Python Tutorial 4

February 23, 2020

This tutorial is for Prof. Xin Tong's DSO 530 class at the University of Southern California in spring 2020. It contains two parts: the first part is about HW1 and the second part is to show you how to plot the *ROC curve* and calculate the *AUC*.

1 Explanation about HW1

1.1 Question 3

Some students try to use *normalize()* from *sklearn.preprocessing* instead of using *MinMaxScaler()* as we taught in tutorials. But *normalize()* doesn't work as we thought here.

The function *normalize()* provides a quick and easy way to perform other kinds of **normalization** like *l1 norms* or *l2 norms*:

normalize() has *axis* parameter and its default value = 1, which means that it operates the data according to the rows.

For example:

[1]:
```
from sklearn.preprocessing import normalize
X = [[ 1., -1.,  2.],
     [ 2.,  0.,  0.],
     [ 0.,  1., -1.]]
```

[2]:
```
X_normalized_1 = normalize(X, norm='l1')
X_normalized_1
```

[2]:
```
array([[ 0.25, -0.25,  0.5 ],
       [ 1.  ,  0.  ,  0.  ],
       [ 0.  ,  0.5 , -0.5 ]])
```

When *norm='l1'*, it calculates the *Absolute-value Norm*. In this example: *l1 norm* of the first row is |1| + |-1| + |2| = 4; *l1 norm* of the second row is |2| + |0| + |0| = 2; *l1 norm* of the third row is |0| + |1| + |-1| = 2. The function calculates the values that equal to the original values divided by the *l1 norm* of each row.

Therefore, it returns the array:[[1/4, -1/4, 2/4],[2/2, 0/2, 0/2],[0/2, 1/2, -1/2]]

[3]:
```
X_normalized_2 = normalize(X, norm='l2')
X_normalized_2
```

[3]:
```
array([[ 0.40824829, -0.40824829,  0.81649658],
       [ 1.        ,  0.        ,  0.        ],
       [ 0.        ,  0.70710678, -0.70710678]])
```

When *norm='l2'*, it calculates the *Euclidean Norm*. In this example: *l2 norm* of the first row is $\sqrt{1^2 + (-1)^2 + 2^2} = \sqrt{6}$; *l2 norm* of the second row is $\sqrt{2^2 + 0^2 + 0^2} = 2$; *l2 norm* of the third row is $\sqrt{0^2 + 1^2 + (-1)^2} = \sqrt{2}$. The function calculates the values that equal to the original values divided by the *l1 norm* of each row.

Therefore, it returns the array:[[1/sqrt(6), -1/sqrt(6), 2/sqrt(6)],[2/2, 0/2, 0/2],[0/sqrt(2), 1/sqrt(2), -1/sqrt(2)]]

[4]:
```
X_normalized_3 = normalize(X, norm='max')
X_normalized_3
```

[4]:
```
array([[ 0.5, -0.5,  1. ],
       [ 1. ,  0. ,  0. ],
       [ 0. ,  1. , -1. ]])
```

When *norm='max'*, it calculates the *Maximum Norm*. In this example: *max norm* of the first row is *max(1, -1, 2) = 2*; *max norm* of the second row is *max(2, 0, 0) = 2*; *max norm* of the third row is *max(0, 1, -1) = 1*. The function calculates the values that equal to the original values divided by the *l1 norm* of each row.

Therefore, it returns the array:[[1/2, -1/2, 2/2],[2/2, 0/2, 0/2],[0/1, 1/1, -1/1]]

We can see that it's different from what we taught in class.

Most importantly, with the default axis, this function *normalize()* on the instances as opposed to the variables.

1.2 Question 4

This part aims to address a counterintuitive observation in question 4 of HW1. We will review and solve this question first.

Question 4 in HW1 is as follows:

4. Split the Boston housing data into two parts with 30% as test data. Use random_state = 1 in this split. Because this is a regression problem, you don't want to use stratify = y part of the code from our Python tutorial. Regress medv on LSTAT and RM using the training data. Compute R2 on both the training data and the test data.

We first import the *boston* dataset:

[5]:
```
import pandas as pd
import numpy as np
from sklearn.datasets import load_boston
boston_dataset = load_boston();

boston = pd.DataFrame(boston_dataset.data, columns=boston_dataset.feature_names)
```

```
boston['MEDV'] = boston_dataset.target
```

Then we split the *Boston* housing data into training and test datasets:

[6]:
```
from sklearn.model_selection import train_test_split

X, y = boston[['LSTAT','RM']].values, boston['MEDV'].values

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,␣
 ↪random_state=1)
```

At last we use linear regression to regress *MEDV* on *LSTAT* and *RM* and we calculate the R^2:

[7]:
```
# regress medv on LSTAT and RM

from sklearn.linear_model import LinearRegression

linear_model_4 = LinearRegression()
linear_model_4.fit(X_train, y_train)

r_sq_train = linear_model_4.score(X_train,y_train)
print(f'R-square on training data: {r_sq_train}')

r_sq_test = linear_model_4.score(X_test, y_test)
print(f'R-square on test data: {r_sq_test}')
```

```
R-square on training data: 0.6099162694401526
R-square on test data: 0.6843090583339466
```

Note that here R-square on test data is larger than R-square on training data. Generally, the R-square on test data should be smaller than the R-square on training data. But test data itself involves randomness, and some random seeds might just result in a test data that is somewhat more linearly dependent than the training set. Now, we run the whole process 1000 times and calculate the average R-square on training data and the average R-square on test data. We can see that the average R-square on train data is larger than the average R-square on test data.

[8]:
```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
N = 1000
linear_model = LinearRegression()
r_sq_train = np.zeros(N)
r_sq_test = np.zeros(N)
for i in range(N):
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,␣
 ↪random_state=i)
    linear_model.fit(X_train, y_train)
    r_sq_train[i] = linear_model.score(X_train,y_train)
    r_sq_test[i] = linear_model.score(X_test, y_test)
```

[9]:
```
r_sq_train.mean()
```

[9]: 0.6403754075184912

[10]:
```
r_sq_test.mean()
```

[10]: 0.623810448145141

2 Plot the ROC Curve and Calculate the AUC

This part aims to teach you how to plot the **receiver operating characteristic (ROC) curve** and calculate the **area under the ROC curve (AUC)** using python.

Here, we will use the *smarket* dataset in *Python Tutorial 3* as an example.

[11]:
```
smarket = pd.read_csv('smarket.csv')
smarket['Up'] = np.where(smarket['Direction'] == 'Up', 1, 0)
smarket.head()
```

[11]:

	Year	Lag1	Lag2	Lag3	Lag4	Lag5	Volume	Today	Direction	Up
0	2001	0.381	-0.192	-2.624	-1.055	5.010	1.1913	0.959	Up	1
1	2001	0.959	0.381	-0.192	-2.624	-1.055	1.2965	1.032	Up	1
2	2001	1.032	0.959	0.381	-0.192	-2.624	1.4112	-0.623	Down	0
3	2001	-0.623	1.032	0.959	0.381	-0.192	1.2760	0.614	Up	1
4	2001	0.614	-0.623	1.032	0.959	0.381	1.2057	0.213	Up	1

[12]:
```
X = smarket[['Lag1', 'Lag2', 'Lag3', 'Lag4', 'Lag5','Volume']]
y = smarket['Up']

train_bool = smarket['Year'] < 2005

X_test = X[~train_bool]
y_test = y[~train_bool]
```

[13]:
```
print("X_test.shape: ", X_test.shape)
print("y_test.shape: ", y_test.shape)
```

```
X_test.shape:  (252, 6)
y_test.shape:  (252,)
```

[14]:
```
import statsmodels.formula.api as smf
result = smf.logit('Up ~ Lag1 + Lag2', data=smarket, subset = train_bool).fit()
result.summary()
```

```
Optimization terminated successfully.
         Current function value: 0.692085
         Iterations 3
```

[14]:
```
<class 'statsmodels.iolib.summary.Summary'>
"""
                           Logit Regression Results
==============================================================================
Dep. Variable:                     Up   No. Observations:                  998
Model:                          Logit   Df Residuals:                      995
Method:                           MLE   Df Model:                            2
Date:                Sun, 23 Feb 2020   Pseudo R-squ.:                0.001347
Time:                        20:30:33   Log-Likelihood:                -690.70
converged:                       True   LL-Null:                       -691.63
Covariance Type:            nonrobust   LLR p-value:                    0.3939
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept      0.0322      0.063      0.508      0.611      -0.092       0.156
Lag1          -0.0556      0.052     -1.076      0.282      -0.157       0.046
Lag2          -0.0445      0.052     -0.861      0.389      -0.146       0.057
==============================================================================
"""
```

[15]:
```python
result_prob = result.predict(X_test)
result_prob.head()
```

[15]:
```
998     0.509827
999     0.520824
1000    0.533263
1001    0.526057
1002    0.507210
dtype: float64
```

[16]:
```python
result_prob.max()
```

[16]:
```
0.5423242345233326
```

The code above is exactly the same in *Python Tutorial 3*. So far, we get the probabilities that the market will go up, given values of the predictors using *predict()* function. We can import *roc_curve, auc* from *sklearn.metrics* and use *roc_curve()* to calculate the false positive rate (type I error rate) and true positive rate (1 - false nagetive rate = 1 - type II error rate).

[17]:
```python
from sklearn.metrics import roc_curve
import matplotlib.pyplot as plt

fpr,tpr,threshold = roc_curve(y_test, result_prob)
```

fpr and *tpr* are lists of the false positive rates and the true positive rates, and *threshold* is a list of the corresponding decreasing thresholds on the decision function used to compute fpr and tpr. *threshold[0]* represents no instances being predicted and is arbitrarily set to *max(result_prob) + 1*.

[18]:
```
fpr[:10]
```

[18]:
```
array([0.        , 0.        , 0.03603604, 0.03603604, 0.04504505,
       0.04504505, 0.05405405, 0.05405405, 0.06306306, 0.06306306])
```

[19]:
```
tpr[:10]
```

[19]:
```
array([0.        , 0.0070922 , 0.0070922 , 0.0212766 , 0.0212766 ,
       0.05673759, 0.05673759, 0.06382979, 0.06382979, 0.07092199])
```

[20]:
```
threshold[:10]
```

[20]:
```
array([1.54232423, 0.54232423, 0.53067039, 0.52944358, 0.52939328,
       0.52739314, 0.52684888, 0.52605741, 0.52559797, 0.5255558 ])
```

[21]:
```
fpr.shape
```

[21]:
```
(127,)
```

[22]:
```
tpr.shape
```

[22]:
```
(127,)
```

[23]:
```
threshold.shape
```

[23]:
```
(127,)
```

Then we use *auc()* function to calculate the AUC. The code is as follows:

[24]:
```
from sklearn.metrics import auc
roc_auc = auc(fpr,tpr)
```

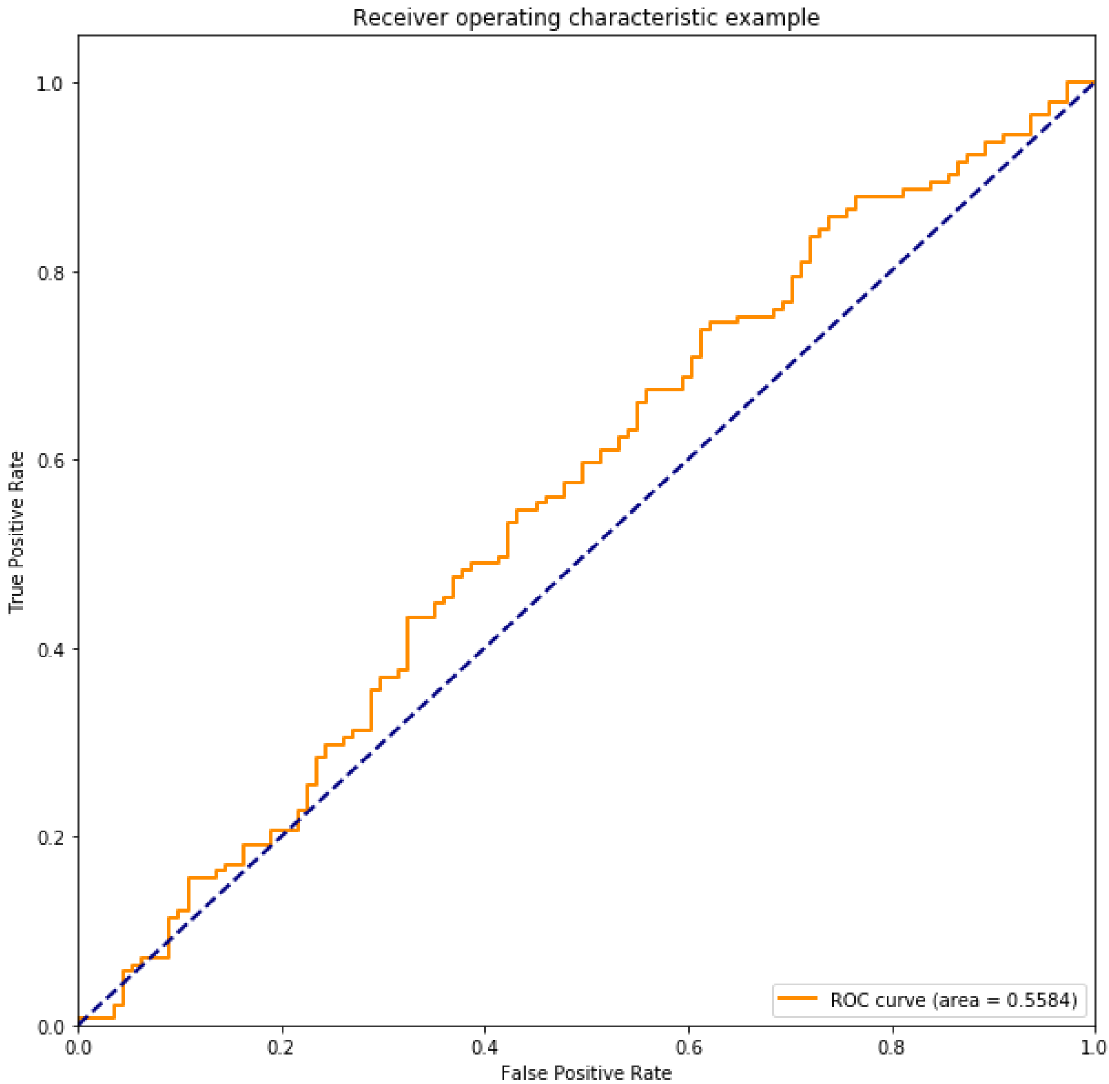
[25]:
```
roc_auc
```

[25]:
```
0.5584307711967287
```

[26]:
```
plt.figure()
plt.figure(figsize=(10,10))
plt.plot(fpr, tpr, color='darkorange',
         lw=2, label='ROC curve (area = {0:.4f})'.format(roc_auc))
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--') # lw is linewidth
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic example')
plt.legend(loc="lower right")
```

```
plt.show()
```

<Figure size 432x288 with 0 Axes>



References:

https://en.wikipedia.org/wiki/Receiver_operating_characteristic

https://scikit-learn.org/stable/modules/model_evaluation.html#roc-metrics

https://matplotlib.org/tutorials/introductory/pyplot.html