Q1

June 23, 2020

1 Question 1: Kernels and splines: brothers, or distant cousins? (35 points)

The wheat dataset contains 100 wheat samples with specified protein and moisture content. Samples were measured by diffuse reflectance as log (1/R) from 1100 to 2500 nm in 20 nm intervals (70 data points). We want to use Kernel regression with the Epanechnikov kernel and smoothing splines to build two regression models to predict the protein and moisture content. For this purpose, we randomly split the data into training and test sets. The training set includes 80 functional data in "yTrain.csv" and their corresponding responses in "proteinTrain.csv". The test data set includes 20 observations in "yTest.csv" and "proteinTest.csv".

[1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.patheffects as pe
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import MinMaxScaler
from sklearn.linear_model import LinearRegression
import rpy2.robjects as robjects
from IPython.core.display import display, HTML
display(HTML("<style>.container { width:100% !important; }</style>"))
```

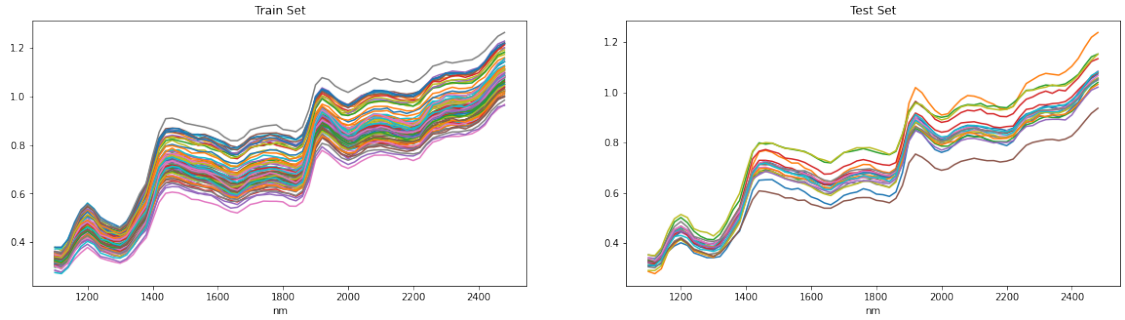
```
<IPython.core.display.HTML object>
```

[2]:
```
#load data and conver to numpy arrays
response_test = pd.read_csv('proteinTest.csv', header=None).values
response_train = pd.read_csv('proteinTrain.csv', header=None).values
y_test = pd.read_csv('yTest.csv', header=None).values
y_train = pd.read_csv('yTrain.csv', header=None).values
```

[3]:
```
#Create the domain representative of each points abscissa
# 1100 tot 2500 nm in increments of 20nm
domain =np.arange(1100,2500,20)
```

1.0.1 Visualize train and test sets

[4]:
```
plt.subplots(nrows=1, ncols=2, figsize=(20,5))
plt.subplot(121)
for x in range(len(y_train)):
    plt.plot(domain, y_train[x,:])
plt.title('Train Set')
plt.xlabel('nm')

plt.subplot(122)
for x in range(len(y_test)):
    plt.plot(domain, y_test[x,:])
plt.title('Test Set')
plt.xlabel('nm')
plt.show()
```



1.1 1)

Code your own Kernel regression with Epanechnikov kernel and find the optimal bandwidth fitted to the mean signal of the training data. Report the optimal bandwidth and plot the estimated mean function along with the sample average signal. Epanechnikov kernel is defined as:

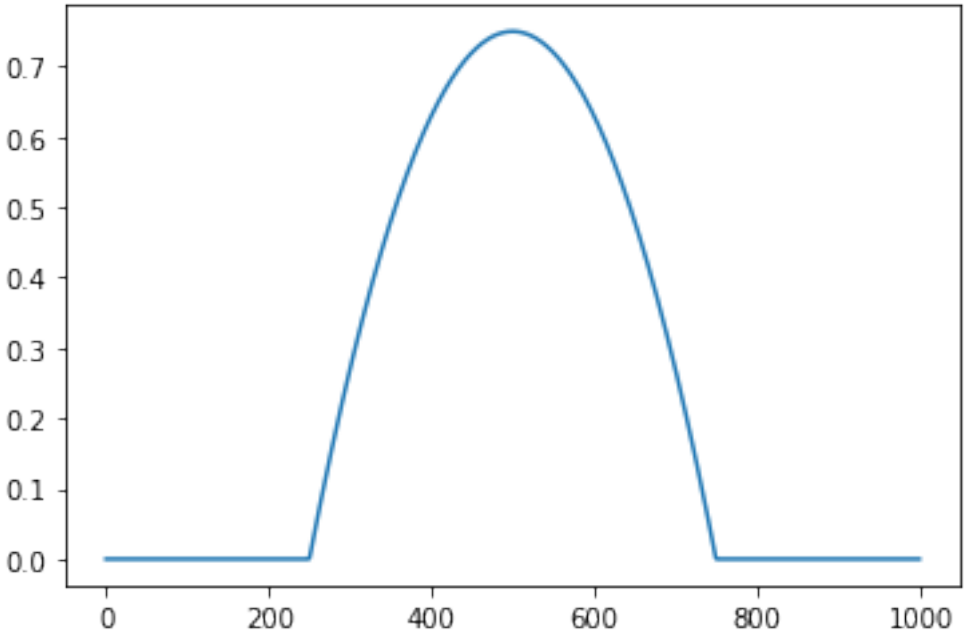
$$k(z) = \begin{cases} \frac{3}{4}(1-z^2) & if\ |z| \leq 1 \\ 0 & \text{else} \end{cases} \tag{1}$$

[5]:
```
# Epanechnikov kernel function
def ekov(z):
    if np.abs(z) <= 1: return (3.0/4.0)*(1-z**2)
    else: return 0
```

Test kernel function

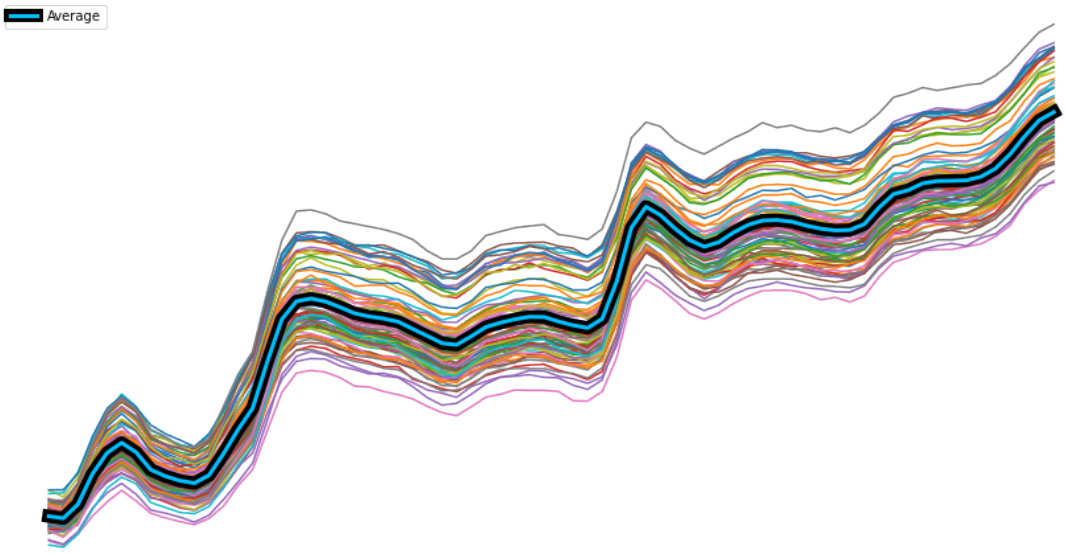
[6]:
```
#Randomly generated data
nonsense = np.linspace(-2,2,1000)
```

[7]:
```
plt.figure()
plt.plot([ekov(x) for x in nonsense])
plt.show()
```



Average traing data

[8]:
```
#Average train data
y_train_avg = y_train.mean(axis=0)

plt.figure(figsize=(15,8))
for x in range(len(y_train)):
    plt.plot(y_train[x,:],zorder=1)
plt.plot(y_train_avg,lw=3,color='deepskyblue',path_effects=[pe.
 ↪Stroke(linewidth=10, foreground='k'), pe.Normal()], label='Average',␣
 ↪zorder=2)
plt.legend()
plt.axis('off')
plt.show()
```



Fit Averaged Data with Epanechnikov kernel and find best LAMDA (λ) by LOOCV

[9]:
```
#Closure function to return prediction function on fitted data
def ekov_fit(x,y,lambda_):

    #return closure function
    def wrapper(x,y,lambda_):
        def inner(xtest):
            if not isinstance(xtest, (list, np.ndarray)):
                xtest = [xtest]
            N = len(xtest)
            f = np.zeros(N)

            for k in range(N):
                #apply Epanechnikov kernel to all values
                z = np.array([ekov(v) for v in ((xtest[k] - x) / lambda_)])
                f[k] = np.sum(z * y) / np.sum(z)

            return f
        return inner
    #establish function call
    func = wrapper(x, y, lambda_)

    #return established closure function
    return func
```

[10]:
```
#LOOCV
def epanechnikov_mse_LOOCV(x, y, lambda_):
    '''
    inputs
    ------
    x          -> the independent variable vector
    y          -> the dependent variable vector
    lambda_    -> lambda parameter


    returns
    ------
    mse -> the mse for all errors for LOOCV
    '''
    preds = np.zeros(len(y))
    y_true = np.zeros(len(y))

    #LOOCV
    for pos, i in enumerate(range(len(y))):
        X = np.delete(x, i)
        Y = np.delete(y, i)

        xtest = x[i]
        pred_obj = ekov_fit(x,y,lambda_)

        pred = pred_obj(xtest)
        preds[i] = pred
        y_true[i] = y[i]

    mse = mean_squared_error(y_true, preds)


    return mse
```

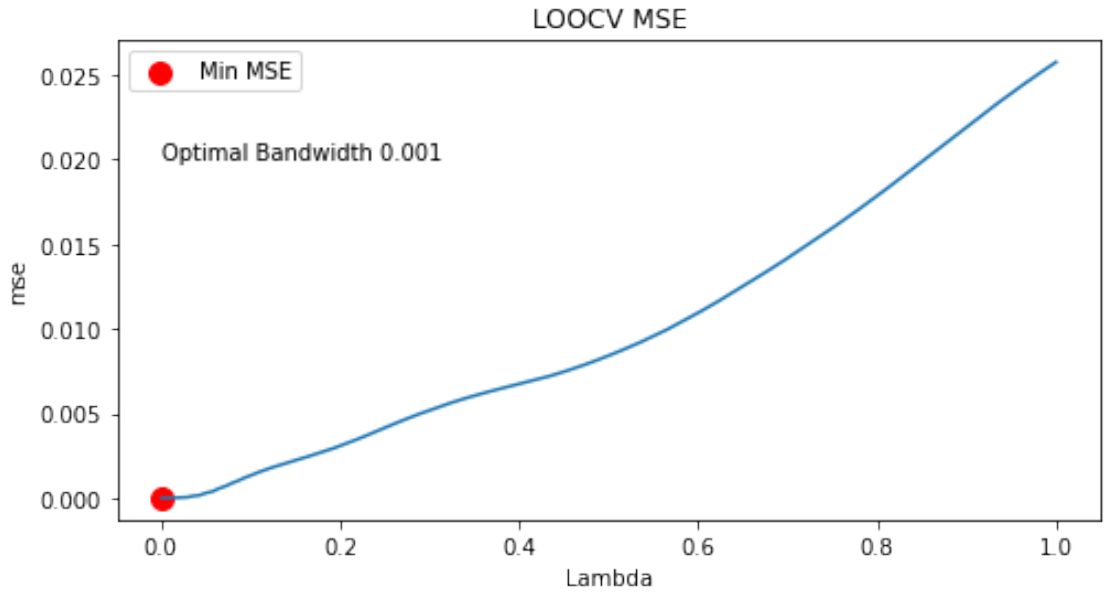
[11]:
```
#scale the domain
scaler = MinMaxScaler()
scaler.fit(domain.reshape(-1,1))
domain_scaled = scaler.transform(domain.reshape(-1,1)).ravel()
print('Domain Min: {}\nDomain Max: {}'.format(domain_scaled.min(),␣
 ↪domain_scaled.max()))
```

```
Domain Min: 0.0
Domain Max: 1.0
```

[12]:
```
# Different values of lambda to test
lams = np.linspace(0.001,1,1000)
```

[13]:
```
#perform LOOCV
mses = []

for l in lams:
    mses.append(epanechnikov_mse_LOOCV(x=domain_scaled, y=y_train_avg,␣
 ↪lambda_=l))
```

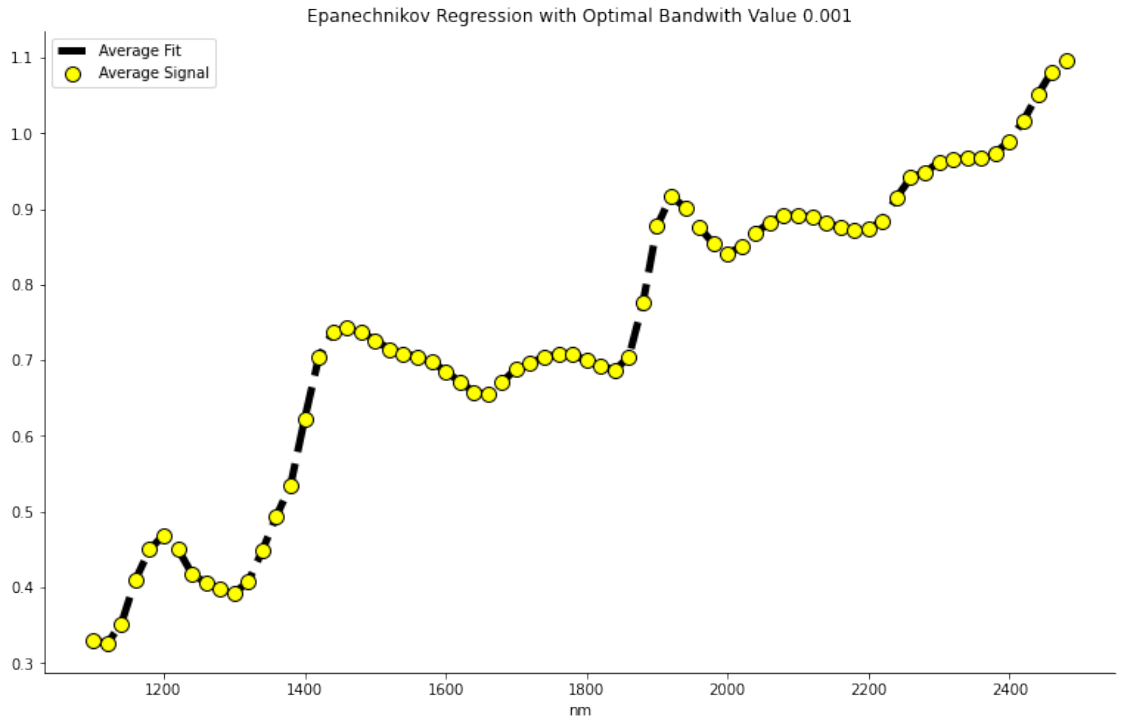
[14]:
```
plt.figure(figsize=(8,4))
plt.plot(lams,mses)
plt.scatter(lams[np.argmin(mses)], mses[np.argmin(mses)], color='r', s=100,␣
 ↪label='Min MSE')
plt.xlabel('Lambda')
plt.ylabel('mse')
plt.title('LOOCV MSE')
plt.text(0,0.02, 'Optimal Bandwidth {}'.format(lams[np.argmin(mses)]))
plt.legend()
plt.show()
```



[15]:
```
# Create a kernel function with the best found lambda (kernel bandwidth) value
ekov_mapper = ekov_fit(x=domain_scaled, y=y_train_avg, lambda_=lams[np.
 ↪argmin(mses)])
```

[16]:
```
plt.figure(figsize=(13,8))
plt.plot(domain, ekov_mapper(domain_scaled), color = 'k', lw=5, ls='--'␣
 ↪,zorder=1, label='Average Fit')
plt.scatter(domain, y_train_avg, color='yellow', edgecolor='k', s=100,␣
 ↪zorder=2, label='Average Signal')
```

```
plt.title('Epanechnikov Regression with Optimal Bandwith Value {}'.
 ↪format(lams[np.argmin(mses)]))
plt.gca().spines['right'].set_visible(False)
plt.gca().spines['top'].set_visible(False)
plt.xlabel('nm')
plt.legend()
plt.show()
```



Result The optimal bandwidth when fitting Epanechnikov Regression on the mean signal of the training data was found to be 0.001. This low bandwidth is a cause for concern however as it appears to be a value which induces high variance (extreme overfitting). The level of 0.001 is also due to rescaling the domain from magnitudes of 1100-2500 to 0-1.

1.2 2)

Develop a prediction model to predict the protein and moisture content based on the kernel you found in part (1) and the training data.

[17]:
```
# Gaussian Kernel on abscissa
H = ekov_mapper(domain_scaled)
```

[18]:
```
#OLS to solve for weights on train set
betas = np.linalg.lstsq(H.T.reshape(-1,1), y_train.T, rcond=None)[0].T

#OLS to solve for weights on test set
betas_test = np.linalg.lstsq(H.T.reshape(-1,1), y_test.T, rcond=None)[0].T
```

Linear Regression model on weights

[19]:
```
#fit a linear regression model on the training weights (independent variable)␣
 ↪and the training protein values (dependent variable)
model = LinearRegression()
model.fit(betas, response_train)
```

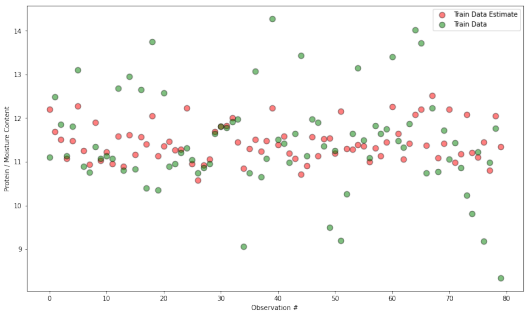
[19]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
```

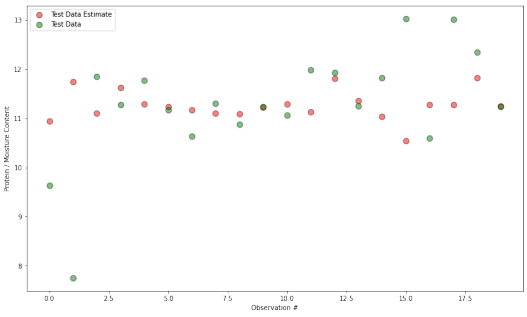
[20]:
```
#predict on the train and test sets
train_preds = model.predict(betas)
test_preds = model.predict(betas_test)
```

[21]:
```
print('Training data MSE: {}'.format(mean_squared_error(response_train, model.
 ↪predict(betas))))
print('Test data MSE: {}'.format(mean_squared_error(response_test, model.
 ↪predict(betas_test))))
```

```
Training data MSE: 1.0232399728722652
Test data MSE: 1.5184823164384993
```

[22]:
```
plt.subplots(nrows=1, ncols=2, figsize=(30,8))
plt.subplot(121)
plt.scatter(range(len(train_preds)), train_preds, label='Train Data Estimate',␣
 ↪color='red', alpha=0.5, edgecolor='k',s=75)
plt.scatter(range(len(train_preds)), response_train, label='Train␣
 ↪Data',color='green', alpha=0.5, edgecolor='k', s=75)
plt.legend()
plt.xlabel('Observation #')
plt.ylabel('Protein / Moisture Content')
plt.subplot(122)
plt.scatter(range(len(test_preds)), test_preds, label='Test Data Estimate',␣
 ↪color='red', alpha=0.5, edgecolor='k', s=75)
plt.scatter(range(len(test_preds)), response_test, label='Test␣
 ↪Data',color='green', alpha=0.5, edgecolor='k', s=75)
plt.legend()
plt.xlabel('Observation #')
plt.ylabel('Protein / Moisture Content')
plt.show()
```





1.3 3)

Find the optimal lambda using GCV for the smoothing splines fitted to the mean signal of the training data. Report the optimal lambda and the number of spline coefficients corresponding to the optimal lambda.

[23]:
```
def smoothing_splines(x, y, spar):
    '''
    inputs
    ------
    x -> independent variables
    y -> dependent variables
    spar -> smoothing parameter

    outputs
    -------
    yhat -> the fitted values
    pred_obj -> function used to predict on new unseen data

    '''

    #get the smooth.spline function used by r
    smooth_spline = robjects.r['smooth.spline']

    #conver x and y to r objects
    x = robjects.FloatVector(list(x))
    y = robjects.FloatVector(list(y))

    #fit the data
    spline_obj = smooth_spline(x=x, y=y, spar=float(spar))


    '''
```

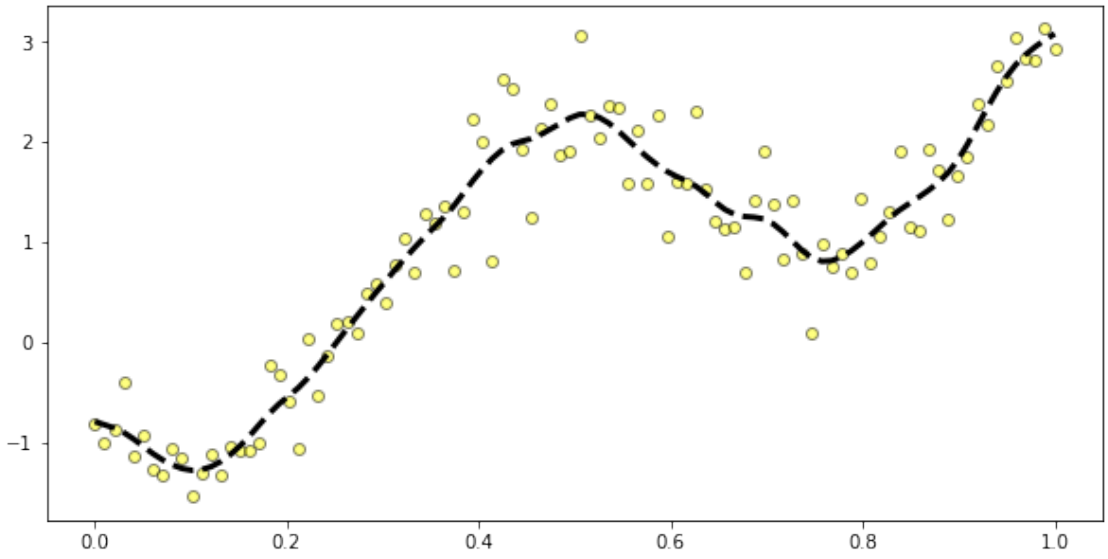
```
    This closure function is returned so predictions can be made on new unseen␣
↪data.
    '''
    def wrapper(spline_obj):
        def inner(x):
            x = robjects.FloatVector(list(x))
            yspline = robjects.r['predict'](spline_obj,x).rx2('y')
            return np.array(yspline)
        return inner

    #create the prediction function
    predict_obj = wrapper(spline_obj)

    #return yhat, df, and prediction function
    return np.array(spline_obj.rx2('y')),np.array(spline_obj.rx2('df'))[0] ,␣
↪predict_obj, spline_obj
```

Test out smoothing spline implementation

[24]:
```
#arbitrary data to test out smoothing spline implementation
n=100; k=40; x = np.linspace(0,1,n); y=2.5*x-np.sin(10*x)-np.exp(-10*x);␣
↪sigma=0.3; ynoise = y + np.random.randn(n)*sigma
```

[25]:
```
fit, _, _, _ = smoothing_splines(x,ynoise,0.5)
```

[26]:
```
plt.figure(figsize=(10,5))
plt.scatter(x,ynoise,edgecolor='k', alpha=0.5, color='yellow')
plt.plot(x,fit, ls='--', lw=3, color='k')
plt.show()
```



Hyperparameter search across multiple smoothing parameter values

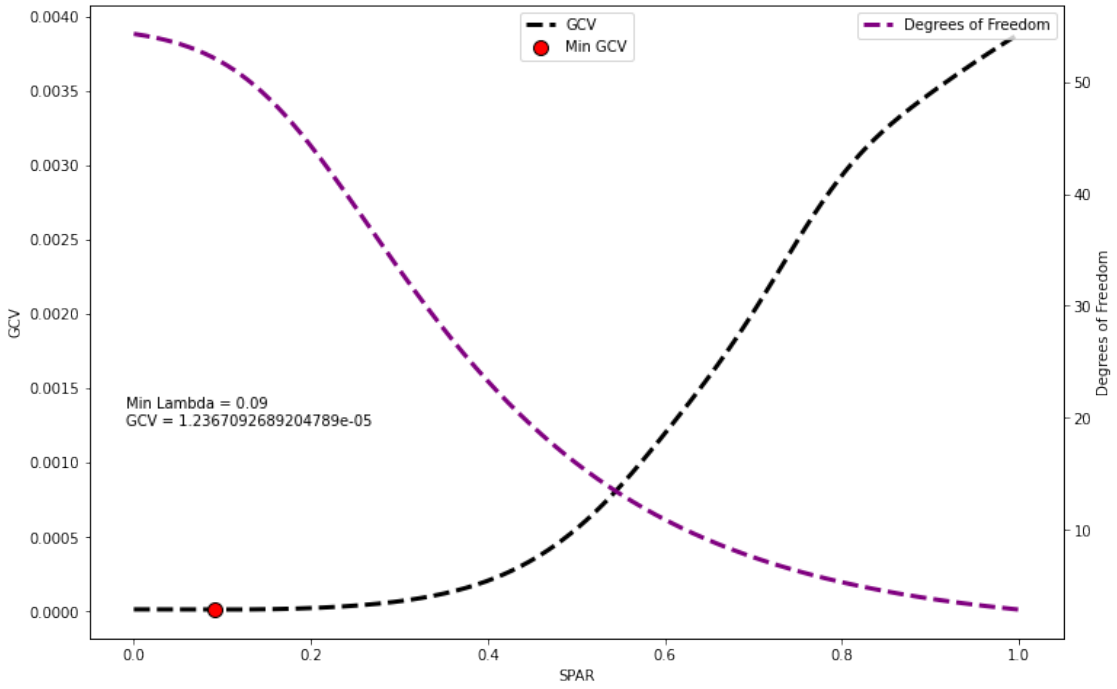
[27]:
```
# 1000 smoothing parameters from [0,1]
lams = np.linspace(0,1,2000)

#RSS placeholder
RSS = np.zeros(len(lams))

#df placeholder
dfs = np.zeros(len(lams))

n = len(y_train_avg)
```

[28]:
```
# Hyperparam search across all values of lambda
for pos, l in enumerate(lams):
    yhat, df, _ , _= smoothing_splines(x=domain_scaled, y=y_train_avg, spar=l)
    dfs[pos] = df
    RSS[pos] = np.sum((yhat - y_train_avg)**2)
```

GCV

[29]:
```
GCV = (RSS/n)/((1- dfs /n)**2)
```

[30]:
```
plt.figure(figsize=(12,8))
plt.plot(lams,GCV,zorder=1,color='k', ls='--', label='GCV', lw=3)
plt.scatter(lams[np.argmin(GCV)], GCV[np.argmin(GCV)], color='red', s=100,␣
 ↪edgecolor='k', label='Min GCV',zorder=2)
plt.text(lams[np.argmin(GCV)] - 1.1 * lams[np.argmin(GCV)], GCV[np.argmin(GCV)]␣
 ↪+ 100 * GCV[np.argmin(GCV)], 'Min Lambda = {}\nGCV = {}'.
 ↪format(round(lams[np.argmin(GCV)],2), GCV[np.argmin(GCV)]))
plt.xlabel('SPAR')
plt.ylabel('GCV')
plt.legend(loc='upper center')
plt.twinx()
plt.plot(lams,dfs,ls='--', lw=3, color='purple', label='Degrees of Freedom')
plt.legend()
plt.ylabel('Degrees of Freedom')
plt.show()
```



[31]:
```
#create smoothing spline fit with the optimal lambda
yhat, df, pred , spl = smoothing_splines(x=domain_scaled, y=y_train_avg, spar=0.
↪09)
```

[32]:
```
print('''
Optimal lambda value found to be 0.09 when fit on scaled domain [0,1]\n
Number of coefficients found to be {}
'''.format(len(spl.rx2('fit').rx2('coef'))))
```

```
Optimal lambda value found to be 0.09 when fit on scaled domain [0,1]

Number of coefficients found to be 56
```

1.4 4)

Develop a prediction model to predict the protein and moisture content based on the extracted features from the training data using smoothing spline.

[33]:
```
#OLS to solve for weights on train set
betas = np.linalg.lstsq(yhat.T.reshape(-1,1), y_train.T, rcond=None)[0].T

#OLS to solve for weights on test set
```

```
betas_test = np.linalg.lstsq(yhat.T.reshape(-1,1), y_test.T, rcond=None)[0].T
```

[34]:
```
#fit a linear regression model on the training weights (independent variable)␣
 ↪and the training protein values (dependent variable)
model = LinearRegression()
model.fit(betas, response_train)
```

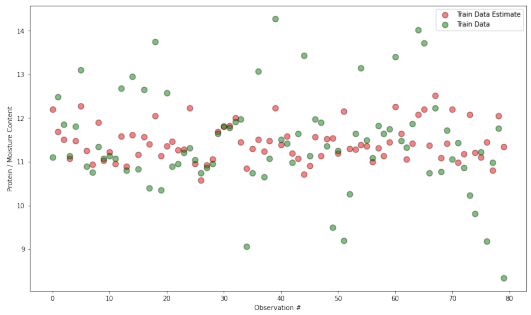
[34]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
```

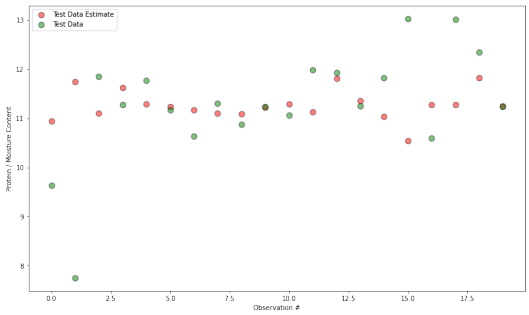
[35]:
```
#predict on the train and test sets
train_preds = model.predict(betas)
test_preds = model.predict(betas_test)
```

[36]:
```
print('Training data MSE: {}'.format(mean_squared_error(response_train, model.
 ↪predict(betas))))
print('Test data MSE: {}'.format(mean_squared_error(response_test, model.
 ↪predict(betas_test))))
```

```
Training data MSE: 1.0232377216190047
Test data MSE: 1.5184736501562002
```

[37]:
```
plt.subplots(nrows=1, ncols=2, figsize=(30,8))
plt.subplot(121)
plt.scatter(range(len(train_preds)), train_preds, label='Train Data Estimate',␣
 ↪color='red', alpha=0.5, edgecolor='k',s=75)
plt.scatter(range(len(train_preds)), response_train, label='Train␣
 ↪Data',color='green', alpha=0.5, edgecolor='k', s=75)
plt.legend()
plt.xlabel('Observation #')
plt.ylabel('Protein / Moisture Content')
plt.subplot(122)
plt.scatter(range(len(test_preds)), test_preds, label='Test Data Estimate',␣
 ↪color='red', alpha=0.5, edgecolor='k', s=75)
plt.scatter(range(len(test_preds)), response_test, label='Test␣
 ↪Data',color='green', alpha=0.5, edgecolor='k', s=75)
plt.legend()
plt.xlabel('Observation #')
plt.ylabel('Protein / Moisture Content')
plt.show()
```





1.5 5)

Evaluate and compare the performance of the estimated prediction models in terms of mean-squared error on the test set. Which model do you recommend?

1.5.1 Result

Smoothing Splines	
Train MSE	1.023
Test MSE	1.518

Smoothing Splines	
Train MSE	1.028
Test MSE	1.526

Both methods returned close to the same results when trained using the same regression model (Linear Regression). Smoothing splines came in a little lower in terms of MSE for Test set. However, no one method proves significantly superior over another

[ ]: