COMP 219 Coursework 1

Yifan Xu

Student Number : 201377026

20, Nov , 2018

Contents

Check list

Step 1.Loading data	implement successfully
Step 2.Training	implement successfully
Step 2.1 Code for training	implement successfully
Step 2.2 Successful training	implement successfully
Step 3 Model Evaluation	implement successfully
Step 3.1 Explain your experimental design	implement successfully
Step 3.2 Document your evaluation results	implement successfully
Extra	implement successfully

Step 1. Loading data

1. Import some libraries

```
#Numpy
import numpy as np

#Datafram operations
import pandas as pd

#Data visualization
from matplotlib import pyplot as plt
import seaborn as sns

#Common Model Helpers
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import StandardScaler

#Models
from sklearn.neighbors import KNeighborsClassifier #KNN
```

2. Import dataset

```
train_df = pd.read_csv('train.csv')
test_df = pd.read_csv('test.csv')
data = train_df.append(test_df)# The entire data = train + test
```

3. Feature Engineering

- Calculate the missing data and drop some columns

```
#Calculate the missing data
data = data.fillna(np.nan)
data.isnull().sum()
```

```
Age             263
Cabin          1014
Embarked          2
Fare              1
Name              0
Parch             0
PassengerId       0
Pclass            0
Sex               0
SibSp             0
Survived        418
Ticket            0
dtype: int64
```

let's drop some columns which don't have many missing values

```
drop_elements = ['Name', 'PassengerId', 'SibSp', 'Parch', 'Ticket', 'Cabin',
                 'Embarked']
train_df.drop(drop_elements, axis = 1, inplace = True)
test_df.drop(drop_elements, axis = 1, inplace = True)
```

- Processing fare, age, title, family size, sex features

Step 2.Training

Step 2.1 Code for training

1. Feature scaling

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(train_df.drop('Survived', axis =1))
```

```
StandardScaler(copy=True, with_mean=True, with_std=True)
```

```
scaled = scaler.transform(train_df.drop('Survived', axis =1))
```

```
df_scaled = pd.DataFrame(scaled, columns = train_df.columns.drop('Survived'))
```

```
X = df_scaled
y = train_df['Survived']
X_test = test_df.copy()
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2)
```

Step 2.2 Successful training

2. Using KNN Model

```
knn = KNeighborsClassifier(n_neighbors=2)
knn.fit(X, y)
```

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=1, n_neighbors=2, p=2,
           weights='uniform')
```

```
y_pred = knn.predict(X_test)
```

Step 3. Model Evaluation

Step 3.1 Explain your experimental design

1. Using confusion matrix

   Directly using sklearn and calculate the TP, TN, FP, TN

```
#Accuracy, classification_report, confusion_matrix
from sklearn.metrics import accuracy_score,classification_report,confusion_matrix
confusion = confusion_matrix(y_test, y_pred)
TP = confusion[1, 1]
TN = confusion[0, 0]
FP = confusion[0, 1]
FN = confusion[1, 0]
print ("TP:", TP)
print ("TN:", TN)
print ("FP:", FP)
print ("FN:", FN)
```

2. Classification Accuracy

```
print ((TP+TN) / float(TP+TN+FN+FP))
accuracy = accuracy_score(y_test,y_pred)
```

- **True positives (TP):** These are cases in which we predicted yes , and the actual is also positive.
- **True negatives (TN):** We predicted no, and the actual is also negative.
- **False positives (FP):** We predicted yes, but the actual is negative(Also known as a "Type I error.")
- **False negatives (FN):** We predicted no, but the actual is positive. (Also known as a "Type II error.")

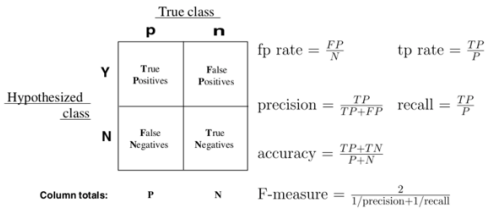


Fig. 1. Confusion matrix and common performance metrics calculated from it.

3. Classification Error

```
print ((FP+FN) / float(TP+TN+FN+FP))
error = 1-accuracy_score(y_test, y_pred)
```

4. Using ROC curve

X– axis: the True Positive-rate ; Y-axis: the False Positive Rate; Threshold = 0.5

In the result we can see that over 80% of cases, the model scores higher on positive samples than on negative samples

```
from sklearn import metrics
def plot_roc_curve(fpr, tpr, label= None):
    threshold = 0.5
    plt.figure(figsize = (12,5))
    plt.plot(fpr, tpr, label = label)
    plt.axis([0, 1, 0, 1])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('ROC Curve')
    plt.show()
```

Step 3.2 Document your evaluation results

1.Confusion matrix result

```
[[106   4]
 [ 27  42]]
```

The left graph is the basic concept of accuracy rate and confusion matrix, the right graph is the result of confusion matrix:

- There are 106 cases that predicted true and actual true,
- There are 4 cases that predicted yes but actual false
- There are 27 cases that predicted false but actual true
- There are 42 cases that predicted false and actual false

2.Accuracy and error results

```
print(confusion)
print("Accuracy: "+ str(accuracy))
print("Error: "+ str(error))
print(classification_report(y_test, y_pred))
```

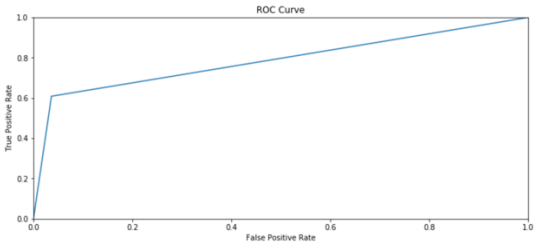
```
[[106   4]
 [ 27  42]]
Accuracy: 0.8268156424581006
Error: 0.17318435754189943
             precision    recall  f1-score   support

          0       0.80      0.96      0.87       110
          1       0.91      0.61      0.73        69

avg / total       0.84      0.83      0.82       179
```

3.ROC curve result

```
fpr, tpr, thresholds = metrics.roc_curve(y_test, y_pred)
plot_roc_curve(fpr, tpr)
plt.show()
print("AUC Score:\n",metrics.roc_auc_score(y_test, y_pred))
```




```
AUC Score:
 0.7861660079051384
```

Extra

Train new models via KNN

```
from sklearn.neighbors import KNeighborsClassifier
clf=KNeighborsClassifier(n_neighbors=5)
clf.fit(X, y)

from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import cross_val_score

k_range = range(1,50)
k_scores = []
for k in k_range:
    knn=KNeighborsClassifier(n_neighbors=k)
    scores = cross_val_score(knn, X, y, cv = 4, scoring='accuracy')
    print("k = " + str(k) + ", score = " + str(scores) + ", mean = " +  str(scores.mean())+"\n"+"Accuracy: "+str(accura
          +", AUC Score:",metrics.roc_auc_score(y_test, y_pred))
    k_scores.append(scores.mean())
```

```
k = 1, score = [0.69196429 0.73542601 0.6981982  0.71621622], mean = 0.7104511772743275
Accuracy: 0.8268156424581006, AUC Score: 0.7861660079051384
k = 2, score = [0.74107143 0.80269058 0.79279279 0.8018018 ], mean = 0.7845891515314161
Accuracy: 0.8268156424581006, AUC Score: 0.7861660079051384
k = 3, score = [0.78571429 0.81165919 0.75225225 0.76126126], mean = 0.7777217480132278
Accuracy: 0.8268156424581006, AUC Score: 0.7861660079051384
k = 4, score = [0.77678571 0.8206278  0.78828829 0.8018018 ], mean = 0.796875901766597
Accuracy: 0.8268156424581006, AUC Score: 0.7861660079051384
k = 5, score = [0.79017857 0.8206278  0.82432432 0.82882883], mean = 0.815989881818077
Accuracy: 0.8268156424581006, AUC Score: 0.7861660079051384
k = 6, score = [0.79910714 0.8206278  0.78828829 0.81531532], mean = 0.8058346372878323
Accuracy: 0.8268156424581006, AUC Score: 0.7861660079051384
k = 7, score = [0.76785714 0.81165919 0.8018018  0.81981982], mean = 0.8002844893259692
Accuracy: 0.8268156424581006, AUC Score: 0.7861660079051384
k = 8, score = [0.76339286 0.81165919 0.79279279 0.81531532], mean = 0.7957900395190193
Accuracy: 0.8268156424581006, AUC Score: 0.7861660079051384
k = 9, score = [0.77232143 0.80717489 0.80630631 0.83333333], mean = 0.8047839890258612
Accuracy: 0.8268156424581006, AUC Score: 0.7861660079051384
k = 10, score = [0.78125    0.81165919 0.78828829 0.84234234], mean = 0.8058849558639357
Accuracy: 0.8268156424581006, AUC Score: 0.7861660079051384
```

Model Compare

```
plt.plot(k_range, k_scores)
plt.xlabel('K for KNN')
plt.ylabel('Cross Validation Accuracy')
plt.show()
```