Q3

May 1, 2024

The goal of this question is predicting the heart health of patients in a hospital. In the homework package, you can access the data file "HeartData.csv", which consists of 13 features and one response variable (num). The features represent some measurements of the patients' health atributes and num is an indication of the heart health. If num = 0, the heart is healthy, and if num = 1, it reports an issue.

Consider splitting the data into a a training and test set. Samples 1 to 200 form the training set and samples 201 to 297 form the test set. Try the following classification models to predict "num" in terms of the other features in the dataset:

```
- Use logistic regression for your classification. Report the p-values associated with the inte
- Apply LDA and QDA, and again report your model accuracies using the test data.
- Among logistic regression, LDA, and QDA which model(s) seems the most accurate one(s)?
```

[1]:
```
import numpy as np
import pandas as pd
import statsmodels.api as sm
from ISLP.models import (ModelSpec as MS,
                         summarize)
from ISLP import confusion_table
from sklearn.discriminant_analysis import \
     (LinearDiscriminantAnalysis as LDA,
      QuadraticDiscriminantAnalysis as QDA)
```

1 = = = = = Logistic Regression = = = = =

[2]:
```
Data = pd.read_csv('HeartData.csv')
train = (Data.index < 200)
data_train = Data.loc[train]
data_test = Data.loc[~train]
print('Training Data Shape:', data_train.shape)
X = MS(Data.columns.drop(['num'])).fit_transform(Data)
Y = Data['num']
```

```
Training Data Shape: (200, 14)
```

[3]:
```
y_train, X_train = Y.loc[train] , X.loc[train]
y_test, X_test = Y.loc[~train] , X.loc[~train]
```

```
# print(y_test)
LogisticRegressionModel = sm.GLM(y_train, X_train, family=sm.families.
    Binomial()).fit()
LogisticRegressionModel.summary()
```

[3]:

Dep. Variable:	num	**No. Observations:**	200
Model:	GLM	**Df Residuals:**	186
Model Family:	Binomial	**Df Model:**	13
Link Function:	Logit	**Scale:**	1.0000
Method:	IRLS	**Log-Likelihood:**	-63.478
Date:	Wed, 01 May 2024	**Deviance:**	126.96
Time:	20:56:38	**Pearson chi2:**	174.
No. Iterations:	6	**Pseudo R-squ. (CS):**	0.5236
Covariance Type:	nonrobust		

	coef	**std err**	**z**	**P> |z|**	**[0.025**	**0.975]**
intercept	-10.3711	3.716	-2.791	0.005	-17.655	-3.087
age	-0.0073	0.031	-0.236	0.813	-0.068	0.054
sex	1.8161	0.703	2.582	0.010	0.438	3.195
cp	0.9642	0.290	3.324	0.001	0.396	1.533
trestbps	0.0341	0.014	2.432	0.015	0.007	0.062
chol	0.0075	0.005	1.583	0.113	-0.002	0.017
fbs	-1.0563	0.654	-1.616	0.106	-2.337	0.225
restecg	0.4627	0.244	1.894	0.058	-0.016	0.942
thalach	-0.0285	0.014	-1.967	0.049	-0.057	-9.99e-05
exang	0.6358	0.524	1.214	0.225	-0.390	1.662
oldpeak	0.2416	0.260	0.928	0.353	-0.268	0.752
slope	0.5692	0.454	1.252	0.210	-0.322	1.460
ca	0.9591	0.316	3.031	0.002	0.339	1.579
thal	0.3448	0.128	2.689	0.007	0.093	0.596

[4]:

```
probs = LogisticRegressionModel.predict(exog=X_test)
print(probs)
print('=============================================================================')
labels = np.array([0]*y_test.shape[0])
labels[probs>0.5] = 1
# print(labels)
print(confusion_table(labels, y_test))
print('=============================================================================')
print('True rate:', np.mean(labels == y_test), ', False rate:', np.mean(labels !
    = y_test))
```

```
200    0.167715
201    0.214472
202    0.997312
203    0.994625
204    0.955727
         …
```

```
292    0.590421
293    0.106836
294    0.914073
295    0.920404
296    0.027289
Length: 97, dtype: float64
===========================================================================
Truth       0   1
Predicted
0          46  15
1           4  32
===========================================================================
True rate: 0.8041237113402062 , False rate: 0.1958762886597938
```

2 = = = = = Running LDA = = = = =

[5]:
```python
lda = LDA(store_covariance=True)
# Since the LDA estimator automatically adds an intercept, we should remove the␣
↪column corresponding to
# the intercept in both X_train and X_test. We can also directly use the labels␣
↪rather than the Boolean
# vectors y_train.

if 'intercept' in X_train:
    X_train, X_test = [M.drop(columns=['intercept'], axis = 1) for M in␣
↪[X_train, X_test]]
    # print(X_test)
print('===========================================================================')
# print(y_train)
lda.fit(X_train, y_train)
lda_pred = lda.predict(X_test)
print(confusion_table(lda_pred, y_test))
print('True rate:', np.mean(lda_pred == y_test), ', False rate:', np.
↪mean(lda_pred != y_test))
```

```
===========================================================================
Truth       0   1
Predicted
0          46  14
1           4  33
True rate: 0.8144329896907216 , False rate: 0.18556701030927836
```

3 = = = = = Running QDA = = = = =

[6]:
```
qda = QDA(store_covariance=True)
qda.fit(X_train, y_train)
qda_pred = qda.predict(X_test)
print(confusion_table(qda_pred, y_test))
print(np.mean(qda_pred == y_test), np.mean(qda_pred != y_test))
```

```
Truth       0   1
Predicted
0          46  15
1           4  32
0.8041237113402062 0.1958762886597938
```

Among these models, *LDA* seems to be the most accurate model.

***QDA* is less accurate since it might be overfitting.**