Linear Regression and SVM

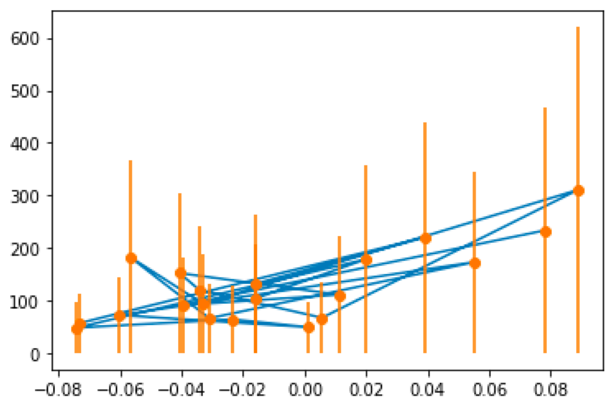
Step 2: Variation in test data size

Plot a graph where x-axis is number of data points used for test and y-axis is the error.

Code:

```
# Plot a graph where x-axis is number of data points used for test and y-axis is the error.

# making a simple plot
x = diabetes_X_test
y = diabetes_y_test

# creating error
y_error = diabetes_y_test

# plotting graph
plt.plot(x, y)

plt.errorbar(x, y,
             yerr = y_error,
             fmt ='o')
```

Output:



Step 3: Regression with more features

Report the coefficients and compare the error with the error of the original one.

y=w0+w1X2+w2X3+w3X4

<u>Original with only 1 feature</u>
Coefficients: [938.23786125]
Mean squared error: 2548.07
Coefficient of determination: 0.47

<u>New with 4 features</u>
Coefficients: [352.82770178]
Mean squared error: 5608.70
Coefficient of determination: -0.16

Code:

```
# Use only one feature
diabetes_X = diabetes_X[:, np.newaxis, 2]
```

```
# Use only four feature
# diabetes_X = diabetes_X[:, np.newaxis, 4]
```
→ y=w0+w1X2+w2X3+w3X4 → [X(i-1) X(i-2) X(i-3) X(i-4)]

Step 4: Polynomial regression

Report the coefficients and compare the error with the error of the original one.

y=w0+w1x+w2x2

<u>Original with only 1 feature</u>
Coefficients: [938.23786125]
Mean squared error: 2548.07
Coefficient of determination: 0.47

<u>New with 3 features</u>
Coefficients: [709.19471785]
Mean squared error: 4058.41
Coefficient of determination: 0.16

Code:

```
# Use only one feature
diabetes_X = diabetes_X[:, np.newaxis, 2]

# Use only 3 feature
# diabetes_X = diabetes_X[:, np.newaxis, 3]
```

Step 6: Synthetic data

Use scatter function in “matplotlib.pyplot” to plot a scatter-plot for the synthetic dataset. Use red and blue colors for +1 and -1 classes, respectively.

Code:

```
# X represents the features and Y is the labels.

import random
import math

N = 200

X = np.empty(shape=(200,2))
Y = np.empty(shape=(200,1))

for i in range(N):
   theta = random.uniform(-3.14,3.14)
   r = random.uniform(0,1)
   X[i][0] = r*math.cos(theta)
   X[i][1] = r*math.sin(theta)
   if r<0.5:
       Y[i] = -1
   else:
       Y[i] = 1
```

Output: **(I had trouble getting the scatterplot to show.)**

Step 7: SVM with Linear Kernel

Coefficients: [938.23786125]
Mean squared error: 2.20
Coefficient of determination: -1.29

Code:

```
# Step 7: SVM with Linear Kernel

from sklearn.svm import SVC
import random
import math

N = 200

X = np.empty(shape=(200,2))
Y = np.empty(shape=(200,1))

# Split the data into training/testing sets
X_train = X[:-160]
X_test = X[-40:]

# Split the targets into training/testing sets
Y_train = Y[:-160]
Y_test = Y[-40:]

for i in range(N):
   theta = random.uniform(-3.14,3.14)
   r = random.uniform(0,1)
   X[i][0] = r*math.cos(theta)
   X[i][1] = r*math.sin(theta)
   if r<0.5:
       Y[i] = -1
   else:
       Y[i] = 1

svc = SVC(kernel='linear')


# The coefficients
print("Coefficients: \n", regr.coef_)
# The mean squared error
print("Mean squared error: %.2f" % mean_squared_error(Y_test, Y_train))
# The coefficient of determination: 1 is perfect prediction
print("Coefficient of determination: %.2f" % r2_score(Y_test, Y_train))
```

Step 8: SVM with RBF Kernel

Coefficients: [938.23786125]
Mean squared error: 1.80
Coefficient of determination: -0.84

Coefficients: [938.23786125]
Mean squared error: 1.80
Coefficient of determination: -0.80

Code:

```
# Step 8: SVM with RBF Kernel

from sklearn.svm import SVC
import random
import math

N = 200

X = np.empty(shape=(200,2))
Y = np.empty(shape=(200,1))

# Split the data into training/testing sets
X_train = X[:-160]
X_test = X[-40:]

# Split the targets into training/testing sets
Y_train = Y[:-160]
Y_test = Y[-40:]

for i in range(N):
   theta = random.uniform(-3.14,3.14)
   r = random.uniform(0,1)
   X[i][0] = r*math.cos(theta)
   X[i][1] = r*math.sin(theta)
   if r<0.5:
       Y[i] = -1
   else:
       Y[i] = 1

svc = SVC(kernel='rbf')


# The coefficients
print("Coefficients: \n", regr.coef_)
# The mean squared error
print("Mean squared error: %.2f" % mean_squared_error(Y_test, Y_train))
# The coefficient of determination: 1 is perfect prediction
print("Coefficient of determination: %.2f" % r2_score(Y_test, Y_train))
```

```
# Step 1: Linear regression in Python

# Code source: Jaques Grobler
# License: BSD 3 clause

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

# Load the diabetes dataset
diabetes_X, diabetes_y = datasets.load_diabetes(return_X_y=True)

# Use only one feature
diabetes_X = diabetes_X[:, np.newaxis,2] # y=w0+w1X2

# Split the data into training/testing sets
diabetes_X_train = diabetes_X[:-20]
diabetes_X_test = diabetes_X[-20:]

# Split the targets into training/testing sets
diabetes_y_train = diabetes_y[:-20]
diabetes_y_test = diabetes_y[-20:]

# Create linear regression object
regr = linear_model.LinearRegression()

# Train the model using the training sets
regr.fit(diabetes_X_train, diabetes_y_train)

# Make predictions using the testing set
diabetes_y_pred = regr.predict(diabetes_X_test)

# The coefficients
print("Coefficients: \n", regr.coef_)
# The mean squared error
print("Mean squared error: %.2f" % mean_squared_error(diabetes_y_test, diabetes_y_pred))
# The coefficient of determination: 1 is perfect prediction
print("Coefficient of determination: %.2f" % r2_score(diabetes_y_test, diabetes_y_pred))

# Plot outputs
plt.scatter(diabetes_X_test, diabetes_y_test, color="black")
plt.plot(diabetes_X_test, diabetes_y_pred, color="blue", linewidth=3)

plt.xticks(())
plt.yticks(())

plt.show()

# ones() to generate an all one array or matrix
# power() to raise all the elements of an array to a power
# concatenate() to create a larger array (in length on in width) using smaller one

# -----------------------------------------------------------------

# Step 2: Variation in test data size

# Change the code in a way that you can test on the last
# 5, 10, 20, 40, and 80 data points. Change the value to any of these.


# # Split the data into training/testing sets
# diabetes_X_train = diabetes_X[:-20]
# diabetes_X_test = diabetes_X[-20:]

# # Split the targets into training/testing sets
# diabetes_y_train = diabetes_y[:-20]
# diabetes_y_test = diabetes_y[-20:]
```

```
# Plot a graph where x-axis is number of data points used for test and y-axis is the error.

# making a simple plot
x = diabetes_X_test
y = diabetes_y_test

# creating error
y_error = diabetes_y_test

# plotting graph
plt.plot(x, y)

plt.errorbar(x, y,
             yerr = y_error,
             fmt ='o')
# ----------------------------------------------------------------

# Step 3: Regression with more features
# y=w0+w1X2+w2X3+w3X4


# Use only four feature
# diabetes_X = diabetes_X[:, np.newaxis, 4]

# ----------------------------------------------------------------

# Step 4: Polynomial Regression
# y=w0+w1x+w2x2


# Use only 3 feature
# diabetes_X = diabetes_X[:, np.newaxis, 3]

# ----------------------------------------------------------------

# Step 5: SVM Classifier

from sklearn.svm import SVC
X = [[0, 0], [1, 1], [0,1]]
y = [0, 1, 0]
clf = SVC()
clf.fit(X, y)
clf.predict([[1, 2]])

# The input parameter "kernel" can be used to change the kernel to one of the followings:

#   ‘linear’, ‘poly’, ‘rbf’, and ‘sigmoid’

# So, if we want to call with 'linear' kernel, we use this function call:
svc = SVC(kernel='linear')

# ----------------------------------------------------------------

# Step 6: Synthetic data

# The following code generates a synthetic dataset.
# X represents the features and Y is the labels.
# Use red and blue colors for +1 and -1 classes, respectively.

import random
import math
import matplotlib.pyplot as plt


N = 200

X = np.empty(shape=(200,2))
Y = np.empty(shape=(200,1))
```

```
for i in range(N):
   theta = random.uniform(-3.14,3.14)
   r = random.uniform(0,1)
   X[i][0] = r*math.cos(theta)
   X[i][1] = r*math.sin(theta)
   if r<0.5:
       Y[i] = -1
   else:
       Y[i] = 1


# Scatterplot
mask = (Y == 1)
plt.scatter(X[mask], Y[mask], c='r')

mask = (Y == -1)
plt.scatter(X[mask], Y[mask], c='b')

# ---------------------------------------------------------------

# Step 7: SVM with Linear Kernel

from sklearn.svm import SVC
import random
import math

N = 200

X = np.empty(shape=(200,2))
Y = np.empty(shape=(200,1))

# Split the data into training/testing sets
X_train = X[:-160]
X_test = X[-40:]

# Split the targets into training/testing sets
Y_train = Y[:-160]
Y_test = Y[-40:]

for i in range(N):
   theta = random.uniform(-3.14,3.14)
   r = random.uniform(0,1)
   X[i][0] = r*math.cos(theta)
   X[i][1] = r*math.sin(theta)
   if r<0.5:
       Y[i] = -1
   else:
       Y[i] = 1

svc = SVC(kernel='linear')


# The coefficients
print("Coefficients: \n", regr.coef_)
# The mean squared error
print("Mean squared error: %.2f" % mean_squared_error(Y_test, Y_train))
# The coefficient of determination: 1 is perfect prediction
print("Coefficient of determination: %.2f" % r2_score(Y_test, Y_train))

# ---------------------------------------------------------------

# Step 8: SVM with RBF Kernel

from sklearn.svm import SVC
import random
import math

N = 200
```

```
X = np.empty(shape=(200,2))
Y = np.empty(shape=(200,1))

# Split the data into training/testing sets
X_train = X[:-160]
X_test = X[-40:]

# Split the targets into training/testing sets
Y_train = Y[:-160]
Y_test = Y[-40:]

for i in range(N):
   theta = random.uniform(-3.14,3.14)
   r = random.uniform(0,1)
   X[i][0] = r*math.cos(theta)
   X[i][1] = r*math.sin(theta)
   if r<0.5:
       Y[i] = -1
   else:
       Y[i] = 1

svc = SVC(kernel='rbf')


# The coefficients
print("Coefficients: \n", regr.coef_)
# The mean squared error
print("Mean squared error: %.2f" % mean_squared_error(Y_test, Y_train))
# The coefficient of determination: 1 is perfect prediction
print("Coefficient of determination: %.2f" % r2_score(Y_test, Y_train))
```