

華東理工大學

模式识别大作业

题 目	Titanic 号乘客存活预测
学 院	信息科学与工程
专 业	控制科学与工程
组 员	孙浩
指导教师	赵海涛

完成日期： 2018 年 10 月 24 日

模式识别大作业——Titanic 号乘客存活预测

一.课题背景

泰坦尼克号是英国白星航运公司下辖的一艘奥林匹克邮轮，在其处女航行中，因与一座冰山相撞而至沉船。在这次事故中，有约三分之二的人丧生。近百年来，关于泰坦尼克号沉没的原因，一直是人类关注的话题。正因为如此，泰坦尼克号问题应运而生，根据已知的乘客的个人信息，通过使用模式识别相关的方法，学习一个分类模型，来预测乘客是否在本次沉船灾难中最终存活。这是一个典型的二分类问题，只存在两种预测结果，存活或死亡。

模式 (Pattern) 这个词的意思有两层，第一层是代表事物 (个体或一组事物) 的模板或原型；第二层是表征事物特点的特征或性状的组合。在模式识别学科中，模式可以看作是对象的组成成分或影响因素间存在的规律性关系，或者是因素间存在的确定性或随机性规律的对象、过程或事件的集合。因此，模式也被称为模式类，模式识别也被称作为模式分类 (Pattern Classification)。

赵老师在几次模式识别课中，已经为我们详细讲解了模式识别中的几种典型方法：最小二乘法、BP 神经网络、PCA 主成分分析、最大似然估计、贝叶斯决策和贝叶斯估计等相关的理论。让我具备了模式识别的基础知识，为研究泰坦尼克号问题打下了基础。

二.课题分析

从课题的背景可知，这是一个二分类的问题，不妨从数据集看看已知的信息都有哪些。数据集共分为训练数据集和测试数据集，其中包含了乘客的个人信息如下：

1. PassengerId：乘客的唯一 ID
2. Survived：乘客最终是否存活(0 = No, 1 = Yes, 仅 train.csv 中包含此信息)
3. Pclass：乘客的船票的等级(1 = 1st, 2 = 2nd, 3 = 3rd)
4. Name：乘客名字
5. Sex：乘客性别(male, female)
6. Age：乘客年龄(Year)
7. Sibsp：船上兄弟姐妹/配偶的人数
8. Parch：船上父母/儿女的人数
9. Ticket：船票号码
10. Fare：船票价格
11. Cabin：船舱号

12. Embarked : 出发港口(C = Cherbourg, Q = Queenstown, S = Southampton)

训练集中共包含了 12 种信息，但是并非每一种信息都可以作为训练集的特征，例如乘客的 ID 似乎是随机分配的，对他们是否幸存的结果影响并不大。因此选择合适的信息作为训练集的训练特征是首要任务，但是也可以根据训练过程中发现的问题和可能提高分类器性能的角度重新修正训练特征的选取。对于一个典型的二分类问题，首先想到的就是 logistic 回归模型，logistic 回归模型针对线性可分问题来说，是一种易于实现且性能优异的分类模型。因此，对于泰坦尼克号问题，首先想到的就是 logistic 回归模型，但是分类性能可能会强差人意，因为很可能训练模型的线性可分性不是特别明显，因此还需要考虑改进或者是另外的分类模型做备选。通过学习赵老师在 GitHub 上给出的关于 XGBOOST 的模式识别的程序实例，发现这个模型似乎控制模型过拟合效果较好，因此 XGBOOST 也作为模型备选之一。

选取好特征和分类模型后，利用训练数据对模型进行训练。模型训练好后，将测试数据 test 送入训练模型，可以得到对测试集的结果预测。

三.模型构建方案

对于一个典型的二分类问题，并且在训练数据中是含有标签的，因此是一个有监督学习任务。有监督学习任务的过程如下图所示：

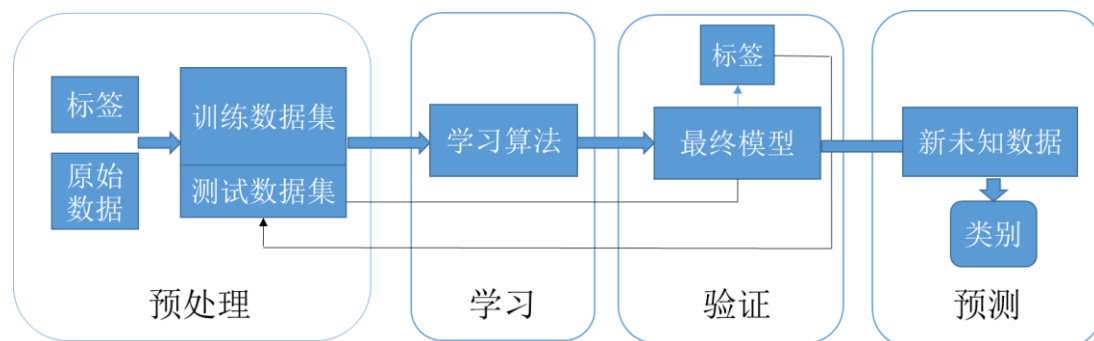


图 1 监督学习任务

从上图中可以看出，有监督学习任务主要分为数据预处理，利用学习算法训练模型，验证模型和预测新数据几个步骤。

3.1 数据预处理

在 lintcode 上下载下来的泰坦尼克号乘客的相关训练和测试数据集,但是通过分析这些数据，我们可以看出有些数据其实是存在缺失的，因此如果想使用这些包含缺失项的数据集，我们必须对这些缺失的数据进行补全。从图 2 可以看出训练集中乘客数量一共是 891，但是年龄（Age）和船舱号（Cabin）以及出发港口（Embarked）信息有部分缺失。因此考虑这三个信息是否可以作为特征，船舱号似乎对乘客是否幸存影响不大，且船舱号（Cabin）信息确实过大，故将船舱号（Cabin）这组信息暂时不作为特征。年龄（Age）对于是否幸存影响较大，毕竟年轻的人生命力旺盛一些，因此年龄缺失的部分需要想办法补全

缺失的信息。

```
In [1]: import pandas as pd
        train = pd.read_csv('/Users/Administrator/Desktop/train.csv')

In [3]: train.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
PassengerId    891 non-null int64
Survived       891 non-null int64
Pclass         891 non-null int64
Name           891 non-null object
Sex            891 non-null object
Age           714 non-null float64
SibSp          891 non-null int64
Parch          891 non-null int64
Ticket         891 non-null object
Fare           891 non-null float64
Cabin         204 non-null object
Embarked       889 non-null object
dtypes: float64(2), int64(5), object(5)
memory usage: 83.6+ KB
```

图 2.train 数据信息

3.1.1 补全年龄的策略

年龄缺失部分的补全主要考虑了两种策略，一种是简单的平均值补全的方法，另一种是采用随机森林法补全缺失的数据。

平均值补全法较为简单实现，只需要将缺失部分的年龄用年龄的平均值进行填充就可以，代码如下：

```
X_train['Age'].fillna(X_train['Age'].mean())
```

```
X_test['Age'].fillna(X_test['Age'].mean())
```

随机森林算法的实质是基于决策树的分类器集成算法，其中每一棵树都依赖于一个随机向量，随机森林的所有向量都是独立同分布的。随机森林就是对数据集的列变量和行观测进行随机化，生成多个分类数，最终将分类树结果进行汇总。随机森林相比于神经网络，降低了运算量的同时也提高了预测精度，而且该算法对多元共线性不敏感以及对缺失数据和非平衡数据比较稳健，可以很好地适应多达几千个解释变量数据集。在本课题中，将已知的训练集根据年龄的有无分成两个数据集，年龄信息完整的数据集用来训练随机森林模型，利用训练好的模型对训练集中缺失年龄的数据进行预测，将预测的年龄填充到缺失的年龄。代码如下：

```
def set_missing_ages(df): # 把已有的数值型特征取出来丢进Random Forest Regressor中
    age_df = df[['Age', 'Fare', 'Parch', 'SibSp', 'Pclass']] # 乘客分成已知年龄和未知年龄两部分
    known_age = age_df[age_df.Age.notnull()].as_matrix()
```

```

unknown_age = age_df[age_df.Age.isnull()].as_matrix()
# y即目标年龄
y = known_age[:, 0]
# X即特征属性值
X = known_age[:, 1:]
# fit到RandomForestRegressor之中
rfr = RandomForestRegressor(random_state=0, n_estimators=2000, n_jobs=-1)
rfr.fit(X, y)
# 用得到的模型进行未知年龄结果预测
predictedAges = rfr.predict(unknown_age[:, 1:])
# 用得到的预测结果填补原缺失数据
df.loc[ (df.Age.isnull()), 'Age' ] = predictedAges
return df, rfr
train ,rfr = set_missing_ages(train)#将train中的缺失的年龄补全

```

3.1.2 特征归一化

选取的特征有一些为文本信息，如果想使用 logistic 回归模型来进行分类，需要将这些文本类的分类信息转化为 0 或 1，而不能直接使用字符串，在 python 中可以使用 pandas 中的 `get_dummies` 的方法，对字符串列进行转换。实质是通过引入虚拟变量，即构造 0/1 的人工变量，将不能量化的变量进行量化，虽然可能让模型复杂化，但是对于问题的描述却起到了简化的作用。例如本课题中的 Embarked 属性：

```
dummies_Embarked = pd.get_dummies(data_train['Embarked'], prefix= 'Embarked')
```

```
print(data_train.Embarked)
```

```

0      S
1      C
2      S
3      S
4      S
5      Q
6      S
7      S
8      S
9      C
10     S
11     S
12     S

```

```
dummies_Embarked
```

	Embarked_C	Embarked_Q	Embarked_S
0	0	0	1
1	1	0	0
2	0	0	1
3	0	0	1
4	0	0	1
5	0	1	0
6	0	0	1
7	0	0	1
8	0	0	1
9	1	0	0
10	0	0	1
11	0	0	1
12	0	0	1

图 3.特征归一化

从图三可以看出，原本在 12 名乘客的信息中含有三类 Embarked,分别为

S、C、Q，但是经过转化后，就变成了只有 0/1 来表示的变量。虽然增加了变量的数量，但是让数据可操作性增强了。同时对特征 Sex 和 Pclass 也做归一化处理，代码如下：

```
dummies_Embarked = pd.get_dummies(data_train['Embarked'], prefix= 'Embarked')
dummies_Sex = pd.get_dummies(data_train['Sex'], prefix= 'Sex')
dummies_Pclass = pd.get_dummies(data_train['Pclass'], prefix= 'Pclass')
df = pd.concat([data_train, dummies_Embarked, dummies_Sex, dummies_Pclass], axis=1)
df.drop(['Pclass', 'Name', 'Sex', 'Ticket', 'Embarked'], axis=1, inplace=True)
```

对于特征归一化，也可以采用 DictVectorizer 方法，DictVectorizer 对非数字化的处理方式是，借助原特征的名称，组合成新的特征，并采用 0/1 的方式进行量化，而数值型的特征转化比较方便，一般情况维持原值即可

对于年龄（Age）和票价（Fare）这两个特征来说，本身是数字，需要将其规整到（-1，1）之间，从而方便 logistic 模型进行训练。代码如下：

```
scaler = preprocessing.StandardScaler()#将 Age 和 Fare 的值压缩到（-1 和 1 之间）
df['Age_scaled'] = scaler.fit_transform(df['Age'].values.reshape(-1,1))
df['Fare_scaled'] = scaler.fit_transform(df['Fare'].values.reshape(-1,1))
```

至此，数据特征的预处理基本完成，可供模型训练使用。

3.2 模型原理和程序

3.2.1 logistic 回归模型

Logistic 回归模型，虽然称为回归，但是却用来解决分类问题。针对本课题，是一个二分类问题，但是 logistic 回归模型也可以处理多分类问题。类比于线性回归，我们希望 logistic 回归模型也有一个假设函数（Hypothesis） $H_{\theta}(x)$ ，但是对于这个假设函数，我们希望它的取值范围为 $0 \leq H_{\theta}(x) \leq 1$ ，（稍后作出解释）， $H_{\theta}(x)$ 的形式如下：

$$H_{\theta}(x) = g(\theta^T X)$$

$$g(z) = \frac{1}{1 + e^{-z}}$$

只需要将 z 换掉，因此：

$$H_{\theta}(x) = \frac{1}{1 + e^{-\theta^T X}}$$

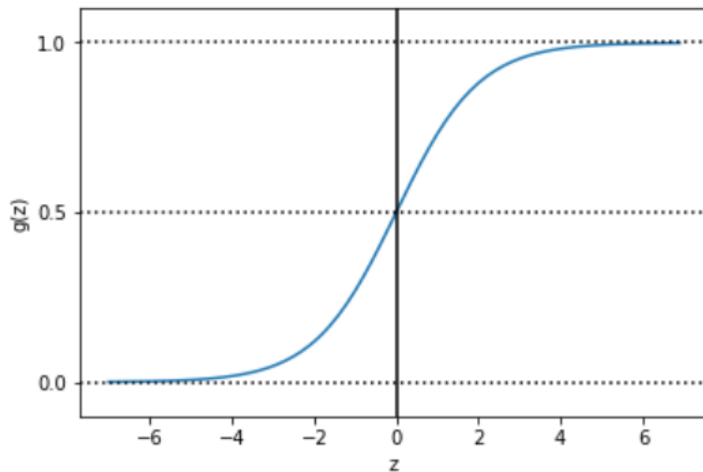


图 4. $g(z)$ 的图像

从图 4 中可以看出， $g(z)$ 的取值范围是 $0 \leq g(z) \leq 1$ 符合我们刚才的要求。因此，对于 $H_{\theta(x)}$ 的输出值，我们将它类似的看成一种概率的估计，即输入为 X ，并且类别标签 $y=1$ 时的概率。（对于二分类问题，只有两个类别，我们选择标签为 0 和 1，0 代表消极类别（negative class），1 代表积极类别（positive class））。例如 $H_{\theta(x)} = 0.7$ 则表示在输入特征 X （特征大多数是多维矩阵，所以此处采用大写 X ，在公式中为了与 $H_{\theta(x)}$ 统一，采用小写 x ）的情况下，属于类别 1 的概率为 70%。如果采用确切的概率形式表示：

$$H_{\theta}(x) = P(y = 1|x, \theta)$$

含义是：给定 X 的情况下，参数 θ 的作用下，属于类别 $y=1$ 的概率，其中 x 表示的是输入特征向量， θ 为参数， y 是类别标签。

$$P(y = 0|x, \theta) + P(y = 1|x, \theta) = 1$$

$$P(y = 0|x, \theta) = 1 - P(y = 1|x, \theta)$$

对于分类问题来说，最重要的是构建出分类边界，也被称为决策边界。我们所构造的 $H_{\theta(x)}$ 假设函数，通过观察 $g(z) > 0.5$ 时， $z > 0$ ； $g(z) < 0.5$ 时， $z < 0$ 。可以很明显的看出 $g(z) = 0.5$ 是一个临界值，类似一个分界面，将这个映射到 H 函数上，因此构建如下策略：

$$H_{\theta}(x) \geq 0.5 \text{ 即 } \theta^T X \geq 0 \text{ 此时判定 } y=1$$

$$H_{\theta}(x) < 0.5 \text{ 即 } \theta^T X < 0 \text{ 此时判定 } y=0$$

从概率的角度来说，我们假设的是当前的目标属于类别 1，因此当概率大于 50% 的情况下，判定属于类别 1，当概率小于 50% 的情况下，判定为类别 0。貌似在理论上行得通。

但是需要注意的是，我们所构建的这个决策边界实际上是假设函数的属性，决定的是分类面，跟数据集属性无关，与参数 θ 有关。

参数 θ 如何来获得，将是 logistic 回归模型中最重要的一环。模型的参数

实际上是需要通过训练集的数据训练获得的。

假设有训练集 $\{(x^1, y^1), (x^2, y^2) \dots \dots (x^m, y^m)\}$ ，共有 m 组数据，每一组数据中包含特征 x 又是 n 维的列向量， $y \in \{0,1\}$ ，则：

$$x = \begin{bmatrix} x^0 \\ x^1 \\ \dots \\ x^n \end{bmatrix} \quad H_{\theta}(x) = \frac{1}{1 + e^{-\theta^T x}}$$

想要确定参数 θ 的值，需要构建一个代价函数，优化的目标就是让代价函数最小，换句话说，在做决策的时候，希望决策者做的决策所付出的代价要尽可能小，对于线性回归来说，预测值和真实值之间的误差越小，决策者付出的代价也就越小，拟合的效果也就越好。因此对于线性回归来说，其代价函数如下：

$$J(\theta) = \frac{1}{m} \sum_{i=1}^m \frac{1}{2} (H_{\theta}(x^{(i)}) - y^{(i)})^2 = \frac{1}{m} \sum_{i=1}^m \text{Cost}(H_{\theta}(x^{(i)}), y^{(i)})$$

$$\text{Cost}(H_{\theta}(x^{(i)}), y^{(i)}) = \frac{1}{2} (H_{\theta}(x^{(i)}) - y^{(i)})^2$$

但是对于 logistic 回归来说， $H_{\theta}(x)$ 是非线性函数，如果也采用上面线性回归的代价函数 $\text{Cost}(H_{\theta}(x^{(i)}), y^{(i)})$ 的话，会生成非凸形式的 $J(\theta)$ ，采用梯度下降法不能收敛到全局最优，因此需要重新构建一个 $\text{Cost}(H_{\theta}(x^{(i)}), y^{(i)})$ 。构造如下的代价函数：

$$\text{Cost}(H_{\theta}(x), y) = \begin{cases} -\log(H_{\theta}(x)) & \text{if } y = 1 \\ -\log(1 - H_{\theta}(x)) & \text{if } y = 0 \end{cases}$$

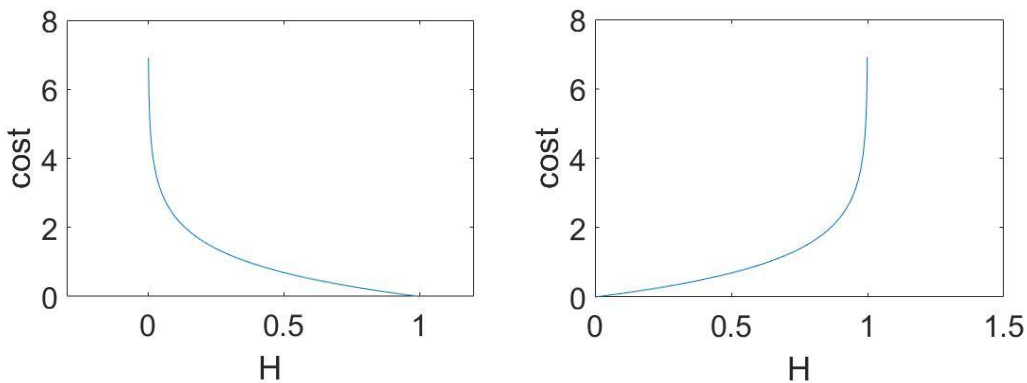


图 6.代价函数

图 6 的图中描绘的分别是 $y=1$ 的情况下（左侧的图）， $y=0$ 的情况下的（右侧的图）代价函数的图形。横坐标 H 代表的就是 $H_{\theta}(x)$ ，纵坐标 cost 代表的 $\text{Cost}(H_{\theta}(x), y)$ 。

从左侧的图（对应的是 $y=1$ ）中可以看出,在 $y=1$ 时，同时 $H_{\theta}(x) = 1$ ，也就是说我们预测出的 $y=1$ 的概率是 100%（预测结果为 $y=1$ ），此时代价函数为 0；而如果我们 在 $y=1$ 时， $H_{\theta}(x) = 0$ ，此时说明我们预测出的 $y=1$ 的概率为 0

($P(y = 1|x, \theta) = 0$ 也就是预测出结果为 $y=0$)，此时我们要付出的代价是无穷大的，说明错误的决策损失很严重。

同样的在右侧的图（对应的是 $y=0$ ）中也可以看出，如果实际情况是 $y=0$ ，而预测的结果 $H_\theta(x) = 0$ ，此时 $P(y = 1|x, \theta) = 0$ ，预测结果和实际符合，代价为 0；而如果预测出的结果为 $P(y = 1|x, \theta) = 1$ ，此时预测结果和实际结果完全相反，则付出的代价是无穷大的。

通过以上的分析，构建的代价函数可以作为目标函数，作为优化的目标，将代价函数最小化，就可以求得较为理想的模型。为了方便后续的优化处理，将上面的代价函数写在一起：

$$\text{Cost}(H_\theta(x), y) = -y \log(H_\theta(x)) - (1 - y) \log(1 - H_\theta(x))$$

其实上式是通过统计学中极大似然估计得出的一个公式，其实很容易理解，因为 y 的取值无非两种 0 或 1，当 $y=1$ 时，第二项中的 $1-y=0$ ，就消去了第二项；当 $y=0$ 时，自动消去了第一项。

由于训练集中有 m 组数据，因此构建优化目标函数：

$$J(\theta) = -\frac{1}{m} \left[\sum_{i=1}^m y^{(i)} \log(H_\theta(x^{(i)})) - (1 - y^{(i)}) \log(1 - H_\theta(x^{(i)})) \right]$$

上式的 $J(\theta)$ 就是我们需要优化的目标函数，尽可能的让 $J(\theta)$ 的小，从而得到的参数 θ 用于构建分类模型。

可以采用梯度下降法来将 $J(\theta)$ 最小化：

$$\min_{\theta} J(\theta)$$
$$\theta_j = \theta_j - \alpha \sum_{i=1}^m (H_\theta(x^{(i)}) - y^{(i)}) x_j^i$$

需要注意的是，同时更新所有的 θ_j 。为方便运算可以采用向量化方法求解，赵老师在前两次课上详细讲述了向量化处理梯度下降法的公式。

在求出参数 θ 后，将参数带回到最初的 $H_\theta(x)$ ，此时就完成了模型的训练，可以使用该模型对测试集数据进行预测。

3.2.2 代码实现

Python 中的 `scikit-learn` 库中已经含有 `logistic` 回归封装好的代码，使用起来较为方便。

```
# y 即 Survival 结果
y = train_np[:, 0]
# X 即特征属性值
X = train_np[:, 1:]
```

```
clf = linear_model.LogisticRegression(C=1.0, penalty='l1', tol=1e-6) #C 为正则化系数
clf.fit(X, y)
```

3.2 XGBOOST 方法

这种方法是我在学习赵老师给的模式识别编程实例中看到的一种方法，因此就作为一种尝试，也用在了这次作业的过程中，因此在这里不做详细的理论分析，只介绍相关的代码。没想到用这种方法做出的效果比 **logistic** 的效果要好一些。

```
#模型选择 XGB
xgb_model = xgb.XGBClassifier()
#设置参数
params = dict(booster='gbtree',
               objective='multi:softmax',
               num_class=2,
               learning_rate=0.1,
               max_depth=2,
               silent=0,)
# 设置迭代次数
plst = list(params.items())
num_rounds = 2000
# train_test_split 进行训练数据集划分，训练集和交叉验证集比例
train_x, val_X, train_y, val_y = train_test_split(X_train, y_train, test_size=0.2, random_state=1)
# xgb 矩阵赋值
xgb_val = xgb.DMatrix(val_X, label=val_y)
xgb_train = xgb.DMatrix(train_x, label=train_y)
xgb_test = xgb.DMatrix(X_test)
#watchlist 方便查看运行情况
watchlist = [(xgb_train, 'train'), (xgb_val, 'val')]
# training model
# early_stopping_rounds 当设置的迭代次数较大时，early_stopping_rounds 可在一定的迭代次数内准确率没有提升就停止训练
model = xgb.train(plst, xgb_train, num_rounds, watchlist, early_stopping_rounds=100)
```

在具体的参数设置中 **booster: gbtree**，选择基于树的模型；参数 **objective** 设置为 **multi:softmax**，旨在使用 **softmax** 的多分类器，返回预测的类别；参数 **num_class**，用于设置类别数量，在本课题中是二分类的问题，因此选择参数为 2；参数 **learning_rate**，代表的是学习率，也可以理解为步长，通过减少每一步

的权重，可以提高模型的鲁棒性，初始值选择为 0.1；参数 `max_depth`，这个也是用来避免过拟合的。`max_depth` 越大，模型会学到更具体更局部的样本；参数 `silent` 设置为 0 能显示运行情况，让我们更好地理解模型。

四 模型性能测试和检验

4.1 logistic 模型



在 logistic 回归模型的构建中，为了检验模型的性能，我才用了交叉验证的方法来检验模型训练的效果，代码如下：

```
scores = cross_val_score(clf, X, y, cv=10)#进行 10 次交叉验证
print(scores)
b=len(scores)
sum_score = 0
for i in scores:
    sum_score=sum_score+i
mean_score=sum_score/b
print(mean_score)
```

结果如下图所示：

```
[ 0.81111111  0.78888889  0.7752809   0.85393258  0.80898876  0.76404494
 0.79775281  0.79775281  0.83146067  0.82954545]
0.805875893769
```

通过结果可以看出交叉验证的平均准确率有 80.58%，貌似还可以接受，于是我将模型应用在了测试数据上，并将结果传到了 lintcode 上，以此来验证结果。

 predictions.csv 2 days ago	成功	0.76555
second		
 predictions.csv 2 days ago	成功	0.76555
first		

可惜的是，在测试数据集上的准确率只有 76.555%，和交叉验证得到的准确率差距还有些大，不过也可以理解，因为交叉验证使用的数据集仍然是训练用的训练数据集，因此肯定会准确率较高。由此也可以看出训练的模型存在过拟合的现象。分析原因，可能是前期的数据处理不够，丢掉的数据特征可能也会对结果产生影响，填充缺失的数据时，没能很好的训练随机森林的模型，导致填充的缺



失数据和真实数据相差较大。

4.2 XGBOOST 模型

Logistic 模型没有能很好的完成预测任务，因此选择赵老师给出编程实例上的 XGBOOST 的方法进行尝试。在 XGBOOST 的方法中我尝试了两种不同的补偿缺失数据的方法，因此在附录代码中会出现 XGBOOST1 和 XGBOOST2，其中 XGBOOST1 中，采用的是平均值补缺失数据的方法，而在 XGBOOST2 中则是采用了 logistic 回归模型使用的随机森林的方法来补全缺失的年龄数据。

```
[182]    train-merror:0.123596    val-merror:0.206704
[21:16:03] d:\build\xgboost\xgboost-0.80.git\src\tre
xtra nodes, 0 pruned nodes, max_depth=2
[21:16:03] d:\build\xgboost\xgboost-0.80.git\src\tre
xtra nodes, 0 pruned nodes, max_depth=2
[183]    train-merror:0.123596    val-merror:0.206704
[21:16:03] d:\build\xgboost\xgboost-0.80.git\src\tre
xtra nodes, 0 pruned nodes, max_depth=2
[21:16:03] d:\build\xgboost\xgboost-0.80.git\src\tre
xtra nodes, 0 pruned nodes, max_depth=2
[184]    train-merror:0.123596    val-merror:0.206704
[21:16:03] d:\build\xgboost\xgboost-0.80.git\src\tre
xtra nodes, 0 pruned nodes, max_depth=2
[21:16:03] d:\build\xgboost\xgboost-0.80.git\src\tre
xtra nodes, 0 pruned nodes, max_depth=2
[185]    train-merror:0.123596    val-merror:0.206704
```

如图所示训练过程中的部分状态，在一定次数的训练后，误差基本趋于稳定。将模型应用在测试数据上，并上传到 lintcode 上。

 predictions.csv a day ago	成功	0.77273
SEVEY		
 predictions.csv a day ago	成功	0.77273
six		

结果还不错，比之前的 logistic 的结果要好，准确率达到了 77.273%，由此可以看出赵老师给出的方法是非常有效果的。赵老师给出的结果将准确率做到了 78.7%，为了努力向老师的准确率靠近，我尝试的将随机森林法融入到 XGBOOST 中，在 XGBOOST2 的代码中，采用随机森林的方法填充缺失的数据。

 predictions.csv 1天前	成功	0.77751
---	----	---------

算法改进过后，准确率提升到了 77.751%，相比于 XGBOOST1,提升了将近

0.5%，证明随机森林法填补数据要比单纯的使用平均值填补数据更能接近真实的数据，由此得到的数据训练出来的模型预测的准确率更高。

五. 总结与感想

通过采用 `logistic` 回归模型对泰坦尼克号问题进行预测，预测率达到了 76.555%，进而采用赵海涛老师给出的实例 `XGBOOST` 的方法对相同的数据集进行了预测，通过不断调整学习率和迭代次数，最终取得了 77.751% 的准确率。由此可见，针对泰坦尼克号问题，`XGBOOST` 算法的分类准确率更高。同时也比较了两种填充缺失数据的方法，随机森林法补全缺失数据要比平均值法补全缺失数据的效果更好一些。

通过学习模式识别这门课，让我进一步了解模式识别的理论基础和相关算法，在赵海涛老师生动的讲解下，原本难懂枯燥的公式推导，也变得相对容易理解。赵老师布置的这次大作业，意在让我们通过自己动手编程，加深对相关算法的理解。由于这是第一次做和模式识别相关的编程作业，还是遇到了很多问题，虽然之前有接触过 `python2`，但是和模式识别相关的程序都是以 `python3` 为基础的，因此在编程的过程中出现了混淆两个版本的错误和警告。

在做作业的过程中，其实机器学习相关算法已经有很多完善的程序包可以供我们使用，但是如果针对具体问题，还是要在原有的算法框架基础上进行修改。要想写出性能优越的程序，必须要有强大的理论支撑作为依靠，否则修改出的算法很有可能存在不被觉察的低级错误。我深刻地认识到了自己在模式识别方面仍然存在很多欠缺，还有许多需要努力的方向。在今后的学习时光里，我要继续努力学习模式识别相关的知识，并致力于将其应用在自己的研究方向上——生物电信的模式识别。最后再一次感谢赵老师在学习和编程上给予的帮助。

附录说明

1 模式识别大作业报告

2 python 代码 logistic

3 python 代码 XGBOOST1

4 python 代码 XGBOOST2