hw3

September 29, 2023

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from   sklearn.linear_model import LogisticRegression
from   sklearn.preprocessing import MinMaxScaler, LabelEncoder
from   sklearn.model_selection import train_test_split
from   sklearn.metrics import confusion_matrix, accuracy_score
```

0.1 Problem 1

The file caesarian data.txt contains 80 records; each record lists 5 features used by a doctor to determine whether or not to recommend that a baby be delivered by Caesarian section

- **age**: patient's age in years;
- **num**: number of previous deliveries by patient;
- **tim**: delivery date (0=timely, 1 = premature, 2 = late);
- **pre**: blood pressure (0=low, 1=normal, 2=high);
- **hrt**: heart (0=healthy, 1=unhealthy);
- **cae**: decision (0=normal delivery, 1=use Caesarian);

Use features "age", "num", and "hrt" to perform the following tasks:

Setup

[2]:
```
data = pd.read_csv("./caesarian_data.txt",
                   delimiter=" ",
                   header=None,
                   names=["age", "num", "tim", "pre", "hrt", "cae"])
data.head()
```

[2]:

	age	num	tim	pre	hrt	cae
0	22	1	0	2	0	0
1	26	2	0	1	0	1
2	26	2	1	1	0	0
3	28	1	0	2	0	0
4	22	2	0	1	0	1

[3]:
```
data.describe()
```

[3]:

	age	num	tim	pre	hrt	cae
count	80.000000	80.000000	80.000000	80.000000	80.000000	80.000000
mean	27.687500	1.662500	0.637500	1.000000	0.375000	0.575000
std	5.017927	0.794662	0.815107	0.711568	0.487177	0.497462
min	17.000000	1.000000	0.000000	0.000000	0.000000	0.000000
25%	25.000000	1.000000	0.000000	0.750000	0.000000	0.000000
50%	27.000000	1.000000	0.000000	1.000000	0.000000	1.000000
75%	32.000000	2.000000	1.000000	1.250000	1.000000	1.000000
max	40.000000	4.000000	2.000000	2.000000	1.000000	1.000000

[4]:
```python
# Select the features that the problem specified
features = data[["age", "num", "hrt"]]
features.head(5)
```

[4]:

	age	num	hrt
0	22	1	0
1	26	2	0
2	26	2	0
3	28	1	0
4	22	2	0

(a) Perform the Logistic Regression (note normalize data first). What are the learned model parameters?

[5]:
```python
# Normalizing data using MinMax
scaler = MinMaxScaler().fit(features)
X = scaler.transform(features)
y = data["cae"].to_numpy()

print(f"Example head of features matrix X: \n{X[:5]}\n")
print(f"Example head of labels vector y: \n{y[:5]}")
```

```
Example head of features matrix X:
[[0.2173913  0.         0.        ]
 [0.39130435 0.33333333 0.        ]
 [0.39130435 0.33333333 0.        ]
 [0.47826087 0.         0.        ]
 [0.2173913  0.33333333 0.        ]]

Example head of labels vector y:
[0 1 0 0 1]
```

[6]:
```python
x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪shuffle=True, random_state=42)
model = LogisticRegression()
model.fit(x_train, y_train)
a = model.coef_
b = model.intercept_
```

```python
print("trained model parameters are:")
print(a, b)
```

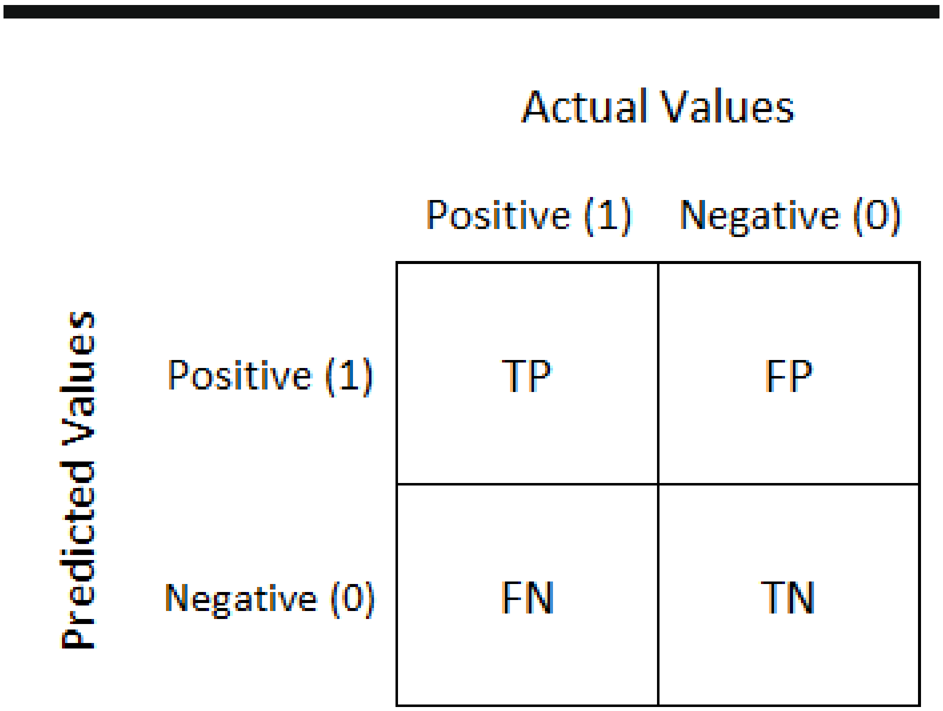
```
trained model parameters are:
[[0.04524739 0.13766913 1.13222396]] [-0.18050479]
```

(b) Evaluate the performance of your Logistic Regression classifier by using the metrics discussed in class.

[7]:
```python
y_predict       = model.predict(x_train)
y_predict_prob  = model.predict_proba(x_train)

print(f"Printing first few:")
print(f"Actual class\tPredicted class\t\tPredicted probabilities")
for n in range(0, 10):
    print(f"{y_train[n]}\t\t{y_predict[n]}\t\t\t{y_predict_prob[n]}")
```

```
Printing first few:
Actual class    Predicted class         Predicted probabilities
0               1                       [0.25483151 0.74516849]
1               0                       [0.54402823 0.45597177]
0               0                       [0.52968426 0.47031574]
1               1                       [0.27460314 0.72539686]
1               0                       [0.54012155 0.45987845]
0               1                       [0.27303843 0.72696157]
0               0                       [0.54061017 0.45938983]
1               1                       [0.26364321 0.73635679]
1               0                       [0.52919415 0.47080585]
1               0                       [0.52625231 0.47374769]
```



[8]:
```
ypred_train        = model.predict(x_train)
accuracy_train     = accuracy_score(y_train, ypred_train)
conf_matrix_train  = confusion_matrix(y_train, ypred_train)
print('Accuracy for training data (R^2): ', accuracy_train, '\n')
print('Confusion matrix for training data:\n', conf_matrix_train, '\n')

ypred_test         = model.predict(x_test)
accuracy_test      = accuracy_score(y_test,ypred_test)
conf_matrix_test   = confusion_matrix(y_test, ypred_test)
print('Accuracy for test data (R^2): ', accuracy_test, '\n')
print('Confusion matrix for test data:\n', conf_matrix_test, '\n')
```

```
Accuracy for training data (R^2):  0.640625

Confusion matrix for training data:
 [[23  5]
 [18 18]]

Accuracy for test data (R^2):  0.6875

Confusion matrix for test data:
 [[5 1]
 [4 6]]
```

pretty bad …

(c) How does the hyperparameter of l2 penalty affect the performance of your classifier?

[9]:
```
model2 = LogisticRegression(penalty="l2", C=0.3)
model2.fit(x_train, y_train)
a = model2.coef_
b = model2.intercept_

print("trained model parameters are:")
print(a, b)
```

```
trained model parameters are:
[[0.04275134 0.08657218 0.73315448]] [-0.04122827]
```

[10]:
```
ypred_train        = model2.predict(x_train)
accuracy_train     = accuracy_score(y_train, ypred_train)
conf_matrix_train  = confusion_matrix(y_train, ypred_train)
print('Accuracy for training data (R^2): ', accuracy_train, '\n')
print('Confusion matrix for training data:\n', conf_matrix_train, '\n')

ypred_test         = model2.predict(x_test)
accuracy_test      = accuracy_score(y_test,ypred_test)
conf_matrix_test   = confusion_matrix(y_test, ypred_test)
print('Accuracy for test data (R^2): ', accuracy_test, '\n')
print('Confusion matrix for test data:\n', conf_matrix_test, '\n')
```

```
Accuracy for training data (R^2):  0.59375

Confusion matrix for training data:
 [[12 16]
 [10 26]]

Accuracy for test data (R^2):  0.75

Confusion matrix for test data:
 [[5 1]
 [3 7]]
```

C value of .9 and above seem to yeild the same results as before. But once I go lower than that, the results start to differ. In this case, I chose a C of 0.3, and it looks like the accracy for testing sample got better but the training data accuracy dropped about 5%.

(d) What is the threshold used in logistic regression? How does the model perform if we set the threshold of estimated probability to 0.6 in part (a) for Caesarian? By deafult, the threshold is 0.5

[11]:
```
threshold = 0.6

ypred_train_new = (model.predict_proba(x_train)[:, 1] >= threshold).astype(int)
```

```
accuracy_train = accuracy_score(y_train, ypred_train_new)
conf_matrix_train  = confusion_matrix(y_train, ypred_train_new)
print('Accuracy for training data (R^2): ', accuracy_train, '\n')
print('Confusion matrix for training data:\n', conf_matrix_train, '\n')


ypred_test_new     = (model.predict_proba(x_test)[:, 1] >= threshold).
 ↪astype(int)
accuracy_test      = accuracy_score(y_test,ypred_test_new)
conf_matrix_test   = confusion_matrix(y_test, ypred_test_new)
print('Accuracy for test data (R^2): ', accuracy_test, '\n')
print('Confusion matrix for test data:\n', conf_matrix_test, '\n')
```

```
Accuracy for training data (R^2):  0.640625

Confusion matrix for training data:
 [[23  5]
 [18 18]]

Accuracy for test data (R^2):  0.6875

Confusion matrix for test data:
 [[5 1]
 [4 6]]
```

In my case of increasing the threshold to 0.6, there is no difference. That is because ALL of those classified as 1 are already at 0.7+ probability. In the following cell, I try to do .73 and we can see now that we get different results.

[12]:
```
threshold = 0.73

ypred_train_new = (model.predict_proba(x_train)[:, 1] >= threshold).astype(int)
accuracy_train = accuracy_score(y_train, ypred_train_new)
conf_matrix_train  = confusion_matrix(y_train, ypred_train_new)
print('Accuracy for training data (R^2): ', accuracy_train, '\n')
print('Confusion matrix for training data:\n', conf_matrix_train, '\n')


ypred_test_new     = (model.predict_proba(x_test)[:, 1] >= threshold).
 ↪astype(int)
accuracy_test      = accuracy_score(y_test,ypred_test_new)
conf_matrix_test   = confusion_matrix(y_test, ypred_test_new)
print('Accuracy for test data (R^2): ', accuracy_test, '\n')
print('Confusion matrix for test data:\n', conf_matrix_test, '\n')
```

```
Accuracy for training data (R^2):  0.546875

Confusion matrix for training data:
```

```
 [[25  3]
 [26 10]]

Accuracy for test data (R^2):  0.625

Confusion matrix for test data:
 [[5 1]
 [5 5]]
```

0.2 Problem 2.

The file iris.csv contains the data for three different species of iris flowers. Use features "petal length" and "petal width" to perform the following tasks

Setup

[13]:
```
iris = pd.read_csv("./iris.csv")
iris
```

[13]:
```
     sepal_length  sepal_width  petal_length  petal_width    species
0             5.1          3.5           1.4          0.2     setosa
1             4.9          3.0           1.4          0.2     setosa
2             4.7          3.2           1.3          0.2     setosa
3             4.6          3.1           1.5          0.2     setosa
4             5.0          3.6           1.4          0.2     setosa
..            …            …             …            …         …
145           6.7          3.0           5.2          2.3  virginica
146           6.3          2.5           5.0          1.9  virginica
147           6.5          3.0           5.2          2.0  virginica
148           6.2          3.4           5.4          2.3  virginica
149           5.9          3.0           5.1          1.8  virginica

[150 rows x 5 columns]
```

[14]:
```
features = iris[["petal_length", "petal_width"]]
scaler = MinMaxScaler().fit(features)
X = scaler.transform(features)
y = iris[["species"]].to_numpy().ravel()
y1 = iris[["species"]].to_numpy()


print(f"Example head of features matrix X: \n{X[:5]}\n")
print(f"Example head of labels vector y  (we need to use this): \n{y[:5]}\n")
print(f"Example head of labels vector y1 (As opposed to this): \n{y1[:5]}")
```

```
Example head of features matrix X:
[[0.06779661 0.04166667]
 [0.06779661 0.04166667]
```

```
 [0.05084746 0.04166667]
 [0.08474576 0.04166667]
 [0.06779661 0.04166667]]

Example head of labels vector y  (we need to use this):
['setosa' 'setosa' 'setosa' 'setosa' 'setosa']

Example head of labels vector y1 (As opposed to this):
[['setosa']
 ['setosa']
 ['setosa']
 ['setosa']
 ['setosa']]
```

[15]:
```
# Encode the target labels
labelEncoder = LabelEncoder()
y_encoded = labelEncoder.fit_transform(y)
y_encoded
```

[15]:
```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2])
```

(a) Use the one-versus-the-rest strategy to classify the iris flowers. Evaluate the performance of your classifier. One-versus-rest (OvR)

[16]:
```
x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪shuffle=True, random_state=42)
ovr_model = LogisticRegression(multi_class='ovr')
ovr_model.fit(x_train, y_train)
a = ovr_model.coef_
b = ovr_model.intercept_

print("trained model parameters are:")
print(f"Model Coef: {a}")
print(f"Model intercept: {b}")
```

```
trained model parameters are:
Model Coef: [[-3.59018237 -3.4457668 ]
 [ 1.1397102  -0.07786512]
 [ 2.7747137   3.40510757]]
Model intercept: [ 1.93300185 -1.16363257 -4.13259285]
```

[17]:
```
ypred_train        = ovr_model.predict(x_train)
accuracy_train     = accuracy_score(y_train, ypred_train)
conf_matrix_train  = confusion_matrix(y_train, ypred_train)
print('Accuracy for training data (R^2): ', accuracy_train, '\n')
print('Confusion matrix for training data:\n', conf_matrix_train, '\n')

ypred_test         = ovr_model.predict(x_test)
accuracy_test      = accuracy_score(y_test,ypred_test)
conf_matrix_test   = confusion_matrix(y_test, ypred_test)
print('Accuracy for test data (R^2): ', accuracy_test, '\n')
print('Confusion matrix for test data:\n', conf_matrix_test, '\n')
```

```
Accuracy for training data (R^2):  0.9

Confusion matrix for training data:
 [[40  0  0]
 [ 1 29 11]
 [ 0  0 39]]

Accuracy for test data (R^2):  0.9

Confusion matrix for test data:
 [[10  0  0]
 [ 0  6  3]
 [ 0  0 11]]
```

[18]:
```
y_predict       = ovr_model.predict(x_train)
y_predict_prob  = ovr_model.predict_proba(x_train)

print(f"Printing first few:")
print(f"features\t\tActual class\tPredicted class\t\tPredicted probabilities")
for n in range(0, 10):
    output =␣
 ↪f"{x_train[n]}\t{y_train[n]}\t\t{y_predict[n]}\t\t\t{y_predict_prob[n]}"
    print(output)
```

```
Printing first few:
features                Actual class    Predicted class         Predicted
probabilities
[0.         0.04166667] setosa          setosa                  [0.77025847
0.21342499 0.01631654]
[0.08474576 0.125     ] setosa          setosa                  [0.7299155
0.24145978 0.02862472]
[0.57627119 0.54166667] versicolor              versicolor
[0.14519655 0.44685389 0.40794956]
[0.10169492 0.04166667] setosa          setosa                  [0.74015615
0.23787211 0.02197173]
```

```
[0.05084746 0.04166667] setosa         setosa                 [0.7559515
0.22513198 0.01891652]
[0.6779661 0.75      ]  virginica           virginica
[0.04335764 0.38632472 0.57031764]
[0.59322034 0.58333333] versicolor          virginica
[0.11712667 0.43691736 0.44595597]
[0.08474576 0.04166667] setosa         setosa                 [0.74559964
0.23350271 0.02089765]
[0.06779661 0.04166667] setosa         setosa                 [0.75086207
0.2292576  0.01988033]
[0.08474576 0.        ] setosa         setosa                 [0.75189615
0.23021479 0.01788905]
```

(b) Use the softmax regression to classify the iris flowers. Evaluate the performance of your classifier. From Docs: For a multi_class problem, if multi_class is set to be "multinomial" the softmax function is used to find the predicted probability of each class.

[19]:
```
x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪shuffle=True, random_state=42)
multinomial_model = LogisticRegression(multi_class='multinomial')
multinomial_model.fit(x_train, y_train)
a = multinomial_model.coef_
b = multinomial_model.intercept_

print("trained model parameters are:")
print(f"Model Coef: {a}")
print(f"Model intercept: {b}")
```

```
trained model parameters are:
Model Coef: [[-3.12343671 -2.97550723]
 [ 0.62100299 -0.13665518]
 [ 2.50243372  3.11216241]]
Model intercept: [ 2.47751014  0.3955824  -2.87309254]
```

[20]:
```
ypred_train        = multinomial_model.predict(x_train)
accuracy_train     = accuracy_score(y_train, ypred_train)
conf_matrix_train  = confusion_matrix(y_train, ypred_train)
print('Accuracy for training data (R^2): ', accuracy_train, '\n')
print('Confusion matrix for training data:\n', conf_matrix_train, '\n')

ypred_test         = multinomial_model.predict(x_test)
accuracy_test      = accuracy_score(y_test,ypred_test)
conf_matrix_test   = confusion_matrix(y_test, ypred_test)
print('Accuracy for test data (R^2): ', accuracy_test, '\n')
print('Confusion matrix for test data:\n', conf_matrix_test, '\n')
```

```
Accuracy for training data (R^2):  0.95
```

```
Confusion matrix for training data:
 [[40  0  0]
 [ 0 38  3]
 [ 0  3 36]]

Accuracy for test data (R^2):  1.0

Confusion matrix for test data:
 [[10  0  0]
 [ 0  9  0]
 [ 0  0 11]]
```

[21]:

```python
y_predict       = multinomial_model.predict(x_train)
y_predict_prob  = multinomial_model.predict_proba(x_train)

print(f"Printing first few:")
print(f"features\t\tActual class\tPredicted class\t\tPredicted probabilities")
for n in range(0, 10):
    output =␣
 ↪f"{x_train[n]}\t{y_train[n]}\t\t{y_predict[n]}\t\t\t{y_predict_prob[n]}"
    print(output)
```

```
Printing first few:
features                Actual class    Predicted class         Predicted
probabilities
[0.         0.04166667] setosa          setosa                  [0.87224895
0.12241693 0.00533413]
[0.08474576 0.125     ] setosa          setosa                  [0.79329443
0.19372674 0.01297882]
[0.57627119 0.54166667] versicolor              versicolor
[0.10747669 0.53962285 0.35290046]
[0.10169492 0.04166667] setosa          setosa                  [0.8222162
0.16887379 0.00891002]
[0.05084746 0.04166667] setosa          setosa                  [0.84895305
0.14413594 0.00691101]
[0.6779661 0.75      ]  virginica               virginica
[0.02860672 0.37972831 0.59166497]
[0.59322034 0.58333333] versicolor              versicolor
[0.08564242 0.51570135 0.39865623]
[0.08474576 0.04166667] setosa          setosa                  [0.83152506
0.16028358 0.00819136]
[0.06779661 0.04166667] setosa          setosa                  [0.84043479
0.15203906 0.00752616]
[0.08474576 0.        ] setosa          setosa                  [0.84824918
0.14526681 0.00648402]
```

(c) Compare and Discuss your results. **One-versus-many (OvR)** is an approach for multi-class regression where, for example, the classification would look like: target and everything else is bundled as not-target

e.g. if we have banana, apple, and orange

we will have: (1) banana vs not banana (2) apple vs not apple (3) orange vs not orange

Softmax (multinomial) is like dividing a pie into n number of categories. Calculating probability looks like dividing the expression of certain features by the expressions of all features.

In my evalution, it seems that Softmax outperforms OvR by 5% on the training data and 10% on the testing data. Which makes sense because Softmax is better suited, in my understanding, to handle multi-class categroization.