A: Simple Linear Regression

Diabetes Dataset - can be loaded from sklearn (mean centred and scaled - read here: http://scikit-learn.org/stable/datasets/index.html). Original data can be found in https://www4.stat.ncsu.edu/~boos/var.select/diabetes.html

For Regression, one feature out of 10 provided is selected, namely the *bmi* (body mass index). The response of interest (y) is a quantitative measure of disease progression one year after baseline, which, hypothetically is linearly dependent on *bmi*.

Tasks

1) Plot *bmi* and *y*
2) Execute provided code and discuss:
   a) the coefficients for the linear model (intercept and slope).
   b) the Coefficient of Determination (R^2).
   c) is the linear model provided a good predictive model, why?

```python
# print(__doc__)
# Code source: Jaques Grobler
# License: BSD 3 clause
# changes made for teaching purpose by Hisham Ihshaish

import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from sklearn import datasets
from sklearn.metrics import mean_squared_error, r2_score

from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression

# Load the diabetes dataset
diabetes = datasets.load_diabetes()
#if you want to display dataset keys uncomment below.
#print (diabetes.keys())

data = pd.DataFrame(diabetes.data, columns=[diabetes.feature_names])
target = pd.DataFrame(diabetes.target)
#print(data.describe())
X = diabetes.data[:, np.newaxis, 2]
y = np.array(target)

X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=42)
lr = LinearRegression().fit(X_train, y_train)

print("lr.coef_: {}".format(lr.coef_))
print("lr.intercept_: {}".format(lr.intercept_))

# Make predictions using the testing set
y_pred = lr.predict(X_test)

print('Coefficient of Determination R Squared: %.2f' % r2_score(y_test, y_pred))

# Plot outputs
plt.scatter(X_test, y_test,  color='black')
plt.plot(X_test, y_pred, color='blue', linewidth=3)
plt.xticks(())
plt.yticks(())
plt.show()
```

B1: Multiple Linear Regression

Boston House Prices Dataset - can be loaded from sklearn (http://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_boston.html). Original data can be explored in https://www.cs.toronto.edu/~delve/data/boston/bostonDetail.html

I extended the number of features, originally 14, from the features themselves - using a method 'PolynomialFeatures' provided sklearn (may be out of scope here, but the main idea is that we want the high-dimensional features' space, ie, number of independent variables is large = 104 produced). The response of interest (y) is an estimate for House Price, which is, supposedly, linearly dependent on the features provided.

Tasks

1) Execute the provided code and discuss
   a) what is the test score metric?
   b) why scores are different on training and on testing datasets?
   c) what's the reason behind the produced difference?

```python
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from sklearn import datasets
from sklearn.preprocessing import MinMaxScaler,PolynomialFeatures
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression

#boston dataset extended: added features using PolynomialFeatures
def load_extended_boston():
    boston = datasets.load_boston()
    X = boston.data

    X = MinMaxScaler().fit_transform(boston.data)
    X = PolynomialFeatures(degree=2, include_bias=False).fit_transform(X)
    return X, boston.target

X, y = load_extended_boston()

X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)
lr = LinearRegression().fit(X_train, y_train)
print("Training set score: {:.2f}".format(lr.score(X_train, y_train)))
print("Test set score: {:.2f}".format(lr.score(X_test, y_test)))
```

B2: Multiple Linear Regression

Use Ridge Regression and tune alpha hyper-parameter.

Tasks

1) Produce a scatterplot matrix for the set of features/attributes - use **pandas.plotting.scatter_matrix** class (check examples and documentation).
2) Can your produce a VIF table for your independent variables — you can use:

```
from statsmodels.stats.outliers_influence import
variance_inflation_factor
```

3) Explore potential multicollinearity between the attributes — how would you design your linear regression model if evident multicollinearity exists.

4) Execute the provided code and discuss
   a) Are scores provided by both regression models different? Explain why?
   b) Tune alpha hyper-parameter and explore prediction scores - explain results.
   c) Produce a residuals plot for each case — you either implement a calculation yourself or use yellowbrick's class — check here.

```python
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from sklearn import datasets
from sklearn.preprocessing import MinMaxScaler,PolynomialFeatures
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Ridge


#boston dataset extended: added features using PolynomialFeatures
def load_extended_boston():
    boston = datasets.load_boston()
    X = boston.data

    X = MinMaxScaler().fit_transform(boston.data)
    X = PolynomialFeatures(degree=2, include_bias=False).fit_transform(X)
    return X, boston.target

X, y = load_extended_boston()

X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)
lr = LinearRegression().fit(X_train, y_train)

ridge = Ridge(alpha=1.0).fit(X_train, y_train)

print("Training set score: {:.2f}".format(lr.score(X_train, y_train)))
print("Test set score: {:.2f}".format(lr.score(X_test, y_test)))
print("Ridge Training set score: {:.2f}".format(ridge.score(X_train, y_train)))
print("Ridge Test set score: {:.2f}".format(ridge.score(X_test, y_test)))

```

C: Lasso Regression

Tasks

>> lasso = Lasso().fit(X_train, y_train)

Apply Lasso Regression and tune its alpha hyper-parameter: compare with the scores you obtained earlier.

B3: Produce Figure in lecture notes - slide no 33

```python
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from sklearn import datasets
from sklearn.preprocessing import MinMaxScaler,PolynomialFeatures
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Ridge


#boston dataset extended: added features using PolynomialFeatures
def load_extended_boston():
    boston = datasets.load_boston()
    X = boston.data

    X = MinMaxScaler().fit_transform(boston.data)
    X = PolynomialFeatures(degree=2, include_bias=False).fit_transform(X)
    return X, boston.target

X, y = load_extended_boston()

X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)
lr = LinearRegression().fit(X_train, y_train)

ridge = Ridge(alpha=1.0).fit(X_train, y_train)
ridge10 = Ridge(alpha=10).fit(X_train, y_train)
ridge01 = Ridge(alpha=0.1).fit(X_train, y_train)
plt.plot(ridge.coef_, 's', label="Ridge alpha=1")
plt.plot(ridge10.coef_, '^', label="Ridge alpha=10")
plt.plot(ridge01.coef_, 'v', label="Ridge alpha=0.1")

plt.plot(lr.coef_, 'o', label="LinearRegression")
plt.xlabel("Coefficient index")
plt.ylabel("Coefficient magnitude")
xlims = plt.xlim()
plt.hlines(0, xlims[0], xlims[1])
plt.xlim(xlims)
plt.ylim(-25, 25)
plt.legend()
plt.show()
```