hw6-programming-solved

November 9, 2024

1 Kernel Regression

Given a training dataset $\{x_i, y_i\}_{i=1}^n$, kernel regression approximates the unknown nolinear relation between x and y with a function of form

$$y \approx f(x; w) = \sum_{i=1}^{n} w_i k(x, x_i),$$

where $k(x, x')$ is a positive definite kernel specified by the users, and w_i is a set of weights. We will use the simple Gaussian radius basis funciton (RBF) kernel,

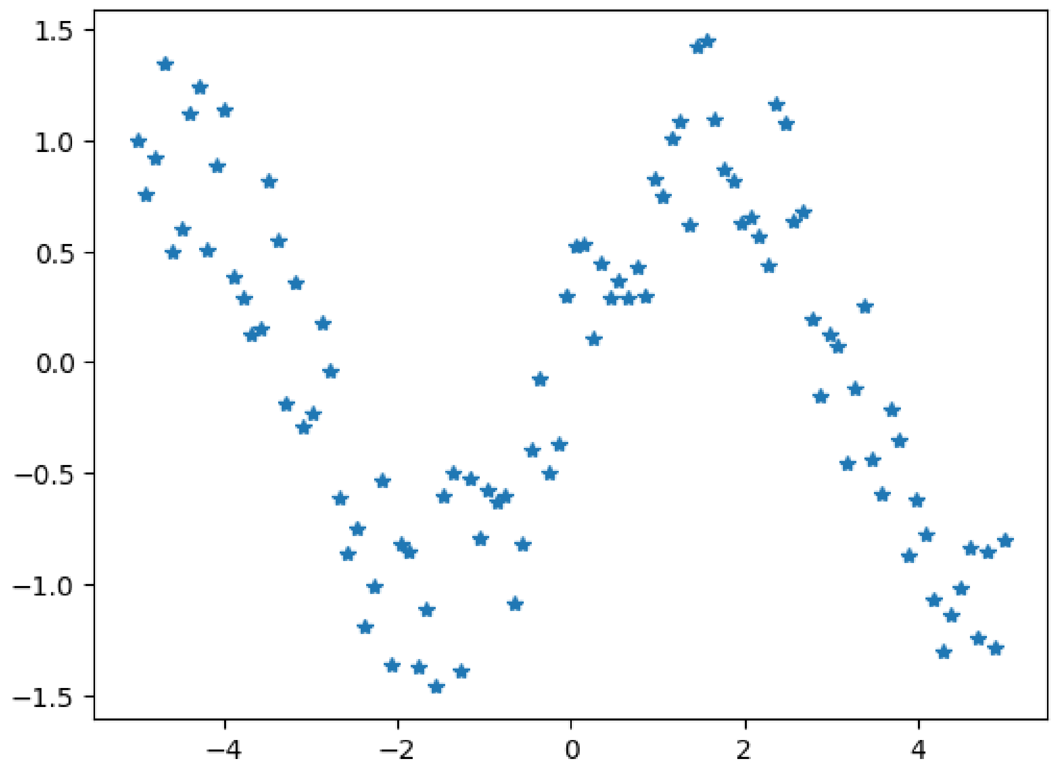
$$k(x, x') = exp(-\frac{||x - x'||^2}{2h^2}),$$

where h is a bandwith parameter.

1.0.1 Step 1. Simulate a 1-dimensional dataset

[28]:
```
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
%matplotlib inline

np.random.seed(100)

### Step 1: Simulate a simple 1D data ###
xTrain = np.expand_dims(np.linspace(-5, 5, 100), 1)  # 100*1
yTrain = np.sin(xTrain) + 0.5*np.random.uniform(-1, 1, size=xTrain.shape) ##␣
 ↪100 *1

print('xTrain shape', xTrain.shape, 'yTrain shape', yTrain.shape)
plt.plot(xTrain, yTrain, '*')
plt.show()
```

```
xTrain shape (100, 1) yTrain shape (100, 1)
```



Now we have a dataset with 100 training data points. Let us calculate the kernel function.

1.0.2 Step 2. Kernel function

Your task is to complete the following rbf_kernel function that takes two sets of points X (of size n) and X' (of size m) and the bandwidth h and ouputs their pairwise kernel matrix $K = [k(x_i, x_j)]_{ij}$, which is of size $n \times m$. We will represent input data as matrices, with $X = [x_i]_{i=1}^n \in R^{n\times 1}$ denoting the input features and $Y = [y_i]_{i=1}^n \in R^{n\times 1}$ the input labels.)

[29]:
```
"""
    calcuating kernel matrix between X and Xp
"""
def rbf_kernel(X, Xp, h):
    # X: n*1 matrix
    # Xp: m*1 matrix
    # h: scalar value

    dists = np.square(X - Xp.T)        # Obtain pairwise differences between X␣
↪and Xp,
                                       # then square to get pairwise squared␣
↪distances
    K = np.exp(-dists/(2 * h**2))      # Apply the RBF kernel formula
```

```
    return K # n*m matrix



### evaluation: if your implementation is correct, you should expect the output␣
 ↪is a 2X3 matrix
# [[0.60653066 1.         0.60653066]
# [0.13533528 0.60653066 1.        ]]
k_test = rbf_kernel(np.array([[2],[1]]), np.array([[3],[2],[1]]), 1)
print(k_test)
```

```
[[0.60653066 1.         0.60653066]
 [0.13533528 0.60653066 1.        ]]
```

1.0.3 Step 3. The median trick for bandwith

The choice of the bandwidth h A common way to set the bandwith h in practice is the so called median trick, which sets h to be the median of the pairwise distance on the training data, that is

$$h_{med} = median(\{||x_i - x_j|| : i \neq j, \qquad i, j = 1, ..., n\}).$$

- Task: Compelete the median distance function.

[30]:
```
from scipy.spatial import distance

def median_distance(X):
    # X: n*1 matrix

    #TODO: Calculate the median of the pairwise distance of $X$ below
    #(hint: use '[dist[i, j] for i in range(len(X)) for j in range(len(X)) if i␣
 ↪!= j]' to remove the diagonal terms; use np.median)

    # Compute pairwise distances using scipy's distance function, resulting in␣
 ↪n x n matrix
    pairwise_distances = distance.cdist(X, X, metric='euclidean')

    # Extract all non-self-distance elements (i.e., i != j)
    off_diagonal_distances = [
        pairwise_distances[i, j]
        for i in range(len(X))
        for j in range(len(X))
        if i != j
    ]

    # Compute the median of these distances
    h = np.median(off_diagonal_distances)
```

```python
    return h


### Test your functions
#evaluation: if your implementation is correct, your answer should be [2.0]
h_test = median_distance(np.array([[1],[2],[4]]))
print(h_test)
```

```
2.0
```

1.0.4 Step 4. Kernel regression

The weights w_i are estimated by minimizing a regularized mean square error:

$$\min_{w} \left(\sum_{i=1}^{n} (y_i - f(x_i; w))^2 \right) + \beta w^\top K w,$$

where w is the column vector formed by $w = [w_i]_{i=1}^{n}$ and K is the kernel matrix.

- Please derive the optimal solution of w using matrix inverseion (no need to show the work)
- Complete the following function to implement the calculation of w

[31]:
```python
def kernel_regression_fitting(xTrain, yTrain, h, beta=1):
    # X: input data, numpy array, n*1
    # Y: input labels, numpy array, n*1

    # Step 1: Compute the kernel matrix K (n x n)
    K = rbf_kernel(xTrain, xTrain, h)

    # Step 2: Add the regularization term to the kernel matrix
    n = K.shape[0]
    K_regularized = K + beta * np.eye(n)

    # Step 3: Compute the weights w using matrix inversion
    W = np.linalg.inv(K_regularized).dot(yTrain)

    return W

### evaluating your code, the shape should be (100, 1) (check the values␣
↪yourself)
h = median_distance(xTrain)
W_test = kernel_regression_fitting(xTrain, yTrain, h)
print(W_test.shape)
```

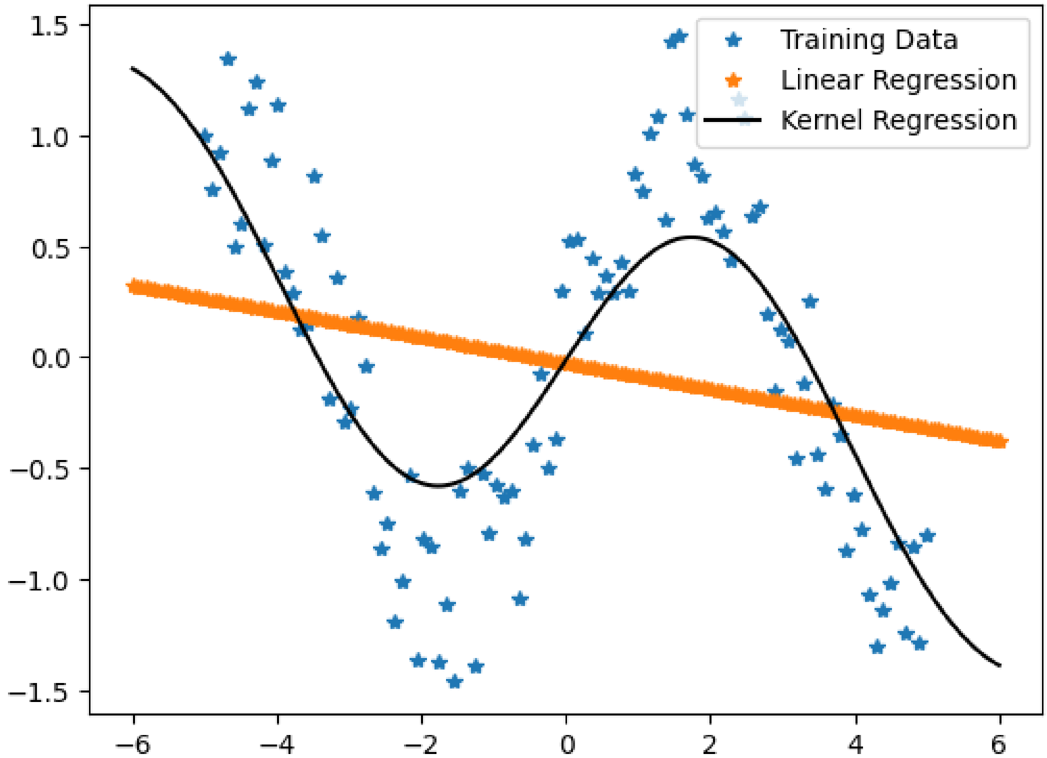
```
(100, 1)
```

1.0.5 Step 5. Evaluation and Cross Validation

We now need to evaluate the algorithm on the testing data and select the hyperparameters (bandwidth and regularization coefficient) using cross validation

[32]:

```python
# Please run and read the following base code

def kernel_regression_fit_and_predict(xTrain, yTrain, xTest, h, beta):

    #fitting on the training data
    W = kernel_regression_fitting(xTrain, yTrain, h, beta)

    # computing the kernel matrix between xTrain and xTest
    K_xTrain_xTest = rbf_kernel(xTrain, xTest, h)

    # predict the label of xTest
    yPred = np.dot( K_xTrain_xTest.T, W)
    return yPred

# generate random testing data
xTest = np.expand_dims(np.linspace(-6, 6, 200), 1) ## 200*1


beta = 1.
# calculating bandwith
h_med = median_distance(xTrain)
yHatk = kernel_regression_fit_and_predict(xTrain, yTrain, xTest, h_med, beta)


# we also add linear regression for comparision
from sklearn.linear_model import LinearRegression
lr = LinearRegression()
lr.fit(xTrain, yTrain)
yHat = lr.predict(xTest) # prediction

# visulization
plt.plot(xTrain, yTrain, '*', label='Training Data')
plt.plot(xTest, yHat, '*', label='Linear Regression')
plt.plot(xTest, yHatk, '-k', label='Kernel Regression')
plt.legend()
plt.show()
```



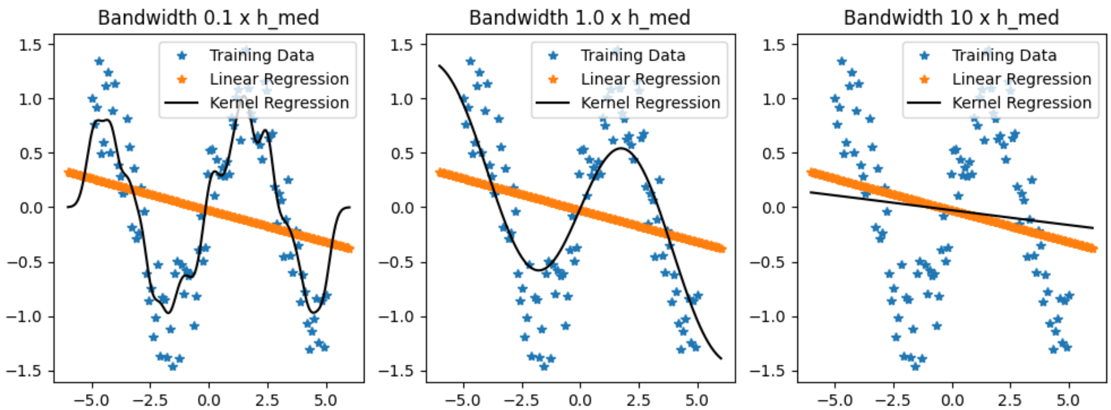
1.0.6 Step 5.1. Impact of bandwith

Run the kernel regression with regularization coefficient $\beta = 1$ and bandwidth $h \in \{0.1h_{med}, h_{med}, 10h_{med}\}$.

- Task: Show the curve learned by different h. Comment on how h influences the smoothness of h.

[33]:
```
### fitting on the training data ###
beta = 1

plt.figure(figsize=(12, 4))

for i, coff in enumerate([0.1, 1., 10]):
    plt.subplot(1, 3, i+1)

    ### TODO: run kernel regression with bandwith h = coff * h_med.
    h = coff * h_med
    yHatk_i = kernel_regression_fit_and_predict(xTrain, yTrain, xTest, h, beta)

    # visulization
    plt.plot(xTrain, yTrain, '*', label='Training Data')
    plt.plot(xTest, yHat, '*', label='Linear Regression')
```

```
    plt.plot(xTest, yHatk_i, '-k', label='Kernel Regression')
    plt.title('Bandwidth {} x h_med'.format(coff))
    plt.legend()

plt.show()
```



1.0.7 How the Bandwidth, h, Influences the Smoothness of the Curve

It is apparent that the higher the value of h, the smoother the curve will be. When $h = 0.1 \times h_med$, we can see that the curve is quite rough, but has the advantage of fitting the training data very closely. However, it should be noted that this may lead to overfitting.

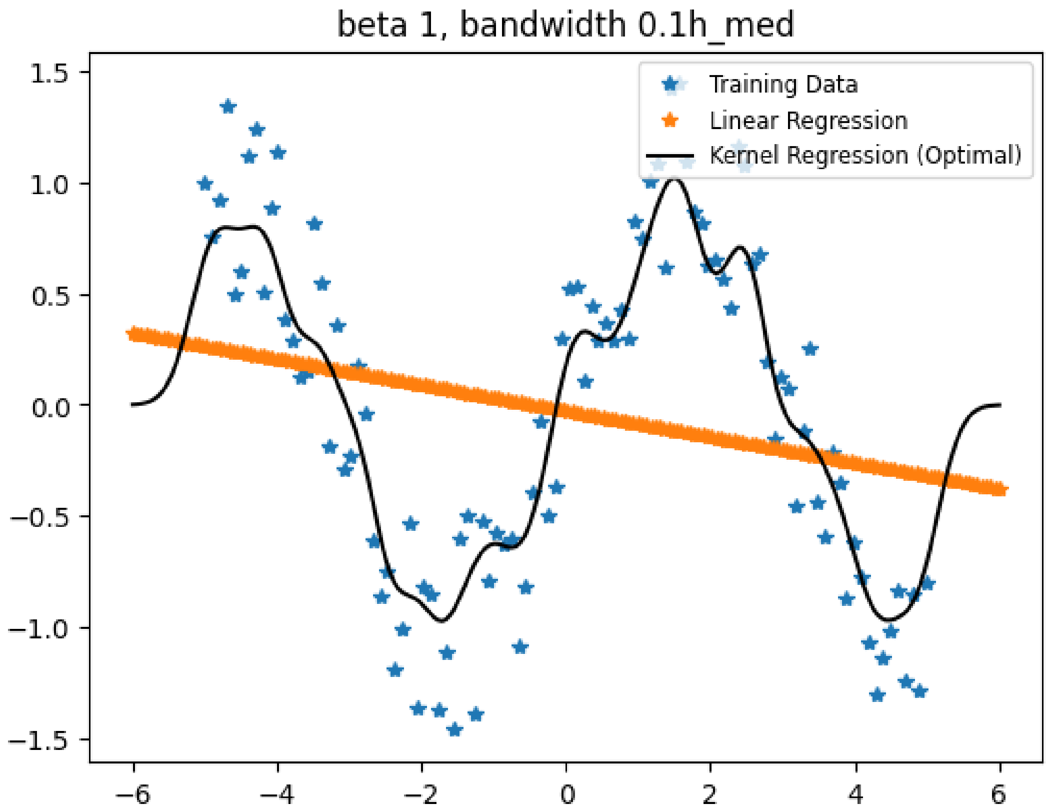
1.0.8 Step 5.2. Cross Validation (CV)

Use 5-fold cross validation to find the optimal combination of h and β within $h \in \{0.1h_{med}, h_{med}, 10h_{med}\}$ and $\beta \in \{0.1, 1\}$. - Task: complete the code of cross validation and find the best h and β. Plot the curve fit with the optimal hyperparameters.

[34]:
```
best_beta, best_coff = 1., 1.
best_mse = 1e8

for beta in [0.1, 1]:
    for coff in [0.1, 1., 10.]:
        # 5-fold cross validation
        max_fold = 5
        mse = []

        for i in range(max_fold):

            ##TODO: calculate the index of the training/testing partition␣
↪within 5 fold CV.
            # (hint: set trnIdx to be these index with idx%max_fold!=i, and␣
↪testIdx with idx%max_fold==i)
```

```python
            trnIdx = [j for j in range(len(xTrain)) if j % max_fold != i]
            testIdx = [j for j in range(len(xTrain)) if j % max_fold == i]

            i_xTrain, i_yTrain = xTrain[trnIdx], yTrain[trnIdx]
            i_xValid, i_yValid = xTrain[testIdx], yTrain[testIdx]


            ##TODO: run kernel regression on (i_xTrain, i_yTrain) and calculate␣
 ↪the mean square error on (i_xValid, i_yValid)
            h = coff * h_med
            i_yPred = kernel_regression_fit_and_predict(i_xTrain, i_yTrain,␣
 ↪i_xValid, h, beta)
            mse.append((i_yValid - i_yPred)**2)

        mse = np.mean(mse)

        # keep track of the combination with the best MSE
        if mse < best_mse:
            best_beta, best_coff = beta, coff
            best_mse = mse

print('Beta beta', best_beta, 'Best bandwith', '{}*h_med'.format(best_coff),␣
 ↪'mse', best_mse)

# bandwith
h = best_coff * median_distance(xTrain)
yHatk_i = kernel_regression_fit_and_predict(xTrain, yTrain, xTest, h, best_beta)

# visulization
plt.plot(xTrain, yTrain, '*', label='Training Data')
plt.plot(xTest, yHat, '*', label='Linear Regression')
plt.plot(xTest, yHatk_i, '-k', label='Kernel Regression (Optimal)')
plt.title('beta {}, bandwidth {}h_med'.format(best_beta, best_coff))
plt.legend(loc='upper right', fontsize='small')
plt.show()
```

```
Beta beta 1 Best bandwith 0.1*h_med mse 0.1116622935589619
```


beta 1, bandwidth 0.1h_med
Training Data
Linear Regression
Kernel Regression (Optimal)
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−6
−4
−2
0
2
4
6