Machine Learning Sheet 5. Basharat Mubashir Ahmed

Monday, 6. December 2021 12:39

Task 1

To prove,
$$E[(\hat{R}-R)^2] = Bias[\hat{R}]^2 + Var[\hat{R}]$$

where,
$$Bias[\hat{R}]^2 = (E[\hat{R}]-R)^2$$
$$Var[\hat{R}] = E[(\hat{R}-E[\hat{R}])^2]$$

$$E[(\hat{R}-R)^2] = E[\{(\hat{R}-E[\hat{R}]) + (E[\hat{R}]-R)\}^2]$$

$$= E[(\hat{R}-E[\hat{R}])^2 + (E[\hat{R}]-R)^2 + 2(R-E[\hat{R}])(E[\hat{R}]-R)]$$

$$= E[(\hat{R}-E[\hat{R}])^2] + E[E[\hat{R}]-R]^2 + 2E[(R-E[\hat{R}])(E[\hat{R}]-R)]$$

Now,
$$E[(\hat{R}-E[\hat{R}])(E[\hat{R}]-R)]$$
$$= \underbrace{(E[\hat{R}] - E[\hat{R}])}_{=0}(E[\hat{R}]-R)$$

and $E[E[\hat{R}]-R]^2 = [E[\hat{R}]-R]^2$

$$\therefore \quad E[(\hat{R}-R)^2] = (E[\hat{R}]-R)^2 + E[(\hat{R}-E[\hat{R}])^2]$$

$$= Bias[\hat{R}]^2 + Var[\hat{R}]$$

Task 3

Error estimate $\hat{R}_T(f_\theta) = \frac{1}{\bar{N}} \sum_{n=1}^{N} l_{eval}(\bar{y}_n, f_\theta(\bar{x}_n))$

where, $l_{eval} = \begin{cases} 0, & y = f_\theta(x) \\ 1, & \text{otherwise} \end{cases}$

Here, $\hat{R}_T(f_\theta) = \frac{1}{10}\left[1+1+1+1\right] = 0.4$

we estimate variance as $S^2_{\hat{R}_T} = \frac{\hat{R}(f_\theta)(1-\hat{R}(f_\theta))}{N}$

$$S^2_{\hat{R}_T} = \frac{0.4 \times 0.6}{10} = 0.024$$

For 2-sided 95% CI, $1-\alpha = 0.95$
$\Rightarrow \alpha = 0.05$

$\therefore \quad \frac{\alpha}{2} = 0.025$

$Z_{\frac{\alpha}{2}} = Z_{0.025} = 1.96$

Now, $\varepsilon = S_{\hat{R}_T} Z_{0.025} = 1.96 \times \sqrt{0.024}$
$\therefore \varepsilon = 0.303$

Thus, 95% CI is:

$= [0.4-0.303, 0.4+0.303]$

$[0.097, 0.703]$

Basharat Mubashir Ahmed

Machine Learning Sheet 5

Question 2.

In [1]:

```
import numpy as np
import matplotlib.pyplot as plt
title=['Plot of the model using optimal tuned d with no regularization $\lambda$',
      'Plot of the model using optimal tuned $\lambda$ with d = 10']
```

Input: number of samples N

Output: one-dimensional data set of N points where $y_n = sin(2\pi x_n) + \epsilon_n$ as in lecture

In [2]:

```
def sine_data_set(N):
  np.random.seed(1234)
  x = np.random.uniform(0,1,(N))
  y = np.sin(x*2*np.pi)+np.random.normal(scale=0.2,size=(N))
  return x,y
```

Input: One-dimensional inputs x as vector of length N, polynomial degree d

Output: polynomial feature representation of the inputs as $N \times (d+1)$ matrix

In [3]:

```
def poly_features(x,d):
  X = np.zeros((x.shape[0],d+1))
  for i in range(0,d+1):
    X[:,i] = np.power(x,i)
  return X
```

Input: instances X as $N \times M$ matrix, labels y as vector of length N, λ for regularization

Output: learned parameter vector

In [4]:

```
def fit_ridge_regression(X,y,lambda_param):
  N = X.shape[0]
  M = X.shape[1]
  return np.linalg.solve(X.T @ X + N*lambda_param*np.eye(M), X.T @ y)
```

Input: instances X as $N \times M$ matrix, model θ

Output: predictions of model θ on X

In [5]:

```
def predict_regression(X,theta):
  return X @ theta
```

Function to plot the results.

In [6]:

```
def plot(x,y,d,Lambda,label):
    #plt.xlim([0,1])
    #plt.ylim([-1.2,1.2])
    plt.scatter(x,y)
    X = poly_features(x,d)
    theta = fit_ridge_regression(X,y,Lambda)
    grid = np.arange(0,1,0.001)
    plt.plot(grid,predict_regression(poly_features(grid,d),theta),'y')
    plt.xlabel('x');plt.ylabel('y');plt.grid();plt.title(label)
```

Function to find the Mean Square Error.

In [7]:

```
def MSE(Y,Yhat):
    return np.mean((Y-Yhat)**2)
```

Function which partitions a given data into k parts.

In [8]:

```
def partition(data,k):
    n=len(data)//k
    start=0
    parts=[]
    for i in range(k):
        parts.append(data[start:n+start,:])
        start=n+start
    return parts
```

Input: instances X, labels y, number of cross-validation folds K, current fold k

Output: train and test sets for fold k

In [9]:

```
def crossval_split(X,y,K,k):
    data=np.hstack((X,y))
    parts=partition(data,K)
    temp=list(parts)
    X_test=temp[k][:,:-1]
    y_test=temp[k][:,-1:]
    temp.pop(k)
    X_train=np.vstack(temp[::])[:,:-1]
    y_train=np.vstack(temp[::])[:,-1:]
    return X_train, y_train, X_test, y_test
```

Function performing K-fold cross-validation.

In [10]:

```
def KFoldCV(Xtrain,Ytrain,param,K,par):
    k_loss=[]
    data=np.hstack((Xtrain,Ytrain))
    for j in range(K):
        xtrain_k,ytrain_k,xtest_k,ytest_k=crossval_split(Xtrain,Ytrain,K,j)
        if par==1:
            beta_k=fit_ridge_regression(xtrain_k,ytrain_k,0)
        else:
            beta_k=fit_ridge_regression(xtrain_k,ytrain_k,param)
        temp_loss=MSE(ytest_k,predict_regression(xtest_k,beta_k))
        k_loss.append(temp_loss)
    if par==1:
        beta_total=fit_ridge_regression(Xtrain,Ytrain,0)
    else:
        beta_total=fit_ridge_regression(Xtrain,Ytrain,param)
    tot_acc=(sum(k_loss)/len(k_loss))
    return beta_total,tot_acc
```

Function to find the optimal hyperparameters using grid search.

In [11]:

```
def GridSearch(Xtrain,Ytrain,par):
    if par==1:
        param=np.arange(0,11)
        Lambda=0
    elif par==0:
        param=10**(np.arange(-9,1).astype('float'))
        d=10
    tot_acc=[]
    for row in range(len(param)):
        if par==1:
            X=poly_features(Xtrain,param[row])
        else:
            X=poly_features(Xtrain,10)
        beta_total,fold_loss=KFoldCV(X,Ytrain,param[row],5,par)
        tot_acc.append((fold_loss,param[row]))
    return tot_acc
```

Generate toy data set

In [12]:

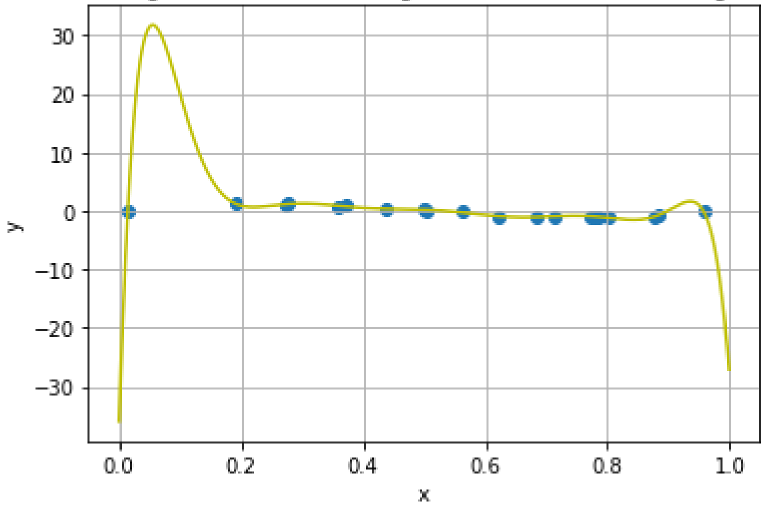
```
N = 20
x,y = sine_data_set(N)
y=y.reshape(len(y),1)
```

Plot of the original model with a high order d = 10 and no regularization.

In [13]:

```
plot(x,y,10,0,'Plot of the original model with a high order d = 10 and no regularization.')
```

Plot of the original model with a high order d = 10 and no regularization.



Tune the hyper parameter d by passing parameter 1 as the argument to Grid Search.

In [14]:

```
output=GridSearch(x,y,1)
```

In [15]:

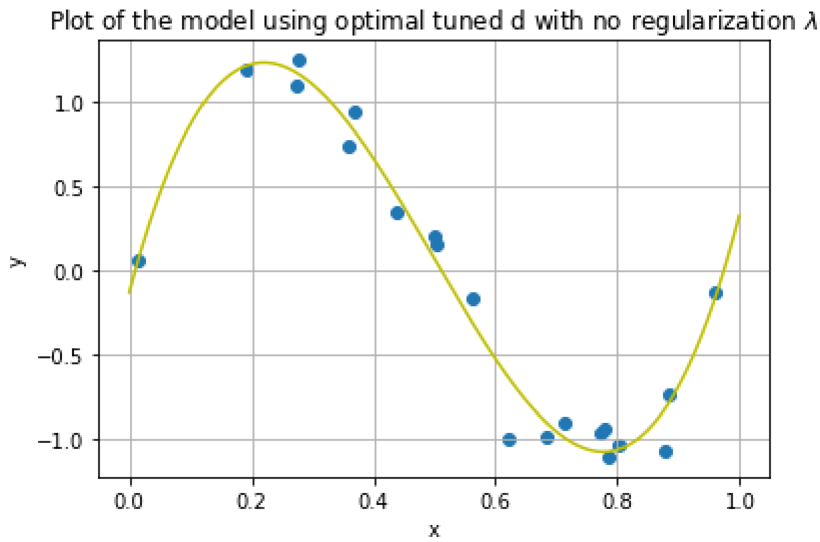
```
optimal=min(output)
dopt=optimal[1]
print('Minimum loss is: %0.5f and occurs when d = %d'%(optimal[0],dopt))
```

```
Minimum loss is: 0.02737 and occurs when d = 3
```

Calling the function to plot and re-train the model with the tuned parameter.

In [16]:

```
plot(x,y,dopt,0,title[0])
```



Tune the hyper parameter λ with d = 10 by passing parameter 0 as the argument to Grid Search.

In [17]:

```
output=GridSearch(x,y,0)
```

In [18]:

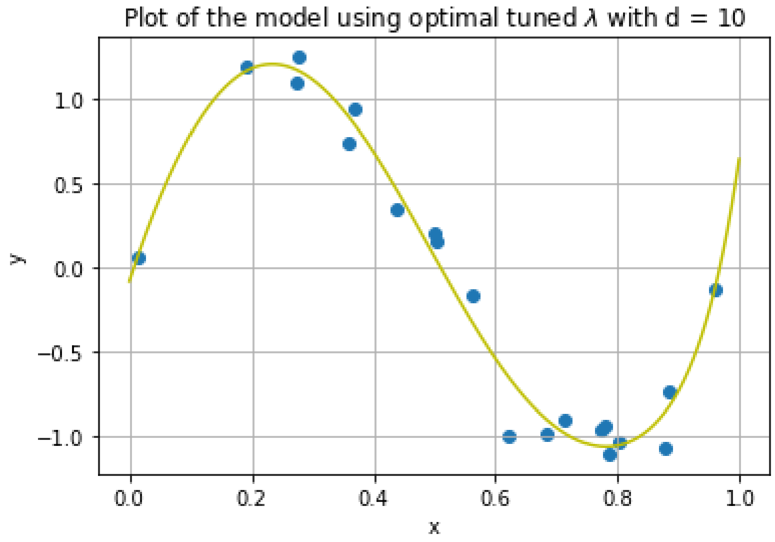
```
optimal=min(output)
lambda_opt=optimal[1]
print(r'Minimum loss is: %0.9f and occurs when d = 10 at regularization = %0.9f'%(optimal[0
```

```
Minimum loss is: 0.064505052 and occurs when d = 10 at regularization = 0.00
0000100
```

Calling the function to plot and re-train the model with the tuned parameter.

In [19]:

```
plot(x,y,10,lambda_opt,title[1])
```



In [ ]: