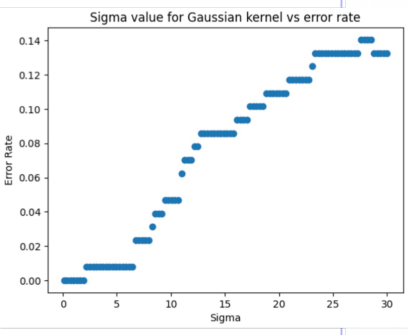
Assignment 9

1. Face Emotion Classification with Kernel Classifier. In this problem you will apply a kernel classifier to the face emotion dataset. You may find it very helpful to use code from an activity.

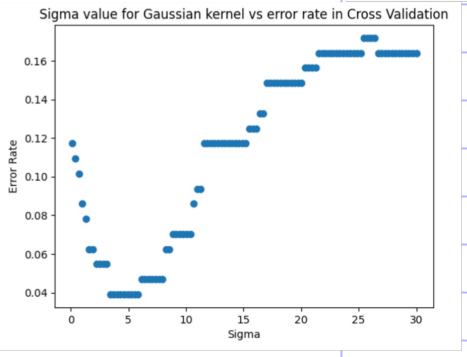


a) Build a kernel classifier using

- the squared error loss function
- an ℓ_2 regularizer with $\lambda = 0.5$.
- the Guassian Kernel $K(\boldsymbol{u}, \boldsymbol{v}) = \exp(-||\boldsymbol{u} - \boldsymbol{v}||^2/(2\sigma^2))$.

b) Train your classifier choosing for different values of σ and create a plot with σ on the horizontal axis and accuracy on the vertical axis and comment on the plot. Does your classifier achieve 0% training error? Yes, with a small σ, <about 2, the classifier has 0% training error.

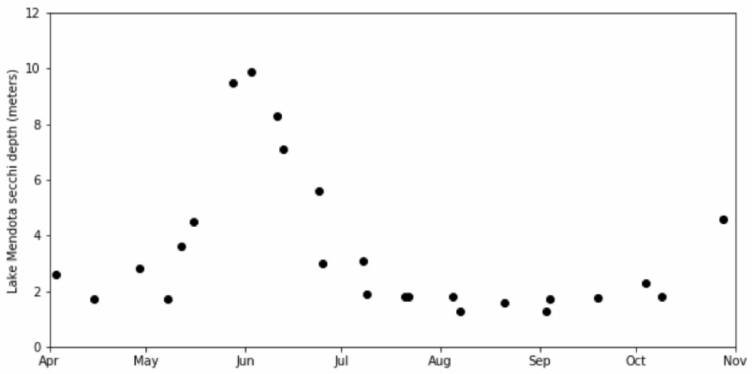
c) Find a more realistic estimate of the accuracy of your classifier by using 8-fold cross validation. Can you achieve perfect test accuracy?



There is no σ that has perfect test accuracy. The lowest error is around 0.04, so the optimal σ is between 3 and 6.

2. Kernel Regression, Lake Mendota Clarity. The *Secchi depth* is a measure of water clarity obtained by lowering a black and white disk off the shady side of a boat and recording the depth at which the disk is no longer visible.

A dataset obtained from the University of Wisconsin's Limnology department contains Secchi disk readings (in meters) on Lake Mendota from 2019 and 2020. A Secchi depth of less than 2 meters is consider poor clarity, while a Secchi depth greater than 6 meters is consider very clear. Lake Mendota can have very clear water in late spring when native zooplankton *daphnia pulicaria* consume large amounts of algae and phytoplankton (for more details, see https://blog.limnology.wisc.edu/2019/06/12/whats-behind-this-extended-phase-of-crazy-clear-water-in-lake-mendota/).

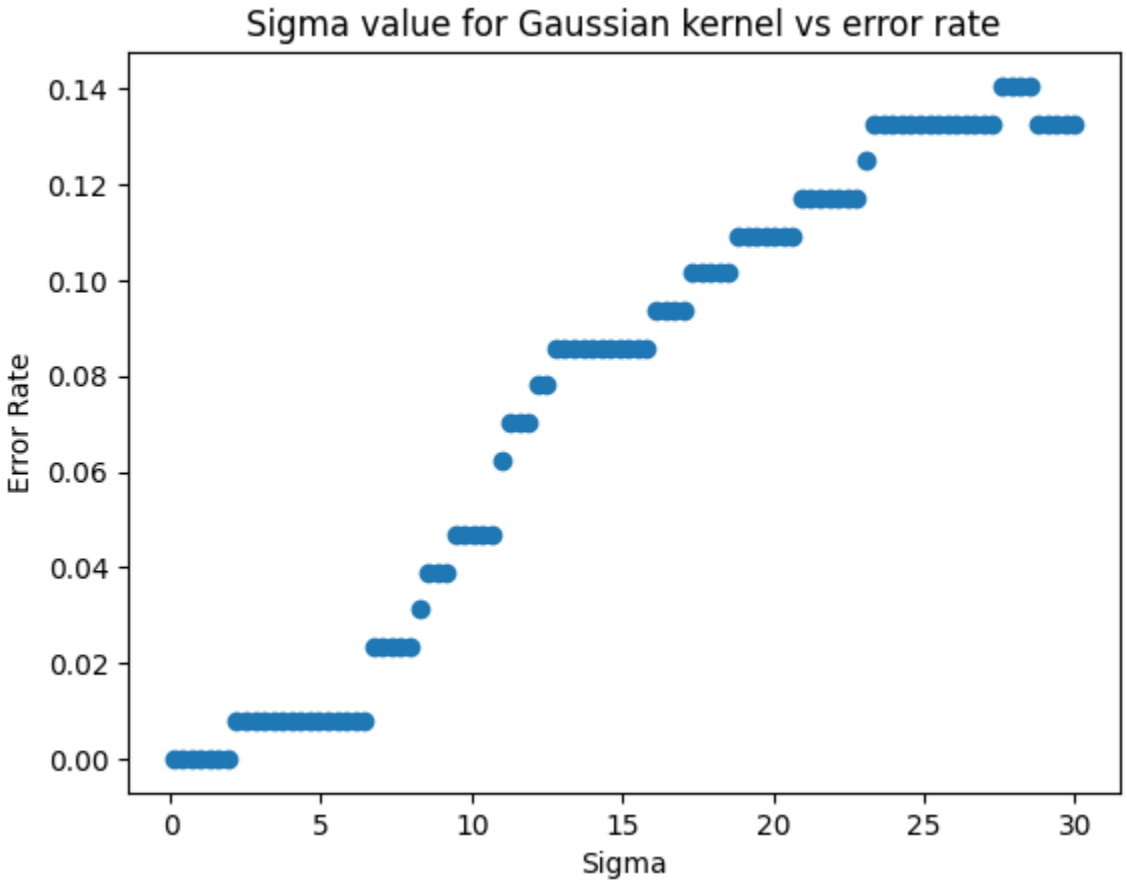


```
import numpy as np
import matplotlib.pyplot as plt
from scipy.io import loadmat

dataset = loadmat('face_emotion_data.mat')

X, y = dataset['X'], dataset['y']
n, p = np.shape(X)

X = np.hstack((np.ones((n,1)), X))  # append a column of ones


lam = 0.5
sig_vals = np.linspace(0.1, 30, 100)

error_rate = []

for sigma in sig_vals:
    distsq=np.zeros((n,n),dtype=float)

    for i in range(0,n):
        for j in range(0,n):
            d = np.linalg.norm(X[i,:]-X[j,:])
            distsq[i,j]=d**2

    K = np.exp(-distsq/(2*sigma**2))
    alpha = np.linalg.inv(K+lam*np.identity(n))@y

    # prediction
    y_hat_pred = []
    g = n
    y_hat_pred = np.zeros((g,1))

    for i,x1 in enumerate(X):
        y_hat_pred[i] = np.exp(-np.linalg.norm(
            X - x1, axis = 1)**2/(2*sigma**2))@alpha

    y_hat = np.sign(y_hat_pred)

    errors = [0 if i[0]==i[1] else 1 for i in np.hstack((y, y_hat))]
    error_rate.append(sum(errors)/len(errors))

for i in range(len(error_rate)):
  if (error_rate[i]>0): break
  print(sig_vals[i])
```

```
0.1
0.40202020202020206
0.704040404040404
1.006060606060606
1.3080808080808082
1.6101010101010103
1.912121212121212
```

```
plt.scatter(sig_vals, error_rate)

plt.title("Sigma value for Gaussian kernel vs error rate")
plt.xlabel("Sigma")
plt.ylabel("Error Rate")
```

```
Text(0, 0.5, 'Error Rate')
```

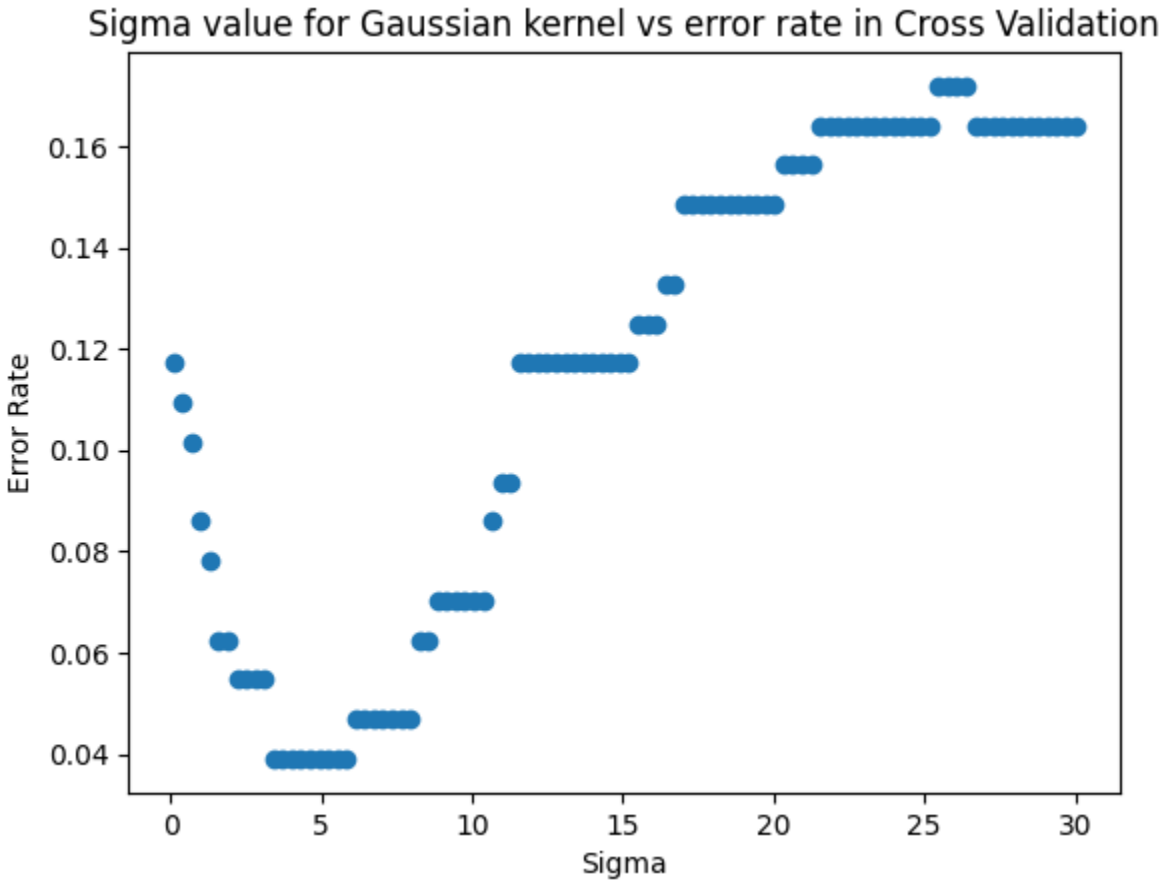


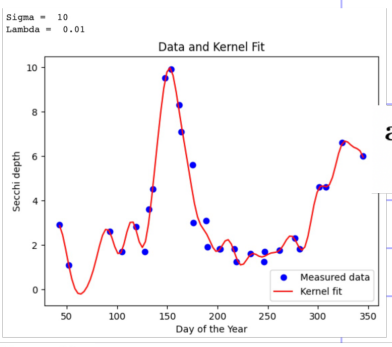
1c: 8-fold

```
lam = 0.5
sig_vals = np.linspace(0.1, 30, 100)

error_rate = []

# for each sig value
for sigma in sig_vals:
  eight_sets = np.split(X, 8)
  eight_y = np.split(y, 8)
  total_error = 0

  # cross validation
  for i in range(8):
    # remove one set from the eight
    training = np.delete(eight_sets, i, axis=0).reshape(-1, 10)
    training_y = np.delete(eight_y, i, axis=0).reshape(112, 1)
```

```
    training_y = np.delete(eight_y, i, axis=0).reshape(112, -1)

    n, p = np.shape(training)
    distsq=np.zeros((n,n),dtype=float)

    for j in range(0,n):
        for k in range(0,n):
            d = np.linalg.norm(training[j,:]-training[k,:])
            distsq[j,k]=d**2

    K = np.exp(-distsq/(2*sigma**2))
    alpha = np.linalg.inv(K+lam*np.identity(n))@training_y

    # prediction on holdout
    y_hat_pred = []
    g = len(eight_sets[i])
    y_hat_pred = np.zeros((g,1))

    for l,x1 in enumerate(eight_sets[i]):
        y_hat_pred[l] = np.exp(-np.linalg.norm(
            training - x1, axis = 1)**2/(2*sigma**2))@alpha

    y_hat = np.sign(y_hat_pred)

    # errors for the holdout set
    errors = [0 if i[0]==i[1] else 1 for i in np.hstack((eight_y[i], y_hat))]
    total_error += sum(errors)/16

  error_rate.append(total_error/8) # average error of the eight holdout sets


plt.scatter(sig_vals, error_rate)

plt.title("Sigma value for Gaussian kernel vs error rate in Cross Validation")
plt.xlabel("Sigma")
plt.ylabel("Error Rate")
```

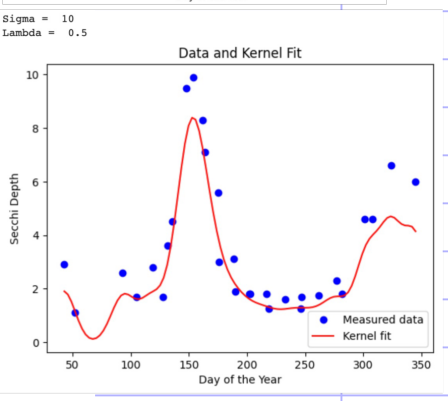
```
Text(0, 0.5, 'Error Rate')
```





a) Use kernel ridge regression with a Gaussian kernel to fit the measurements. You may find it useful to use code from an activity. Use regularization parameter $\lambda = 0.01$ and scale parameter $\sigma = 10$. Plot the resulting fit, and comment on the results. Do these parameters overfit or underfit the data? Adjust the regularization parameter to find a visually better fit.

The regression with these parameters appears to be overfitting the data. The line follows the location of each point rather than a general trend.



A lambda value of 0.5 smooths the line out a bit without any changes to sigma. In general, increasing λ will flatten the curve. However, the trend at the beginning of the year is not well approximated.

Changing both σ and λ would find an even better fit than just changing the regularization parameters.

b) Describe how you could use k-fold cross validation to systematically find a good value of σ and λ.

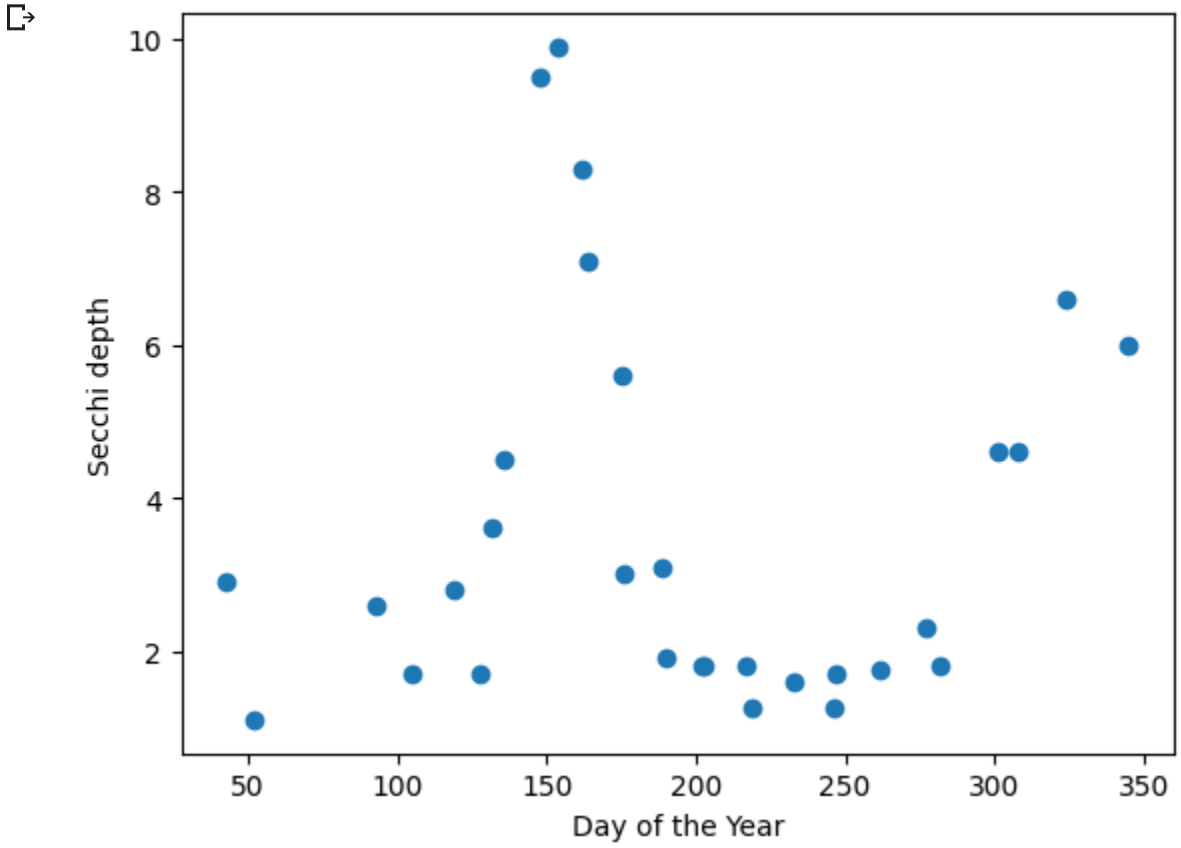
To find a good value for σ and λ, we need to run the k-fold cross validation procedure on multiple combinations of σ and λ.

first 2 loops →
1. Choose a selection of σ and λ values.
2. Fix σ and loop through λ values.

k-fold cross validation loop →
3. Split the 30 point dataset into K groups.
4. Loop through the K groups.
5. Take one group out as a test / holdout group.
6. Train the regression on the remaining k-1 groups.
7. Evaluate on the holdout set and calculate some type of error. Repeat for all k groups.
8. Find the average of the error for this σ and λ combination and repeat the cross validation for all different λ values.
9. Continue to the next value of σ and repeat.

best σ and λ value →
10. The best values for σ and λ will be the combination with the lowest error.

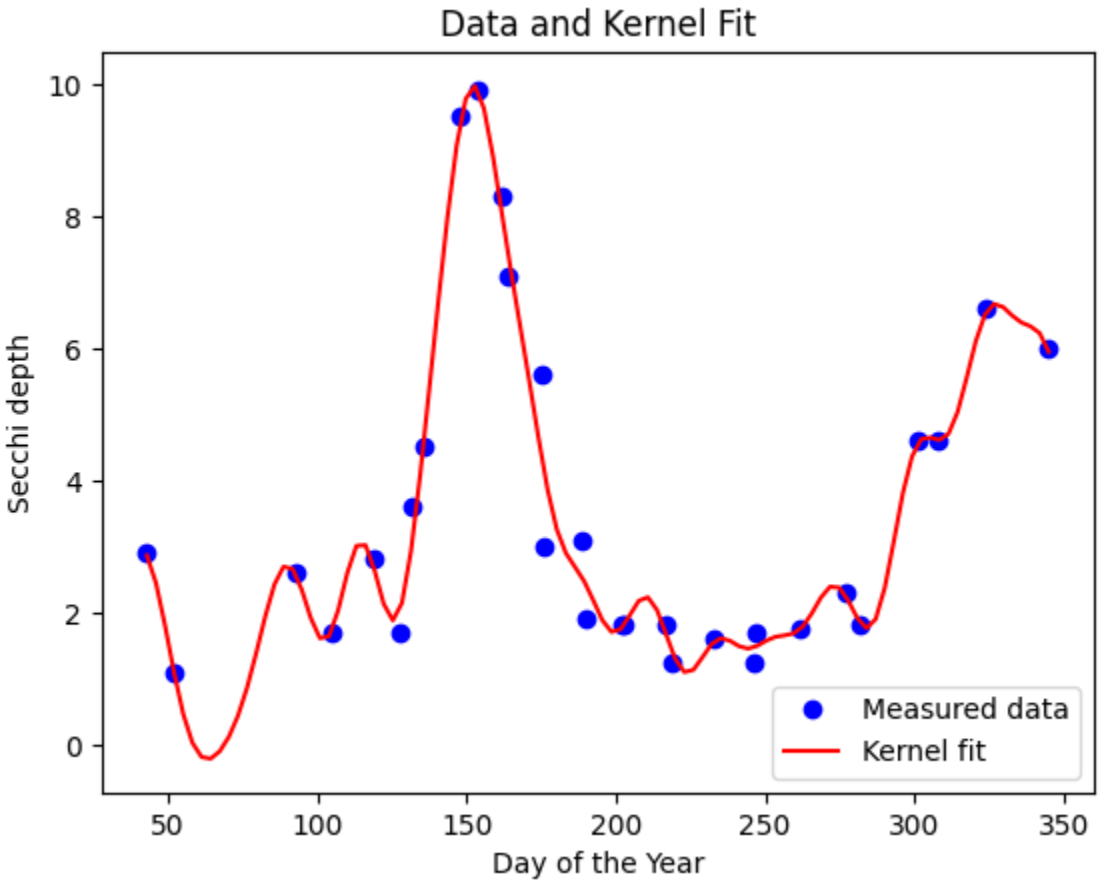
```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd

df = pd.read_csv('mendota_secchi_depth.txt', delimiter='\t')
x = df['day_of_year']
y = df['secchi_depth']


# plot result
plt.scatter(x,y)
plt.xlabel('Day of the Year')
plt.ylabel('Secchi depth')
plt.show()
```




```
# Kernel fitting to data
sigma = 10 #defines Gaussian kernel width
lam = 0.01 #ridge regression parameter
n = 30

distsq=np.zeros((n,n),dtype=float)

for i in range(0,n):
    for j in range(0,n):
        distsq[i,j]=(x[i]-x[j])**2

K = np.exp(-distsq/(2*sigma**2))
```

```
alpha = np.linalg.inv(K+lam*np.identity(n))@y

# Generate smooth curve corresponding to data fit
p = 100
x_test = np.linspace(min(x),max(x),p)  # uniformly sample interval

distsq_xtest = np.zeros((p,n),dtype=float)
for i in range(0,p):
    for j in range(0,n):
        distsq_xtest[i,j] = (x_test[i]-x[j])**2

dtest = np.exp(-distsq_xtest/(2*sigma**2))@alpha

print('Sigma = ',sigma)
print('Lambda = ',lam)
plt.plot(x,y,'bo',label='Measured data')
plt.plot(x_test,dtest,'r',label='Kernel fit')
plt.title('Data and Kernel Fit')
plt.legend(loc='lower right')
plt.xlabel('Day of the Year')
plt.ylabel('Secchi depth')
plt.show()
```

```
Sigma =  10
Lambda =  0.01
```




```
# Kernel fitting to data
sigma = 10 #defines Gaussian kernel width
lam = 0.5 #ridge regression parameter
```

```
lam = 0.5 #ridge regression parameter
n = 30

distsq=np.zeros((n,n),dtype=float)

for i in range(0,n):
    for j in range(0,n):
        distsq[i,j]=(x[i]-x[j])**2

K = np.exp(-distsq/(2*sigma**2))

alpha = np.linalg.inv(K+lam*np.identity(n))@y

# Generate smooth curve corresponding to data fit
p = 100
x_test = np.linspace(min(x),max(x),p)  # uniformly sample interval

distsq_xtest = np.zeros((p,n),dtype=float)
for i in range(0,p):
    for j in range(0,n):
        distsq_xtest[i,j] = (x_test[i]-x[j])**2

dtest = np.exp(-distsq_xtest/(2*sigma**2))@alpha

print('Sigma = ',sigma)
print('Lambda = ',lam)
plt.plot(x,y,'bo',label='Measured data')
plt.plot(x_test,dtest,'r',label='Kernel fit')
plt.title('Data and Kernel Fit')
plt.legend(loc='lower right')
plt.xlabel('Day of the Year')
plt.ylabel('Secchi Depth')
plt.show()
```

```
Sigma =  10
Lambda =  0.5
```