hw6_arkam2

March 15, 2021

1 STAT 542: Homework 6

1.1 Name- Arka Mitra (netid - arkam2)

1.1.1 Spring 2021, by Ruoqing Zhu (rqzhu)

1.1.2 Due: Tuesday, Mar 16, 11:59 PM CT

2 Question 1 [100 Points] Write Your Own Spline Basis

We will fit and compare different spline models to the `Ozone` dataset form the `mlbench` package. The dataset is already ordered by date, and we will use this index as the x variable, named as `time`.

```
library(mlbench)
data(Ozone)

# Wind will only be used for Q2
mydata = data.frame("time" = seq(1:nrow(Ozone))/nrow(Ozone), "ozone" = Ozone$V4, "wind" = Ozon

trainid = sample(1:nrow(Ozone), 250)
train = mydata[trainid, ]
test = mydata[-trainid, ]
par(mfrow=c(1,2))

plot(train$time, train$ozone, pch = 19, cex = 0.5)
plot(train$wind, train$ozone, pch = 19, cex = 0.5)
```

2.1 a. [80 points] Univariate Spline Fit

Let's consider several different spline methods to model the `ozone` level using `time`. To test your model, use the train/test split provided above. If you use Python, please generate your split with the same mechanism and save your seed. Use the mean squared error as the metric for evaluation and report it for each method. For the basis that you write with your own code, make sure to include the intercept term. For each method, produce a figure consists of training data, testing data and your fitted curve.

i. Write your own code (you cannot use `bs()` or similar functions) to implement a continuous piecewise linear fitting. Pick 3 knots using your own judgment.

ii. Write your own code to implement a quadratic spline fitting. Your spline should be continuous up to the first derivative. Pick 4 knots using your own judgment.

iii. Produce a same set of basis as (ii) using the `bs()` function. Note that they do not have to be exactly the same as yours. Verify (figure out how) that the column spaces are the same.

Use existing functions (e.g. `ns()`)to implement a natural cubic spline with 6 knots. Choose your own knots.

Use existing functions to implement a smoothing spline. Use the built-in generalized cross-validation method to select the best tuning parameter.

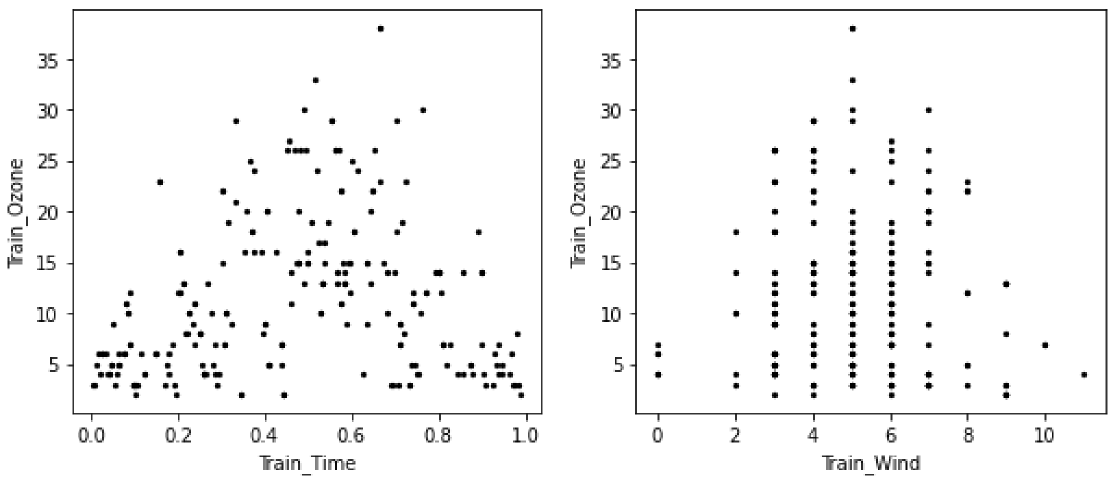
3 Solution

Since the `mlbench` library does not exist in Python, we ran the following code to write out the necessary dataset into a .csv file. The rest of the analysis is done on the `oz.csv` file (attached with submission).

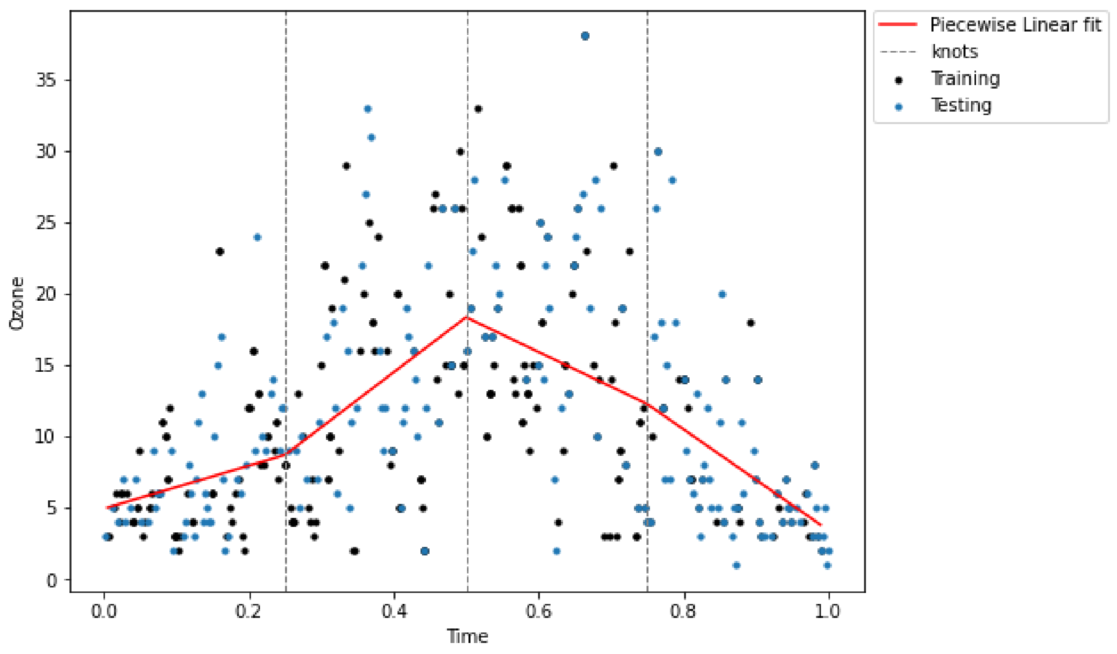
```
library(mlbench)
data(Ozone)
mydata = data.frame("time" = seq(1:nrow(Ozone))/nrow(Ozone), "ozone" = Ozone$V4, "wind" = Ozone$
write.csv(mydata, "oz.csv")
```

To check the veracity of the data from R, identical figures to the ones provided are recreated below.

[2]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

mydata = pd.read_csv("oz.csv").dropna()
np.random.seed(5)
train_id = np.random.randint(0, len(mydata), 250)
train_df, test_df = mydata.iloc[train_id] , mydata[~mydata.index.isin(train_id)]
train_df = train_df.sort_values('time')

fig, axes = plt.subplots(1,2,figsize = (10,4))
ax1, ax2 = axes[0], axes[1]
ax1.scatter(train_df['time'], train_df['ozone'], s=5, color='k')
ax1.set_xlabel('Train_Time')
ax1.set_ylabel('Train_Ozone')
ax2.scatter(train_df['wind'], train_df['ozone'], s=5, color='k')
ax2.set_xlabel('Train_Wind')
ax2.set_ylabel('Train_Ozone')
plt.show()
```

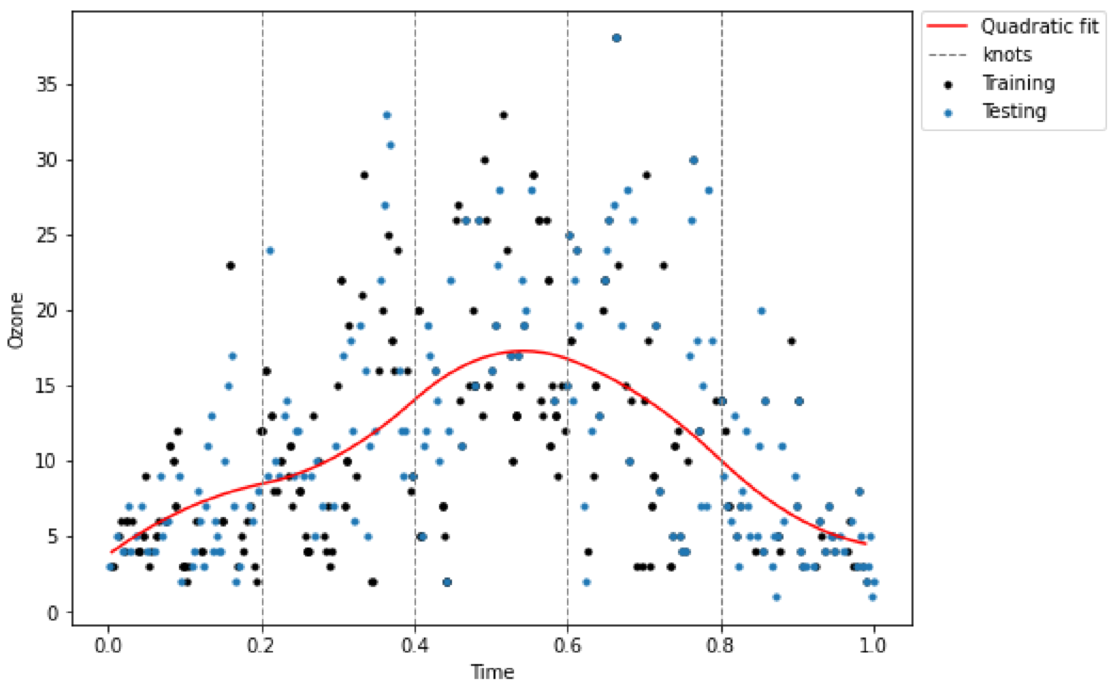


[75]:
```
from sklearn.metrics import mean_squared_error as MSE

def basis(xtrain, degree, knots):
    '''
    Function to generate the basis X

    Variables:
    xtrain (Input) :: Training covariates
    degree (Input) :: Degree of spline
    knots (Input) :: Positions of split
    X (Output) :: Constructed basis
    '''
    df = len(knots)+degree+1
    X = np.zeros((len(xtrain), df))

    for i in range(df):
        if i<(degree+1):
            X[:,i] = xtrain**i
        else:
            idxmap = (xtrain>knots[i-(degree+1)])
            X[idxmap,i] = (xtrain[idxmap]-knots[i-(degree+1)])**degree

    return X


def fit_spline(xtrain, ytrain, degree, knots):
    '''
    Function to fit a piecewise 1/2-degree continuous spline,
    continuous upto (n-1)th derivative. xtrain must be sorted.
    Variables:
    xtrain (Input) :: Training covariates
    degree (Input) :: Degree of spline
    knots (Input) :: Positions of split
    X (Output) :: Constructed basis

    Other outputs :: Fitted spline yfit, Coefficients beta
    '''

    X = basis(xtrain, degree, knots) # We directly use the function above to␣
 ↪generate basis
    beta = np.linalg.inv(X.T.dot(X)).dot(X.T.dot(ytrain))

    return X.dot(beta), X, beta

#Continuous piecewise linear fit
knots = [0.25, 0.5, 0.75]
yfit, Xtrain, beta = fit_spline(train_df['time'], train_df['ozone'], 1, knots)

plt.figure(figsize = (8,6))
plt.scatter(train_df['time'], train_df['ozone'], s = 10, color = 'black', marker␣
 ↪= 'o', label = 'Training')
plt.scatter(test_df['time'], test_df['ozone'], s = 10, marker = 'o', label =␣
 ↪'Testing')
plt.plot(train_df['time'] , yfit, color = 'red', label = 'Piecewise Linear fit')
plt.axvline(knots[0], ls = '--', color = 'gray', label = 'knots', lw = 1)
for knot in knots[1:]:
    plt.axvline(knot, ls = '--', color = 'gray', lw = 1)
plt.legend(bbox_to_anchor=(1.01, 1), loc='upper left', borderaxespad=0)
plt.xlabel('Time')
plt.ylabel('Ozone')
plt.show()

training_error = MSE(yfit, train_df['ozone'])
Xtest = basis(test_df['time'], 1, knots)
testing_error = MSE(Xtest.dot(beta) , test_df['ozone'])
print('Training Error = ',np.round(training_error,3))
print('Testing Error = ',np.round(testing_error,3))
```



```
Training Error =  40.171
Testing Error =  37.293
```

[76]:
```
#Quadratic Spline fit
knots = [0.2, 0.4, 0.6, 0.8]
yfit, Xtrain, beta = fit_spline(train_df['time'], train_df['ozone'], 2, knots)

plt.figure(figsize = (8,6))
plt.scatter(train_df['time'], train_df['ozone'], s = 10, color = 'black', marker␣
 ↪= 'o', label = 'Training')
plt.scatter(test_df['time'], test_df['ozone'], s = 10, marker = 'o', label =␣
 ↪'Testing')
plt.plot(train_df['time'] , yfit, color = 'red', label = 'Quadratic fit')
plt.axvline(knots[0], ls = '--', color = 'gray', label = 'knots', lw = 1)
for knot in knots[1:]:
    plt.axvline(knot, ls = '--', color = 'gray', lw = 1)
plt.legend(bbox_to_anchor=(1.01, 1), loc='upper left', borderaxespad=0)
plt.xlabel('Time')
plt.ylabel('Ozone')
plt.show()

training_error = MSE(yfit, train_df['ozone'])
Xtest = basis(test_df['time'], 2, knots)
testing_error = MSE(Xtest.dot(beta) , test_df['ozone'])
print('Training Error = ', np.round(training_error,3))
print('Testing Error = ', np.round(testing_error,3))
```



```
Training Error =  40.038
Testing Error =  36.926
```

[77]:
```
#Inbuilt basis generation in Python

import patsy

Xtrain_python = patsy.bs(train_df['time'], knots = knots, degree=2,␣
 ↪include_intercept=True).values
myBasis = Xtrain
pyBasis = Xtrain_python

#Checking if each basis vector in our basis can be written as a linear␣
 ↪combination of the Python basis
RSS1 = []
for i in range(myBasis.shape[1]):
    y = myBasis[:,i]
    RSS1.append(np.linalg.norm(y - pyBasis.dot(np.linalg.inv(pyBasis.T.
 ↪dot(pyBasis)).dot(pyBasis.T.dot(y)))))
print(np.round(RSS1, 8))


#Checking if each basis vector in Python basis can be written as a linear␣
 ↪combination of our basis
RSS2 = []
```

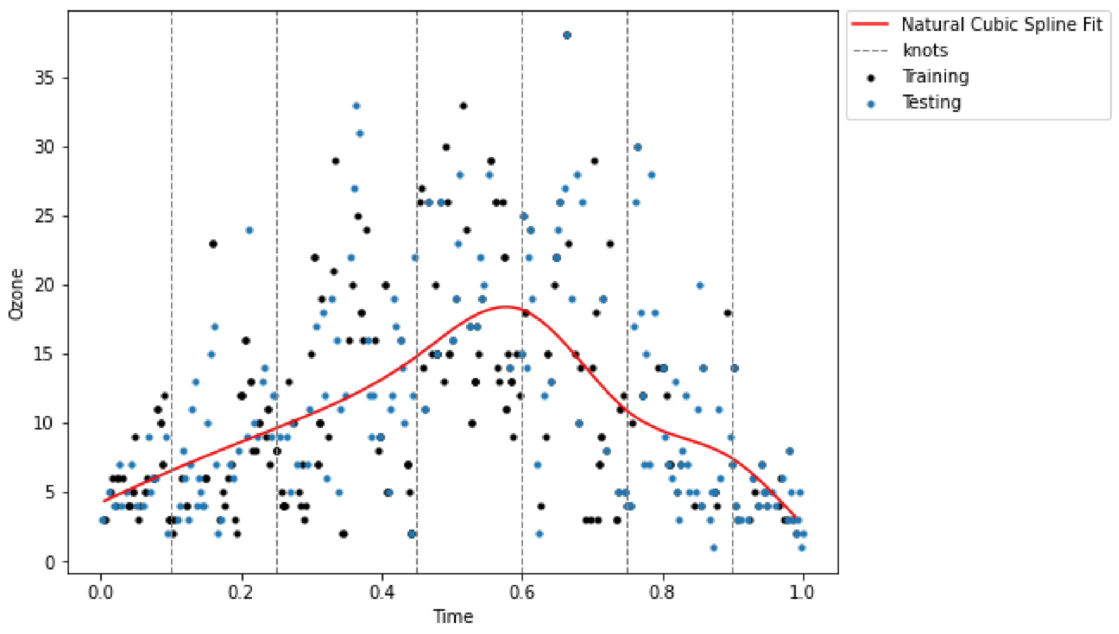
```
for i in range(pyBasis.shape[1]):
    y = pyBasis[:,i]
    RSS2.append(np.linalg.norm(y - myBasis.dot(np.linalg.inv(myBasis.T.
 ↪dot(myBasis)).dot(myBasis.T.dot(y)))))
print(np.round(RSS2, 8))
```

```
[0. 0. 0. 0. 0. 0. 0.]
[0. 0. 0. 0. 0. 0. 0.]
```

We see that the penalized criterion, RSS = 0 when we fit each basis vector with the other basis. This means that each basis vector can be expressed as a linear combination of the Python basis, and vice versa.

Thus, both basis occupy the same space.

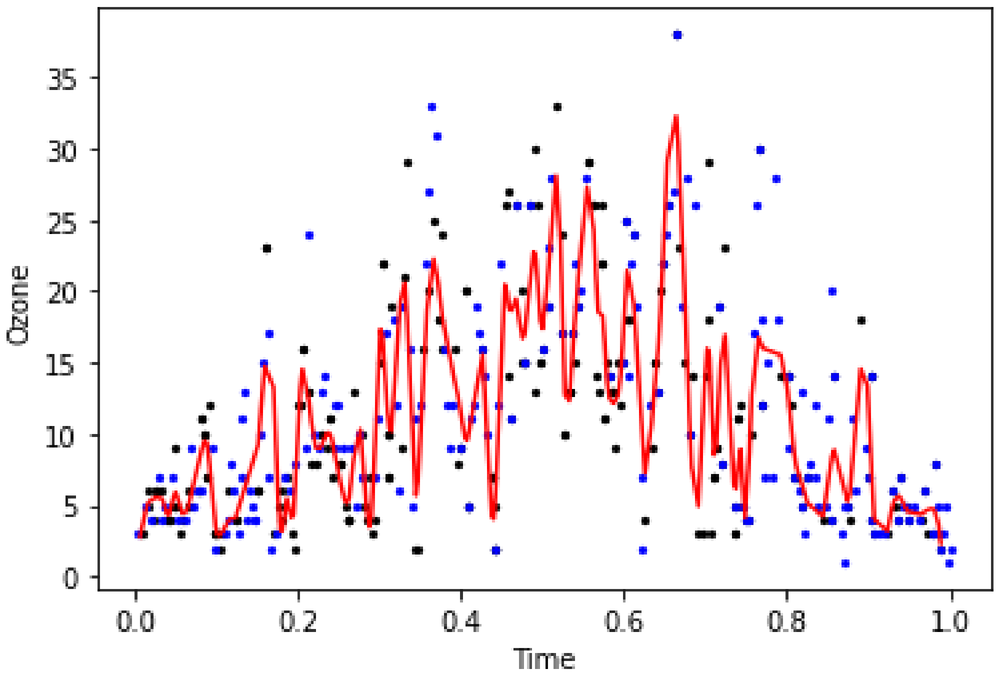
[79]:
```
#Cubic Spline

knots = [0.1,0.25,0.45,0.6,0.75,0.9]
X_cr = patsy.cr(train_df['time'], knots = knots).values
beta_fit = np.linalg.inv(X_cr.T.dot(X_cr)).dot(X_cr.T.dot(train_df['ozone']))

plt.figure(figsize = (8,6))
plt.scatter(train_df['time'], train_df['ozone'], s = 10, color = 'black', marker␣
 ↪= 'o', label = 'Training')
plt.scatter(test_df['time'], test_df['ozone'], s = 10, marker = 'o', label =␣
 ↪'Testing')
plt.plot(train_df['time'] , X_cr.dot(beta_fit), color = 'red', label = 'Cubic␣
 ↪Spline Fit')
plt.axvline(knots[0], ls = '--', color = 'gray', label = 'knots', lw = 1)
for knot in knots[1:]:
    plt.axvline(knot, ls = '--', color = 'gray', lw = 1)
plt.legend(bbox_to_anchor=(1.01, 1), loc='upper left', borderaxespad=0)
plt.xlabel('Time')
plt.ylabel('Ozone')
plt.show()
```



Since there is no `smooth.spline` equivalent in Python, I called the R `smooth.spline` operator directly from Python (using `rpy2`) to do the next task. Note this is not done in many other instances, as `rpy2` often struggles with non-native R functionality, such as in calling external libraries such as `mlbench`. The figure and parameters of fit are reported.

[80]:
```
#Smoothing Spline

import rpy2.robjects as robjects

x = train_df['time']
y = train_df['ozone']
r_x = robjects.FloatVector(x)
r_y = robjects.FloatVector(y)
r_smooth_spline = robjects.r['smooth.spline'] #extract R function
sp = r_smooth_spline(x=r_x, y=r_y, cv=False) # cv=False ensures GCV criterion
plt.scatter(train_df['time'], train_df['ozone'], s=5, color='k')
plt.scatter(test_df['time'], test_df['ozone'], s=5, color='b')
plt.gca().set_xlabel('Time')
plt.gca().set_ylabel('Ozone')
plt.plot(train_df['time'], np.array(robjects.r['predict'](sp,robjects.
 ↪FloatVector(x)).rx2('y')), color='r')
plt.show()

# print('Writing out the list of outputs in the sp robject :')
# for i in range(len(sp.names)):
#     print(sp.names[i])
```

```python
# Writing out the list of outputs in the sp robject :
# x, y, w, yin, tol, data, no.weights, lev, cv.crit, pen.crit
# crit, df, spar, ratio, lambda, iparms, auxM, fit, call

print('--------------------------------------------------')
print('--------------------------------------------------')
print('Necessary best parameters for tuning :')
print('Smoothing Parameter = ',np.array(sp.rx2('spar'))[0])
print('Penalty (lambda) = ',np.array(sp.rx2('lambda'))[0])
print('Degrees of freedom = ',np.array(sp.rx2('df'))[0])
print('GCV Criterion = ',np.array(sp.rx2('cv.crit'))[0])
print('Penalized criterion = ',np.array(sp.rx2('pen.crit')[0]))
```



```
--------------------------------------------------
--------------------------------------------------
Necessary best parameters for tuning :
Smoothing Parameter =  0.17826575305955175
Penalty (lambda) =  2.73854026421821e-08
Degrees of freedom =  78.39222510171496
GCV Criterion =  28.865727618099818
Penalized criterion =  3400.293622870157
```

4 b. [20 points] Multivariate Spline Fit With Additive Structure

Consider using both `time` and `wind` as the covariate. Use the additive model structure, with continuous piecewise linear for `time` and quadratic spline for `wind`. Both should be done using the code you developed previously. Pick your number of knots, but no more than 5. Fit and predict the ozone outcome and report the prediction error.

[84]:
```
from sklearn.linear_model import LinearRegression
#We first get the spline basis from both - time (linear) and wind (quadratic)

knots_time = [0.25, 0.5, 0.75]
knots_wind = [2.5, 5, 7.5]

X_time = basis(train_df['time'], 1, knots_time)
X_wind = basis(train_df['wind'], 2, knots_wind)

#merge the 2 bases
X_mv = np.append(X_time, X_wind, axis = 1)

#Fit a linear regression on the combined basis
reg = LinearRegression().fit(X_mv, train_df['ozone'])
X_time_test = basis(test_df['time'], 1, knots_time)
X_wind_test = basis(test_df['wind'], 2, knots_wind)
X_mv_test = np.append(X_time_test, X_wind_test, axis = 1)

#Predict the test points
y_predicted = reg.predict(X_mv_test)

#Find and print the testing error
testing_error = MSE(y_predicted, test_df['ozone'])
print("The prediction error is =", np.round(testing_error, 3))
```

```
The prediction error is = 34.347
```

[ ]: