Solutions 2

Jumping Rivers

For this practical we will explore models for the prediction of progression of diabetes for 442 patients. Measurements of their age, gender, body mass index, blood pressure and size blood serum measurements were taken to gether with a numeric measurement of disease progression one year after a baseline.

The data are available in the *jrpyml* package and can be accessed with

```
import jrpyml
diabetes = jrpyml.datasets.diabetes.load_data()
```

The data have already been normalised, so we do not need to worry about this. However we should separate the inputs from the output ready for modelling.

```
X, y = diabetes.drop('y', axis=1), diabetes['y']
```

- It is good practice to have a dedicated test set for final assessment of our chosen models. We can create training and test sets from data using `sklearn.model_selection.train_test_split()`. The following code will partition our data with 10% held out for final testing. The other 90% we will use for training and cross validation of different models.

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(
  X, y, test_size=0.1,
  random_state=2019,  # ensures same random subset
)
```

- Begin by fitting a linear regression to the training set using all available predictor variables.

```
from sklearn.linear_model import LinearRegression

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

- Use the `mean_squared_error` function from the *sklearn.metrics* module on the full training set. This will give us the training error.

```
from sklearn.metrics import mean_squared_error
mean_squared_error(y_train, model.predict(X_train))
```

```
## 2798.6714131129825
```

- Training error gives us a measure of how far from the original data our model is. However it is typically different to test error, which would give us a better idea of how our model generalises to new data. Use 10 fold cross validation to estimate the test error rate for this model.

```
from sklearn.model_selection import cross_validate
from sklearn.metrics import make_scorer
score = make_scorer(mean_squared_error)
scores = cross_validate(
  model, X_train, y_train,
  scoring=score, cv=10
)
scores['test_score'].mean()
```

```
## 2939.891936497848
```

- How does this compare to the training error

```
##
##  Test error is larger, this is typically true.
##  training error tends to under estimate test error.
##
```

- Fit a lasso regression model to the diabetes data.

```
from sklearn.linear_model import LassoCV
lasso_model = LassoCV(cv=10)
lasso_model.fit(X_train, y_train)
```

- How do the coefficients of the lasso model compare to those of the standard linear regression?

```
lasso_model.coef_
```

```
## array([   0.        , -150.4344357 ,  550.02447118,  290.81253559,
##         -96.19308788,   -0.        , -180.92291918,    0.        ,
##         547.21048248,   12.60989253])
```

```
model.coef_
```

```
# 3 of the coefficients have been chosen as 0
# for lasso. The remaining coefficients have been
# "shrunk"
```

```
## array([  -4.64533072, -183.27587176,  545.42457602,  307.97143272,
##        -644.25638471,  429.79494741,   19.85978671,   34.65533001,
##         760.80145466,   25.99043256])
```

- Try fitting ridge and elastic net models too

```
from sklearn.linear_model import RidgeCV
from sklearn.linear_model import ElasticNetCV

ridge_model = RidgeCV(cv=10)
enet_model = ElasticNetCV(cv=10)

ridge_model.fit(X_train, y_train)

enet_model.fit(X_train, y_train)
```

- Which model performs best on the test set in terms of mean squared error?

```
mean_squared_error(y_test, model.predict(X_test))
```

```
## 3546.657941607803
```

```
mean_squared_error(y_test, lasso_model.predict(X_test))
```

```
## 3675.1104481698912
```

```
mean_squared_error(y_test, ridge_model.predict(X_test))
```

```
## 3549.0846216338086
```

```
mean_squared_error(y_test, enet_model.predict(X_test))

# Standard linear regression was best here, although ridge was
# close. Perhaps finding better parameters for the ridge
# penalty might yield a better model.
```

```
## 3886.5752154203346
```