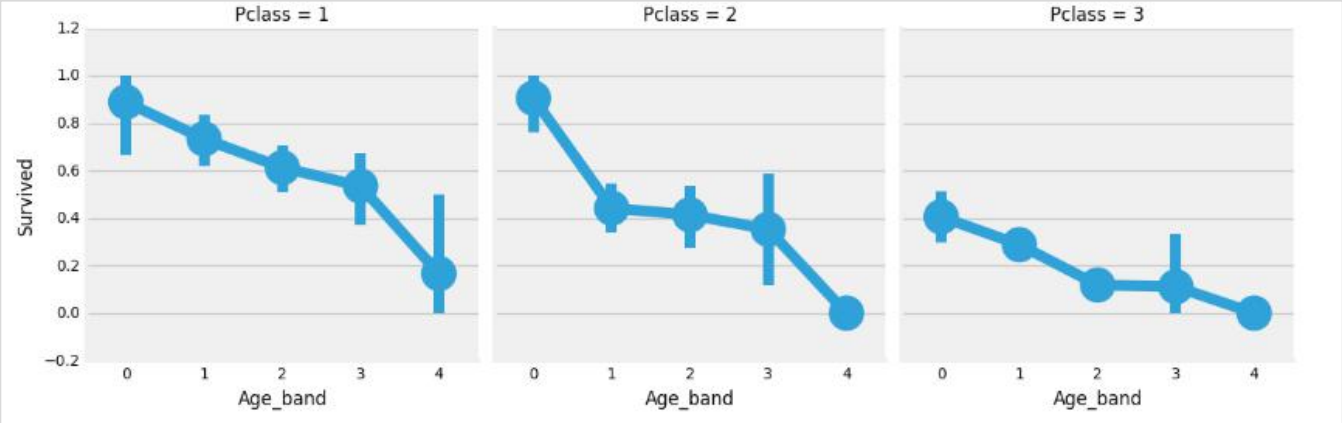
检查分段年龄的人数分布：

```
data['Age_band'].value_counts().to_frame().style.background_gradient(cmap='summer')
```

	Age_band
1	382
2	325
0	104
3	69
4	11

分析年龄段的存活情况：

```
sns.factorplot('Age_band','Survived',data=data,col='Pclass')
plt.show()
```

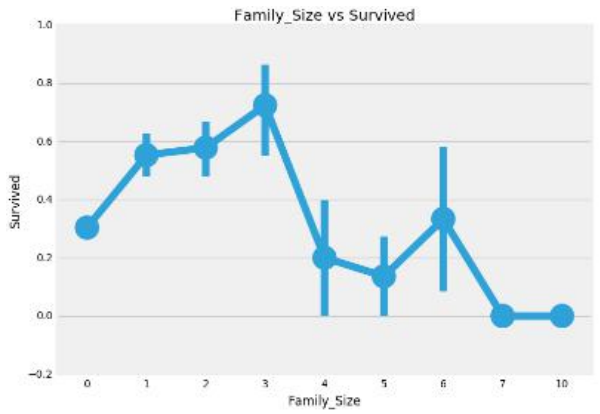


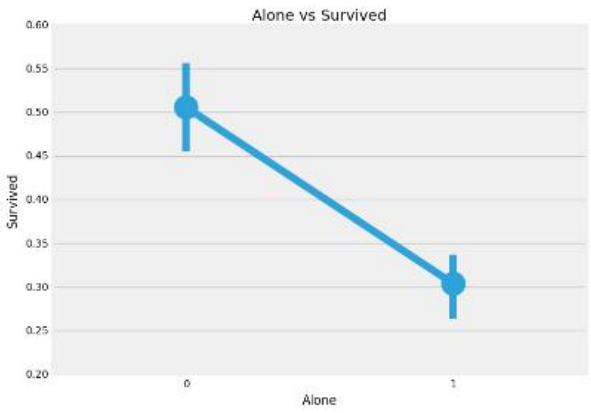
结论：不管在那个船舱，生存率随年龄的增加而减少。

2. Family_size：家庭总人数

光看兄弟姐妹和老人孩子看不太直接，咱们直接看全家的人数。

```
data['Family_Size']=0
data['Family_Size']=data['Parch']+data['SibSp']#family size
data['Alone']=0
data.loc[data.Family_Size==0,'Alone']=1#Alone

f,ax=plt.subplots(1,2,figsize=(18,6))
sns.factorplot('Family_Size','Survived',data=data,ax=ax[0])
ax[0].set_title('Family_Size vs Survived')
sns.factorplot('Alone','Survived',data=data,ax=ax[1])
ax[1].set_title('Alone vs Survived')
plt.close(2)
plt.close(3)
plt.show()
```

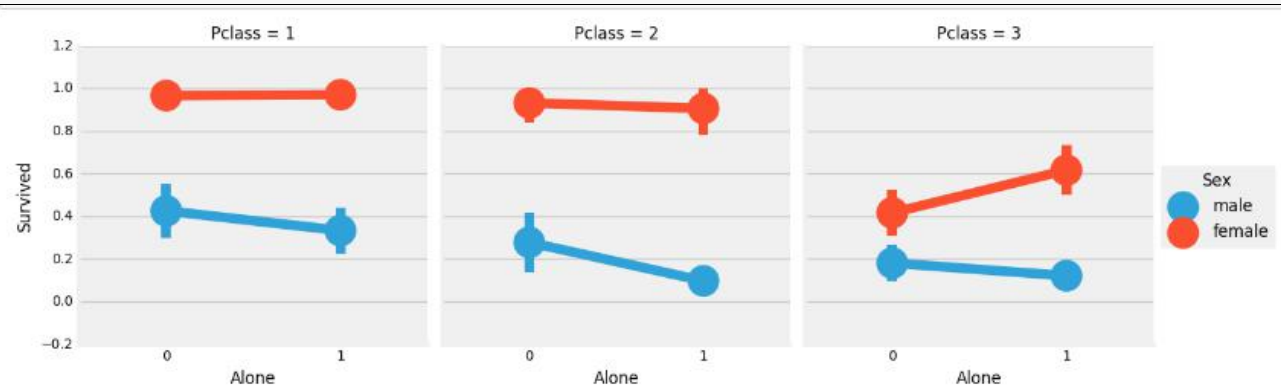




显然，如果你是单独（family_size = 0），那么生存的机会很低。家庭规模4以上，机会也减少。这看起来也是模型的一个重要特性。

让我们进一步研究这个问题，下面通过因子图，按船舱等级和性别来分析单独和非单独的存活率。

```
sns.factorplot('Alone','Survived',data=data,hue='Sex',col='Pclass')
plt.show()
```



女性在船舱等级1和2，不管是单身还是非单身，生存率持平；女性在pclass=3时候单身生存率更高；男性不管在哪个船舱等级，单身的生存率均下降。

3. 船票价格

票价也是连续的特性，所以我们需要做离散化处理，将它转换为数值。

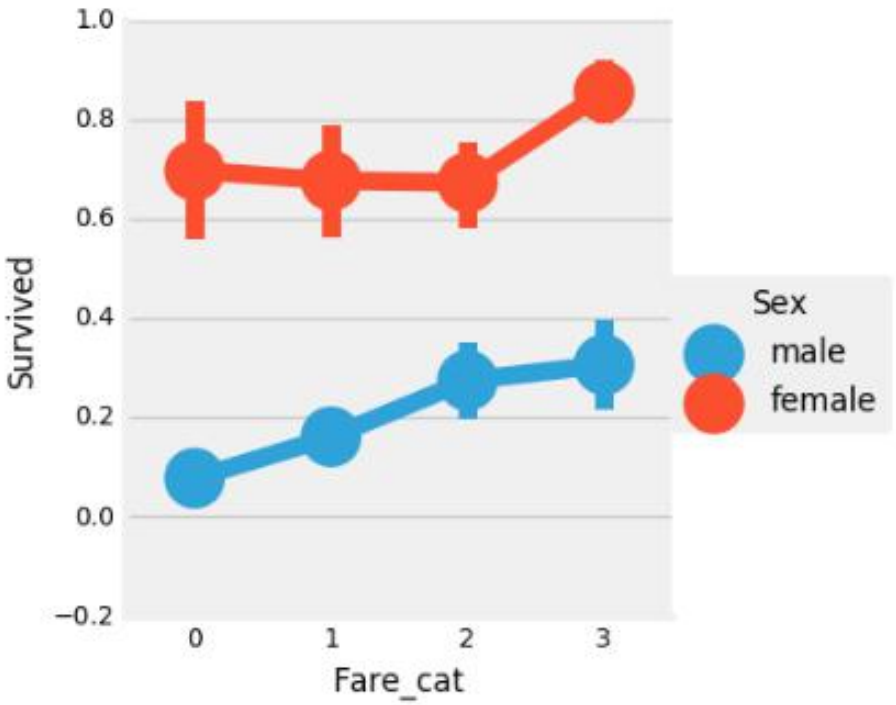
```
data['Fare_Range']=pd.qcut(data['Fare'],4)#基于样本数将变量离散成相等大小的桶
data.groupby(['Fare_Range'])['Survived'].mean().to_frame().style.background_gradient(cmap='summer_r') #分组统计生存率均值
```

	Survived
Fare_Range	
(-0.001, 7.91]	0.197309
(7.91, 14.454]	0.303571
(14.454, 31.0]	0.454955
(31.0, 512.329]	0.581081

清楚地看到，船票价格增加生存的机会增加。

离散化处理：

```
data['Fare_cat']=0
data.loc[data['Fare']<=7.91,'Fare_cat']=0
data.loc[(data['Fare']>7.91)&(data['Fare']<=14.454),'Fare_cat']=1
data.loc[(data['Fare']>14.454)&(data['Fare']<=31),'Fare_cat']=2
data.loc[(data['Fare']>31)&(data['Fare']<=513),'Fare_cat']=3

sns.factorplot('Fare_cat','Survived',data=data,hue='Sex')
plt.show()
```



显然，随着fare_cat增加，存活的几率增加。这一特性可能成为建模过程中的一个重要特征。

4. 数据清洗

特征数字化

将字符串值转换为数字，数字类型在机器学习模型中会更方便的处理。

```
data['Sex'].replace(['male','female'],[0,1],inplace=True)
data['Embarked'].replace(['S','C','Q'],[0,1,2],inplace=True)
data['Initial'].replace(['Mr','Mrs','Miss','Master','Other'],[0,1,2,3,4],inplace=True)
```

去掉不必要的特征

名称>我们不需要name特性，因为它不能转换成任何分类值

年龄——>我们有age_band特征，所以不需要这个

票号-->这是任意的字符串，不能被归类

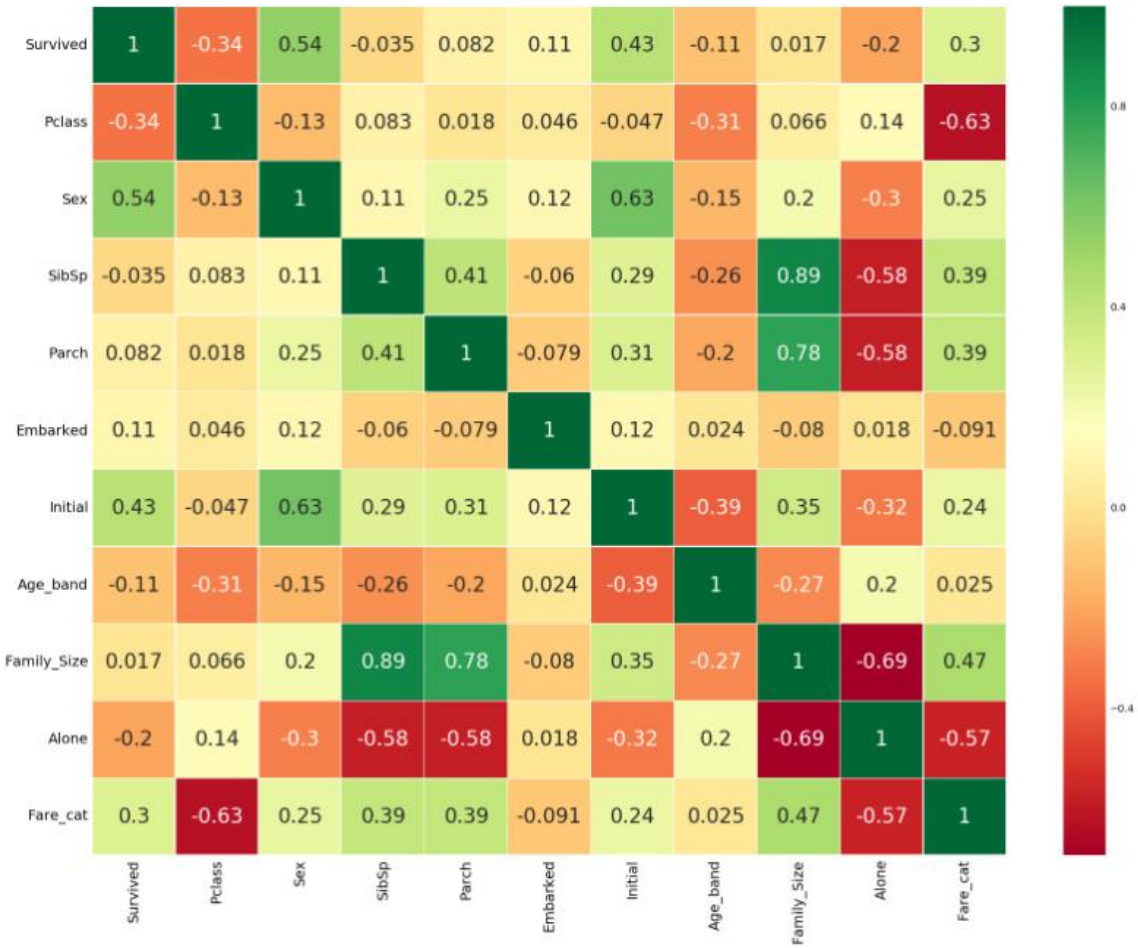
票价——>我们有fare_cat特征，所以不需要

船仓号——>这个也不要没啥含义

passengerid -->不能被归类

去掉后重新观察热力图

```
data.drop(['Name','Age','Ticket','Fare','Cabin','Fare_Range','PassengerId'],axis=1,inplace=True)
sns.heatmap(data.corr(),annot=True,cmap='RdYlGn',linewidths=0.2,annot_kws={'size':20})
fig=plt.gcf()
fig.set_size_inches(18,15)
plt.xticks(fontsize=14)
plt.yticks(fontsize=14)
plt.show()
```



现在以上的相关图，我们可以看到一些正相关的特征。他们中的一些人sibsp和family_size，和一些负面的Alone和family_size。

5.机器学习建模

我们在前面对数据进行了探索性的数据分析(EDA - Exploratory Data Analysis,EDA)，得到了一些结论。

现在我们建模来预测乘客是否能够生存下来，使用到的模型包括：

1）logistic回归

2）支持向量机（线性和径向）

3）随机森林
4）k-近邻
5）朴素贝叶斯
6）决策树
7）神经网络

```
#导入全部用到的机器学习包
from sklearn.linear_model import LogisticRegression #logistic regression
from sklearn import svm #support vector Machine
from sklearn.ensemble import RandomForestClassifier #Random Forest
from sklearn.neighbors import KNeighborsClassifier #KNN
from sklearn.naive_bayes import GaussianNB #Naive bayes
from sklearn.tree import DecisionTreeClassifier #Decision Tree
from sklearn.model_selection import train_test_split #训练和测试数据分拆包
from sklearn import metrics #accuracy measure
from sklearn.metrics import confusion_matrix #for confusion matrix
```

得到训练集和测试集，测试分割中包含的数据集的比例为30%

```
train,test=train_test_split(data,test_size=0.3,random_state=0,stratify=data['Survived'])
train_X=train[train.columns[1:]] #特征
train_Y=train[train.columns[:1]] #目标值
test_X=test[test.columns[1:]]
test_Y=test[test.columns[:1]]
X=data[data.columns[1:]]
Y=data['Survived']
```

#使用径向支持向量机建模

```
#训练
model=svm.SVC(kernel='rbf',C=1,gamma=0.1)
model.fit(train_X,train_Y)

#测试
prediction1=model.predict(test_X)

#准确率
print('Accuracy for rbf SVM is ',metrics.accuracy_score(prediction1,test_Y))
```

Accuracy for rbf SVM is 0.835820895522

使用线性支持向量机

```
model=svm.SVC(kernel='linear',C=0.1,gamma=0.1)
model.fit(train_X,train_Y)
prediction2=model.predict(test_X)
print('Accuracy for linear SVM is',metrics.accuracy_score(prediction2,test_Y))
```

Accuracy for linear SVM is 0.817164179104

使用逻辑回归

```
model = LogisticRegression()
model.fit(train_X,train_Y)
prediction3=model.predict(test_X)
print('The accuracy of the Logistic Regression is',metrics.accuracy_score(prediction3,test_Y))
```

The accuracy of the Logistic Regression is 0.817164179104

决策树

```
model=DecisionTreeClassifier()
model.fit(train_X,train_Y)
prediction4=model.predict(test_X)
print('The accuracy of the Decision Tree is',metrics.accuracy_score(prediction4,test_Y))
```

The accuracy of the Decision Tree is 0.805970149254

使用KNN

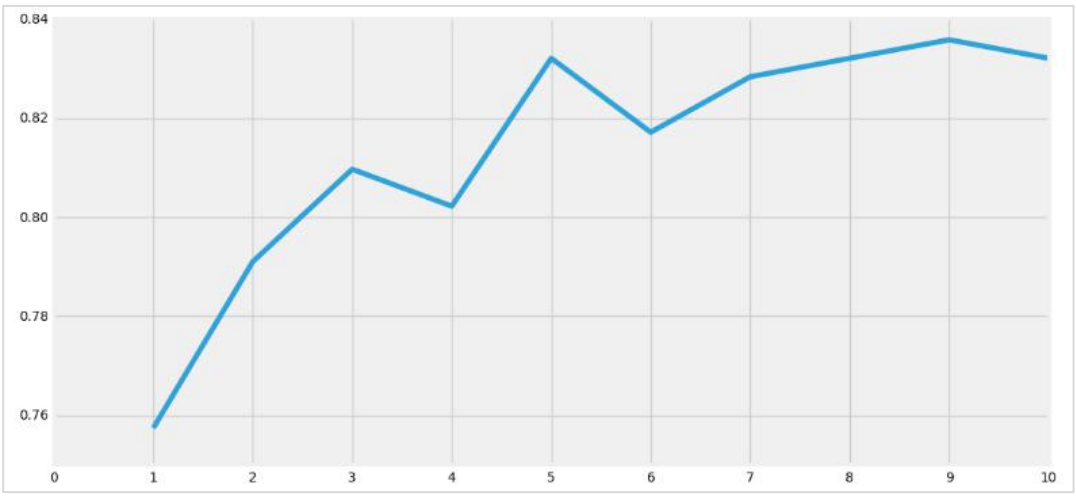
```
model=KNeighborsClassifier()
model.fit(train_X,train_Y)
prediction5=model.predict(test_X)
print('The accuracy of the KNN is',metrics.accuracy_score(prediction5,test_Y))
```

The accuracy of the KNN is 0.832089552239

我们改变KNN的n_neighbours值属性（默认值是5）。让我们检查的精度在n_neighbours不同时的结果。

```
a_index=list(range(1,11))
a=pd.Series()
x=[0,1,2,3,4,5,6,7,8,9,10]
for i in list(range(1,11)):
    model=KNeighborsClassifier(n_neighbors=i)
    model.fit(train_X,train_Y)
    prediction=model.predict(test_X)
    a=a.append(pd.Series(metrics.accuracy_score(prediction,test_Y)))
```

```
plt.plot(a_index, a)
plt.xticks(x)
fig=plt.gcf()
fig.set_size_inches(12,6)
plt.show()
print('Accuracies for different values of n are:',a.values,'with the max value as ',a.values.max())
```




```
Accuracies for different values of n are: [ 0.75746269  0.79104478  0.80970149  0.80223881  0.83208955  0.81716418
  0.82835821  0.83208955  0.8358209   0.83208955] with the max value as  0.835820895522
```

使用朴素贝叶斯

```
model=GaussianNB()
model.fit(train_X,train_Y)
prediction6=model.predict(test_X)
print('The accuracy of the NaiveBayes is',metrics.accuracy_score(prediction6,test_Y))
```

The accuracy of the NaiveBayes is 0.813432835821

使用随机深林

```
model=RandomForestClassifier(n_estimators=100)
model.fit(train_X,train_Y)
prediction7=model.predict(test_X)
print('The accuracy of the Random Forests is',metrics.accuracy_score(prediction7,test_Y))
```

The accuracy of the Random Forests is 0.820895522388

以上精度最高的是径向支持向量机。

我们不能确定分类器在不同数据源上的结果。当训练和测试数据发生变化时，精确度也会改变。它可能会增加或减少。

6. 交叉验证

为了克服数据集对模型的影响，我们需要引入交叉验证策略。

1）交叉验证的工作原理是首先将数据集分成k个子集。

2）假设我们将数据集划分为（k = 5）部分。我们预留1个部分进行测试，并对这4个部分进行训练。

3）我们通过在每次迭代中改变测试部分并在其他部分中训练算法来继续这个过程。然后对衡量结果求平均值，得到算法的平均精度。

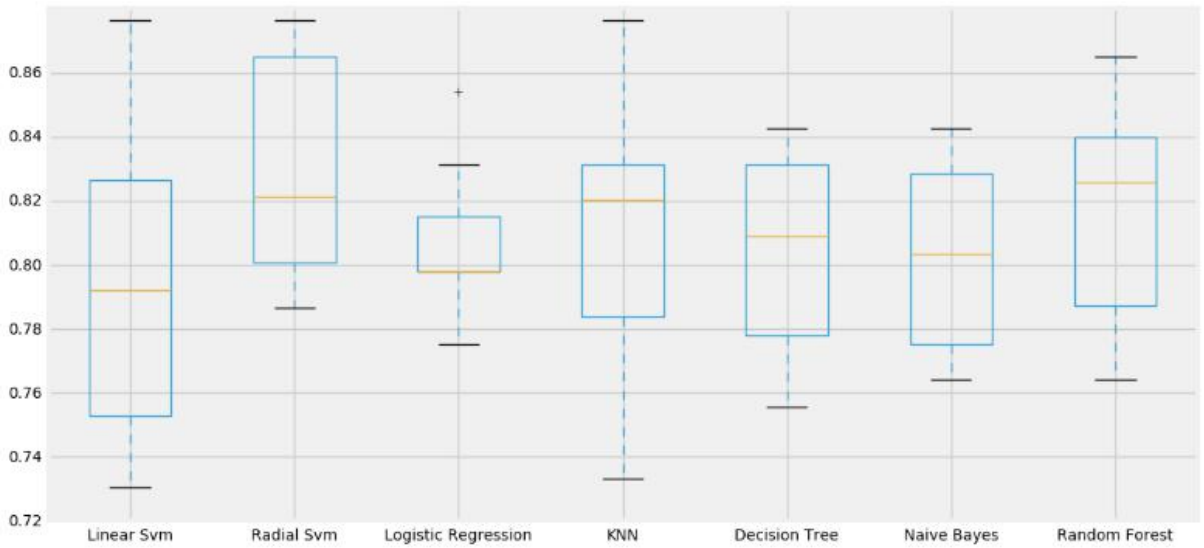
```
from sklearn.model_selection import KFold #for K-fold cross validation
from sklearn.model_selection import cross_val_score #score evaluation
from sklearn.model_selection import cross_val_predict #prediction

kfold = KFold(n_splits=10, random_state=22) # k=10,分拆数据为 10 个等分
xyz=[]
accuracy=[]
std=[]

#分类器
classifiers=['Linear Svm','Radial Svm','Logistic Regression','KNN','Decision Tree','Naive Bayes','Random Forest']

models=[svm.SVC(kernel='linear'),svm.SVC(kernel='rbf'),LogisticRegression(),KNeighborsClassifier(n_neighbors=9),DecisionTreeClassifier(),GaussianNB(),RandomForestClassifier(n_estimators=100)]

for i in models:
        model = i
        #验证模型在某个训练集上的稳定性，输出预测精度
    cv_result = cross_val_score(model,X,Y, cv = kfold,scoring = "accuracy")
    cv_result=cv_result
    xyz.append(cv_result.mean()) #均值
    std.append(cv_result.std()) #标准差
        accuracy.append(cv_result)

new_models_dataframe2=pd.DataFrame({'CV Mean':xyz,'Std':std},index=classifiers)
new_models_dataframe2
```

结果：

	CV Mean	Std
Linear Svm	0.793471	0.047797
Radial Svm	0.828290	0.034427
Logistic Regression	0.805843	0.021861
KNN	0.813783	0.041210
Decision Tree	0.805893	0.030119
Naive Bayes	0.801386	0.028999
Random Forest	0.817091	0.031978

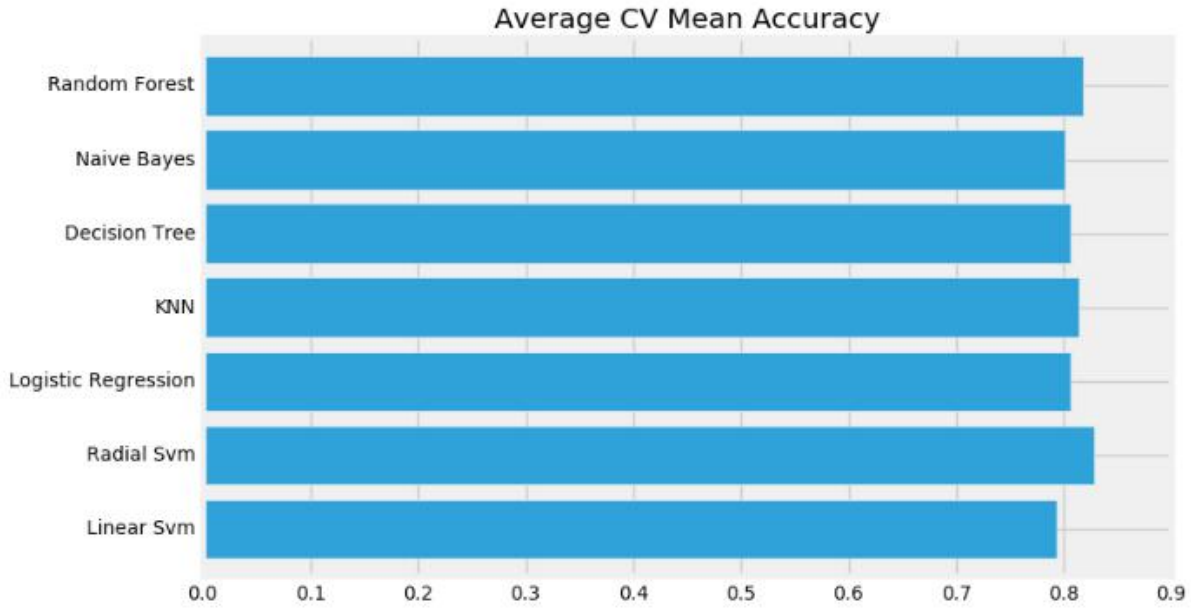
绘制盒图（箱线图）观察精度

```
plt.subplots(figsize=(12,6))
box=pd.DataFrame(accuracy,index=[classifiers])
box.T.boxplot()
```



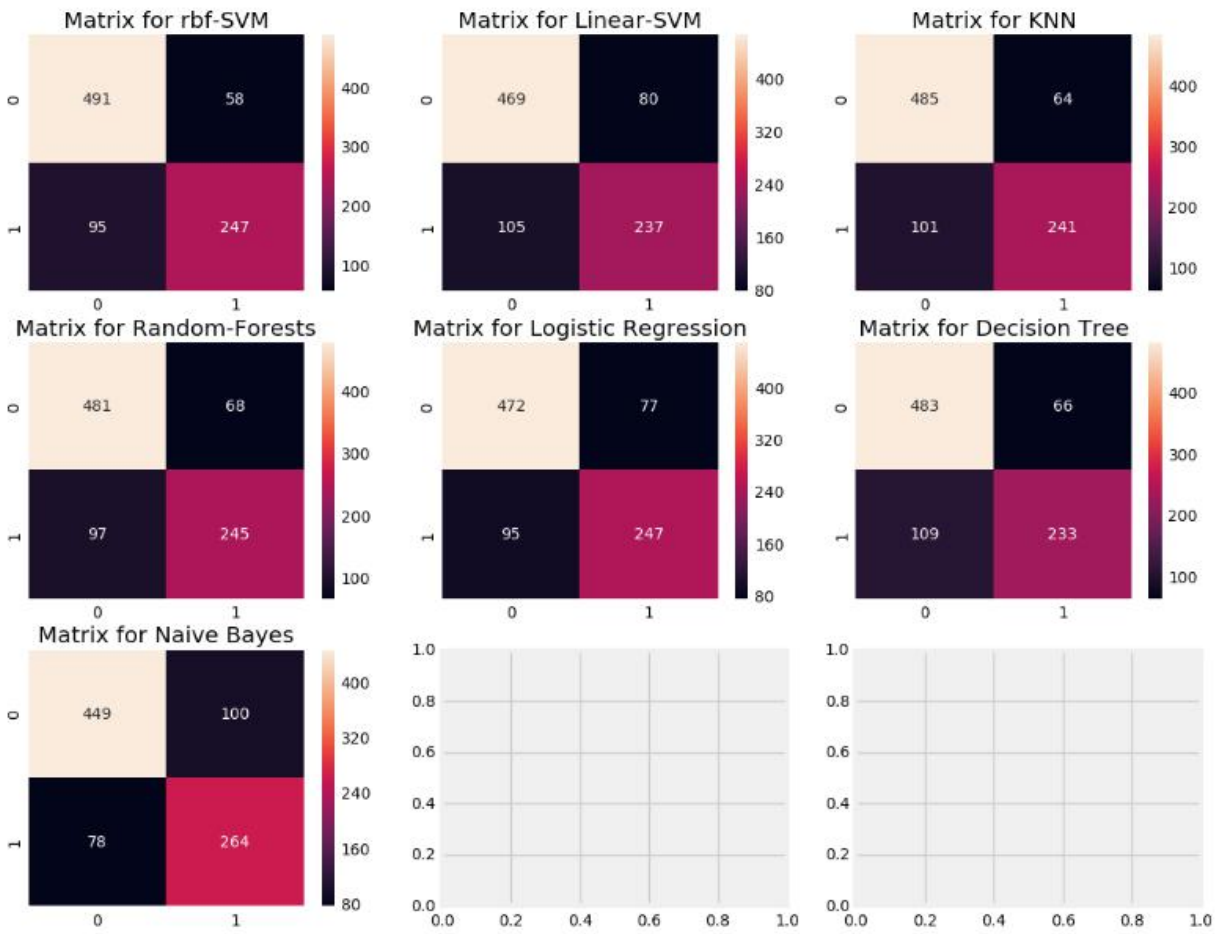
绘制横杆图，观察各分类器的平均精度

```
new_models_dataframe2['CV Mean'].plot.barh(width=0.8) #绘图
plt.title('Average CV Mean Accuracy')
fig=plt.gcf() #得到当前 FIgure 的引用
fig.set_size_inches(8,5)
plt.show()
```



混淆矩阵 它给出各分类器的正确和不正确分类的数量。

```
f,ax=plt.subplots(3,3,figsize=(12,10))
y_pred = cross_val_predict(svm.SVC(kernel='rbf'),X,Y,cv=10)
sns.heatmap(confusion_matrix(Y,y_pred),ax=ax[0,0],annot=True,fmt='2.0f')
ax[0,0].set_title('Matrix for rbf-SVM')
y_pred = cross_val_predict(svm.SVC(kernel='linear'),X,Y,cv=10)
sns.heatmap(confusion_matrix(Y,y_pred),ax=ax[0,1],annot=True,fmt='2.0f')
ax[0,1].set_title('Matrix for Linear-SVM')
y_pred = cross_val_predict(KNeighborsClassifier(n_neighbors=9),X,Y,cv=10)
sns.heatmap(confusion_matrix(Y,y_pred),ax=ax[0,2],annot=True,fmt='2.0f')
ax[0,2].set_title('Matrix for KNN')
y_pred = cross_val_predict(RandomForestClassifier(n_estimators=100),X,Y,cv=10)
sns.heatmap(confusion_matrix(Y,y_pred),ax=ax[1,0],annot=True,fmt='2.0f')
ax[1,0].set_title('Matrix for Random-Forests')
y_pred = cross_val_predict(LogisticRegression(),X,Y,cv=10)
sns.heatmap(confusion_matrix(Y,y_pred),ax=ax[1,1],annot=True,fmt='2.0f')
ax[1,1].set_title('Matrix for Logistic Regression')
y_pred = cross_val_predict(DecisionTreeClassifier(),X,Y,cv=10)
sns.heatmap(confusion_matrix(Y,y_pred),ax=ax[1,2],annot=True,fmt='2.0f')
ax[1,2].set_title('Matrix for Decision Tree')
y_pred = cross_val_predict(GaussianNB(),X,Y,cv=10)
sns.heatmap(confusion_matrix(Y,y_pred),ax=ax[2,0],annot=True,fmt='2.0f')
ax[2,0].set_title('Matrix for Naive Bayes')
plt.subplots_adjust(hspace=0.2,wspace=0.2)
plt.show()
```



混淆矩阵是一种特定的矩阵用来呈现算法性能的可视化效果，每一列代表预测值，每一行代表的是实际的类别。

来看第一个图：

1）预测的正确率为491（死亡）+ 247（存活），交叉验证（CV）平均准确率为（491+247）/ 891 = 82.8%。

2）58和95都是预测错了的。

7. 超参数调整

机器学习模型就像一个黑盒子。这个黑盒有一些默认参数值，我们可以调整或更改以获得更好的模型。比如支持向量机模型中的C和γ，我们称之为超参数，他们对结果可能产生非常大的影响。

支持向量机自动调参：

```
from sklearn.model_selection import GridSearchCV
C=[0.05,0.1,0.2,0.3,0.25,0.4,0.5,0.6,0.7,0.8,0.9,1]
gamma=[0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1.0]
kernel=['rbf','linear']
hyper={'kernel':kernel,'C':C,'gamma':gamma}

#自动调参,只要把参数输进去,就能给出最优化的结果和参数
gd=GridSearchCV(estimator=svm.SVC(),param_grid=hyper,verbose=True)
```

```
gd.fit(X,Y)
print(gd.best_score_)
print(gd.best_estimator_)
```

```
Fitting 3 folds for each of 240 candidates, totalling 720 fits
0.828282828283
SVC(C=0.5, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape=None, degree=3, gamma=0.1, kernel='rbf',
  max_iter=-1, probability=False, random_state=None, shrinking=True,
  tol=0.001, verbose=False)
[Parallel(n_jobs=1)]: Done 720 out of 720 | elapsed:   16.2s finished
```

随机深林自动调参：

```
n_estimators=range(100,1000,100)
hyper={'n_estimators':n_estimators}
gd=GridSearchCV(estimator=RandomForestClassifier(random_state=0),param_grid=hyper,verbose=True)
gd.fit(X,Y)
print(gd.best_score_)
print(gd.best_estimator_)
```

```
Fitting 3 folds for each of 9 candidates, totalling 27 fits
[Parallel(n_jobs=1)]: Done  27 out of  27 | elapsed:   29.8s finished
0.817059483726
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=None, max_features='auto', max_leaf_nodes=None,
            min_impurity_split=1e-07, min_samples_leaf=1,
            min_samples_split=2, min_weight_fraction_leaf=0.0,
            n_estimators=900, n_jobs=1, oob_score=False, random_state=0,
            verbose=0, warm_start=False)
```

总结：

支持向量机的最佳得分为82.82%，C = 0.5，γ = 0.1。

RandomForest，成绩是81.7%，n_estimators=900

8. 集成

集成是提高模型的精度和性能的一个很好的方式。简单地说，是各种简单模型的结合创造了一个强大的模型。

1）随机森林类型的，并行的集成

2）提升类型

3）堆叠类型

投票分类器

这是将许多不同的简单机器学习模型的预测结合起来的最简单方法。它基于各子模型的预测，给出了一个平均预测结果。

```
from sklearn.ensemble import VotingClassifier
ensemble_lin_rbf=VotingClassifier(estimators=[('KNN',KNeighborsClassifier(n_neighbors=10)),
                                              ('RBF',svm.SVC(probability=True,kernel='rbf',C=0.5,gamma=0.1)),
                                              ('RFor',RandomForestClassifier(n_estimators=500,random_state=0)),
                                              ('LR',LogisticRegression(C=0.05)),
                                              ('DT',DecisionTreeClassifier(random_state=0)),
                                              ('NB',GaussianNB()),
                                              ('svm',svm.SVC(kernel='linear',probability=True))
                                             ],
                        voting='soft').fit(train_X,train_Y)
print('The accuracy for ensembled model is:',ensemble_lin_rbf.score(test_X,test_Y))
cross=cross_val_score(ensemble_lin_rbf,X,Y, cv = 10,scoring = "accuracy")
print('The cross validated score is',cross.mean())
```

```
The accuracy for ensembled model is: 0.824626865672
The cross validated score is 0.823766031097
```

Bagging算法

Bagging算法（英语：Bootstrap aggregating，引导聚集算法），又称装袋算法，是机器学习领域的一种团体学习算法。最初由Leo Breiman于1994年提出。Bagging算法可与其他分类、回归算法结合，提高其准确率、稳定性的同时，通过降低结果的方差，避免过拟合的发生。

数学基础：

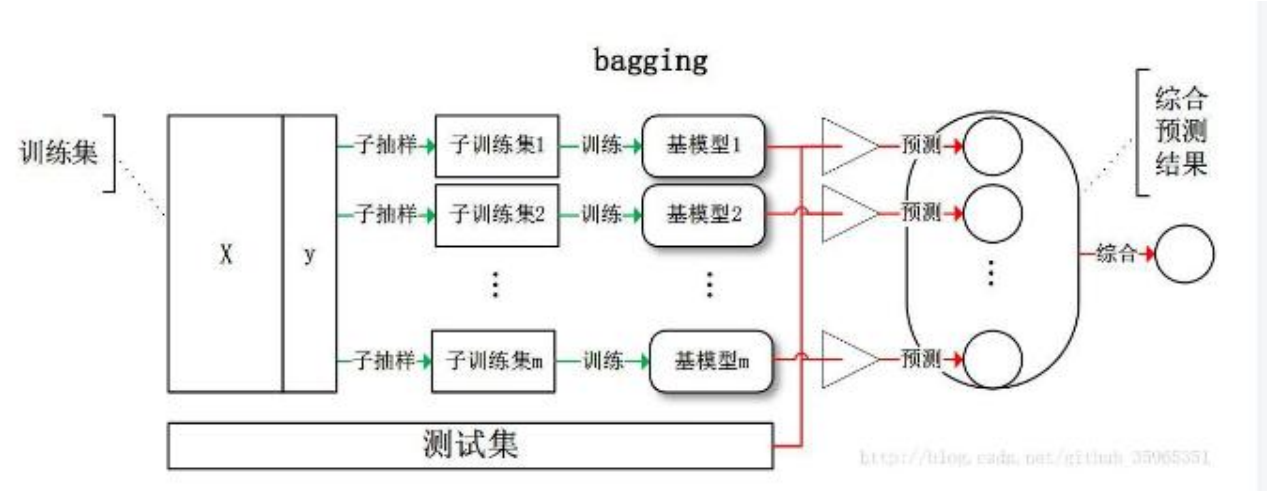
输入： 训练集 $D=\{(\boldsymbol{x}_1,y_1),(\boldsymbol{x}_2,y_2),\ldots,(\boldsymbol{x}_m,y_m)\}$;
基学习算法 $\mathfrak{L}$;
训练轮数 T.

过程：

1: **for** $t=1,2,\ldots,T$ **do**
2: $\quad h_t=\mathfrak{L}(D,\mathcal{D}_{bs})$
3: **end for**

输出： $H(\boldsymbol{x})=\arg\max_{y\in\mathcal{Y}}\sum_{t=1}^{T}\mathbb{I}(h_t(\boldsymbol{x})=y)$

图示：



scikit-learn中， 参数 max_samples 和 max_features 控制子集的大小（在样本和特征方面）
参数 bootstrap 和 bootstrap_features 控制是否在有或没有替换的情况下绘制样本和特征。
API：

sklearn.ensemble.**BaggingClassifier**(*base_estimator=None*, *n_estimators=10*, *max_samples=1.0*, *max_features=1.0*, *bootstrap=True*, *bootstrap_features=False*, *oob_score=False*, *warm_start=False*, *n_jobs=1*, *random_state=None*, *verbose=0*)[source]

使用Bagging(即bootstrap aggregatin) KNN

```
from sklearn.ensemble import BaggingClassifier
#使用 KNN（默认是决策树）
model=BaggingClassifier(base_estimator=KNeighborsClassifier(n_neighbors=3),random_state=0,n_estimators=700)
model.fit(train_X,train_Y)
prediction=model.predict(test_X)
print('The accuracy for bagged KNN is:',metrics.accuracy_score(prediction,test_Y))
result=cross_val_score(model,X,Y,cv=10,scoring='accuracy')
print('The cross validated score for bagged KNN is:',result.mean())
```

```
The accuracy for bagged KNN is: 0.835820895522
The cross validated score for bagged KNN is: 0.814889342867
```

Bagging决策树

```
model=BaggingClassifier(base_estimator=DecisionTreeClassifier(),random_state=0,n_estimators=100)
model.fit(train_X,train_Y)
prediction=model.predict(test_X)
print('The accuracy for bagged Decision Tree is:',metrics.accuracy_score(prediction,test_Y))
result=cross_val_score(model,X,Y,cv=10,scoring='accuracy')
print('The cross validated score for bagged Decision Tree is:',result.mean())
```

```
The accuracy for bagged Decision Tree is: 0.824626865672
The cross validated score for bagged Decision Tree is: 0.820482635342
```

AdaBoost（自适应增强）

Adaptive boosting(自适应增强)是一种迭代算法，其核心思想是针对同一个训练集训练不同的弱分类器，然后把这些弱分类器集合起来，构成一个强分类器，Adaboost可处理分类和回归问题。

AdaBoost（自适应增强）在这种情况下，弱学习或估计是一个决策树。但我们可以改变缺省base_estimator任何算法的选择。

AdaBoost

```
from sklearn.ensemble import AdaBoostClassifier
ada=AdaBoostClassifier(n_estimators=200,random_state=0,learning_rate=0.1)
result=cross_val_score(ada,X,Y,cv=10,scoring='accuracy')
print('The cross validated score for AdaBoost is:',result.mean())
```

The cross validated score for AdaBoost is: 0.824952616048

梯度提升

```
from sklearn.ensemble import GradientBoostingClassifier
grad=GradientBoostingClassifier(n_estimators=500,random_state=0,learning_rate=0.1)
result=cross_val_score(grad,X,Y,cv=10,scoring='accuracy')
print('The cross validated score for Gradient Boosting is:',result.mean())
```

The cross validated score for Gradient Boosting is: 0.818286233118

我们得到了最高的精度为AdaBoost。我们将尝试用超参数调整来增加它。

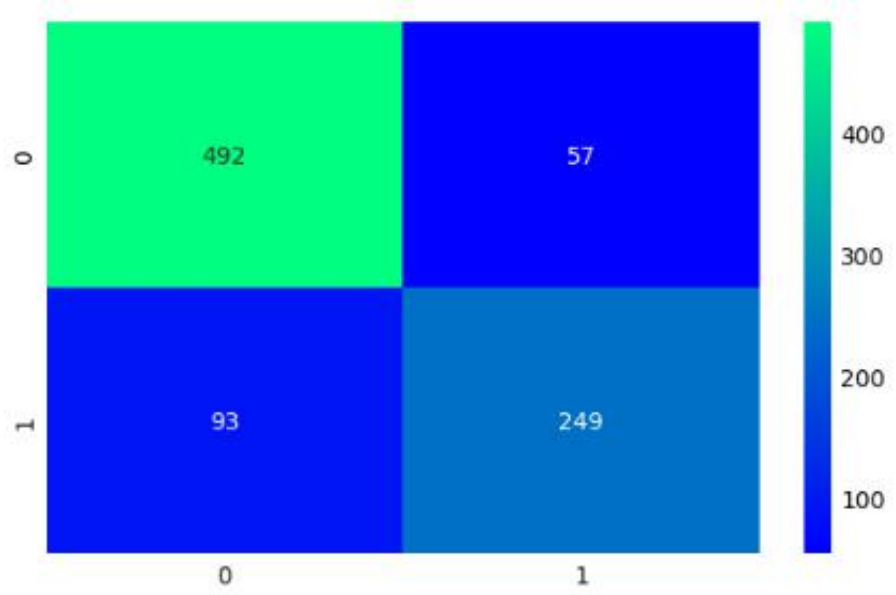
```
n_estimators=list(range(100,1100,100))
learn_rate=[0.05,0.1,0.2,0.3,0.25,0.4,0.5,0.6,0.7,0.8,0.9,1]
hyper={'n_estimators':n_estimators,'learning_rate':learn_rate}
gd=GridSearchCV(estimator=AdaBoostClassifier(),param_grid=hyper,verbose=True)
gd.fit(X,Y)
print(gd.best_score_)
print(gd.best_estimator_)
```

```
Fitting 3 folds for each of 120 candidates, totalling 360 fits
[Parallel(n_jobs=1)]: Done 360 out of 360 | elapsed:  6.0min finished
0.83164983165
AdaBoostClassifier(algorithm='SAMME.R', base_estimator=None,
          learning_rate=0.05, n_estimators=200, random_state=None)
```

我们可以从AdaBoost的最高精度是83.16%，n_estimators = 200和learning_rate = 0.05

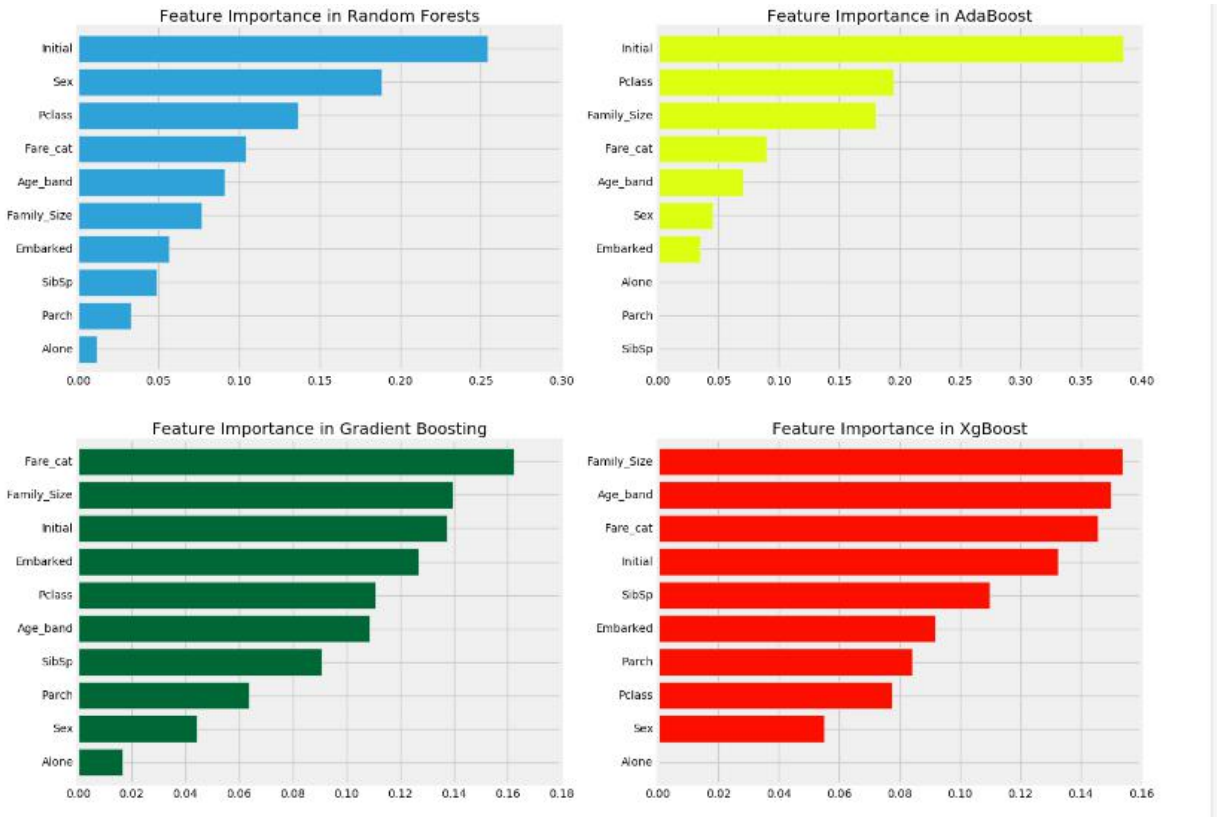
混淆矩阵来评估模型

```
ada=AdaBoostClassifier(n_estimators=200,random_state=0,learning_rate=0.05)
result=cross_val_predict(ada,X,Y,cv=10)
sns.heatmap(confusion_matrix(Y,result),cmap='winter',annot=True,fmt='2.0f')
plt.show()
```



各个特征在不同模型中的重要程度分析

```
f,ax=plt.subplots(2,2,figsize=(15,12))
model=RandomForestClassifier(n_estimators=500,random_state=0)
model.fit(X,Y)
pd.Series(model.feature_importances_,X.columns).sort_values(ascending=True).plot.barh(width=0.8,
ax=ax[0,0])
ax[0,0].set_title('Feature Importance in Random Forests')
model=AdaBoostClassifier(n_estimators=200,learning_rate=0.05,random_state=0)
model.fit(X,Y)
pd.Series(model.feature_importances_,X.columns).sort_values(ascending=True).plot.barh(width=0.8,
ax=ax[0,1],color='#ddff11')
ax[0,1].set_title('Feature Importance in AdaBoost')
model=GradientBoostingClassifier(n_estimators=500,learning_rate=0.1,random_state=0)
model.fit(X,Y)
pd.Series(model.feature_importances_,X.columns).sort_values(ascending=True).plot.barh(width=0.8,
ax=ax[1,0],cmap='RdYlGn_r')
ax[1,0].set_title('Feature Importance in Gradient Boosting')
model=xg.XGBClassifier(n_estimators=900,learning_rate=0.1)
model.fit(X,Y)
pd.Series(model.feature_importances_,X.columns).sort_values(ascending=True).plot.barh(width=0.8,
ax=ax[1,1],color='#FD0F00')
ax[1,1].set_title('Feature Importance in XgBoost')
plt.show()
```



可以看到，头衔、票价、家庭成员数量等特征，在不同的算法中发挥重要程度是不一样的。