Python Tutorial 6

February 18, 2022

This tutorial is for Prof. Xin Tong's DSO 530 class at the University of Southern California in spring 2022. It aims to provide Python code to implement *multi-class logistic regression* and *linear discriminant analysis*. It also guides you to construct an ROC curve.

We use the *Wine* dataset introduced in *Python Tutorial 2*. The Wine dataset is an open-source dataset that is available from the UCI machine learning repository (https://archive.ics.uci.edu/ml/datasets/Wine); it consists of 178 wine observations with 13 features describing their different chemical properties.

Using the pandas library, we will directly read in the open-source Wine dataset from the UCI machine learning repository:

[1]:
```
import pandas as pd
import numpy as np

df_wine = pd.read_csv('https://archive.ics.uci.edu/ml/'␣
↪'machine-learning-databases/wine/wine.data', header=None)

# if the Wine dataset is temporarily unavailable from the
# UCI machine learning repository, un-comment the following line
# of code to load the dataset from a local path:

# df_wine = pd.read_csv('wine.data', header=None)

df_wine.columns = ['Class label', 'Alcohol', 'Malic acid', 'Ash',
                   'Alcalinity of ash', 'Magnesium', 'Total phenols',
                   'Flavanoids', 'Nonflavanoid phenols', 'Proanthocyanins',
                   'Color intensity', 'Hue', 'OD280/OD315 of diluted wines',
                   'Proline']

print('Class labels', np.unique(df_wine['Class label']))
df_wine.head()
```

```
Class labels [1 2 3]
```

[1]:
```
   Class label  Alcohol  Malic acid   Ash  Alcalinity of ash  Magnesium  \
0            1    14.23        1.71  2.43               15.6        127
1            1    13.20        1.78  2.14               11.2        100
2            1    13.16        2.36  2.67               18.6        101
```

```
3            1    14.37        1.95  2.50               16.8         113
4            1    13.24        2.59  2.87               21.0         118

   Total phenols  Flavanoids  Nonflavanoid phenols  Proanthocyanins  \
0           2.80        3.06                  0.28             2.29
1           2.65        2.76                  0.26             1.28
2           2.80        3.24                  0.30             2.81
3           3.85        3.49                  0.24             2.18
4           2.80        2.69                  0.39             1.82

   Color intensity   Hue  OD280/OD315 of diluted wines  Proline
0             5.64  1.04                          3.92     1065
1             4.38  1.05                          3.40     1050
2             5.68  1.03                          3.17     1185
3             7.80  0.86                          3.45     1480
4             4.32  1.04                          2.93      735
```

The 13 different features in the Wine dataset, describing the chemical properties of the 178 wine instances, are listed in the above table.

A bottle of wine in the collection belongs to one of three different classes, 1, 2, and 3, which refer to the three different types of grape grown in the same region in Italy but derived from different wine cultivars, as described in the dataset summary (https://archive.ics.uci.edu/ml/machine-learning-databases/wine/wine.names). It is a multi-class classification problem.

We use the `train_test_split` function from scikit-learn's `model_selection` submodule to partition this dataset into separate training and test datasets:

[2]:
```
from sklearn.model_selection import train_test_split

X, y = df_wine.iloc[:, 1:].values, df_wine.iloc[:, 0].values

X_train, X_test, y_train, y_test =\
    train_test_split(X, y,
                     test_size=0.55,
                     random_state=0,
                     stratify=y)
```

1 Multi-class Logistic Regression

Here we can use `LogisticRegression` function from scikit-learn's `linear_model` submodule to achieve multi-class logistic regression. The default value of `penalty` parameter is 'l2', so we set `penalty='none'` to use the original version of logistic regression. `max_iter` parameter represents the maximum number of iterations taken for the solvers to converge and its default value is 100. `fit` function is the function to fit the model according to the given training data. For multi-class logistic regression, the classes are usually coded by $\{1, \ldots, K\}$, and we assume $P(Y = k|X = x) = \exp(\beta_{0k} + \beta_{1k}x)/[\sum_{i=1}^{K} \exp(\beta_{0i} + \beta_{1i}x)]$. And we classify an instance with feature meaasurement x to class k for which $P(Y = k|X = x)$ is the largest.

[3]:
```python
from sklearn.linear_model import LogisticRegression

mlr = LogisticRegression(penalty='none', max_iter=300).fit(X_train, y_train)
# If we set max_iter=100, it will throw out a warning that it failed to converge
```

We can use the `predict` function to predict the label of the first 5 instances in test data.

[4]:
```python
mlr.predict(X_test[0:5, :])
```

[4]:
```
array([2, 2, 3, 2, 3])
```

And we can use the `predict_proba` function to get the probabilities of the first 5 instances in test data.

[5]:
```python
mlr.predict_proba(X_test[0:5, :])
```

[5]:
```
array([[0., 1., 0.],
       [0., 1., 0.],
       [0., 0., 1.],
       [0., 1., 0.],
       [0., 0., 1.]])
```

[6]:
```python
y_test[0:5]
```

[6]:
```
array([2, 2, 3, 2, 3])
```

And we can use `score` function to evaluate the accuracy on the given test data.

[7]:
```python
mlr.score(X_test, y_test)
```

[7]:
```
0.9693877551020408
```

2 Linear Discriminant Analysis (LDA)

Here we can use `LinearDiscriminantAnalysis` function from scikit-learn's `discriminant_analysis` submodule to achieve (multi-class) linear discriminant analysis. `fit` function is the function to fit the model according to the given training data. For multi-class LDA, the model assumption is that $X|(Y=k) \sim \mathcal{N}(\mu_k, \Sigma)$, for $k = 1, \dots, K$.

[8]:
```python
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis

lda = LinearDiscriminantAnalysis().fit(X_train, y_train)
```

We can use `predict`, `predict_proba`, `score` functions as well.

[9]:
```python
lda.predict(X_test[0:5, :])
```

[9]:
```
array([2, 2, 3, 2, 3])
```

[10]:
```
lda.predict_proba(X_test[0:5, :])
```

[10]:
```
array([[3.33963584e-09, 9.99999990e-01, 7.05470498e-09],
       [3.59845788e-05, 9.99903808e-01, 6.02072480e-05],
       [7.68772943e-08, 5.71137120e-05, 9.99942809e-01],
       [3.73169552e-06, 9.99996236e-01, 3.20868500e-08],
       [7.75250594e-11, 1.44702689e-06, 9.99998553e-01]])
```

[11]:
```
lda.score(X_test, y_test)
```

[11]:
```
0.9591836734693877
```

3 ROC curve

We will use the famous email spam data from the UCI machine learning repository (https://archive.ics.uci.edu/ml/datasets/spambase). For simplicity, one can think of the first 57 columns as engineered features from the original emails, while the last column indicates whether an email is spam (1) or not (0).

[12]:
```
df_spam = pd.read_csv("spambase.data", header = None)
```

[13]:
```
df_spam.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4601 entries, 0 to 4600
Data columns (total 58 columns):
0     4601 non-null float64
1     4601 non-null float64
2     4601 non-null float64
3     4601 non-null float64
4     4601 non-null float64
5     4601 non-null float64
6     4601 non-null float64
7     4601 non-null float64
8     4601 non-null float64
9     4601 non-null float64
10    4601 non-null float64
11    4601 non-null float64
12    4601 non-null float64
13    4601 non-null float64
14    4601 non-null float64
15    4601 non-null float64
16    4601 non-null float64
17    4601 non-null float64
18    4601 non-null float64
19    4601 non-null float64
20    4601 non-null float64
```

```
21    4601 non-null float64
22    4601 non-null float64
23    4601 non-null float64
24    4601 non-null float64
25    4601 non-null float64
26    4601 non-null float64
27    4601 non-null float64
28    4601 non-null float64
29    4601 non-null float64
30    4601 non-null float64
31    4601 non-null float64
32    4601 non-null float64
33    4601 non-null float64
34    4601 non-null float64
35    4601 non-null float64
36    4601 non-null float64
37    4601 non-null float64
38    4601 non-null float64
39    4601 non-null float64
40    4601 non-null float64
41    4601 non-null float64
42    4601 non-null float64
43    4601 non-null float64
44    4601 non-null float64
45    4601 non-null float64
46    4601 non-null float64
47    4601 non-null float64
48    4601 non-null float64
49    4601 non-null float64
50    4601 non-null float64
51    4601 non-null float64
52    4601 non-null float64
53    4601 non-null float64
54    4601 non-null float64
55    4601 non-null int64
56    4601 non-null int64
57    4601 non-null int64
dtypes: float64(55), int64(3)
memory usage: 2.0 MB
```

[14]:
```
np.sum(df_spam.iloc[:,-1]==0)##number of non-spam emails
```

[14]: 2788

[15]:
```
np.sum(df_spam.iloc[:,-1]==1)##number of spam emails
```

[15]: 1813

[16]:
```
df_spam.shape
```

[16]: (4601, 58)

[17]:
```
X, y = df_spam.iloc[:, 0:57].values, df_spam.iloc[:, 57].values

X_train, X_test, y_train, y_test =\
    train_test_split(X, y,
                     test_size=0.2,
                     random_state=20,
                     stratify=y)
```

[18]:
```
mlr_spam = LogisticRegression(penalty='none', max_iter=10000).fit(X_train,␣
 ↪y_train)
```

[19]:
```
mlr_spam.score(X_test, y_test)
```

[19]: 0.9402823018458197

[20]:
```
mlr_spam.predict(X_test[0:5, :])
```

[20]: array([1, 0, 0, 0, 0])

[21]:
```
mlr_spam.predict_proba(X_test[0:5, :])
```

[21]:
```
array([[3.59898858e-01, 6.40101142e-01],
       [1.00000000e+00, 1.14908797e-18],
       [9.30585453e-01, 6.94145467e-02],
       [7.96940069e-01, 2.03059931e-01],
       [1.00000000e+00, 7.48362938e-11]])
```

[22]:
```
mlr_spam_prob = mlr_spam.predict_proba(X_test)
mlr_spam_prob1 = mlr_spam_prob[:, 1]
```

[23]:
```
np.max(mlr_spam_prob1)
```

[23]: 1.0

We can import `roc_curve`, `auc` from `sklearn.metrics`. We use `roc_curve()` to calculate the false positive rate (i.e., type I error rate) and true positive rate (i.e., 1 - false nagetive rate = 1 - type II error rate).

[24]:
```
from sklearn.metrics import roc_curve
import matplotlib.pyplot as plt
fpr,tpr,threshold = roc_curve(y_test, mlr_spam_prob1)
```

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(mlr_spam_prob1) + 1.*

[25]:
```
fpr[:10]
```

[25]:
```
array([0.        , 0.        , 0.        , 0.        , 0.        ,
       0.00179211, 0.00179211, 0.00179211, 0.00179211, 0.00358423])
```

[26]:
```
tpr[:10]
```

[26]:
```
array([0.        , 0.00550964, 0.0523416 , 0.05785124, 0.14325069,
       0.14325069, 0.25619835, 0.26170799, 0.28374656, 0.28374656])
```

[27]:
```
threshold[:10]
```

[27]:
```
array([2.        , 1.        , 0.99999868, 0.99999867, 0.99974999,
       0.99973978, 0.99757902, 0.99753451, 0.99684822, 0.99681789])
```

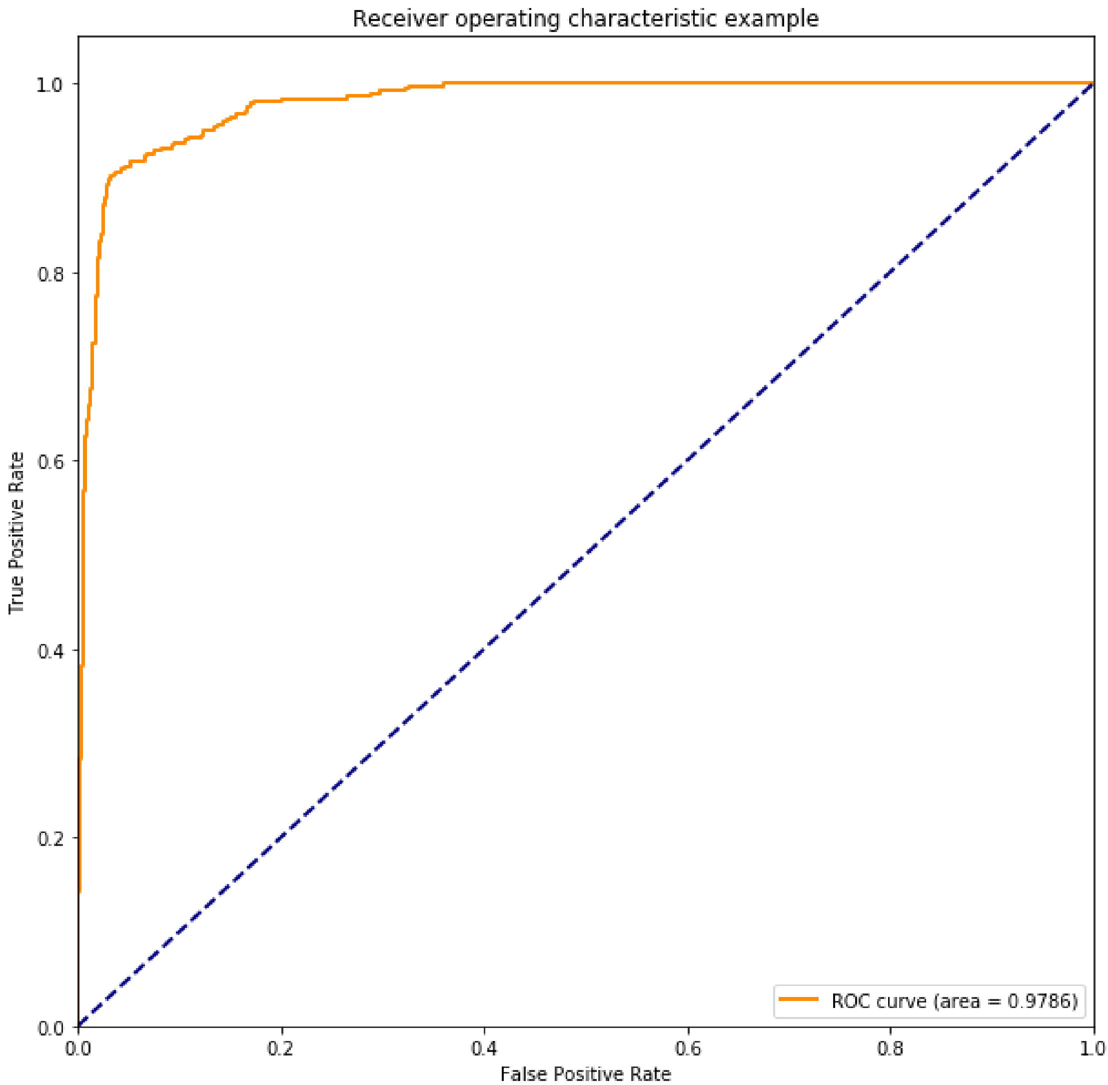
Then we use `auc()` function to calculate the AUC. The code is as follows:

[28]:
```
from sklearn.metrics import auc
roc_auc = auc(fpr,tpr)
roc_auc
```

[28]:
```
0.97861311057792
```

[29]:
```
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>
```


Receiver operating characteristic example
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
ROC curve (area = 0.9786)