Regression - Assignment 2

Data and Package Import

In [27]:

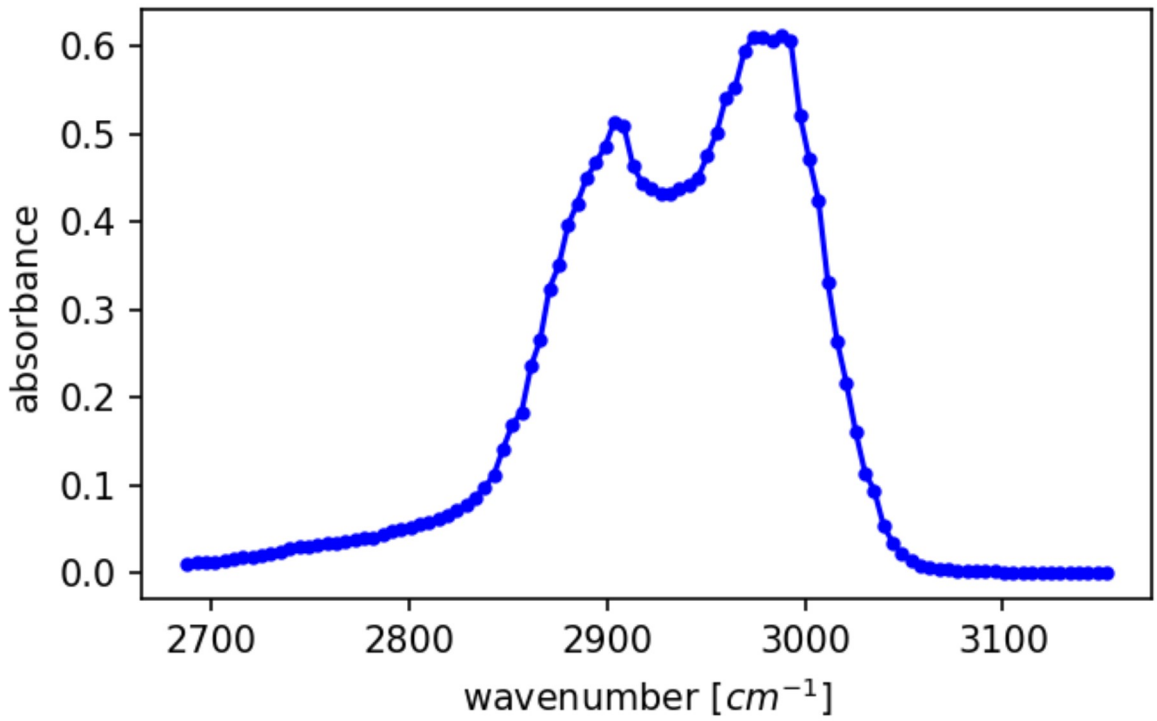
```
%matplotlib inline
import numpy as np
import pandas as pd
import pylab as plt
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.model_selection import KFold
#I am also importing the rbf function and stats made in previous hw
#by pasting it here

def rbf(x_train, x_test=None, gamma=1):
    if x_test is None:
        x_test = x_train
    N = len(x_test) #<- number of data points
    M = len(x_train) #<- number of features
    X = np.zeros((N,M))
    for i in range(N):
        for j in range(M):
            X[i,j] = np.exp(-gamma*(x_test[i] - x_train[j])**2)
    return X
```

In [23]:
```
df = pd.read_csv('data/ethanol_IR.csv')
x_all = df['wavenumber [cm^-1]'].values
y_all = df['absorbance'].values

x_peak = x_all[475:575]
y_peak = y_all[475:575]

fig, ax = plt.subplots(figsize = (5, 3), dpi = 150)
ax.plot(x_peak, y_peak, '-b', marker = '.')
ax.set_xlabel('wavenumber [$cm^{-1}$]')
```



1. Mean Absolute Errors

In [50]:
```
def MAE(actual, prediction):
    N = len(actual)
    sumoferrors = 0
    for i in list(range(N)):
        sumoferrors = sumoferrors + abs(actual[i] - prediction[i])

    mae = sumoferrors/N
    #print(mae)
    return mae
```

Use 8-fold cross-validation to compute the average and standard deviation of the MAE on the spectra dataset.

Use a `LinearRegression` model and an `rbf` kernel with σ=100.

Make sure to pass `shuffle = True` argument when you make a `KFold` object.

In [84]:
```
# Writing some notes on strategy:
# kfold split will split it differently on each iteration
# I will define 8 iterations
# I will do a linear regression model with rbf kernel with sig = 100 on each split
```

```
# Then we will find the MAE for each iterated split
# then will find the stats of each split using a defined calc_stats func

kf = KFold(n_splits = 8, shuffle = True)
sigma = 100
gamma = 1./2/sigma**2
r2_test = []
MAEs = []
#start doing ksplit iterations

for train_index, test_index in kf.split(x_peak):
    x_train, x_test = x_peak[train_index], x_peak[test_index]
    y_train, y_test = y_peak[train_index], y_peak[test_index]

    X_train = rbf(x_train, gamma=gamma)

    model_rbf = LinearRegression() #create a linear regression model instance
    model_rbf.fit(X_train, y_train) #fit the model
    r2 = model_rbf.score(X_train, y_train) #get the "score", which is equivalent t


    X_test = rbf(x_train, x_test=x_test, gamma=gamma)

    yhat_rbf = model_rbf.predict(X_test)
    currentMAE = MAE(X_test[:,0], y_test)


    MAEs.append(currentMAE)


MAEs_mean = np.mean(MAEs)
MAEs_std = np.std(MAEs)
print(MAEs_mean,MAEs_std)
```

```
0.3729140772325844 0.09647579601606301
```

Determine the optimum σ that results in the lowest mean of MAE.

Vary the width of an `rbf` kernel with σ = [1, 10, 50, 100, 150].

In [132]:

```
sigmas = [1, 10, 50, 100, 150]

kf = KFold(n_splits = 8, shuffle = True)

#gamma = 1./2/sigmas**2
r2_test = []
#I did not know how to make a for loop for both the sigmas and the k iterations SO
MAEs1 = []

total_mae_mean = []

MAEs1 = []
for train_index, test_index in kf.split(x_peak):
    x_train, x_test = x_peak[train_index], x_peak[test_index]
    y_train, y_test = y_peak[train_index], y_peak[test_index]

    X_train = rbf(x_train, gamma=1./2/1**2)

    model_rbf = LinearRegression() #create a linear regression model instance
    model_rbf.fit(X_train, y_train) #fit the model
    r2 = model_rbf.score(X_train, y_train) #get the "score", which is equivalent t
    X_test = rbf(x_train, x_test=x_test, gamma=gamma)
    yhat_rbf = model_rbf.predict(X_test)
```

```
    currentMAE = MAE(X_test[:,0], y_test)
    MAEs1.append(currentMAE)
MAEs1_mean = np.mean(MAEs1)
total_mae_mean.append(MAEs1_mean)

MAEs10 = []
for train_index, test_index in kf.split(x_peak):
    x_train, x_test = x_peak[train_index], x_peak[test_index]
    y_train, y_test = y_peak[train_index], y_peak[test_index]

    X_train = rbf(x_train, gamma=1./2/10**2)

    model_rbf = LinearRegression() #create a linear regression model instance
    model_rbf.fit(X_train, y_train) #fit the model
    r2 = model_rbf.score(X_train, y_train) #get the "score", which is equivalent t
    X_test = rbf(x_train, x_test=x_test, gamma=gamma)
    yhat_rbf = model_rbf.predict(X_test)
    currentMAE = MAE(X_test[:,0], y_test)
    MAEs10.append(currentMAE)
MAEs10_mean = np.mean(MAEs10)
total_mae_mean.append(MAEs10_mean)

MAEs50 = []
for train_index, test_index in kf.split(x_peak):
    x_train, x_test = x_peak[train_index], x_peak[test_index]
    y_train, y_test = y_peak[train_index], y_peak[test_index]

    X_train = rbf(x_train, gamma=1./2/50**2)

    model_rbf = LinearRegression() #create a linear regression model instance
    model_rbf.fit(X_train, y_train) #fit the model
    r2 = model_rbf.score(X_train, y_train) #get the "score", which is equivalent t
    X_test = rbf(x_train, x_test=x_test, gamma=gamma)
    yhat_rbf = model_rbf.predict(X_test)
    currentMAE = MAE(X_test[:,0], y_test)
    MAEs50.append(currentMAE)
MAEs50_mean = np.mean(MAEs50)
total_mae_mean.append(MAEs50_mean)

MAEs100 = []
for train_index, test_index in kf.split(x_peak):
    x_train, x_test = x_peak[train_index], x_peak[test_index]
    y_train, y_test = y_peak[train_index], y_peak[test_index]

    X_train = rbf(x_train, gamma=1./2/100**2)

    model_rbf = LinearRegression() #create a linear regression model instance
    model_rbf.fit(X_train, y_train) #fit the model
    r2 = model_rbf.score(X_train, y_train) #get the "score", which is equivalent t
    X_test = rbf(x_train, x_test=x_test, gamma=gamma)
    yhat_rbf = model_rbf.predict(X_test)
    currentMAE = MAE(X_test[:,0], y_test)
    MAEs100.append(currentMAE)
MAEs100_mean = np.mean(MAEs100)
total_mae_mean.append(MAEs100_mean)

MAEs150 = []
for train_index, test_index in kf.split(x_peak):
    x_train, x_test = x_peak[train_index], x_peak[test_index]
    y_train, y_test = y_peak[train_index], y_peak[test_index]

    X_train = rbf(x_train, gamma=1./2/150**2)

    model_rbf = LinearRegression() #create a linear regression model instance
```

```
    model_rbf.fit(X_train, y_train) #fit the model
    r2 = model_rbf.score(X_train, y_train) #get the "score", which is equivalent t
    X_test = rbf(x_train, x_test=x_test, gamma=gamma)
    yhat_rbf = model_rbf.predict(X_test)
    currentMAE = MAE(X_test[:,0], y_test)
    MAEs150.append(currentMAE)
MAEs150_mean = np.mean(MAEs150)
total_mae_mean.append(MAEs150_mean)
low_MAE = total_mae_mean.index(min(total_mae_mean))
print(sigmas[low_MAE])
print("is the optimal sigma value")
print(total_mae_mean)

#     for sigma in sigmas:
#         X_train = rbf(x_train, gamma=1./2/sigma**2)
#         model_rbf = LinearRegression() #create a linear regression model instanc
#         model_rbf.fit(X_train, y_train) #fit the model
#         r2 = model_rbf.score(X_train, y_trin) #get the "score", which is equival


#         X_test = rbf(x_train, x_test=x_test, gamma=gamma)

#         yhat_rbf = model_rbf.predict(X_test)
#         currentMAE = MAE(X_test[:,0], y_test)
```

```
10
is the optimal sigma value
[0.2018660417114409, 0.19752068112562177, 0.19852264831007693, 0.198981975422566
1, 0.20181583263875585]
```

2. Hyperparameter Tuning

Reshape `x_peak` and `y_peak` into 2D array.

In [133]:
```
x_peakT = x_peak.reshape(-1,2)
y_peakT = y_peak.reshape(-1,2)
```

In [134]:
```
x_train, x_test, y_train, y_test = train_test_split(x_peakT, y_peakT, test_size=0.
np.random.seed(0)
xtraino = np.reshape(x_train,(-1,1))
```

Use `for` loop to determine the optimum regularization strength α of a KRR model.

Use an `rbf` kernel with σ=20.

Determine the optimum value of α out of [1e-5, 1e-4, 1e-3, 1e-2, 1e-1, 1].

In [135]:
```
from sklearn.kernel_ridge import KernelRidge
alphas = [1e-5, 1e-4, 1e-3, 1e-2, 1e-1, 1]
sigma = 20
gamma = 1/(2*sigma**2)
r2s = []


for i in range(len(alphas)):
    KRR = KernelRidge(alpha = alphas[i], kernel = 'rbf', gamma = gamma)
```

```
    KRR.fit(xtraino, ytraino)

    x_predict = np.linspace(min(xtraino), max(xtraino), 300)
    yhat_KRR = KRR.predict(x_predict)

    r2_test = KRR.score(xtraino, ytraino)
    r2s.append(r2_test)


bestalpha = r2s.index(max(r2s))
print(alphas[bestalpha])
print("is the best alpha value")
#print(r2s)
#print(alphas[bestalpha])
#KRR = KernelRidge(kernel='rbf')
# parameter_ranges = {'alpha':alphas}
# KRR = KernelRidge(kernel='rbf')
```

```
1e-05
is the best alpha value
```

3. GridSearchCV

Import a LASSO model.

In [136]:

Shuffle the `x_peak` and `y_peak`.

You can get a shuffled array when you run `x_shuffle, y_shuffle = shuffle(x, y)`.

The reason why we shuffle the data is that `GridSearchCV` does not have an option to shuffle the input data. Note that we automatically shuffled the data in the problem 1.

In [137]:
```
from sklearn.utils import shuffle
```

Build a `GridSearchCV` model that optimizes the hyperparameters of a LASSO model for the spectra data.

Search over $\alpha \in$ [1e-5, 1e-4, 1e-3, 1e-2, 1e-1, 1] and $\sigma \in$ [5, 10, 15, 20, 25, 30, 35, 40].

Use 3-fold cross-validation.

Hint: You will need to use a `for` loop over σ values. Unlike KRR, LASSO models do not take neither `gamma` nor `sigma` as a parameter. You have to make an `rbf` kernel manually and input it to a LASSO model.

Obtain the optimum α and the best score for each σ value. Use `GridSearchCV.best_score_` as accuracy metric.

In [167]:
```
from sklearn.model_selection import GridSearchCV
from sklearn.metrics.pairwise import rbf_kernel

# Sorry that this part isn't done, I was very confused by the PIAZZA post
# I had a lot of exams this week

alphas = np.array([1e-5, 1e-4, 1e-3, 1e-2, 1e-1, 1])
sigmas = np.array([5, 10, 15, 20, 25, 30, 35, 40])
gammas = 1./(2*sigmas**2)
```

```
vals = []
for i in range(len(sigmas)):
    gamma = 1/(2*sigmas[i]**2)
    X_train = rbf(x_train, gamma=gamma)
    model_rbf = LinearRegression() #create a linear regression model instance
    model_rbf.fit(X_train, y_train) #fit the model
    r2 = model_rbf.score(X_train, y_train)
    model_rbf_search = GridSearchCV(model_rbf, alphas, cv=3)
    KRR_search.fit(X_train,y_train)
    KRR_search.best_estimator_, KRR_search.best_score_
    vals.append( KRR_search.best_estimator_, KRR_search.best_score_)
```

What is the optimum σ and α?

In [ ]:
```
kf = KFold(n_splits = 8, shuffle = True)
sigma = 100
gamma = 1./2/sigma**2
r2_test = []
MAEs = []
#start doing ksplit iterations

for train_index, test_index in kf.split(x_peak):
    x_train, x_test = x_peak[train_index], x_peak[test_index]
    y_train, y_test = y_peak[train_index], y_peak[test_index]

    X_train = rbf(x_train, gamma=gamma)

    model_rbf = LinearRegression() #create a linear regression model instance
    model_rbf.fit(X_train, y_train) #fit the model
    r2 = model_rbf.score(X_train, y_train) #get the "score", which is equivalent t


    X_test = rbf(x_train, x_test=x_test, gamma=gamma)

    yhat_rbf = model_rbf.predict(X_test)
    currentMAE = MAE(X_test[:,0], y_test)


    MAEs.append(currentMAE)


MAEs_mean = np.mean(MAEs)

MAEs_std = np.std(MAEs)
print(MAEs_mean,MAEs_std)
```

Optional Task

Check what happens if the input data is not shuffled before the `GridSearchCV` .

In [ ]:
```

```

4. Ensemble Kernel Ridge Regression

In this problem you will combine ideas from k-fold cross-validation and bootstrapping with KRR to create an **ensemble** of KRR models.

Reshape `x_peak` and `y_peak` into 2D array.

In [140]:
```
# x_peakT = x_peak.reshape(-1,2)
# y_peakT = y_peak.reshape(-1,2)
x_peak = x_peak.reshape(-1,1) #we need to convert these to columns
```

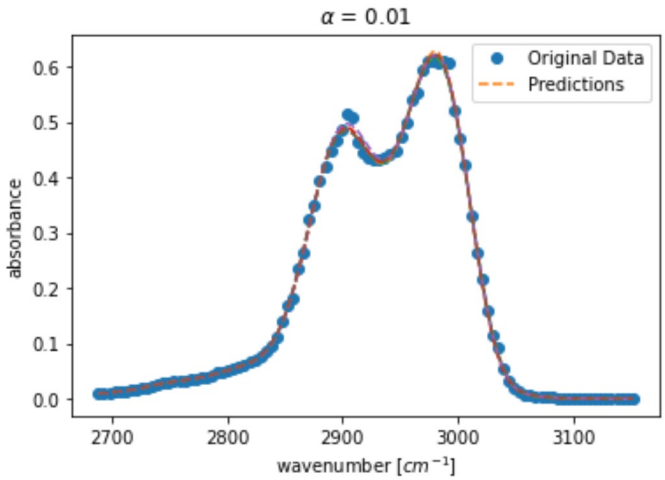
Use 5-fold cross-validation with the spectra data to construct a series of 5 KRR models with a `rbf` kernel with γ=0.0005 and α=0.01.

Each model will be trained with 80% of the data, but the exact training points will vary each time so the models will also vary.

Get the predictions from the whole `x_peak`.

In [159]:
```
gamma = 0.0005
alpha = 0.01
fig, ax = plt.subplots()
ax.plot(x_peak,y_peak,'o')
kf = KFold(n_splits = 5,shuffle = True)
for train_index, test_index in kf.split(x_peak):
    x_train, x_test = x_peak[train_index], x_peak[test_index]
    y_train, y_test = y_peak[train_index], y_peak[test_index]
    #x_train, x_test, y_train, y_test = train_test_split(x_peak, y_peak, test_size
# np.random.seed(0)
# I think the instructions say to get predictions from the entire x_peak
    KRR = KernelRidge(alpha=alpha, kernel='rbf', gamma=gamma)
    KRR.fit(x_train, y_train)
#x_predict = np.linspace(min(x_peak), max(x_peak), 300) #do not use this to predic
    yhat_KRR = KRR.predict(x_peak) #what he means by predict from whole x peak
    ax.plot(x_peak,yhat_KRR, '--', markerfacecolor = 'none')
ax.set_xlabel('wavenumber [$cm^{-1}$]')
ax.set_ylabel('absorbance')
ax.legend(['Original Data', 'Predictions'])
ax.set_title(r'$\alpha$ = {}'.format(alpha))
print('As you can see all the models are quite close to each other')
```

```
As you can see all the models are quite close to each other
so it is hard to see all of the different lines for each k fold
```



Plot the resulting ensemble of models along with the original data.

The plot should consists of 6 different lines (1 from the original data and 5 from KRR models).

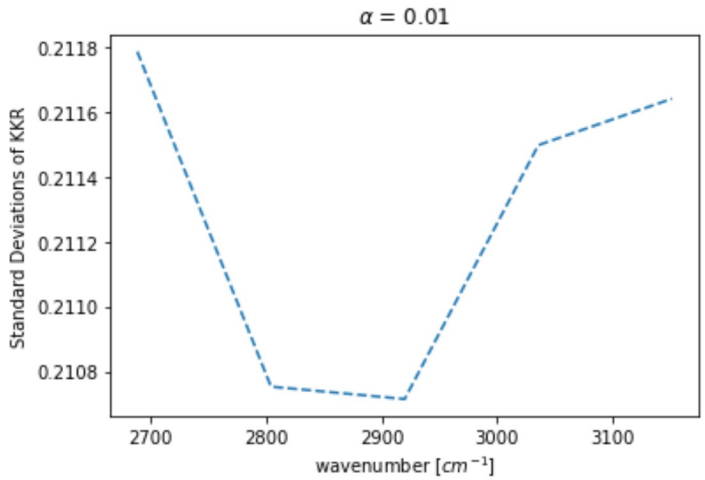
In [152]:
```
#I did not know how to put these codes in separate blocks since
#I plotted the original data in the for loop for the block above!
#If you look closely you can see that there are different colors for the predictio
```

Plot the standard deviation of the 5 KRR models as a function of wavelength.

In [165]:
```
fig, ax = plt.subplots()
#fig, axes = plt.subplots(1,5, figsize = (15, 6))
dev = []
kf = KFold(n_splits = 5, shuffle = True)
for train_index, test_index in kf.split(x_peak):
    x_train, x_test = x_peak[train_index], x_peak[test_index]
    y_train, y_test = y_peak[train_index], y_peak[test_index]
    KRR = KernelRidge(alpha=alpha, kernel='rbf', gamma=gamma)
    KRR.fit(x_train, y_train)
    yhat_KRR = KRR.predict(x_peak) #what he means by predict from whole x peak
    std = np.std(yhat_KRR)
    dev.append(std)
    #ax.plot(x_peak,std, '--', markerfacecolor = 'none')
wavenum = np.linspace(x_peak[0],x_peak[-1],5)
wavenum = wavenum.reshape(-1,1)

ax.plot(wavenum,dev, '--', markerfacecolor = 'none')
ax.set_xlabel('wavenumber [$cm^{-1}$]')
ax.set_ylabel('Standard Deviations of KKR')
#ax.legend(['Original Data', 'Predictions'])
```

Out[165]: Text(0.5, 1.0, '$\\alpha$ = 0.01')



I do not think so, since the standard deviations of the predictions from the error ar
not all the same, and that it varies with the wavenumber. My graph my be wrong, but I