Titanic 数据集分类与聚类

唐正　2120171060

1. 数据预处理

1.1 处理缺失数据

通过 scikit-learn 库的 RandomForestRegressor 来处理数据集中'Age','Fare', 'Parch', 'SibSp', 'Pclass'等字段的缺失数据：

```
def set_missing_ages(df):
    age_df = df[['Age','Fare', 'Parch', 'SibSp', 'Pclass']]
    known_age = age_df[age_df.Age.notnull()].as_matrix()
    unknown_age = age_df[age_df.Age.isnull()].as_matrix()
    y = known_age[:, 0]
    X = known_age[:, 1:]
    rfr = RandomForestRegressor(random_state=0, n_estimators=2000, n_jobs=-1)
    rfr.fit(X, y)
    predictedAges = rfr.predict(unknown_age[:, 1::])
    df.loc[ (df.Age.isnull()), 'Age' ] = predictedAges
    return df, rfr
```

依据 Cabin 有无将该属性处理成 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
```

1.2 将标称数据转化为数值数据

```
df.loc[df['Sex'] == 'male','Sex'] = 1
df.loc[df['Sex'] == 'female','Sex'] = 0
```

1.3 对类目型的特征因子化

```
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)
```

1.4 数值数据归一化处理

```
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)
```

2. 分类算法

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：

```
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：

```
print(dt.score(X_test, y_test))
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：

```
                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见”决策树.pdf”：

```
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("决策树.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：

```
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：

```
print(clf.score(X_test, y_test))
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：

	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

3. 聚类算法

3.1 K-means

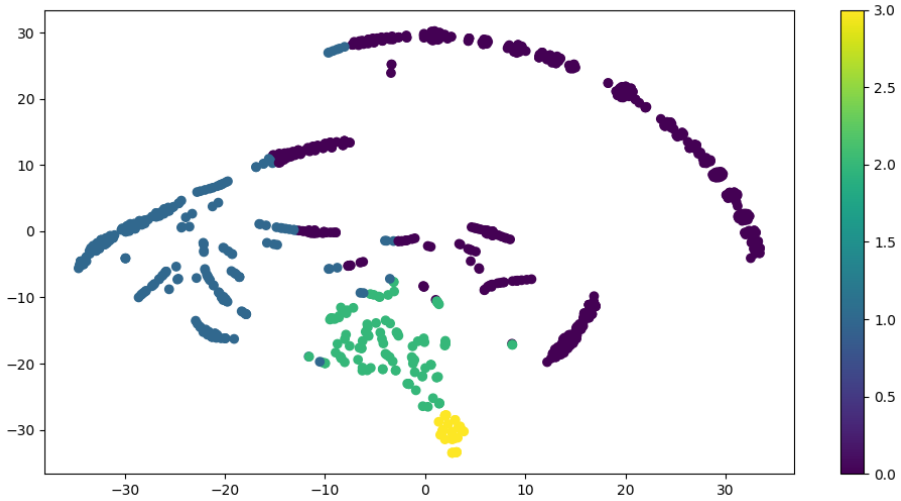
1. 选取数据中用于 K-means 聚类的数据字段，用 Sklearn 中 Kmeans 对数据进行 K-means 聚类：

```
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()
```

3. 可视化效果如下：



3.2 Birch

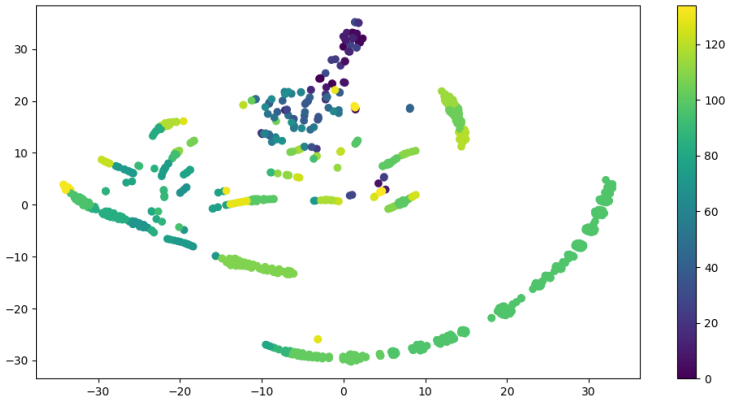
1. 选取数据中用于 Birch 聚类的数据字段，用 Sklearn 中 Birch 对数据进行 Birch 聚类：

```
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()
```

3. 可视化效果如下：


30
20
10
0
−10
−20
−30
−30
−20
−10
0
10
20
30
120
100
80
60
40
20
0