数据挖掘大作业三：分类与聚类

王鹏　　2120171069

一、对数据集进行处理

本次作业选用数据集为泰坦尼克号人员情况，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、读取训练数据集和测试数据集：

```
df_train = pd.read_csv('train.csv')
df_test = pd.merge(pd.read_csv('test.csv'), pd.read_csv('gender_submission.csv'), on='PassengerId')
```

2、处理缺失数据；

用 scikit-learn 中的随机森林算法填充 age 缺失值，

```
def train_rfr(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()

    # 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)

    return rfr

def set_missing_ages(rfr, df):
    # 用得到的模型进行未知年龄结果预测
    age_df = df[['Age', 'Fare', 'Parch', 'SibSp', 'Pclass']]
    unknown_age = age_df[age_df.Age.isnull()].as_matrix()
    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
```

3、按 Cabin 有无数据，将这个属性处理成 Yes 和 No 两种类型，并将“Sex”,”Embarked”属性序列化：

```
#将标称型数据转化为数值型数据
df_train.loc[ (df_train.Cabin.notnull()), 'Cabin' ] = 1
df_train.loc[ (df_train.Cabin.isnull()), 'Cabin' ] = 0
df_train['Embarked'].fillna('ffill', inplace=True)
df_train[['Sex','Embarked']] = df_train[['Sex','Embarked']].apply(lambda X: nominal_to_values(X))

df_test.loc[ (df_test.Cabin.notnull()), 'Cabin' ] = 1
df_test.loc[ (df_test.Cabin.isnull()), 'Cabin' ] = 0
df_test['Embarked'].fillna('ffill', inplace=True)
df_test[['Sex','Embarked']] = df_test[['Sex','Embarked']].apply(lambda X: nominal_to_values(X))
```

二、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：

```
X_train = df_train[["Age", "SibSp", "Parch", "Fare", "Pclass", "Sex", "Cabin", "Embarked"]]
y_train = df_train['Survived']
X_test = df_test[["Age", "SibSp", "Parch", "Fare", "Pclass", "Sex", "Cabin", "Embarked"]]
y_test = df_test['Survived']
# 决策树
dt = tree.DecisionTreeClassifier(max_depth=4)
dt = dt.fit(X_train, y_train)
```

（2）输出预测准确性和详细的分类性能

```
# 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

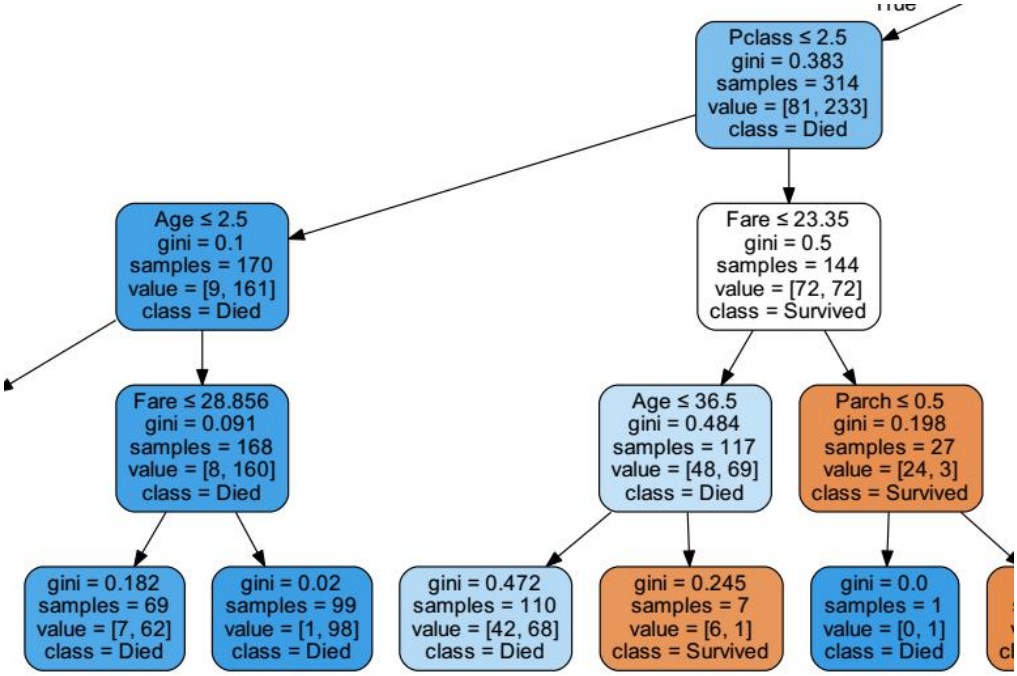
```
0.9593301435406698
             precision    recall  f1-score   support

       died       0.98      0.96      0.97       271
   survived       0.93      0.96      0.94       147

avg / total       0.96      0.96      0.96       418
```

（4）决策树可视化代码及效果

```
feature_name = ["Age", "SibSp", "Parch", "Fare", "Pclass", "Sex", "Cabin", "Embarked"]
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")
```



图保存在 Decision_Tree.pdf 中。

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_train = df_train[["Age", "SibSp", "Parch", "Fare", "Pclass", "Sex", "Cabin", "Embarked"]]
y_train = df_train['Survived']
X_test = df_test[["Age", "SibSp", "Parch", "Fare", "Pclass", "Sex", "Cabin", "Embarked"]]
y_test = df_test['Survived']
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)
# 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

```
0.9354066985645934
             precision    recall  f1-score   support

       died       0.93      0.96      0.95       257
   survived       0.94      0.89      0.91       161

avg / total       0.94      0.94      0.94       418
```

三、聚类算法

在聚类算法的实验中，我将 survived 字段作为训练标签加入到数据集中，并将其权重设置为 10（其余字段权重为 1），期望在聚类过程中聚类结果能表达出其余字段与 survived 字段的关联

3.1 K-means

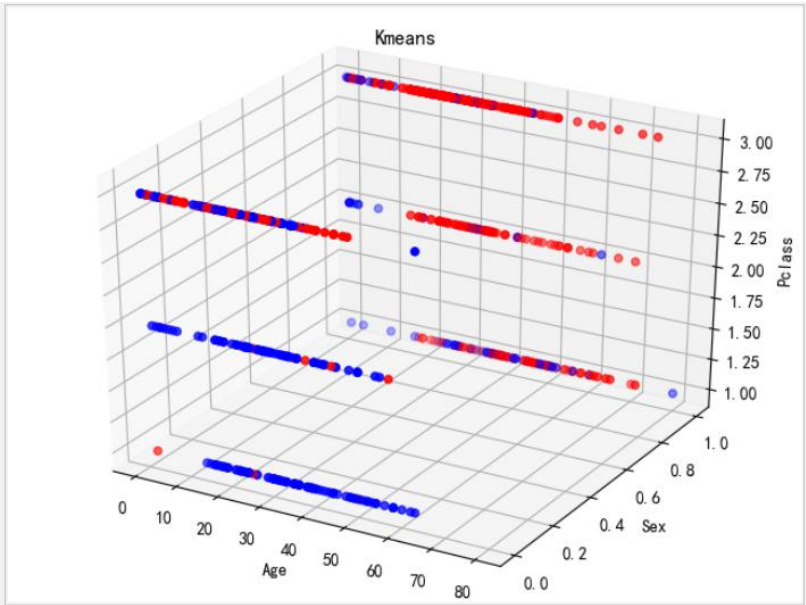
（1）选取数据中用于 K-means 聚类的数据字段，进行归一化并聚类：

```
X_train = df_train[["Age", "SibSp", "Parch", "Fare", "Pclass", "Sex", "Cabin", "Embarked"]]
X_train = X_train.apply(lambda X: preprocessing.scale(X))
X_train['Survived'] = None
X_train['Survived'] = df_train['Survived']*10
clf = KMeans(n_clusters=2)
clf.fit(X_train)
```

（2）利用 Axes3D 进行可视化；

```
def visualize(df, y, name):
    color_value = lambda a: 'r' if a == 1 else 'b'
    color_ori = [color_value(d) for d in y]
    fig = plt.figure()
    ax = Axes3D(fig)
    ax.scatter(df['Age'], df['Sex'], df['Pclass'], c=color_ori, marker='o')
    ax.set_xlabel('Age')
    ax.set_ylabel('Sex')
    ax.set_zlabel('Pclass')
    ax.set_title(name)
    plt.show()
```

可视化效果如下



（3）可视化结果解释

需要先解释一下坐标轴 Sex，0 表示女性，1 表示男性，从中明显可以看出，女性和男性大致上各聚成一类，说明在泰坦尼克沉没时，让女性先走的的意见成为主流。除此以外，我们还可以看到，船舱等级较高的乘客获救概率也比较大，看来有钱人还是率先得到了救助。

在年龄上，年龄与获救的关系不如前面那两个属性这么明显，但也大致可以看出，在 10 岁以下的孩童获救概率较高。此外，值得注意的时，在图上左下角出现了一个 1 号仓的女童未获救，与我们的推论相悖，是值得研究与调查的一个点。

3.2 Birch

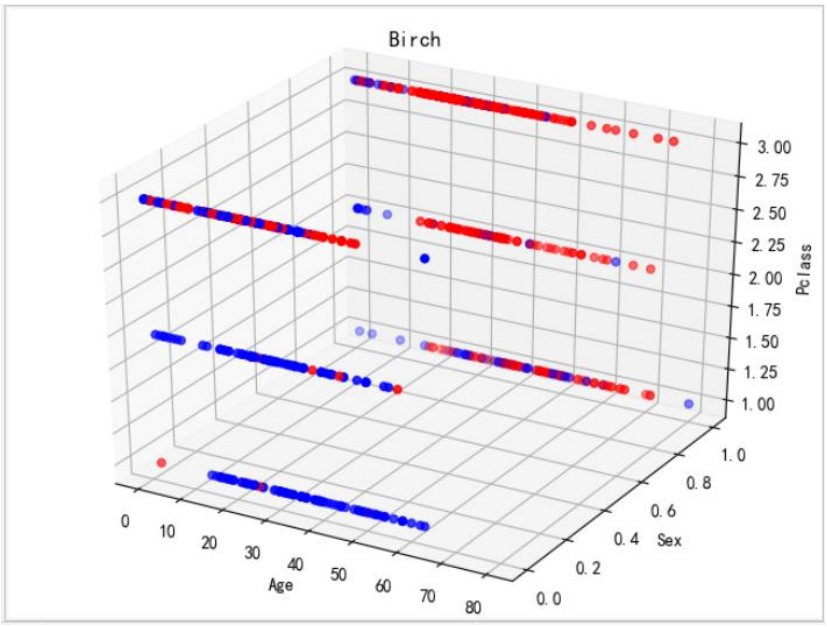
（1）选取数据中用于 K-means 聚类的数据字段，进行归一化并聚类：

```
X_train = df_train[["Age", "SibSp", "Parch", "Fare", "Pclass", "Sex", "Cabin", "Embarked"]]
X_train = X_train.apply(lambda X: preprocessing.scale(X))
X_train['Survived'] = None
X_train['Survived'] = df_train['Survived'] * 10
clf = Birch(n_clusters=2)
clf.fit(X_train)
```

（2）利用 Axes3D 进行可视化；

```
def visualize(df, y, name):
    color_value = lambda a: 'r' if a == 1 else 'b'
    color_ori = [color_value(d) for d in y]
    fig = plt.figure()
    ax = Axes3D(fig)
    ax.scatter(df['Age'], df['Sex'], df['Pclass'], c=color_ori, marker='o')
    ax.set_xlabel('Age')
    ax.set_ylabel('Sex')
    ax.set_zlabel('Pclass')
    ax.set_title(name)
    plt.show()
```

可视化效果如下



（3）利用 Axes3D 进行可视化；

用 Birch 的聚类结果与 Kmeans 差不多，也可以得到上面的解释。