

# 数据挖掘大作业三：分类与聚类

徐晶

2120171083

## 一、对数据集进行处理

本次作业选用数据集[<https://www.kaggle.com/c/titanic/data>], 利用 Python 中的 pandas 库进行 csv 数据文件的读取, 对数据集进行预处理, 以适合分类和聚类算法。

1、处理缺失数据;

用 scikit-learn 中的 RandomForest 来拟合一下 'Age', 'Fare', 'Parch', 'SibSp', 'Pclass' 等字段的缺失数据:

```
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
```

按 Cabin 有无数据, 将这个属性处理成 Yes 和 No 两种类型:

```
def set_Cabin_type(df):  
    df.loc[ (df.Cabin.notnull()), 'Cabin' ] = "Yes"  
    df.loc[ (df.Cabin.isnull()), 'Cabin' ] = "No"  
    return df
```

2、将标称数据转化为数值数据;

```
#将标称型数据转化为数值型数据  
df.loc[df['Sex'] == 'male', 'Sex'] = 1  
df.loc[df['Sex'] == 'female', 'Sex'] = 0
```

3、对类目型的特征因子化;

```
#特征因子化
dummies_Cabin = pandas.get_dummies(data_train['Cabin'], prefix= 'Cabin')

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

dummies_Sex = pandas.get_dummies(data_train['Sex'], prefix= 'Sex')

dummies_Pclass = pandas.get_dummies(data_train['Pclass'], prefix= 'Pclass')

df = pandas.concat([data_train, dummies_Cabin, dummies_Embarked, dummies_Sex, dummies_Pclass], axis=1)
```

4、数值数据归一化处理；

```
#数值数据归一化处理
import sklearn.preprocessing as preprocessing
scaler = preprocessing.StandardScaler()
age_scale_param = scaler.fit(df['Age'].values.reshape(-1, 1))
df['Age_scaled'] = scaler.fit_transform(df['Age'].values.reshape(-1, 1), age_scale_param)
fare_scale_param = scaler.fit(df['Fare'].values.reshape(-1, 1))
df['Fare_scaled'] = scaler.fit_transform(df['Fare'].values.reshape(-1, 1), fare_scale_param)
```

## 二、分类算法

使用 Sklearn 中 train\_test\_split 随机划分训练集和测试集，从样本中随机的按比例选取 train data 和 testdata，用于测试分类算法的预测准确率。以下使用决策树和逻辑回归算法进行模型训练和预测。

```
from sklearn.cross_validation import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=33)
```

### 2.1 决策树

(1) 选取数据中用于决策树模型训练的字段，用 Sklearn 中 DecisionTreeClassifier 进行决策树模型的训练：

```
X = data_train[["Pclass", "Sex", "SibSp", "Parch"]]
y = data_train['Survived']
```

```
#决策树
dt=tree.DecisionTreeClassifier()
dt=dt.fit(X_train,y_train)
```

(2) 输出预测准确性和详细的分类性能

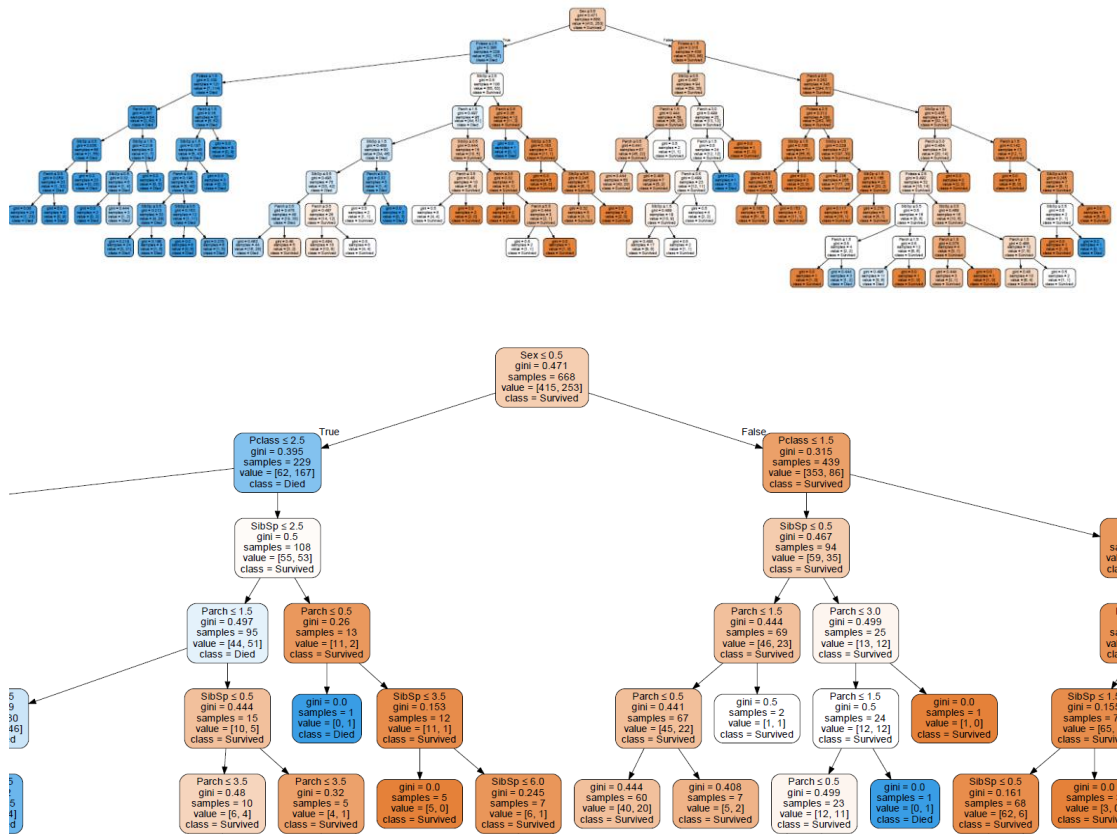
```
# 从sklearn.metrics导入classification_report。
from sklearn.metrics import classification_report
# 输出预测准确性。
print(dt.score(X_test, y_test))
# 输出更加详细的分类性能。
y_predict = dt.predict(X_test)
print(classification_report(y_predict, y_test, target_names = ['died', 'survived']))
```

(3) 预测准确性和详细的性能结果如下

0.8071748878923767					
		precision	recall	f1-score	support
	died	0.91	0.80	0.85	153
	survived	0.65	0.83	0.73	70
	avg / total	0.83	0.81	0.81	223

#### (4) 决策树可视化代码及效果

```
feature_name = ["Pclass", "Sex", "SibSp", "Parch"]
target_name = ["Survived", "Died"]
dot_data = StringIO()
tree.export_graphviz(dt, out_file = dot_data, feature_names=feature_name,
                     class_names=target_name, filled=True, rounded=True,
                     special_characters=True)
graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
graph.write_pdf("Decision_Tree.pdf")
```



## 2.2 逻辑回归

(1) 选取数据中用于逻辑回归模型训练的字段，用 Sklearn 中 LogisticRegression 进行逻辑回归模型的训练：

```
X = df[["Age", "SibSp", "Parch", "Fare", "Pclass", "Sex"]]
y = df['Survived']
```

```
# fit到RandomForestRegressor之中
clf = linear_model.LogisticRegression(C=1.0, penalty='l1', tol=1e-6)
clf.fit(X_train, y_train)
```

(2) 输出预测准确性和详细的分类性能

```
predictions = clf.predict(X_test)
# 从sklearn.metrics导入classification_report。
from sklearn.metrics import classification_report
# 输出预测准确性。
print(clf.score(X_test, y_test))
# 输出更加详细的分类性能。
print(classification_report(predictions, y_test, target_names = ['died', 'survived']))
```

(3) 预测准确性和详细的性能结果如下

0.8161434977578476					
		precision	recall	f1-score	support
	died	0.87	0.83	0.85	141
	survived	0.73	0.79	0.76	82
avg / total		0.82	0.82	0.82	223

## 三、聚类算法

### 3.1 K-means

(1) 选取数据中用于 K-means 聚类的数据字段，用 Sklearn 中 Kmeans 对数据进行 K-means 聚类：

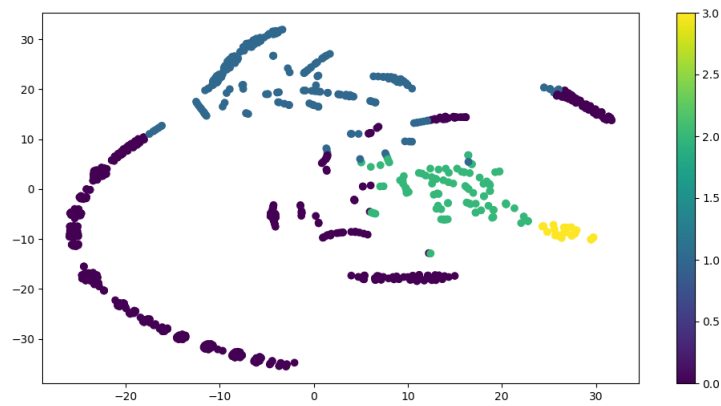
```
X = data_train[["Pclass", "Sex", "SibSp", "Parch", "Age_scaled"]]
y = data_train['Survived']

y_pred = KMeans(n_clusters=4, random_state=0).fit_predict(X_train)
```

(2) 利用 TSNE 进行降维处理并可视化聚类结果

```
iris = chj_load_file(X_train, y_pred)
X_tsne = TSNE(n_components=2, learning_rate=100).fit_transform(iris.data)
plt.figure(figsize=(12, 6))
plt.scatter(X_tsne[:, 0], X_tsne[:, 1], c=iris.target)
plt.colorbar()
plt.show()
```

可视化效果如下



## 3.2 Birch

(1) 选取数据中用于 Birch 聚类的数据字段,用 Sklearn 中 Birch 对数据进行 Birch 聚类:

```
X = data_train[["Pclass", "Sex", "SibSp", "Parch", "Age_scaled"]]
y = data_train['Survived']
```

```
y_Birch = Birch(n_clusters = None).fit_predict(X_train)
```

(2) 利用 TSNE 进行降维处理并可视化聚类结果

```
iris_Birch = chj_load_file(X_train, y_Birch)
X_tsne_Birch = TSNE(n_components=2, learning_rate=100).fit_transform(iris_Birch.data)
plt.figure(figsize=(12, 6))
plt.scatter(X_tsne_Birch[:, 0], X_tsne_Birch[:, 1], c=iris_Birch.target)
plt.colorbar()
plt.show()
```

可视化效果如下

