Assignment 4: Kernels

CS 589 - ML

Shubham Shetty (shubhamshett@umass.edu)
Brinda Murulidhara (bmurulidhara@umass.edu)
Adarsh Kolya (akolya@umass.edu)

April 2021

Preface

Before running our codes, the following packages are imported and test & training data are assigned to variables. Also, some reusable functions are defined -

```
# Import Statements
import numpy as np
from scipy.special import comb
from matplotlib import pyplot as plt
from sklearn import svm
from sklearn.model_selection import KFold
from sklearn.metrics import hinge_loss
from sklearn.svm import SVC


# Load Data
## 1D Data
data = np.load("data_synth.npz")
X_trn = data_synth['X_trn']           # 1D Input Training Dataset
Y_trn = data_synth['Y_train']         # 1D Output Training Dataset
X_val = data_synth['X_val']           # Test Input Dataset
Y_val = data_synth['Y_val']           # Test Output Dataset

## Big data for SVM testing
data_real = np.load("data_real.npz")
X_trn_real = data_real['x_trn']       # 686 training inputs of length
    4
Y_trn_real = data_real['y_trn']       # 686 training outputs of
   length 4
X_val_real = data_real['x_tst']       # 686 test inputs


# Function to perform basis expansion
def get_poly_expansion(P):
    def expand(X):
        tmp = [np.sqrt(comb(P, p)) * X ** p for p in range(P + 1)]
        return np.vstack(tmp).T
    return expand
```

Answer 1

The objective is to find a vector w^* that minimizes :

$$\sum_{\mathrm{n}=1}^{\mathrm{N}} \left(w \cdot \mathrm{x}^{(\mathrm{n})} - \mathrm{y}^{(\mathrm{n})}\right)^2 + \lambda \|w\|^2$$

Let's rewrite this in matrix form. Let y be a single vector containing all of our outputs:

$$y = (y^{(1)}, ..., y^{(N)})$$

Consider X to be a matrix with N rows that is formed by $x^{(1)}, x^{(2)} ... x^{(N)}$.

$$X = \begin{bmatrix} (x^{(1)})^\top \\ \dots \\ (x^{(N)})^\top \end{bmatrix}$$

The above equation $R(w)$ can be rewritten as :

$$\begin{aligned} R(w) &= \|\mathrm{X} \cdot w - \mathrm{y}\|^2 + \lambda \|w\|^2 \\ &= (\mathrm{X} \cdot w - \mathrm{y})^\top (\mathrm{X} \cdot w - \mathrm{y}) + \lambda w^\top w \end{aligned}$$

At w^*,

$$\begin{aligned} &\nabla \mathrm{R}(\mathrm{w}^*) = 0 \\ &2\mathrm{X}^\top (\mathrm{X}w^* - \mathrm{y}) + 2\lambda w^* = 0 \\ &\mathrm{X}^\top \mathrm{X} w^* + \lambda w^* = \mathrm{X}^\top \mathrm{y} \\ &(\mathrm{X}^\top \mathrm{X} + \lambda \mathrm{I}) \cdot w^* = \mathrm{X}^\top \mathrm{y} \end{aligned}$$

Hence,

$$w^* = (\mathrm{X}^\top \mathrm{X} + \lambda \mathrm{I})^{-1} \mathrm{X}^\top \mathrm{y}$$

The above can be rewritten as :

$$\mathrm{w}^* = \left(\sum_{\mathrm{n}=1}^{\mathrm{N}} \left(\mathrm{x}^{(\mathrm{n})}\right) \left(\mathrm{x}^{(\mathrm{n})}\right)^\top + \lambda \mathrm{I}\right)^{-1} \sum_{\mathrm{n}=1}^{\mathrm{N}} \mathrm{x}^{(\mathrm{n})} \mathrm{y}^{(\mathrm{n})}.$$

Answer 2

The objective is to find a vector w^* that minimizes :

$$\sum_{\mathrm{n=1}}^{\mathrm{N}} \left(w \cdot h(\mathrm{x}^{(\mathrm{n})}) - \mathrm{y}^{(\mathrm{n})}\right)^2 + \lambda \|w\|^2$$

Let's rewrite this in matrix form. Let y be a single vector containing all of our outputs:

$$y = (y^{(1)}, ..., y^{(N)})$$

Consider H to be a matrix with N rows that is formed by $h(x^{(1)}), h(x^{(2)})...h(x^{(N)})$.

$$H = \begin{bmatrix} h(x^{(1)})^\top \\ \cdots \\ h(x^{(N)})^\top \end{bmatrix}$$

The above equation $R(w)$ can be rewritten as :

$$\begin{aligned} R(w) &= \|\mathrm{H} \cdot w - \mathrm{y}\|^2 + \lambda \|w\|^2 \\ &= (\mathrm{H} \cdot w - \mathrm{y})^\top (\mathrm{H} \cdot w - \mathrm{y}) + \lambda w^\top w \end{aligned}$$

At w^*,

$$\begin{aligned} &\nabla \mathrm{R}(\mathrm{w}^*) = 0 \\ &2\mathrm{H}^\top(\mathrm{H}w^* - \mathrm{y}) + 2\lambda w^* = 0 \\ &\mathrm{H}^\top \mathrm{H} w^* + \lambda w^* = \mathrm{H}^\top \mathrm{y} \\ &(\mathrm{H}^\top \mathrm{H} + \lambda \mathrm{I}) \cdot w^* = \mathrm{H}^\top \mathrm{y} \end{aligned}$$

Hence,

$$w^* = (\mathrm{H}^\top \mathrm{H} + \lambda \mathrm{I})^{-1} \mathrm{H}^\top \mathrm{y}$$

The above can be rewritten as :

$$\mathrm{w}^* = \left(\sum_{\mathrm{n=1}}^{\mathrm{N}} \left(\mathrm{h}(\mathrm{x}^{(\mathrm{n})})\right) \left(\mathrm{h}(\mathrm{x}^{(\mathrm{n})})\right)^\top + \lambda \mathrm{I}\right)^{-1} \sum_{\mathrm{n=1}}^{\mathrm{N}} \mathrm{h}(\mathrm{x}^{(\mathrm{n})}) \mathrm{y}^{(\mathrm{n})}.$$

Answer 3

The objective is to represent $y^{\text{pred}} = w^* \cdot h(x^{\text{pred}})$ in terms of some kernel function k such that $k(x, x') = h(x) \cdot h(x')$ without referencing $h(\cdot)$ or w^*.

From previous solution, we know that

$$w^* = (\mathrm{H}^\top \mathrm{H} + \lambda \mathrm{I})^{-1} \mathrm{H}^\top \mathrm{y}$$

where H is a matrix with N rows that is formed by $h(x^{(1)}), h(x^{(2)})...h(x^{(N)})$ and y is a single vector containing all of our outputs.

It is known that for given matrices P & Q,

$$(PQ + I)^{-1} P = P(QP + I)^{-1}$$

Using this equality, we get

$$w^* = \mathrm{H}^\top (\mathrm{H}\mathrm{H}^\top + \lambda \mathrm{I})^{-1} \mathrm{y}$$

An element at position (n,m) in $HH^\top$ is equal to $h(x^{(n)}) \cdot h(x^{(m)})$.

Let us define a matrix K with an every element $K_{nm} = k(x^{(n)}, x^{(m)})$.

Since $h(x^{(n)}) \cdot h(x^{(m)}) = k(x^{(n)}, x^{(m)})$, $HH^\top = K$

Let us define $\alpha = (K + \lambda I)^{-1} y$. Hence,

$$w^* = \mathrm{H}^\top \alpha = \mathrm{h(x)} \cdot \alpha$$

Naive prediction can be made as,

$$y^{\text{pred}} = w^* \cdot h(x^{\text{pred}})$$

Substituting value of w^* we get,

$$\begin{aligned} & y^{\text{pred}} = h(x) \cdot \alpha \cdot h(x^{\text{pred}}) \\ \implies & y^{\text{pred}} = \alpha \cdot h(x) \cdot h(x^{\text{pred}}) \\ \implies & y^{\text{pred}} = \alpha \cdot k(x, x^{\text{pred}}) \end{aligned}$$

Hence $y^{\text{pred}} = \alpha \cdot k(x, x^{\text{pred}})$, where α and the kernel function are both independent of w^* and the basis expansion function $h(\cdot)$.

Answer 4

Given that

$$h(x) = [c_0, c_1 x, c_2 x^2, \cdots, c_P x^P], c_p = \sqrt{\binom{P}{p}}$$

Claim : $k(x, x') = (1 + x \cdot x')^P$

Proof :

$$\begin{aligned} h(x) \cdot h(x') &= [c_0, c_1 x, c_2 x^2, \cdots, c_P x^P].[c_0, c_1 x', c_2 x'^2, \cdots, c_P x'^P] \\ &= \sum_{p=0}^{P} \left(c_n^2 x^n x'^n\right) \\ &= \sum_{p=0}^{P} \binom{P}{p} (x \cdot x')^p \end{aligned} \tag{1}$$

Equation (1) can be simplified using binomial theorem to -

$$\begin{aligned} h(x) \cdot h(x') &= (1 + x \cdot x')^P \\ &= k(x, x') \end{aligned}$$

$$\boxed{k(x, x') = (1 + x \cdot x')^P}$$

Answer 5

```
def eval_basis_expanded_ridge(x, w, h):
    y = np.dot(w, h(x)[0])
    return y
```

Answer 6

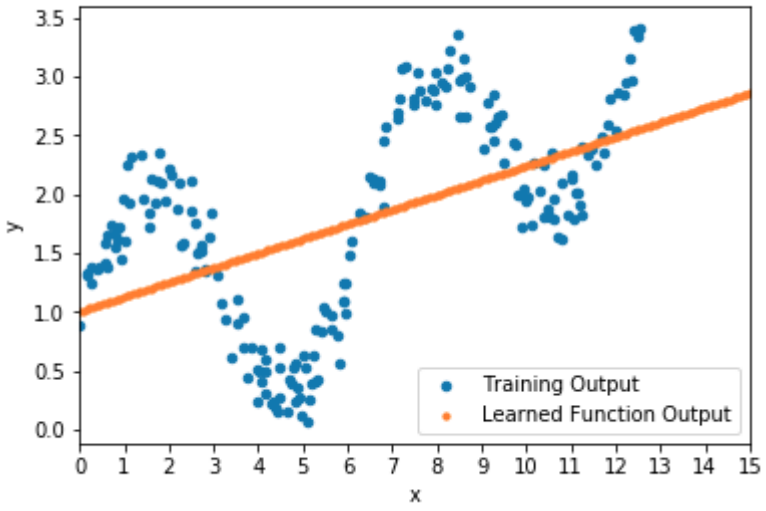
```
def train_basis_expanded_ridge(X, Y, l, h):
    H = h(X)

    C = np.zeros((H[0].shape[0], H[0].shape[0]))
    for item in H:
        C = np.add(C, np.matmul(np.array([item]).T, np.array([item]))
    )

    I = np.identity(len(C))

    w = np.linalg.solve(np.add(C, l * I),
                        np.matmul(np.transpose(H), Y))

    return w
```

Answer 7

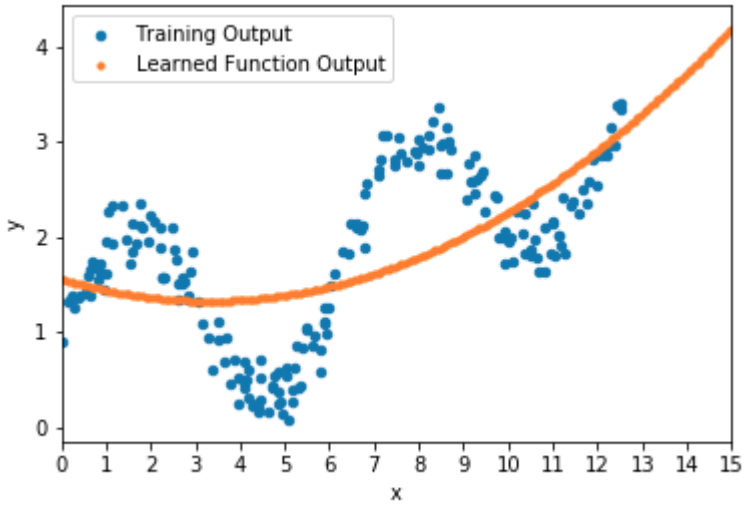
```
def basis_expanded_ridge_reg():
    for P in [1, 2, 3, 5, 10]:
        # Define polynomial basis expansion function
        h = get_poly_expansion(P)

        # Train basis expanded ridge regression function
        W = train_basis_expanded_ridge(X_trn, Y_trn, 0.1, h)

        # Print optimal weights and generate plot
        print(f"P: {P}\nW: {W}\n")
        Y = []
        x_new = np.linspace(0,15,200)
        H = h(x_new)
        for x in x_new:
            Y.append(eval_basis_expanded_ridge(x,W,h))
        Y = np.array(Y)
        plt.figure(1, figsize=(6, 4))
        plt.scatter(X_trn, Y_trn, s=20, label="Training Output")
        plt.scatter(x_new, Y, s=10, label="Learned Function Output")
        plt.xlim([0, 15])
        plt.xlabel("x")
        plt.ylabel("y")
        plt.xticks(np.arange(0, 16, 1))
        plt.legend()
        plt.show()
```

Below are the values of W seen for each value of P and corresponding plot of the final learned function as a function of x superimposed on the training data-

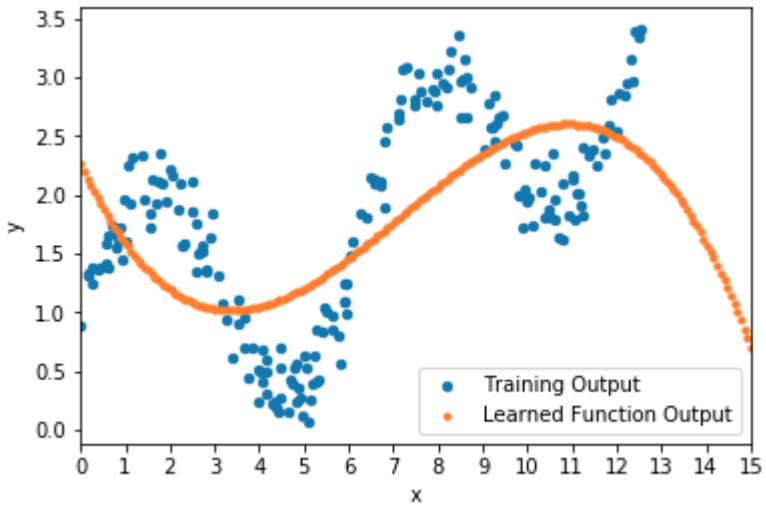
1. P = 1
   W = [1.00565302, 0.12351259]



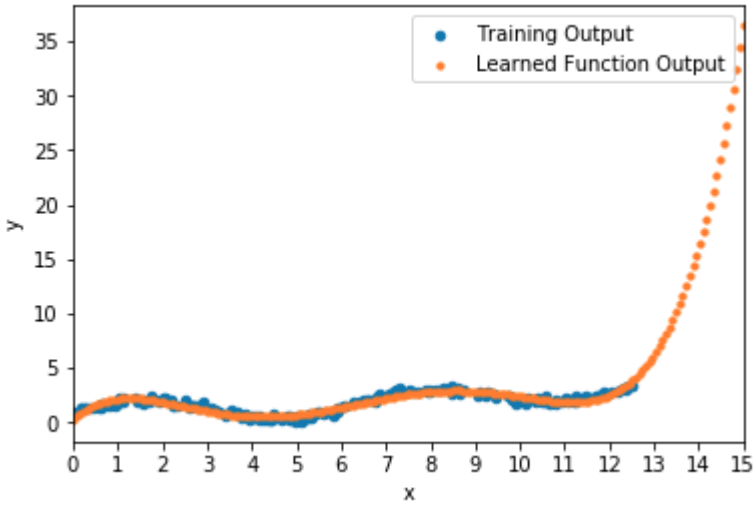
2. P = 2
   W = [1.55636445, -0.09905134, 0.02105954]



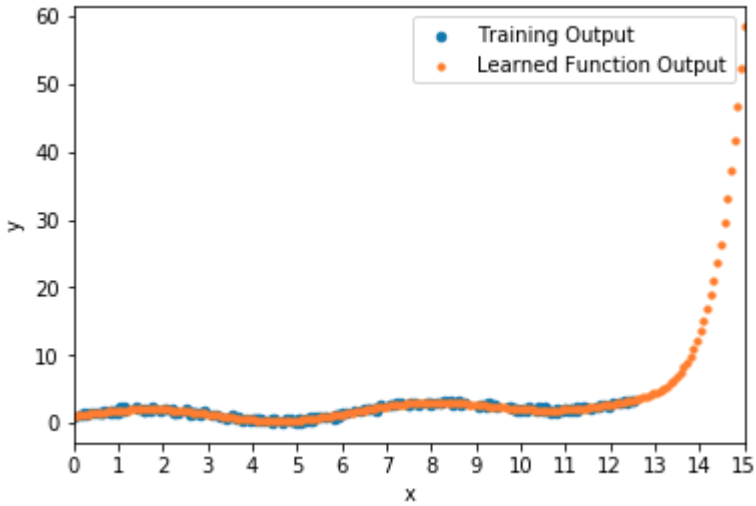
3. P = 3
   W = [2.2585207, -0.4731082, 0.09166343, -0.0074039]



4. P = 5
   W = [2.32031732e-01, 1.70733532e+00, -7.50673264e-01 1.66018570e-01, -2.11820885e-02, 1.49990203e-03]



5. P = 10
   W = [1.00199719e+00, 3.46174032e-01, -6.79997048e-02, 4.04746152e-02, -2.41324409e-02, 7.30654951e-03, -1.30797499e-03, 1.49023358e-04, -1.05582815e-05, 3.71804211e-07, 2.73112041e-09]



Answer 8

```
def get_poly_kernel(P):
    def k(x,xp):
        kernel_value = (1+np.inner(x, xp))**P
        return kernel_value
    return k
```

Answer 9

```
x   = 0.5
xp  = 0.7
k   = get_poly_kernel(5)
h   = get_poly_expansion(5)
out1 = k(x,xp)
out2 = np.inner(h(x),h(xp))
print("output 1", out1)
print("output 2", out2)
```

The outputs observed are -

- output 1 **4.484033437500002**
- output 2 [[**4.48403344**]]

Answer 10

```
from preface import *

def train_kernel_ridge(X,Y,l,k) :
    N = X.shape[0]

    # Compute K
    K = np.empty((N,N))
    for n in range(N) :
        for m in range(N) :
            K[n][m] = k(X[n], X[m])

    a = np.linalg.solve(K + l*np.identity(N), Y)
    return a
```

Answer 11

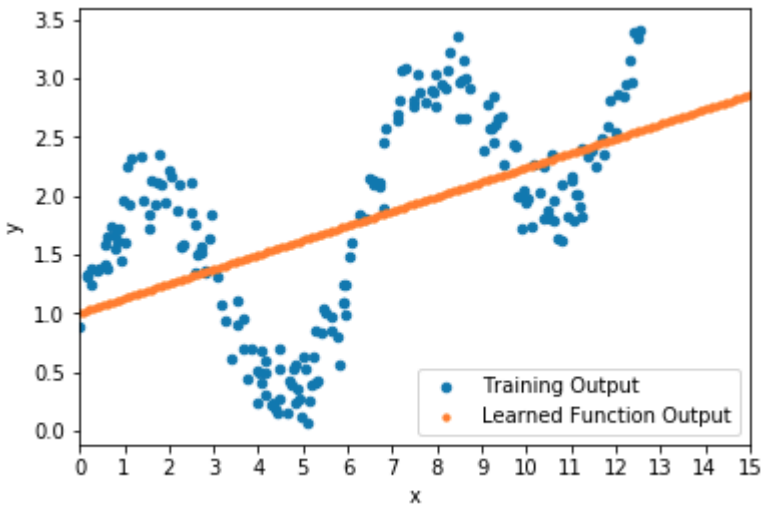
```
def eval_kernel_ridge(x_trn, x, a, k):
    y = np.dot(a, np.array(list(map(lambda x1: k(x1, x), x_trn))))
    return y
```

Answer 12

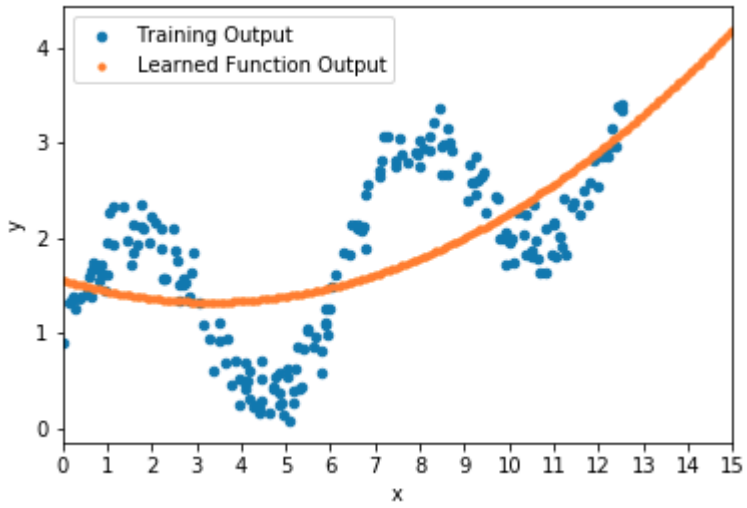
```
def kernel_ridge_reg():
    for P in [1, 2, 3, 5, 10]:
        # Define polynomial kernel function
        k = get_poly_kernel(P)

        # Train Kernel Ridge Regression Model
        a = train_kernel_ridge(X_trn, Y_trn, 0.1, k)
        print(f"P: {P}")
        Y = []
        X_new = np.linspace(0,15,200)

        # Evaluate for new data
        for X in X_new:
            Y.append(eval_kernel_ridge(X_trn, X, a, k))
        Y = np.array(Y)

        # Plot results
        plt.figure(1, figsize=(6, 4))
        plt.scatter(X_trn, Y_trn, s=20, label="Training Output")
        plt.scatter(X_new, Y, s=10, label="Learned Function Output")
        plt.xlim([0, 15])
        plt.xlabel("x")
        plt.ylabel("y")
        plt.xticks(np.arange(0, 16, 1))
        plt.legend()
        plt.show()
```

Plot of the final learned function as a function of x superimposed on the training data for each value of P is given below -

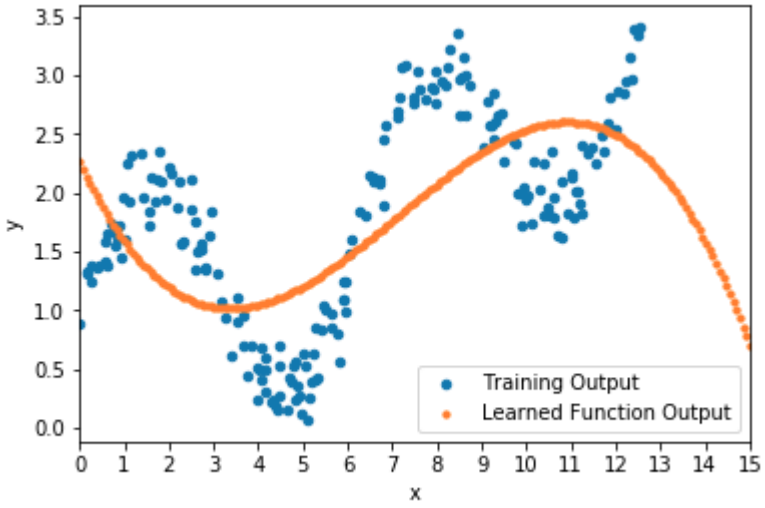
1. P = 1



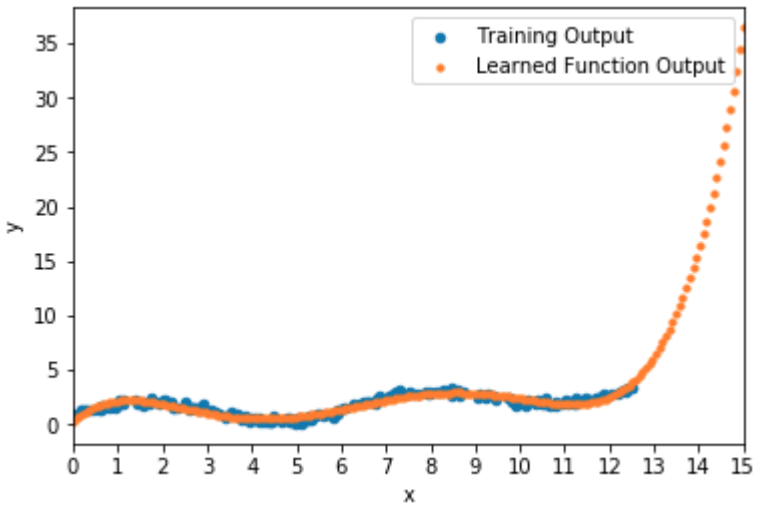
2. P = 2


Training Output
Learned Function Output
y
x
0 1 2 3 4
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

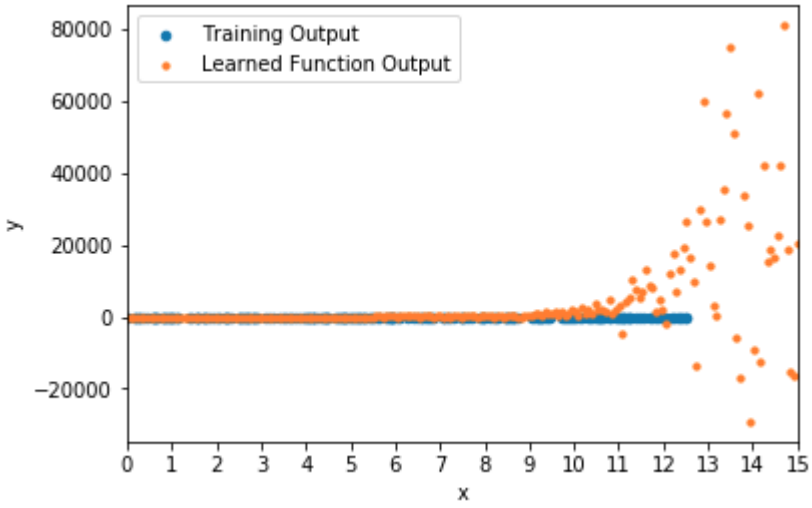
3. P = 3


Training Output
Learned Function Output
y
x
0.0 0.5 1.0 1.5 2.0 2.5 3.0 3.5
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

4. P = 5


Training Output
Learned Function Output
y
x
0
5
10
15
20
25
30
35
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

5. P = 10


Training Output
Learned Function Output
y
x
80000
60000
40000
20000
0
−20000
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

Answer 13

From the previous results, we can see clearly from the plots that the results using kernel ridge regression are similar to results obtained using basis-expanded ridge regression for all P except for the case when $P = 10$.

If we consider the values for α and K generated for each P, we can check whether K is positive definite with respect to α. Following code evaluates $\alpha^\top K\alpha$ for each kernel -

```
def is_kernel_valid(X_trn, Y_trn):
    print("P\tPositive Definite Term")
    for P in [1, 2, 3, 5, 10]:
        k = get_poly_kernel(P)
        a = train_kernel_ridge(X_trn, Y_trn, 0.1, k)
        N = X_trn.shape[0]
        K = np.empty((N,N))
        for n in range(N) :
            for m in range(N) :
                K[n][m] = k(X_trn[n], X_trn[m])
        mer = round(np.matmul(np.matmul(a.transpose(),K),a),10)
        print(f"{P}\t{mer}\n")
```

Results observed from this code are -

P	$\alpha^\top K\alpha$
1	1.0265933456
2	2.4325249636
3	5.3332049447
5	3.5564436669
10	-28.3624896136

According to Mercer's Theorem, a kernel function is valid if and only if K is positive definite. From above table, we can see that for $P = 10$, the value $\alpha^\top K\alpha < 0$. Hence the kernel is not positive definite for $P = 10$, and thus is not safe to use for prediction.

Answer 14

```
from sklearn import svm
from sklearn.metrics import hinge_loss
from sklearn.model_selection import KFold

def train_svm(X_trn, y_trn):
    l_vals = [2, 20, 200]
    splits = 5
    kf = KFold(n_splits=splits, shuffle=True)
    print("Lambda\t\tHinge Loss\n")
    for l in l_vals:
        clf = svm.SVC(kernel='linear', C=1 / (2 * l))
        sum_hinge_loss = 0
        for train_index, test_index in kf.split(X_trn):
            # Split train-test
            X_train, X_test = X_trn[train_index], X_trn[test_index]
            y_train, y_test = y_trn[train_index], y_trn[test_index]

            # Train the model
            clf.fit(X_train, y_train)
            predictions = clf.decision_function(X_test)

            sum_hinge_loss += hinge_loss(y_test, predictions)

        print(f"{l}\t\t{sum_hinge_loss / splits}")
train_svm(X_trn_real, Y_trn_real)
```

Mean validation-set hinge loss using linear kernel :

λ	Hinge loss
2	0.0413724451
20	0.0539554495
200	0.1069876216

Answer 15

Validation-set hinge loss for polynomial basis function of degree 3 :

γ \ λ	2	20	200
0.001	0.0558986632	0.0464007172	0.0728558537
0.01	0.0416938698	0.05478073	0.0551566034
1	0.0443868818	0.0255245765	0.0603758981

Answer 16

Validation-set hinge loss for polynomial basis function of degree 5 :

γ \ λ	2	20	200
0.001	0.2375459622	0.5529637811	0.2828776961
0.01	0.3191144169	0.744456554	0.2543535828
1	0.0571467599	0.1683186749	0.1648740333

Answer 17

Validation-set hinge loss for radial basis function :

γ \ λ	2	20	200
1	0.3843586144	0.8345589747	0.8839056438
0.01	0.0776329575	0.3602880686	0.8367416977
0.001	0.2512348443	0.772343879	0.8779857099

Answer 18

- We chose the polynomial kernel with degree 3, $\gamma = 0.01$ and $\lambda = 2$.
- The estimated 0-1 error is: 0.0043796
- The observed generalization error on the leaderboard is: 0.00584