Titanic Survive project

# Problem definition and goal

根据数据集训练，找出一个人的属性对存活的权重，并进行存活预测

# Get the dataset

# Analyze the data attributes

属性选择

明确数据集属性的含义，分析数据集的属性，确定模型相关的属性

和存活相关的属性确定：

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Survived | PassengerId | Pclass | Name | Sex | Age | SibSp | Parch | Ticket | Fare | Cabin | Embarked |
|  | No | Yes | no | Yes | yes | Yes | yes | no | yes | no | yes |
|  | 号无关 | 座等级有关 | 名字无关 | 性别有关 | 年龄有关 |  |  |  |  |  |  |

1、PassengerId,Cabin,Ticket完全无用，直接去除

2、性别转化为数值型

3、name提取title，并转化为数值型

4、年龄缺失值补充，两种逻辑：1是直接利用age的统计特性来填充缺失值；2是利用其年龄与其他属性的相关性结合年龄的统计特性来填充缺失值

5、SibSp,Parch关系处理，因为这都是有亲缘关系的，可以直接统计有总的有关系的人数，而不必分这么多类

6、费用的缺失值填充，使用出现次数最多的来填充，并根据费用范围进行分区处理，转化为数值型

7、填充Embarked缺失数据，并转化为数值型

# Data clean and Feature engneering

1、PassengerId,Cabin,Ticket完全无用，直接去除

2、性别转化为数值型

3、name提取title，并转化为数值型

4、年龄缺失值补充，两种逻辑：1是直接利用age的统计特性来填充缺失值；2是利用其年龄与其他属性的相关性结合年龄的统计特性来填充缺失值

5、SibSp,Parch关系处理，因为这都是有亲缘关系的，可以直接统计有总的有关系的人数，而不必分这么多类

6、费用的缺失值填充，使用出现次数最多的来填充，并根据费用范围进行分区处理，转化为数值型

7、填充Embarked缺失数据，并转化为数值型

8、丢弃特征工程处理后多余的特征

dataset = [train\_df,test\_df]

#数据清理

#1、PassengerId,Cabin,Ticket完全无用，直接去除

for data in dataset:

data = data.drop(['PassengerId','Ticket','Cabin'],axis =1)

#2、性别转化为数值型

for data in dataset:

data['Sex']=data['Sex'].map({'female':1, 'male':0}).astype(int)

#3、name提取title，并转化为数值型

for data in dataset:

data['Title'] = data.Name.str.extract('([A-Za-z]+)\.', expand=False)

data['Title'] = data['Title'].replace(['Lady', 'Countess', 'Capt',\

'Col', 'Don', 'Dr', 'Major',\

'Rev', 'Sir', 'Jonkheer', 'Dona'], 'Rare')

data['Title'] = data['Title'].replace('Mlle', 'Miss')

data['Title'] = data['Title'].replace('Ms', 'Miss')

data['Title'] = data['Title'].replace('Mme', 'Mrs')

title\_mapping = {"Mr":1, "Miss":2, "Mrs":3, "Master":4, "Rare":5}

data['Title'] = data['Title'].map(title\_mapping)

#4、年龄缺失值补充，两种逻辑：1是直接利用age的统计特性来填充缺失值；2是利用

#其年龄与其他属性的相关性结合年龄的统计特性来填充缺失值

#根据年龄与Pclsaa和Sex的关系来填充缺失值

import numpy as np

for data in dataset:

guess\_ages = np.zeros((2,3))

for i in range(2):

for j in range(3):

guess\_df = data[(data['Sex'] == i) & (data['Pclass'] == j+1)]['Age'].dropna()

age\_guess = guess\_df.median()

# Convert random age float to nearest .5 age

guess\_ages[i, j] = int(age\_guess / 0.5 + 0.5) \* 0.5

data.loc[ (data.Age.isnull()) & (data.Sex == i) & (data.Pclass == j+1),

'Age'] = guess\_ages[i,j]

data['Age'] = data['Age'].astype(int)

#5、SibSp,Parch关系处理，因为这都是有亲缘关系的，可以直接统计有总的有关系的人数，而不必分这么多类

for data in dataset:

data['FamilySize'] = data['SibSp'] + data['Parch'] + 1

data['IsAlone'] = 0

data.loc[data['FamilySize'] == 1, 'IsAlone'] = 1

#6、费用的缺失值填充，使用出现次数最多的来填充，并根据费用范围进行分区处理，转化为数值型

#缺失值填充

for data in dataset:

data['Fare'].fillna(data['Fare'].dropna().mode()[0], inplace=True)

#分区转为数值型

for data in dataset:

data.loc[ data['Fare'] <= 7.91, 'Fare'] = 0

data.loc[(data['Fare'] > 7.91) & (data['Fare'] <= 14.454), 'Fare'] = 1

data.loc[(data['Fare'] > 14.454) & (data['Fare'] <= 31), 'Fare'] = 2

data.loc[data['Fare'] > 31, 'Fare'] = 3

data['Fare'] = data['Fare'].astype(int)

#7、填充Embarked缺失数据，并转化为数值型

freq\_port = train\_df.Embarked.dropna().mode()[0]

for data in dataset:

data['Embarked'] = data['Embarked'].fillna(freq\_port)

for data in dataset:

data['Embarked'] = data['Embarked'].map({'S': 0, 'C': 1, 'Q': 2}).astype(int)

#8、丢弃特征工程处理后多余的特征

train = train\_df.drop(['PassengerId','Ticket','Cabin','Name','SibSp',\

'Parch','Ticket','Cabin','FamilySize',],axis =1)

test = test\_df.drop(['PassengerId','Ticket','Cabin','Name','SibSp',\

'Parch','Ticket','Cabin','FamilySize',],axis =1)

# Split the dataset

分割训练集为训练集和验证集

#划分训练集和验证集

from sklearn.model\_selection import StratifiedShuffleSplit

def MyStratifiedShuffleSplit(data,test\_ratio,attribute):

#attribbute 是数据的属性名称，字符串类型

split = StratifiedShuffleSplit(n\_splits=1, test\_size=test\_ratio, random\_state=42)

for train\_index, test\_index in split.split(data, data[attribute]):

strat\_train\_set = data.loc[train\_index]

strat\_test\_set = data.loc[test\_index]

return strat\_train\_set,strat\_test\_set

train\_set,validate\_set=MyStratifiedShuffleSplit(train,0.2,["Embarked",'Pclass'])#按数据某属性的值的比例分层划分

# 6. Train model

使用多种机器学习方法进行训练，并验证正确率

##################################################################################################

#现在我们已经做好了训练模型的准备，在模型训练完后，我们即可将其应用到解决问题中。对于预测的问题，

#我们至少有60多种算法可供选择。所以我们必须理解问题的类型和解决方案的需求，这样才能缩小模型的选择范围。

#现在这个问题是一个分类与回归的问题，我们希望找出输出（即Survived）与其他特征（即Gender，Age，Port等）

#之间的关系。因为给定了训练集，所以这在机器学习里是一个有监督学习。所以现在对算法的需求是：有监督学习加上

#分类与回归。根据这个条件，我们有以下模型可供选择：

#采用算法

from sklearn.metrics import mean\_squared\_error

################################################################################################################

#1、Logistic Regression

#训练模型

X\_train = train\_set.drop('Survived', axis=1)

Y\_train = train\_set['Survived']

X\_validate = validate\_set.drop('Survived', axis=1)

Y\_validate = validate\_set['Survived']

from sklearn.linear\_model import LogisticRegression

log\_reg = LogisticRegression()

log\_reg.fit(X\_train, Y\_train)

#验证

log\_pred = log\_reg.predict(X\_validate)

log\_mse = mean\_squared\_error(Y\_validate,log\_pred)

log\_rmse = np.sqrt(log\_mse)

print('log\_rmse = ',log\_rmse)

acc\_log = round(log\_reg.score(X\_train, Y\_train) \* 100, 2)

print("acc\_log = ",acc\_log)

########################################################################################################

#2、kNN

from sklearn.neighbors import KNeighborsClassifier

kNN\_reg = KNeighborsClassifier()

kNN\_reg.fit(X\_train, Y\_train)

#验证

kNN\_pred = kNN\_reg.predict(X\_validate)

kNN\_mse = mean\_squared\_error(Y\_validate,kNN\_pred)

kNN\_rmse = np.sqrt(kNN\_mse)

print('kNN\_rmse = ',kNN\_rmse)

acc\_kNN = round(kNN\_reg.score(X\_train, Y\_train) \* 100, 2)

print("acc\_kNN = ",acc\_kNN)

########################################################################################################

#3、SVM

from sklearn.svm import SVC

SVC\_reg = SVC()

SVC\_reg.fit(X\_train, Y\_train)

#验证

SVC\_pred = SVC\_reg.predict(X\_validate)

SVC\_mse = mean\_squared\_error(Y\_validate,SVC\_pred)

SVC\_rmse = np.sqrt(SVC\_mse)

print('SVC\_mse = ',SVC\_mse)

acc\_SVC= round(SVC\_reg.score(X\_train, Y\_train) \* 100, 2)

print("acc\_SVC = ",acc\_SVC)

########################################################################################################

#4、Naïve Bayes classifier

from sklearn.naive\_bayes import GaussianNB

NB\_reg = GaussianNB()

NB\_reg.fit(X\_train, Y\_train)

#验证

NB\_pred = NB\_reg.predict(X\_validate)

NB\_mse = mean\_squared\_error(Y\_validate,NB\_pred)

NB\_rmse = np.sqrt(NB\_mse)

print('NB\_mse = ',NB\_mse)

acc\_NB= round(NB\_reg.score(X\_train, Y\_train) \* 100, 2)

print("acc\_NB = ",acc\_NB)

########################################################################################################

#5、Decision Tree

from sklearn.tree import DecisionTreeClassifier

tree\_reg = DecisionTreeClassifier()

tree\_reg.fit(X\_train, Y\_train)

#验证

tree\_pred = tree\_reg.predict(X\_validate)

tree\_mse = mean\_squared\_error(Y\_validate,tree\_pred)

tree\_rmse = np.sqrt(tree\_mse)

print('tree\_rmse = ',tree\_rmse)

acc\_tree= round(tree\_reg.score(X\_train, Y\_train) \* 100, 2)

print("acc\_tree = ",acc\_tree)

########################################################################################################

#6、Random Forrest

from sklearn.ensemble import RandomForestClassifier

RF\_reg = RandomForestClassifier()

RF\_reg.fit(X\_train, Y\_train)

#验证

RF\_pred = RF\_reg.predict(X\_validate)

RF\_mse = mean\_squared\_error(Y\_validate,RF\_pred)

RF\_rmse = np.sqrt(RF\_mse)

print('RF\_rmse = ',RF\_rmse)

acc\_RF= round(RF\_reg.score(X\_train, Y\_train) \* 100, 2)

print("acc\_RF = ",acc\_RF)

################################################################################################

#7、Perceptron

from sklearn.linear\_model import Perceptron

ppn\_reg=Perceptron(n\_iter=1000,eta0=0.1,random\_state=0)#迭代次数、学习速率、重排训练集

ppn\_reg.fit(X\_train,Y\_train)

#验证

ppn\_pred = ppn\_reg.predict(X\_validate)

ppn\_mse = mean\_squared\_error(Y\_validate,ppn\_pred)

ppn\_rmse = np.sqrt(ppn\_mse)

print('ppn\_rmse = ',ppn\_rmse)

acc\_ppn= round(ppn\_reg.score(X\_train, Y\_train) \* 100, 2)

print("acc\_ppn = ",acc\_ppn)

#8、Artificial neural network

#9、RVM or Relevance Vector Machine

# Predict

#通过上述模型的训练验证得到数据，显示随机森林的精度最高，因此选择随机森林预测

################################################################################################

#预测

################################################################################################

#使用RF来预测

RF\_pred\_test = RF\_reg.predict(test)

StackingSubmission = pd.DataFrame({ 'PassengerId': PassengerId,

'Survived': RF\_pred\_test })

StackingSubmission.to\_csv("StackingSubmission.csv", index=False)