Assignment 4: Kernels

April 16, 2021

0.0.1 Question 1

Proof To find vector w that minimizes $\sum_{n=1}^{N}\left(w \cdot x^{(n)} - y(n)\right)^2 + \lambda\|w\|^2$

We define X with $x^{(n)}$ on the n^{th} row and Y with $y^{(n)}$ on n^{th} row.

Therefore, learning objective becomes: $\|Xw - Y\|^2 + \lambda\|w\|^2$ We calculate gradient wrt w, $\nabla R(w)$

$$\nabla R(w) = 2X^T(Xw - Y) + 2\lambda w. \rightarrow (1)$$

For optimal $w \Rightarrow w^*$

$$\nabla R(w) = 0$$

by solving the equation we get

$$w^* = \left(X^\top X + \lambda I\right)^{-1} X^\top Y$$

We know that,

$$X^T X = \sum_{n=1}^{N} x^{(n)} x^{(n)T}$$

$$X^T Y = \sum_{n=1}^{N} x^{(n)} y^{(n)}$$

Resubstituting in equation (1), we get - $w^* = \left(\sum_{n=1}^{N} x^{(n)} x^{(n)T} + \lambda I\right)^{-1} \sum_{n=1}^{N} x^{(n)} y^{(n)}$

0.0.2 Question 2

Proof To find the vector w that minimizes

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

where a basis expansion $h(x)$ is applied x

We define X with $x^{(n)}$ on the n^{th} row and H with $h(x)^{(n)}$ on the n^{th} row.

To find optimal w^*

$$w^* = \left(X^\top X + \lambda I\right)^{-1} X^\top y \quad \to (1)$$

Substituting for basis expanded x we get

$$w^* = \left(H^\top H + \lambda I\right)^{-1} H^\top y \quad \to (2)$$

$$H^\top H = \sum_{n=1}^{N} h\left(x^{(n)}\right)\cdot h\left(x^{(n)}\right)^\top \quad \to (3)$$

$$H^\top y = \sum_{n=1}^{N} h\left(x^{(n)}\right)\cdot y^{(n)} \quad \to (4)$$

Substituting for $H^\top H$ and $H^\top y$ in (2)

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

0.0.3 Question 3

Proof To derive an expression for $y^{\text{pred}} = w \cdot h\left(x^{\text{pred}}\right)$ in terms of a kernel function k given $k\left(x, x'\right) = h(x) \cdot h\left(x'\right)$

From the derivation in Question 2 we get -

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

Here, we define $h(x)^{(n)}$ on the n^{th} row.

We use a linear algebra trick for two matrices $P\&Q$,

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

By applying this trick to equation (1), we will get -

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

Substituting HH^T with kernel matrix K, we take $\alpha = (K + \lambda I)^{-1} Y$.

$$w^* = H^T \alpha$$

The prediction is given by

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

Substituting value of w^* we get

$$y^{\text{pred}} = (H^T \alpha)^T \cdot h\left(x^{\text{pred}}\right)$$
$$\Rightarrow y^{\text{pred}} = \alpha^T H \cdot h\left(x^{\text{pred}}\right)$$

Transforming it into summation form

$$y^{\text{pred}} = \sum_{n=1}^{N} \alpha_n k(x^{(n)}, x)$$

0.0.4 Question 4

Proof To find - A kernel function $k\left(x, x'\right)$ where

$$k\left(x, x'\right) = h(x) \cdot h\left(x'\right)$$

$h(x)$ is the polynomial basis expansion

$h(x) = \left[c_0, c_1 x, c_2 x^2, \ldots.., c_p x^p\right]$

$cp = \sqrt{\begin{pmatrix} P \\ p \end{pmatrix}}$

We calculate $h\left(x'\right)$ as -

$$h\left(x'\right) = \left[c_0, c_1(x'), c_2(x'^2), \ldots, cp(x'^p)\right]$$

Then, we take the dot product of h and $h\left(x'\right)$ to evaluate $k\left(x, x'\right)$ -

$$k\left(x, x'\right) = \left[c_0, c_1 x, c_1 x^2, \cdots, c_p x^p\right] \cdot \left[c_0, c_1 x', c_2 x^2, \cdots, c_p x'^p\right]$$
$$k\left(x, x'\right) = c_0^2 + c_1^2\left(x \cdot x'\right) + c_2^2\left(x \cdot x'\right)^2 + c_3^2\left(x \cdot x'\right)^3 + \cdots + c_p^2\left(x x'\right)^P$$

Using binomial expansion

$$c_0^2 + c_1^2\left(x \cdot x'\right) + c_2^2\left(x \cdot x'\right)^2 + c_3^2\left(x \cdot x'\right)^3 + \cdots + c_p^2\left(x \cdot x'\right)^P = \left(1 + x x'\right)^P$$

Hence our kernel matrix expression reduces to

$$k\left(x, x'\right) = \left(1 + x x'\right)^P$$

0.0.5 Question 5

[5]:
```
def eval_basis_expanded_ridge(x,w,h):
    expansion = h(x)
    y = np.dot(w,expansion.T)
    return y
```

0.0.6 Question 6

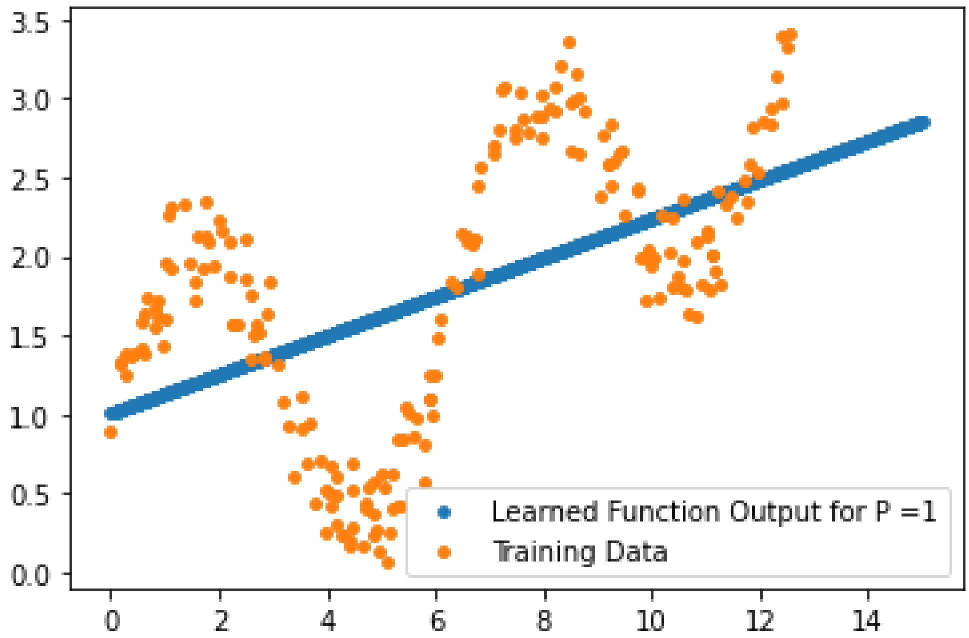
[6]:
```
def train_basis_expanded_ridge(X,Y,,h = get_poly_expansion(3)):
    H = h(X)
    C = np.matmul(H.T,H) + *np.identity(H.shape[1])
    w = np.linalg.solve(C,np.matmul(H.T,Y))
    return w
```

0.0.7 Question 7

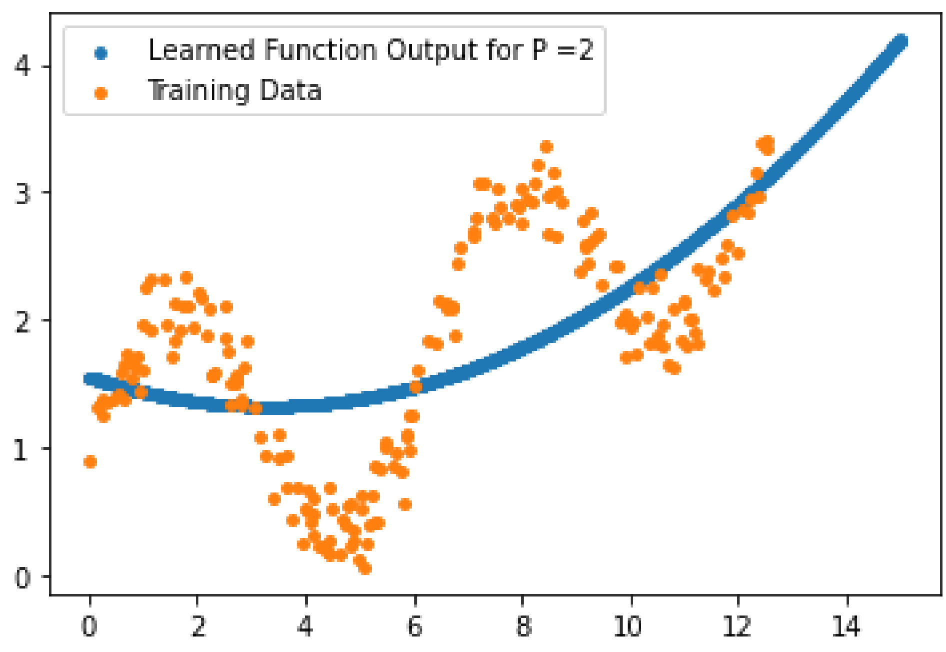
[ ]:
```
P = [1,2,3,5,10]
 = 0.1
for p in P:
    h = get_poly_expansion(p)
    w = train_basis_expanded_ridge(X_trn,Y_trn,,h)
    print("Weight vector = " + str(w) + " for P = " + str(p) + "\n")
```

P	Weight Vector
1	[1.00565302 0.12351259]
2	[1.55636445 -0.09905134 0.02105954]
3	[2.2585207 -0.4731082 0.09166343 -0.0074039]
5	[2.32031734e-01 1.70733532e+00 -7.50673263e-01 1.66018570e-01 -2.11820885e-02 1.49990203e-03]
10	[1.00207181e+00 3.45945567e-01 -6.77619200e-02 4.03404420e-02 -2.40833190e-02 7.29381906e-03 -1.30553266e-03 1.48669952e-04 -1.05198850e-05 3.68789523e-07 2.88162562e-09]

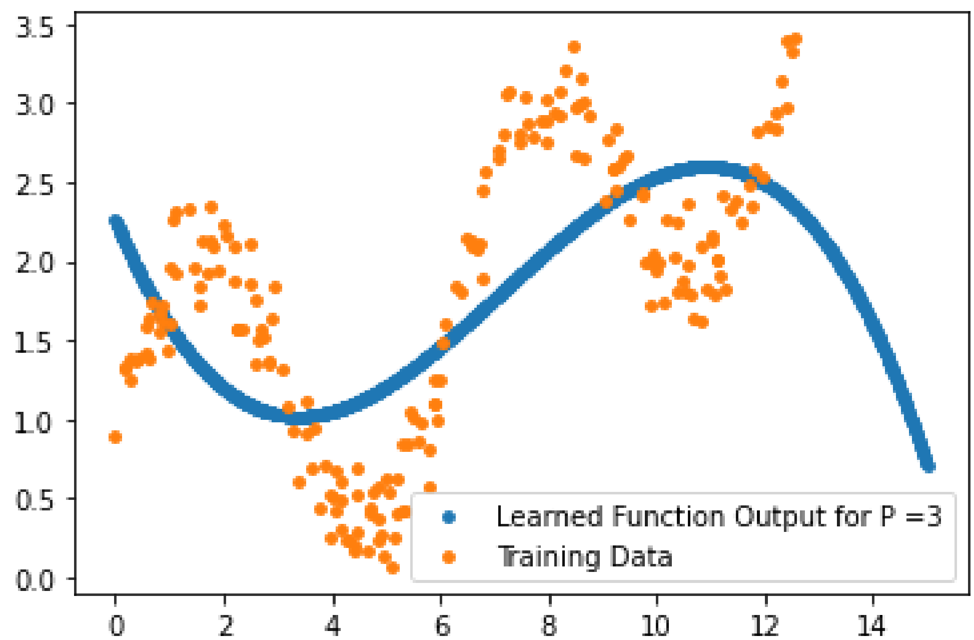
[8]:
```
import matplotlib.pyplot as plt
P = [1,2,3,5,10]
 = 0.1
for p in P:
    h = get_poly_expansion(p)
    w = train_basis_expanded_ridge(X_trn,Y_trn,,h)
    x = np.linspace(0,15,1000,endpoint = True)
    fwx = eval_basis_expanded_ridge(x,w,h)
    plt.scatter(x,fwx,s=15,label = "Learned Function Output for P =" + str(p))
    plt.scatter(X_trn,Y_trn,s =15,label = "Training Data")
    plt.legend()
    plt.show()
```



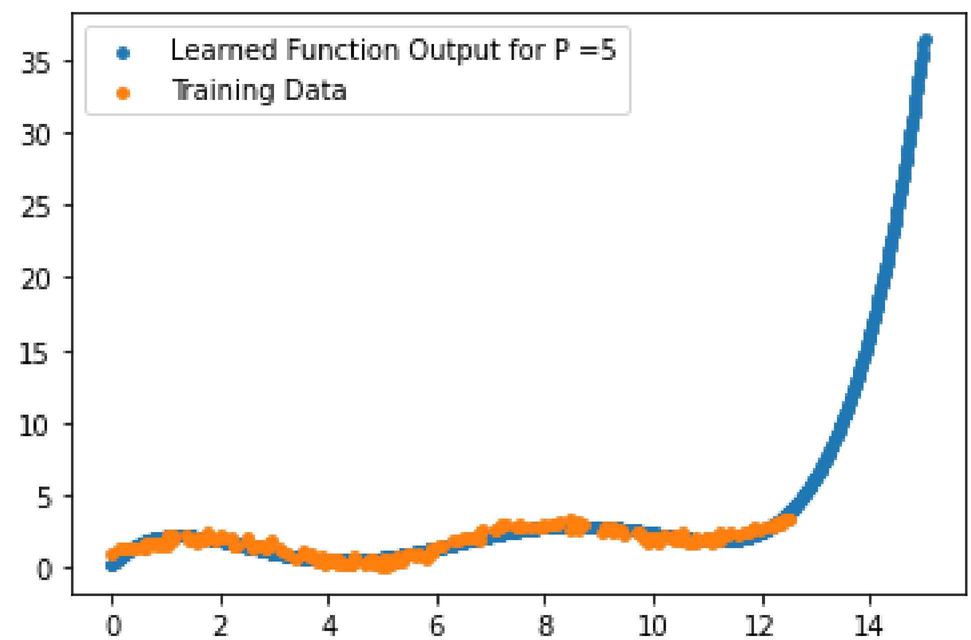
Learned Function Output for P =1
Training Data
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
0
2
4
6
8
10
12
14



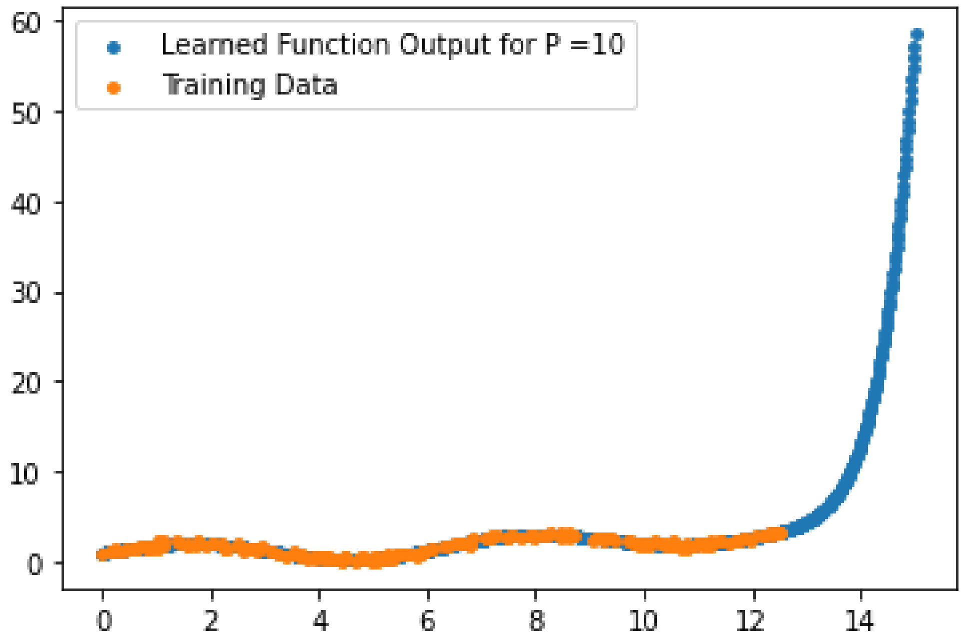
Learned Function Output for P =2
Training Data
0
1
2
3
4
0
2
4
6
8
10
12
14



Learned Function Output for P =3
Training Data



Learned Function Output for P =5
Training Data



Learned Function Output for P =10
Training Data

0.0.8 Question 8

[9]:
```
def get_poly_kernel(P):
    def k(x,xp):
        kernel_value = (1 + np.inner(x, xp)) ** P
        return kernel_value
    return k
```

0.0.9 Question 9

[ ]:
```python
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)
```

Output 1(Kernel)	4.484033437500002
Output 2(Basis Expansion)	4.48403344

0.0.10 Question 10

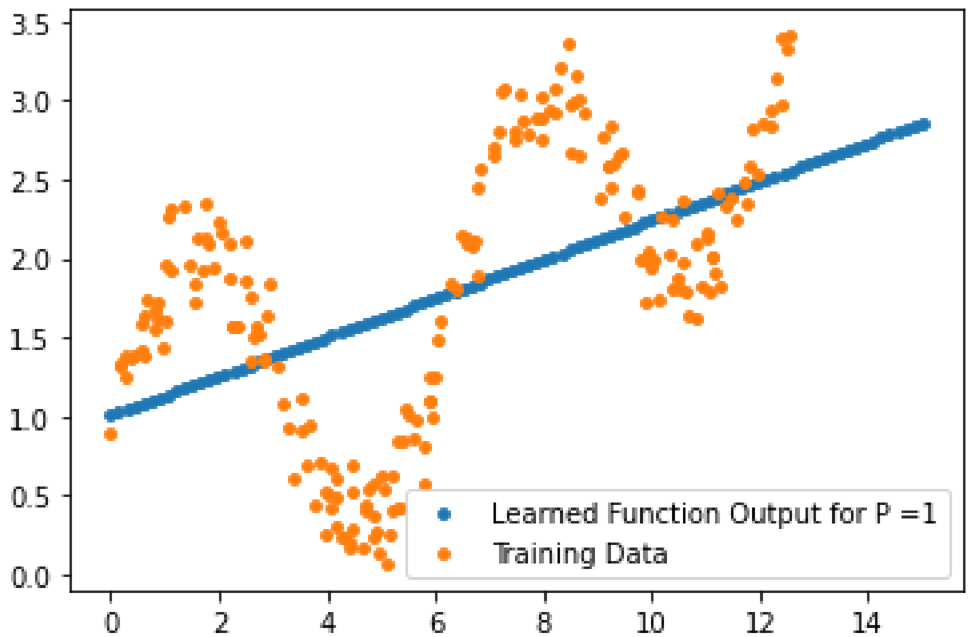
[12]:
```
def train_kernel_ridge(X,Y,,k):
    # Initiate a Kernel matrix
    K = np.zeros((X.shape[0],X.shape[0]))
    # Fill values for kernel matrix using kernel function
    for i in range(X.shape[0]):
        for j in range(X.shape[0]):
            K[i][j] = k(X[i],X[j])
    # Get first part for np.linalg.solve
    kli = K+*np.identity(K.shape[0])
    # Compute alpha
     = np.linalg.solve(kli,Y)
    return
```

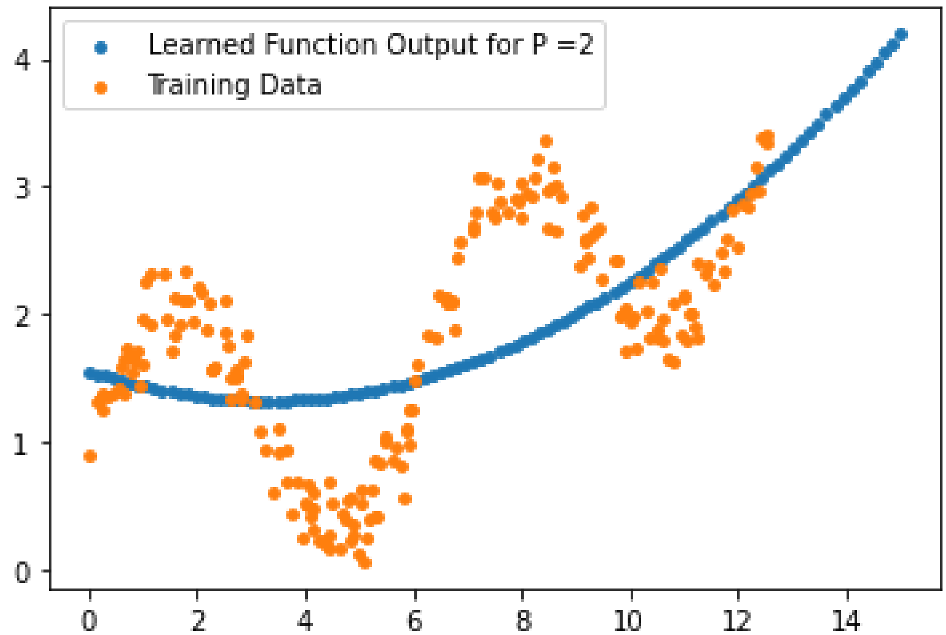
0.0.11 Question 11

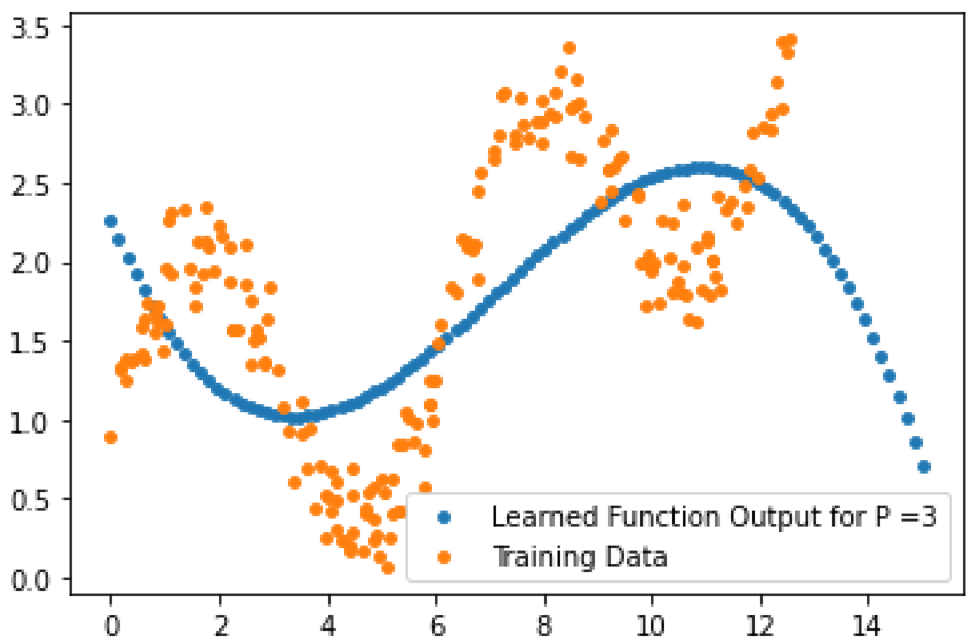
[13]:
```
def eval_kernel_ridge(X_trn, x, , k):
    # Evaluation of kernel ridge regression
    sum_all = 0
    for i in range(len(X_trn)):
        sum_all += [i]*k(X_trn[i],x)
    y = sum_all
    return y
```

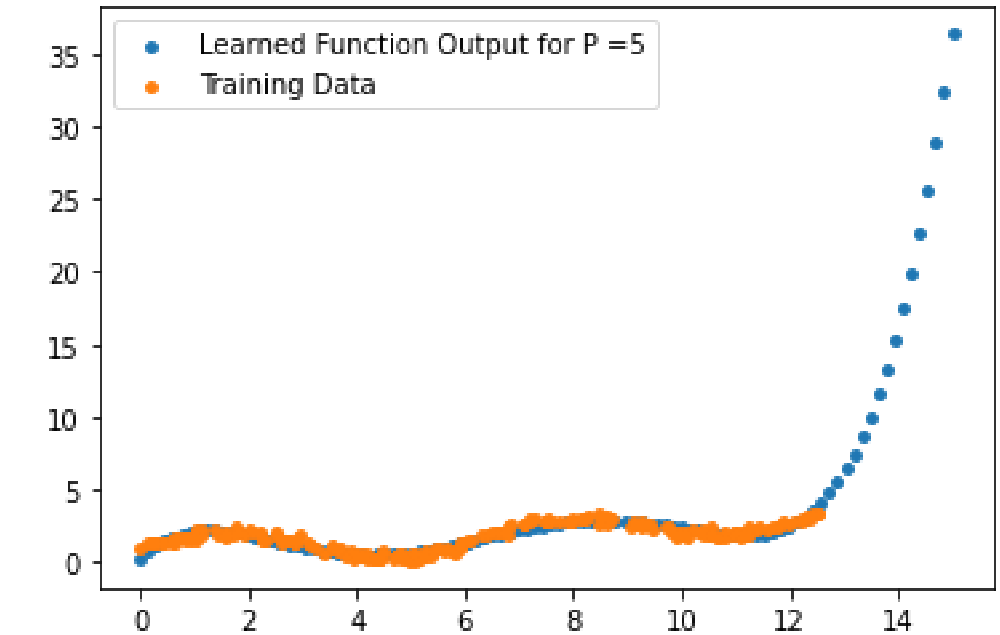
0.0.12 Question 12

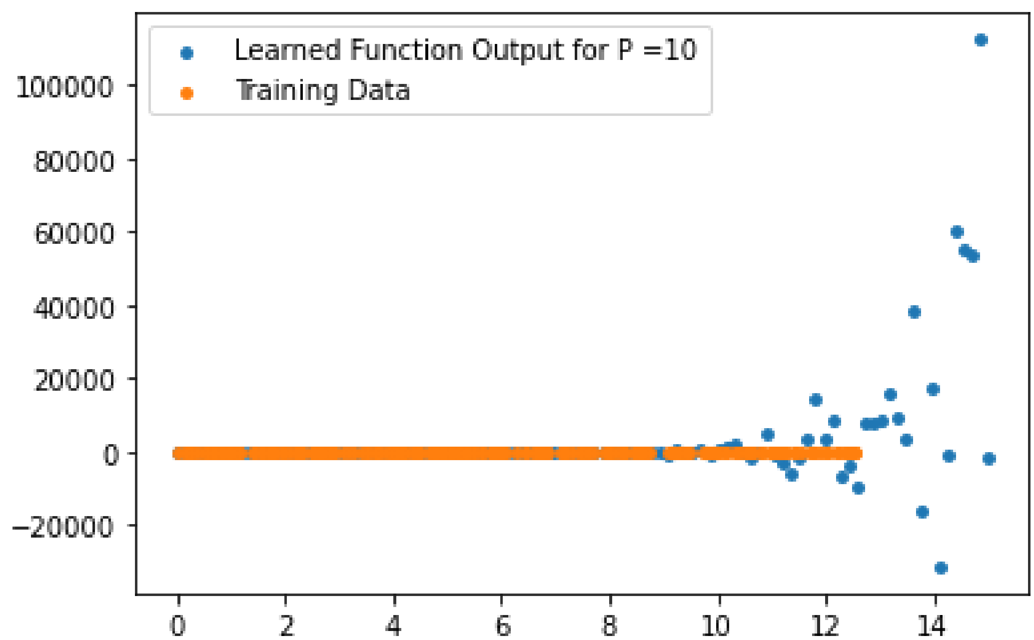
[14]:
```
import matplotlib.pyplot as plt
P = [1,2,3,5,10]
 = 0.1
for p in P:
    k = get_poly_kernel(p)
     = train_kernel_ridge(X_trn,Y_trn,,k = get_poly_kernel(p))
    x = np.linspace(0,15,100)
    Y = []
    for i in range(len(x)):
        Y.append(eval_kernel_ridge(X_trn, x[i], , k))
    Y = np.array(Y)
    plt.scatter(x,Y,s=15,label = "Learned Function Output for P =" + str(p) )
    plt.scatter(X_trn,Y_trn,s =15,label = "Training Data")
    plt.legend()
    plt.show()
```




Learned Function Output for P =2
Training Data
0
1
2
3
4
0
2
4
6
8
10
12
14


0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
Learned Function Output for P =3
Training Data
0
2
4
6
8
10
12
14


Learned Function Output for P =5
Training Data
35
30
25
20
15
10
5
0
0
2
4
6
8
10
12
14


Learned Function Output for P =10
Training Data
100000
80000
60000
40000
20000
0
−20000
0
2
4
6
8
10
12
14

0.0.13 Question 13

The results of kernel ridge regression match that for basis expanded ridge regression for all values of P given in the question except for P=10 where the behaviour is a little erratic. The reason for the same can be found below:

According to Mercer's theorem, a Kernel K is only valid if it forms a positive definite matrix for all the datasets. When we check for all values of P, we observe that the kernel is not positive definite for P greater than 6 and hence it would be unwise to use kernel functions corresponsing to those values of P (In our 5 cases, P=10 would not be valid as kernel matrix corresponding to that is not positive definite)

0.0.14 Question 14

[24]:
```
stuff=np.load("data_real.npz")
x_trn = stuff["x_trn"]
y_trn = stuff["y_trn"]
x_tst = stuff["x_tst"]
```

[ ]:
```
from sklearn import svm
from sklearn.model_selection import KFold
from sklearn.metrics import hinge_loss

c = [2,20,200]

for penalty in c:
    hinge_losses = []

    kf = KFold(n_splits=5, shuffle=True, random_state=3815)
    for train_index, test_index in kf.split(x_trn):


        x_trn_5, x_tst_5 = x_trn[train_index], x_trn[test_index]
        y_trn_5, y_tst_5 = y_trn[train_index], y_trn[test_index]


        clf = svm.SVC(kernel = 'linear', C = 1/penalty)
        clf.fit(x_trn_5,y_trn_5)

        y_pred = clf.decision_function(x_tst_5)
        hinge_losses.append(hinge_loss(y_tst_5,y_pred))



    mean_hinge_loss = (sum(hinge_losses)/5)

    print("The mean hinge loss with 5-Fold cross validation using hinge loss␣
 ↪with lambda = " + str(penalty) + " is: ", mean_hinge_loss)
    print("***************************************************************")
```

	HINGE LOSS FOR LINEAR KERNEL
Lambda	Mean hinge loss
2	0.0392
20	0.0473
200	0.0827

0.0.15 Question 15

```
[ ]: from sklearn import svm
     from sklearn.model_selection import KFold
     from sklearn.metrics import hinge_loss

     c = [2,20,200]
     gamma = [1,0.01,0.001]
     for penalty in c:
           for g in gamma:

             hinge_losses = []

             kf = KFold(n_splits=5, shuffle=True, random_state=3815)
             for train_index, test_index in kf.split(x_trn):

                 x_trn_5, x_tst_5 = x_trn[train_index], x_trn[test_index]
                 y_trn_5, y_tst_5 = y_trn[train_index], y_trn[test_index]

                 clf = svm.SVC(kernel = 'poly', degree = 3, C = 1/penalty, gamma = 1,␣
      ↪coef0 = g)
                 clf.fit(x_trn_5,y_trn_5)
                 y_pred = clf.decision_function(x_tst_5)
                 hinge_losses.append(hinge_loss(y_tst_5,y_pred))


             mean_hinge_loss = (sum(hinge_losses)/5)
             print("The mean hinge loss with 5-Fold cross validation using hinge loss␣
      ↪with lambda = " + str(penalty)  + " and gamma = " + str(g) + " is: " ,␣
      ↪mean_hinge_loss)
            ␣
      ↪print("**************************************************************************
```

Mean Hinge Loss			
Lambda, Gamma	1	0.01	0.001
2	0.0251	0.0684	0.0629
20	0.0509	0.0453	0.0365
200	0.0462	0.0545	0.0582

0.0.16 Question 16

```
[ ]: from sklearn import svm
from sklearn.model_selection import KFold
from sklearn.metrics import hinge_loss

c = [2,20,200]
gamma = [1,0.01,0.001]
for penalty in c:
      for g in gamma:

        hinge_losses = []

        kf = KFold(n_splits=5, shuffle=True, random_state=3815)
        for train_index, test_index in kf.split(x_trn):

            x_trn_5, x_tst_5 = x_trn[train_index], x_trn[test_index]
            y_trn_5, y_tst_5 = y_trn[train_index], y_trn[test_index]

            clf = svm.SVC(kernel = 'poly', degree = 5, C = 1/penalty, gamma = 1,␣
 ↪coef0 = g)
            clf.fit(x_trn_5,y_trn_5)
            y_pred = clf.decision_function(x_tst_5)
            hinge_losses.append(hinge_loss(y_tst_5,y_pred))


        mean_hinge_loss = (sum(hinge_losses)/5)

        print("The mean hinge loss with 5-Fold cross validation using hinge loss␣
 ↪with lambda = " + str(penalty)  + " and gamma = " + str(g) + " is: " ,␣
 ↪mean_hinge_loss)
```

Mean Hinge Loss			
Lambda, Gamma	1	0.01	0.001
2	0.0490	0.3603	0.3183
20	0.4674	1.0210	0.5283
200	0.0705	0.4236	0.4310

0.0.17 Question 17

```
[ ]: from sklearn import svm
from sklearn.model_selection import KFold
from sklearn.metrics import hinge_loss

c = [2,20,200]
gamma = [1,0.01,0.001]
for penalty in c:
    for g in gamma:
        hinge_losses = []
        errors = []

        kf = KFold(n_splits=5, shuffle=True, random_state=3815)
        for train_index, test_index in kf.split(x_trn):

            x_trn_5, x_tst_5 = x_trn[train_index], x_trn[test_index]
            y_trn_5, y_tst_5 = y_trn[train_index], y_trn[test_index]

            clf = svm.SVC(kernel = 'rbf', C = 1/penalty, gamma = g)
            clf.fit(x_trn_5,y_trn_5)

            y_pred = clf.decision_function(x_tst_5)
            hinge_losses.append(hinge_loss(y_tst_5,y_pred))

            y_pred2 = clf.predict(x_tst_5)
            accuracy = (sum(y_pred2 == y_tst_5))/len(y_tst_5)
            errors.append(1 - accuracy)

        mean_hinge_loss = (sum(hinge_losses)/5)
        mean_error = (sum(errors))/5

#         print("The mean hinge loss with 5-Fold cross validation using hinge␣
 ↪loss with lambda = " + str(penalty)  + " and gamma = " + str(g) + " is: " ,␣
 ↪mean_hinge_loss)
#        ␣
 ↪print("*****************************************************************************
```

Mean Hinge Loss for RBF kernel			
Lambda, Gamma	1	0.01	0.001
2	0.2476	0.0568	0.1605
20	0.7811	0.2029	0.6531
200	0.8783	0.7826	0.8655

Generalization Error for RBF Kernel			
Lambda, Gamma	1	0.01	0.001
2	0.0130	0.0102	0.0277
20	0.4445	0.0247	0.3148
200	0.4445	0.445	0.445

0.0.18 Question 18

1. We chose Polynomial kernel with following hyperparameters:

   $$\gamma = 1, \lambda = 2$$

   For this polynomial kernel, the hinge loss that we observe in the K-Fold cross validation data is minimum amongst all the other methods that we tried

2. Estimated Generalization Error is: 0.006%

3. Observed Generalization Error is: 0%