Practical 3 Multinomial Logistic Regression (Iris Dataset)

May 7, 2022

Practical 3 Multinomial Logistic Regression (Iris Dataset)

[1]:
```
#Loading the libraries and the data

import numpy as np
import pandas as pd

from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn import preprocessing

from sklearn.metrics import accuracy_score
from sklearn.model_selection import cross_val_score
from sklearn.metrics import confusion_matrix
import matplotlib as mpl
import matplotlib.pyplot as plt


import statsmodels.api as sm

#for readable figures
pd.set_option('float_format', '{:f}'.format)
iris = pd.read_csv("./Iris_Data.csv")
iris.head()
```

[1]:

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.100000	3.500000	1.400000	0.200000	Iris-setosa
1	4.900000	3.000000	1.400000	0.200000	Iris-setosa
2	4.700000	3.200000	1.300000	0.200000	Iris-setosa
3	4.600000	3.100000	1.500000	0.200000	Iris-setosa
4	5.000000	3.600000	1.400000	0.200000	Iris-setosa

Multinomial logistic regression with scikit-learn

First of all we assign the predictors and the criterion to each object and split the datensatz into a training and a test part.

[4]:
```
x = iris.drop('species', axis=1)
y = iris['species']
trainX, testX, trainY, testY = train_test_split(x, y, test_size = 0.2)
```

[5]:
```
#Fit the model
log_reg = LogisticRegression(solver='newton-cg', multi_class='multinomial')
log_reg.fit(trainX, trainY)
y_pred = log_reg.predict(testX)
```

[6]:
```
# Model validation
# print the accuracy and error rate:
print('Accuracy: {:.2f}'.format(accuracy_score(testY, y_pred)))
print('Error rate: {:.2f}'.format(1 - accuracy_score(testY, y_pred)))
```

```
Accuracy: 0.97
Error rate: 0.03
```

[7]:
```
# look at the scores from cross validation:
clf = LogisticRegression(solver='newton-cg', multi_class='multinomial')
scores = cross_val_score(clf, trainX, trainY, cv=5)
scores
```

[7]:
```
array([0.91666667, 0.95833333, 1.        , 0.95833333, 1.        ])
```

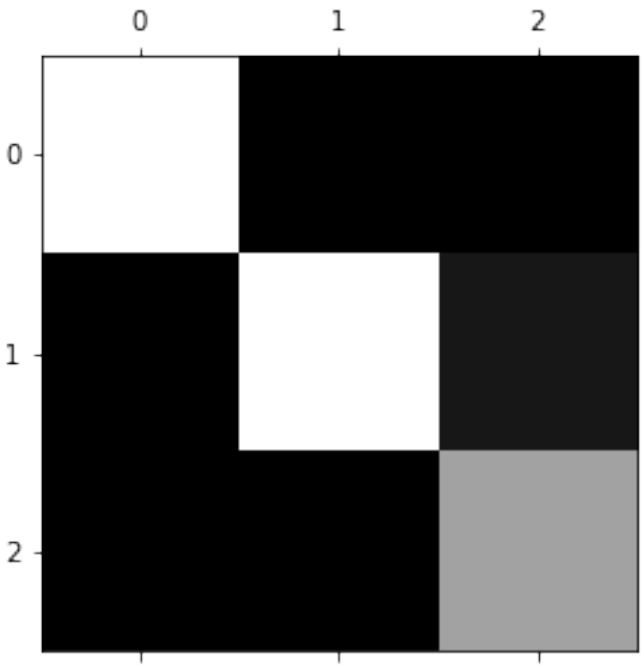
[8]:
```
print("Accuracy: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
```

```
Accuracy: 0.97 (+/- 0.06)
```

[9]:
```
#look at the confusion matrix:
confusion_matrix = confusion_matrix(testY, y_pred)
print(confusion_matrix)
```

```
[[11  0  0]
 [ 0 11  1]
 [ 0  0  7]]
```

[10]:
```
#If you have many variables, it makes sense to plot the confusion matrix:
plt.matshow(confusion_matrix, cmap=plt.cm.gray)
plt.show()
```



[11]:
```
#Calculated probabilities
#get the probabilities of the predicted classes
probability = log_reg.predict_proba(testX)
probability
```

[11]:
```
array([[1.64938718e-01, 8.30797476e-01, 4.26380639e-03],
       [9.65564593e-01, 3.44353123e-02, 9.44168596e-08],
       [3.09577417e-03, 4.25304720e-01, 5.71599506e-01],
       [9.35548860e-01, 6.44508605e-02, 2.79252972e-07],
       [1.41553631e-04, 1.36402116e-01, 8.63456331e-01],
       [9.67998110e-01, 3.20018034e-02, 8.70436747e-08],
       [1.42936187e-03, 3.75814505e-01, 6.22756134e-01],
       [2.30106213e-02, 9.02798839e-01, 7.41905392e-02],
       [2.24599441e-03, 4.29433473e-01, 5.68320533e-01],
       [9.70597746e-01, 2.94021442e-02, 1.09521478e-07],
       [9.70975652e-01, 2.90243105e-02, 3.70783759e-08],
       [6.16361098e-03, 7.92026002e-01, 2.01810387e-01],
       [1.06438523e-03, 5.65753000e-01, 4.33182615e-01],
       [9.80874536e-01, 1.91254377e-02, 2.65125297e-08],
       [4.18397511e-03, 7.39625943e-01, 2.56190081e-01],
       [8.38910568e-05, 6.34226327e-02, 9.36493476e-01],
       [1.03220647e-03, 2.04310768e-01, 7.94657025e-01],
       [9.71896399e-01, 2.81034701e-02, 1.31230843e-07],
       [6.12190643e-03, 8.97949820e-01, 9.59282735e-02],
       [2.45735364e-03, 8.39035579e-01, 1.58507067e-01],
```

```
       [1.19982307e-02, 7.54374159e-01, 2.33627610e-01],
       [9.40712647e-01, 5.92870996e-02, 2.53215628e-07],
       [8.51237074e-02, 9.06089429e-01, 8.78686325e-03],
       [1.81170933e-06, 3.10013813e-02, 9.68996807e-01],
       [9.68708920e-01, 3.12909182e-02, 1.62204034e-07],
       [9.78561762e-01, 2.14381794e-02, 5.84627938e-08],
       [1.97515707e-01, 7.98230337e-01, 4.25395628e-03],
       [9.74923485e-06, 5.62657030e-02, 9.43724548e-01],
       [3.56574881e-03, 8.66698476e-01, 1.29735775e-01],
       [9.61298191e-01, 3.87016155e-02, 1.93201259e-07]])
```

[12]:
```
#Each column here represents a class. The class with the highest probability is␣
↪the output of the predicted class. Here we can see that the length of the␣
↪probability data is the same as the length of the test data.
print(probability.shape[0])
print(testX.shape[0])
```

```
30
30
```

[14]:
```
#output into shape and a readable format
df = pd.DataFrame(log_reg.predict_proba(testX), columns=log_reg.classes_)
df.head()

#with the .classes_ function we get the order of the classes that Python gave.
```

[14]:

	Iris-setosa	Iris-versicolor	Iris-virginica
0	0.164939	0.830797	0.004264
1	0.965565	0.034435	0.000000
2	0.003096	0.425305	0.571600
3	0.935549	0.064451	0.000000
4	0.000142	0.136402	0.863456

[15]:
```
#sum of the probabilities must always be 1
df['sum'] = df.sum(axis=1)
df.head()
```

[15]:

	Iris-setosa	Iris-versicolor	Iris-virginica	sum
0	0.164939	0.830797	0.004264	1.000000
1	0.965565	0.034435	0.000000	1.000000
2	0.003096	0.425305	0.571600	1.000000
3	0.935549	0.064451	0.000000	1.000000
4	0.000142	0.136402	0.863456	1.000000

[16]:
```
# add the predicted classes…
df['predicted_class'] = y_pred
df.head()
```

[16]:
```
   Iris-setosa  Iris-versicolor  Iris-virginica      sum  predicted_class
0     0.164939         0.830797        0.004264 1.000000  Iris-versicolor
1     0.965565         0.034435        0.000000 1.000000      Iris-setosa
2     0.003096         0.425305        0.571600 1.000000   Iris-virginica
3     0.935549         0.064451        0.000000 1.000000      Iris-setosa
4     0.000142         0.136402        0.863456 1.000000   Iris-virginica
```

[17]:
```
#actual classes:
df['actual_class'] = testY.to_frame().reset_index().drop(columns='index')
df.head()
```

[17]:
```
   Iris-setosa  Iris-versicolor  Iris-virginica      sum  predicted_class  \
0     0.164939         0.830797        0.004264 1.000000  Iris-versicolor
1     0.965565         0.034435        0.000000 1.000000      Iris-setosa
2     0.003096         0.425305        0.571600 1.000000   Iris-virginica
3     0.935549         0.064451        0.000000 1.000000      Iris-setosa
4     0.000142         0.136402        0.863456 1.000000   Iris-virginica

      actual_class
0  Iris-versicolor
1      Iris-setosa
2  Iris-versicolor
3      Iris-setosa
4   Iris-virginica
```

[18]:
```
#do a plausibility check whether the classes were predicted correctly.
le = preprocessing.LabelEncoder()

df['label_pred'] = le.fit_transform(df['predicted_class'])
df['label_actual'] = le.fit_transform(df['actual_class'])
df.head()
```

[18]:
```
   Iris-setosa  Iris-versicolor  Iris-virginica      sum  predicted_class  \
0     0.164939         0.830797        0.004264 1.000000  Iris-versicolor
1     0.965565         0.034435        0.000000 1.000000      Iris-setosa
2     0.003096         0.425305        0.571600 1.000000   Iris-virginica
3     0.935549         0.064451        0.000000 1.000000      Iris-setosa
4     0.000142         0.136402        0.863456 1.000000   Iris-virginica

      actual_class  label_pred  label_actual
0  Iris-versicolor           1             1
1      Iris-setosa           0             0
2  Iris-versicolor           2             1
3      Iris-setosa           0             0
4   Iris-virginica           2             2
```

[19]:
```
#see that the two variables (predicted_class & actual_class) were coded the␣
↪same and can therefore be continued properly.
```

[20]:
```
targets = df['predicted_class']
integerEncoded = le.fit_transform(targets)
integerMapping=dict(zip(targets,integerEncoded))
integerMapping
```

[20]:
```
{'Iris-versicolor': 1, 'Iris-setosa': 0, 'Iris-virginica': 2}
```

[21]:
```
targets = df['actual_class']
integerEncoded = le.fit_transform(targets)
integerMapping=dict(zip(targets,integerEncoded))
integerMapping
```

[21]:
```
{'Iris-versicolor': 1, 'Iris-setosa': 0, 'Iris-virginica': 2}
```

[22]:
```
#plausibility check whether the classes were predicted correctly. If the result␣
↪of subtraction is 0, it was a correct estimate of the model.
```

[23]:
```
df['check'] = df['label_actual'] - df['label_pred']
df.head(7)
```

[23]:

	Iris-setosa	Iris-versicolor	Iris-virginica	sum	predicted_class	\
0	0.164939	0.830797	0.004264	1.000000	Iris-versicolor	
1	0.965565	0.034435	0.000000	1.000000	Iris-setosa	
2	0.003096	0.425305	0.571600	1.000000	Iris-virginica	
3	0.935549	0.064451	0.000000	1.000000	Iris-setosa	
4	0.000142	0.136402	0.863456	1.000000	Iris-virginica	
5	0.967998	0.032002	0.000000	1.000000	Iris-setosa	
6	0.001429	0.375815	0.622756	1.000000	Iris-virginica	

	actual_class	label_pred	label_actual	check
0	Iris-versicolor	1	1	0
1	Iris-setosa	0	0	0
2	Iris-versicolor	2	1	-1
3	Iris-setosa	0	0	0
4	Iris-virginica	2	2	0
5	Iris-setosa	0	0	0
6	Iris-virginica	2	2	0

[25]:
```
#For better orientation, we give the observations descriptive names and delete␣
↪unnecessary columns.
df['correct_prediction?'] = np.where(df['check'] == 0, 'True', 'False')
df = df.drop(['label_pred', 'label_actual', 'check'], axis=1)
df.head()
```

[25]:
```
   Iris-setosa  Iris-versicolor  Iris-virginica      sum  predicted_class  \
0     0.164939         0.830797        0.004264 1.000000  Iris-versicolor
1     0.965565         0.034435        0.000000 1.000000      Iris-setosa
2     0.003096         0.425305        0.571600 1.000000   Iris-virginica
3     0.935549         0.064451        0.000000 1.000000      Iris-setosa
4     0.000142         0.136402        0.863456 1.000000   Iris-virginica

      actual_class correct_prediction?
0  Iris-versicolor                True
1      Iris-setosa                True
2  Iris-versicolor               False
3      Iris-setosa                True
4   Iris-virginica                True
```

[29]:
```
#use the generated "values" to manually calculate the accuracy again.
true_predictions = df[(df["correct_prediction?"] == 'True')].shape[0]
false_predictions = df[(df["correct_prediction?"] == 'False')].shape[0]
total = df["correct_prediction?"].shape[0]

print('manual calculated Accuracy is:', (true_predictions / total * 100))
```

```
manual calculated Accuracy is: 96.66666666666667
```

[30]:
```
#take finally a look at the probabilities of the mispredicted classes
wrong_pred = df[(df["correct_prediction?"] == 'False')]
wrong_pred
```

[30]:
```
   Iris-setosa  Iris-versicolor  Iris-virginica      sum predicted_class  \
2     0.003096         0.425305        0.571600 1.000000  Iris-virginica

      actual_class correct_prediction?
2  Iris-versicolor               False
```

Multinomial Logit with the statsmodel library

[31]:
```
#Multinomial Logit with the statsmodel library
#To get the p-values of the model created above we have to use the statsmodel␣
 ↪library again.
x = iris.drop('species', axis=1)
y = iris['species']

x = sm.add_constant(x, prepend = False)

mnlogit_mod = sm.MNLogit(y, x)
mnlogit_fit = mnlogit_mod.fit()

print (mnlogit_fit.summary())
```

```
Warning: Maximum number of iterations has been exceeded.
```

```
         Current function value: 0.039752
         Iterations: 35
                          MNLogit Regression Results
==============================================================================
Dep. Variable:                 species   No. Observations:                  150
Model:                         MNLogit   Df Residuals:                      140
Method:                            MLE   Df Model:                            8
Date:                Sat, 07 May 2022   Pseudo R-squ.:                  0.9638
Time:                        12:43:35   Log-Likelihood:                -5.9627
converged:                      False   LL-Null:                       -164.79
Covariance Type:            nonrobust   LLR p-value:                 7.149e-64
==============================================================================
==========
species=Iris-versicolor       coef    std err          z      P>|z|      [0.025
0.975]
------------------------------------------------------------------------------
-----------
sepal_length                3.4233    349.834      0.010      0.992    -682.239
689.086
sepal_width               -11.3103    207.180     -0.055      0.956    -417.376
394.755
petal_length                7.3579    382.226      0.019      0.985    -741.791
756.507
petal_width                 6.8169    364.361      0.019      0.985    -707.318
720.952
const                      -5.3317   1070.105     -0.005      0.996   -2102.699
2092.035
------------------------------------------------------------------------------
-----------
species=Iris-virginica       coef    std err          z      P>|z|      [0.025
0.975]
------------------------------------------------------------------------------
----------
sepal_length                0.9581    349.842      0.003      0.998    -684.720
686.637
sepal_width               -17.9912    207.229     -0.087      0.931    -424.152
388.169
petal_length               16.7873    382.255      0.044      0.965    -732.419
765.994
petal_width                25.1031    364.492      0.069      0.945    -689.287
739.494
const                     -47.9695   1070.414     -0.045      0.964   -2145.942
2050.003
==============================================================================
==========

c:\users\swadr\appdata\local\programs\python\python38-32\lib\site-
packages\statsmodels\base\model.py:566: ConvergenceWarning: Maximum Likelihood
```

```
optimization failed to converge. Check mle_retvals
  warnings.warn("Maximum Likelihood optimization failed to "
```

We just showed how the Multinomial Logistic Regression can be used to predict multiple classes

[ ]: