Python随机森林预测泰坦尼克海难生还

简介

Titanic是Kaggle竞赛的一道入门题，参赛者需要根据旅客的阶级、性别、年龄、船舱种类等信息预测其是否能在海难中生还，详细信息可以参看https://www.kaggle.com/，本文的分析代码也取自kaggle 中该竞赛的 kernal。

数据介绍

给出的数据格式如下：

PassengerId,Survived,Pclass,Name,Sex,Age,SibSp,Parch,Ticket,Fare,Cabin,Embarked
1,0,3,"Braund, Mr. Owen Harris",male,22,1,0,A/5 21171,7.25,,S
2,1,1,"Cumings, Mrs. John Bradley (Florence Briggs Thayer)",female,38,1,0,PC 17599,71.2833,C85,C

数据项的含义如下：

- PassengerId：乘客ID
- Survived：是否生还，0表示遇难，1表示生还
- Pclass：阶级，1表示最高阶级，3最低
- Name：姓名
- Sex：性别
- Age：年龄
- SibSp：同乘船的兄弟姐妹的数量
- Parch：是否有配偶同乘，1表示是
- Ticket：船票编号
- Fare：恐惧指数
- Cabin：船舱号
- Embarked：登船港口

问题分析

这是一个比较典型的基于特征的分类问题，根据一般的数据处理流程可以将问题的求解分解成为以下步骤：

1. 数据预处理
    1. 读取数据，在本文代码中使用了 python 的 pandas 包管理数据结构
    2. 特征向量化，在本文代码中将性别和登船港口特征转成向量化表示
    3. 处理残缺数据，在本文代码中将残缺年龄用平均年龄表示，残缺的登船港口用频繁项表示
    4. 扔掉多余项，姓名、ID、舱号、票号在本问题中被认为是对分类没有帮助的信息，扔掉了这些特征项
2. 数据训练

在本文代码中使用了随机森林进行分类，随机森林每次随机选取若干特征和数据项生成决策树，最后采用投票的方式来生成预测结果，本文代码中将第一列作为分类项，后n列作为特征项，随机生成100棵决策树对数据进行训练

3. 预测并生成结果

代码实现

```
import pandas as pd
import numpy as np
import csv as csv
from sklearn.ensemble import RandomForestClassifier

# Data cleanup
# TRAIN DATA
train_df = pd.read_csv('train.csv', header=0)        # Load the train file into a
dataframe

# I need to convert all strings to integer classifiers.
# I need to fill in the missing values of the data and make it complete.

# female = 0, Male = 1
train_df['Gender'] = train_df['Sex'].map( {'female': 0, 'male': 1} ).astype(int)

# Embarked from 'C', 'Q', 'S'
# Note this is not ideal: in translating categories to numbers, Port "2" is not 2
times greater than Port "1", etc.

# All missing Embarked -> just make them embark from most common place
if len(train_df.Embarked[ train_df.Embarked.isnull() ]) > 0:
    train_df.Embarked[ train_df.Embarked.isnull() ] =
train_df.Embarked.dropna().mode().values

Ports = list(enumerate(np.unique(train_df['Embarked'])))    # determine all values
of Embarked,
Ports_dict = { name : i for i, name in Ports }              # set up a dictionary
in the form  Ports : index
train_df.Embarked = train_df.Embarked.map( lambda x: Ports_dict[x]).astype(int)
# Convert all Embark strings to int

# All the ages with no data -> make the median of all Ages
median_age = train_df['Age'].dropna().median()
if len(train_df.Age[ train_df.Age.isnull() ]) > 0:
    train_df.loc[ (train_df.Age.isnull()), 'Age'] = median_age

# Remove the Name column, Cabin, Ticket, and Sex (since I copied and filled it to
Gender)
train_df = train_df.drop(['Name', 'Sex', 'Ticket', 'Cabin', 'PassengerId'], axis=1)
```

```
# TEST DATA
test_df = pd.read_csv('test.csv', header=0)        # Load the test file into a
dataframe

# I need to do the same with the test data now, so that the columns are the same as
the training data
# I need to convert all strings to integer classifiers:
# female = 0, Male = 1
test_df['Gender'] = test_df['Sex'].map( {'female': 0, 'male': 1} ).astype(int)

# Embarked from 'C', 'Q', 'S'
# All missing Embarked -> just make them embark from most common place
if len(test_df.Embarked[ test_df.Embarked.isnull() ]) > 0:
    test_df.Embarked[ test_df.Embarked.isnull() ] =
test_df.Embarked.dropna().mode().values
# Again convert all Embarked strings to int
test_df.Embarked = test_df.Embarked.map( lambda x: Ports_dict[x]).astype(int)


# All the ages with no data -> make the median of all Ages
median_age = test_df['Age'].dropna().median()
if len(test_df.Age[ test_df.Age.isnull() ]) > 0:
    test_df.loc[ (test_df.Age.isnull()), 'Age'] = median_age

# All the missing Fares -> assume median of their respective class
if len(test_df.Fare[ test_df.Fare.isnull() ]) > 0:
    median_fare = np.zeros(3)
    for f in range(0,3):                                              # loop 0 to 2
        median_fare[f] = test_df[ test_df.Pclass == f+1 ]['Fare'].dropna().median()
    for f in range(0,3):                                              # loop 0 to 2
        test_df.loc[ (test_df.Fare.isnull()) & (test_df.Pclass == f+1 ), 'Fare'] =
median_fare[f]

# Collect the test data's PassengerIds before dropping it
ids = test_df['PassengerId'].values
# Remove the Name column, Cabin, Ticket, and Sex (since I copied and filled it to
Gender)
test_df = test_df.drop(['Name', 'Sex', 'Ticket', 'Cabin', 'PassengerId'], axis=1)


# The data is now ready to go. So lets fit to the train, then predict to the test!
# Convert back to a numpy array
train_data = train_df.values
test_data = test_df.values


print 'Training...'
forest = RandomForestClassifier(n_estimators=100)
```

```
forest = forest.fit( train_data[0::,1::], train_data[0::,0] )

print 'Predicting...'
output = forest.predict(test_data).astype(int)


predictions_file = open("myfirstforest.csv", "wb")
open_file_object = csv.writer(predictions_file)
open_file_object.writerow(["PassengerId","Survived"])
open_file_object.writerows(zip(ids, output))
predictions_file.close()
print 'Done.'
```