Solutions 13

Jumping Rivers

Logistic Regression

Question 1

We're going to the model the gender of a patient using their head size. The following code will set up the X_train and y_train objects for you

```
import pandas as pd
import jrpyml
hd = jrpyml.datasets.load_head_size()
X = hd.drop(columns = "gender")
y_train = hd["gender"]
X_train = hd[["head_size"]]
```

a) The following code will show you a boxplot of head sizes per gender. What does this tell you about the relationship between gender and head size?

```
sns.boxplot(y = "head_size", x = "gender", data = hd)
```

b) Write a pipeline to standardise the predictor then perform logistic regression. Use that to fit the model

```
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
model = Pipeline([
  ('pre', StandardScaler()),
  ('logis', LogisticRegression(class_weight = 'balanced'))
])
model.fit(X_train, y_train)
```

```
## Pipeline(memory=None,
##          steps=[('pre',
##                  StandardScaler(copy=True, with_mean=True, with_std=True)),
##                 ('logis',
##                  LogisticRegression(C=1.0, class_weight='balanced', dual=False,
##                                     fit_intercept=True, intercept_scaling=1,
##                                     l1_ratio=None, max_iter=100,
##                                     multi_class='warn', n_jobs=None,
##                                     penalty='l2', random_state=None,
##                                     solver='warn', tol=0.0001, verbose=0,
##                                     warm_start=False))],
```

```
##           verbose=False)
##
## /filestore/ubuntu/anaconda3/lib/python3.6/site-packages/sklearn/linear_model/logistic.py:432: FutureW
##   FutureWarning)
```

c) If a patient was to have a head size of 3500, what gender would you predict they were? What is the associated probability?

```
import numpy as np
model.predict(np.array(3500, ndmin = 2))
```

```
## array(['Female'], dtype=object)
```

```
model.predict_proba(np.array(3500, ndmin = 2))
```

```
## array([[0.59534253, 0.40465747]])
```

Questions 2

a) For the model in the previous question, what percentage of predictions did you get right in the training data?

```
from sklearn.metrics import accuracy_score, precision_score, recall_score
y_pred = model.predict(X_train)
accuracy_score(y_train,y_pred)
```

```
## 0.70042194092827
```

b) Of those the model classified as Male, what percentage were actually Male?

```
precision_score(y_train,y_pred,pos_label="Male")
```

```
## 0.7603305785123967
```

c) The following code will set up and perform 10-fold cross validation on the data. How does the average estimate of the accuracy on the test set compare to the accuracy in part a) ?

```
from sklearn.model_selection import cross_validate
from sklearn.metrics import make_scorer, accuracy_score, precision_score, recall_score
import pandas as pd

acc = make_scorer(accuracy_score)

def precision(y_true,y_pred):
  return precision_score(y_true,y_pred,pos_label = "Male")
```

```
def recall(y_true,y_pred):
  return recall_score(y_true, y_pred, pos_label = "Male")

prec = make_scorer(precision)
rec = make_scorer(recall)
output = cross_validate(model,X_train,y_train,scoring={
  'acc' : acc,
  'prec' : prec,
  'rec' : rec
}, cv = 10, return_train_score=False)
```

d) Can you improve your model by including extra terms in your model?

Discriminant Analysis

Question 3

a) Fit a Linear Discriminant Analysis to the head size data to predict gender, using only head size as a predictor variable

```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis

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

```
## LinearDiscriminantAnalysis(n_components=None, priors=None, shrinkage=None,
##                            solver='svd', store_covariance=False, tol=0.0001)
```

b) If a patient was to have a head size of 3500, what gender would you predict they were? What is the associated probability? How does this compare to the logistic regression from question 1?

```
model.predict_proba(np.array(3500, ndmin=2))
```

```
## array([[0.53888812, 0.46111188]])
```

c) Fit a LDA to all the variables and obtain some cross validation estimates of accuracy, precision and recall. How do they compare to the full logistic regression model?

```
X_train = hd.drop(columns='gender')

from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline

lda_pipe = Pipeline([
```

```
  ('prep', ColumnTransformer(
    [
      ('num', StandardScaler(), ['head_size','brain_weight','height']),
      ('cat', OneHotEncoder(), ['age_range'])
    ]
  )),
  ('model', LinearDiscriminantAnalysis())
])

lda_pipe.fit(X_train, y_train)
```

```
## Pipeline(memory=None,
##          steps=[('prep',
##                  ColumnTransformer(n_jobs=None, remainder='drop',
##                                    sparse_threshold=0.3,
##                                    transformer_weights=None,
##                                    transformers=[('num',
##                                                   StandardScaler(copy=True,
##                                                                  with_mean=True,
##                                                                  with_std=True),
##                                                   ['head_size', 'brain_weight',
##                                                    'height']),
##                                                  ('cat',
##                                                   OneHotEncoder(categorical_features=None,
##                                                                 categories=None,
##                                                                 drop=None,
##                                                                 dtype=<class 'numpy.float64'>,
##                                                                 handle_unknown='error',
##                                                                 n_values=None,
##                                                                 sparse=True),
##                                                   ['age_range'])],
##                                    verbose=False)),
##                 ('model',
##                  LinearDiscriminantAnalysis(n_components=None, priors=None,
##                                             shrinkage=None, solver='svd',
##                                             store_covariance=False,
##                                             tol=0.0001))],
##          verbose=False)
##
## /filestore/ubuntu/anaconda3/lib/python3.6/site-packages/sklearn/discriminant_analysis.py:388: UserWa
##   warnings.warn("Variables are collinear.")
```

```
results = cross_validate(lda_pipe, X_train, y_train,scoring={
  'acc' : acc,
  'prec' : prec,
```

```
  'rec' : rec
}, cv = 10, return_train_score=False)
```

```
## /filestore/ubuntu/anaconda3/lib/python3.6/site-packages/sklearn/discriminant_analysis.py:388: UserWa
##   warnings.warn("Variables are collinear.")
## /filestore/ubuntu/anaconda3/lib/python3.6/site-packages/sklearn/discriminant_analysis.py:388: UserWa
##   warnings.warn("Variables are collinear.")
## /filestore/ubuntu/anaconda3/lib/python3.6/site-packages/sklearn/discriminant_analysis.py:388: UserWa
##   warnings.warn("Variables are collinear.")
## /filestore/ubuntu/anaconda3/lib/python3.6/site-packages/sklearn/discriminant_analysis.py:388: UserWa
##   warnings.warn("Variables are collinear.")
## /filestore/ubuntu/anaconda3/lib/python3.6/site-packages/sklearn/discriminant_analysis.py:388: UserWa
##   warnings.warn("Variables are collinear.")
## /filestore/ubuntu/anaconda3/lib/python3.6/site-packages/sklearn/discriminant_analysis.py:388: UserWa
##   warnings.warn("Variables are collinear.")
## /filestore/ubuntu/anaconda3/lib/python3.6/site-packages/sklearn/discriminant_analysis.py:388: UserWa
##   warnings.warn("Variables are collinear.")
## /filestore/ubuntu/anaconda3/lib/python3.6/site-packages/sklearn/discriminant_analysis.py:388: UserWa
##   warnings.warn("Variables are collinear.")
## /filestore/ubuntu/anaconda3/lib/python3.6/site-packages/sklearn/discriminant_analysis.py:388: UserWa
##   warnings.warn("Variables are collinear.")
## /filestore/ubuntu/anaconda3/lib/python3.6/site-packages/sklearn/discriminant_analysis.py:388: UserWa
##   warnings.warn("Variables are collinear.")
```

d) How does QDA compare?

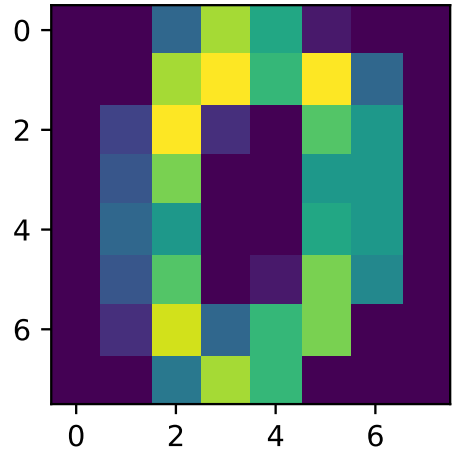
Question 4

The **sklearn** package has some images of hand written digits. It might be interesting to see how well a fairly simple classification model like LDA would do on this. The data can be loaded as below. The plot shows one example image

```
from sklearn.datasets import load_digits

digits = load_digits()
X, y = digits.data, digits.target


import matplotlib.pyplot as plt
plt.imshow(digits.images[0])
```



a) Break your data into a training and testing split

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y)
```

b) Fit a LDA model to predict the numbers in each image using the training set. You should not need any preprocessing for this

```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis

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

```
## LinearDiscriminantAnalysis(n_components=None, priors=None, shrinkage=None,
##                            solver='svd', store_covariance=False, tol=0.0001)
##
## /filestore/ubuntu/anaconda3/lib/python3.6/site-packages/sklearn/discriminant_analysis.py:388: UserWa
##   warnings.warn("Variables are collinear.")
```

c) How well is the model doing?

```
# Surprisingly well

from sklearn.metrics import accuracy_score, confusion_matrix, make_scorer
from sklearn.model_selection import cross_validate

score_fn = make_scorer(accuracy_score)

accuracy_score(y_train, model.predict(X_train))
```

```
## 0.9665924276169265
```

```
confusion_matrix(y_train, model.predict(X_train))
```

```
## array([[134,   0,   0,   0,   0,   0,   1,   0,   0,   0],
##        [  0, 113,   2,   0,   0,   0,   0,   0,   2,   3],
##        [  0,   0, 120,   3,   0,   0,   0,   0,   0,   0],
##        [  0,   0,   1, 137,   0,   1,   0,   0,   3,   0],
##        [  0,   2,   0,   0, 131,   0,   0,   1,   1,   2],
##        [  0,   0,   0,   0,   0, 138,   1,   0,   0,   4],
##        [  0,   0,   0,   0,   0,   0, 135,   0,   0,   1],
##        [  0,   0,   0,   0,   0,   0,   0, 138,   0,   2],
##        [  0,   6,   0,   0,   0,   1,   0,   0, 125,   2],
##        [  0,   0,   0,   2,   0,   1,   0,   2,   1, 131]])
```

```
cross_validate(model, X_train, y_train, scoring=score_fn, cv=10)['test_score'].mean()
```

```
## 0.9500614215200411
##
## /filestore/ubuntu/anaconda3/lib/python3.6/site-packages/sklearn/discriminant_analysis.py:388: UserWa
##   warnings.warn("Variables are collinear.")
## /filestore/ubuntu/anaconda3/lib/python3.6/site-packages/sklearn/discriminant_analysis.py:388: UserWa
##   warnings.warn("Variables are collinear.")
## /filestore/ubuntu/anaconda3/lib/python3.6/site-packages/sklearn/discriminant_analysis.py:388: UserWa
##   warnings.warn("Variables are collinear.")
## /filestore/ubuntu/anaconda3/lib/python3.6/site-packages/sklearn/discriminant_analysis.py:388: UserWa
##   warnings.warn("Variables are collinear.")
## /filestore/ubuntu/anaconda3/lib/python3.6/site-packages/sklearn/discriminant_analysis.py:388: UserWa
##   warnings.warn("Variables are collinear.")
## /filestore/ubuntu/anaconda3/lib/python3.6/site-packages/sklearn/discriminant_analysis.py:388: UserWa
##   warnings.warn("Variables are collinear.")
## /filestore/ubuntu/anaconda3/lib/python3.6/site-packages/sklearn/discriminant_analysis.py:388: UserWa
##   warnings.warn("Variables are collinear.")
## /filestore/ubuntu/anaconda3/lib/python3.6/site-packages/sklearn/discriminant_analysis.py:388: UserWa
##   warnings.warn("Variables are collinear.")
## /filestore/ubuntu/anaconda3/lib/python3.6/site-packages/sklearn/discriminant_analysis.py:388: UserWa
##   warnings.warn("Variables are collinear.")
## /filestore/ubuntu/anaconda3/lib/python3.6/site-packages/sklearn/discriminant_analysis.py:388: UserWa
##   warnings.warn("Variables are collinear.")
```

KNN

a) Fit a KNN to the digits data, make sure to tune the hyper parameter using accuracy of the classifier as the performance metric.

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import GridSearchCV
import numpy as np
```

```
model = KNeighborsClassifier()

search = GridSearchCV(model, {
  'n_neighbors': np.arange(1,51)
  }, cv=10, scoring=score_fn)

search.fit(X_train, y_train)
```

```
## GridSearchCV(cv=10, error_score='raise-deprecating',
##              estimator=KNeighborsClassifier(algorithm='auto', leaf_size=30,
##                                             metric='minkowski',
##                                             metric_params=None, n_jobs=None,
##                                             n_neighbors=5, p=2,
##                                             weights='uniform'),
##              iid='warn', n_jobs=None,
##              param_grid={'n_neighbors': array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14
##        18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34,
##        35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50])},
##              pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
##              scoring=make_scorer(accuracy_score), verbose=0)
##
## /filestore/ubuntu/anaconda3/lib/python3.6/site-packages/sklearn/model_selection/_search.py:813: Depre
##   DeprecationWarning)
```

```
import pandas as pd

pd.DataFrame(search.cv_results_)[['params','mean_test_score']]
```

```
##                  params  mean_test_score
## 0    {'n_neighbors': 1}         0.986637
## 1    {'n_neighbors': 2}         0.981440
## 2    {'n_neighbors': 3}         0.985152
## 3    {'n_neighbors': 4}         0.985895
## 4    {'n_neighbors': 5}         0.985152
## 5    {'n_neighbors': 6}         0.985895
## 6    {'n_neighbors': 7}         0.985152
## 7    {'n_neighbors': 8}         0.980698
## 8    {'n_neighbors': 9}         0.984410
## 9   {'n_neighbors': 10}         0.979955
## 10  {'n_neighbors': 11}         0.979955
## 11  {'n_neighbors': 12}         0.977728
## 12  {'n_neighbors': 13}         0.976986
## 13  {'n_neighbors': 14}         0.976244
## 14  {'n_neighbors': 15}         0.978471
```

```
## 15  {'n_neighbors': 16}        0.976986
## 16  {'n_neighbors': 17}        0.974016
## 17  {'n_neighbors': 18}        0.974016
## 18  {'n_neighbors': 19}        0.974016
## 19  {'n_neighbors': 20}        0.971047
## 20  {'n_neighbors': 21}        0.968820
## 21  {'n_neighbors': 22}        0.968077
## 22  {'n_neighbors': 23}        0.967335
## 23  {'n_neighbors': 24}        0.967335
## 24  {'n_neighbors': 25}        0.966592
## 25  {'n_neighbors': 26}        0.967335
## 26  {'n_neighbors': 27}        0.962880
## 27  {'n_neighbors': 28}        0.963623
## 28  {'n_neighbors': 29}        0.962880
## 29  {'n_neighbors': 30}        0.962138
## 30  {'n_neighbors': 31}        0.959911
## 31  {'n_neighbors': 32}        0.959911
## 32  {'n_neighbors': 33}        0.958426
## 33  {'n_neighbors': 34}        0.956199
## 34  {'n_neighbors': 35}        0.955457
## 35  {'n_neighbors': 36}        0.953972
## 36  {'n_neighbors': 37}        0.951745
## 37  {'n_neighbors': 38}        0.953229
## 38  {'n_neighbors': 39}        0.951745
## 39  {'n_neighbors': 40}        0.951002
## 40  {'n_neighbors': 41}        0.949517
## 41  {'n_neighbors': 42}        0.950260
## 42  {'n_neighbors': 43}        0.946548
## 43  {'n_neighbors': 44}        0.945063
## 44  {'n_neighbors': 45}        0.945063
## 45  {'n_neighbors': 46}        0.942094
## 46  {'n_neighbors': 47}        0.943578
## 47  {'n_neighbors': 48}        0.942094
## 48  {'n_neighbors': 49}        0.942836
## 49  {'n_neighbors': 50}        0.939866
```

b) How does this compare to the LDA result?

```
print("""
We are now doing even better
""")
```

```
##
## We are now doing even better
```

c) Which digit are we struggling with the most?

```
cmat = confusion_matrix(y_train, search.predict(X_train))
(np.diag(cmat)/cmat.sum(axis=1)).argsort()[0]
```

```
## 0
```