MACHINE LEARNING HOMEWORK-5
SUPPORT VECTOR MACHINES

- **SVM on MNIST dataset**

1. Default kernel functions

- Linear Kernel

```
prob = svm_problem(train_y,train_x)
print("Default Linear Kernel(without setting parameter):")
param = svm_parameter('-s 0 -t 0 -q')
m = svm_train(prob,param)

print("train")
p_lable,p_acc,p_val=svm_predict(train_y, train_x,m)
print()
print("test")
p_lable,p_acc,p_val=svm_predict(test_y, test_x,m)
```

```
Default Linear Kernel(without setting parameter):
train
Accuracy = 100% (5000/5000) (classification)

test
Accuracy = 95.08% (2377/2500) (classification)
```

- Polynomial Kernel

```
prob = svm_problem(train_y,train_x)
print("Polynomial Kernel(without setting parameter):")
param = svm_parameter('-s 0 -t 1 -q')
m = svm_train(prob,param)

print("train")
p_lable,p_acc,p_val=svm_predict(train_y, train_x,m)
print()
print("test")
p_lable,p_acc,p_val=svm_predict(test_y, test_x,m)
```

```
Polynomial Kernel(without setting parameter):
train
Accuracy = 34.34% (1717/5000) (classification)

test
Accuracy = 34.68% (867/2500) (classification)
```

- RBF kernel

```
prob = svm_problem(train_y,train_x)
print("RBF Kernel(without setting parameter):")
param = svm_parameter('-s 0 -t 2 -q')
m = svm_train(prob,param)

print("train")
p_lable,p_acc,p_val=svm_predict(train_y, train_x,m)
print()
print("test")
p_lable,p_acc,p_val=svm_predict(test_y, test_x,m)
```

```
RBF Kernel(without setting parameter):
train
Accuracy = 96.88% (4844/5000) (classification)

test
Accuracy = 95.32% (2383/2500) (classification)
```

The prob object is creates by calling the svm_problem and svm parameters are set using svm_parameter function. The parameters –s 0 –t 0 for soft margin svm with linear kernel and default value of hyper-parameter C. Similarly, for polynomial and RBF kernel, the value of parameter –t 1 and –t 2 respectively. The svm is trained using all three kernels. The svm with linear and RBF kernel gives similarly accuracy but the accuracy of svm is degraded with polynomial kernel. As there is large difference between the accuracy of svm with polynomial kernel as compared to linear and RBF kernel. It means that model with polynomial kernel under fits the dataset. What is reason behind this under fitting? It means that model is not complex. The hyper-parameter C control the trade-of between minimizing training errors and controlling model complexity. It means hyper-parameter values needs to be tuned which can be done using grid search.

- Comparison of default kernels

	Linear	Polynomial	RBF
Accuracy on test	95.08%	34.68%	95.32%

2. Kernel functions with grid search

- Linear kernel with grid search C (Penalty)

GridSearch C (penalty) for Linear kernel

```
1  def gridSearch_linear(train_y,train_x):
2      print ("RUNNING gridSearch for SVC Linear Kernel")
3      prob  = svm_problem(train_y, train_x)
4      bestc=-1
5      bestCrossVal=-1
6      grid=[]
7      for parm in range (-8,1,1):
8          print(2**parm)
9          param = svm_parameter("-s 0 -q -t 0 -v 3 -c " + str(2**parm))
10         m = svm_train(prob,param)
11         grid.append([2**parm,m])
12         if (m > bestCrossVal):
13             bestCrossVal=m
14             bestc=2**parm
15     print()
16     print()
17     print("Parameter C  Vs Accuracy:")
18     for C in grid:
19         print(C)
20
21     return bestc
```

```
1  bestc = gridSearch_linear(train_y,train_x)
```

```
RUNNING gridSearch for SVC Linear Kernel
0.00390625
Cross Validation Accuracy = 96.3%
0.0078125
Cross Validation Accuracy = 96.82%
0.015625
Cross Validation Accuracy = 96.9%
0.03125
Cross Validation Accuracy = 97.1%
0.0625
Cross Validation Accuracy = 97.08%
0.125
Cross Validation Accuracy = 96.76%
0.25
Cross Validation Accuracy = 96.32%
0.5
Cross Validation Accuracy = 96.06%
1
Cross Validation Accuracy = 96.24%
```

```
Parameter C  Vs Accuracy:
[0.00390625, 96.3]
[0.0078125, 96.82]
[0.015625, 96.89999999999999]
[0.03125, 97.1]
[0.0625, 97.08]
[0.125, 96.76]
[0.25, 96.32]
[0.5, 96.06]
[1, 96.24000000000001]
```

```
# run on the best c
prob  = svm_problem(train_y, train_x)
param = svm_parameter("-s 0 -q -t 0 -c " + str(bestc) )
m = svm_train(prob, param)
print("Prediction on training data:")
p_label, p_acc, p_val = svm_predict(train_y, train_x, m)
print("Prediction on test data:")
p_label, p_acc, p_val = svm_predict(test_y, test_x, m)
```

```
Prediction on training data:
Accuracy = 98.56% (4928/5000) (classification)
Prediction on test data:
Accuracy = 96% (2400/2500) (classification)
```

- c= 0.03125 gives the best performance 97.1% on cross validation
- Using this value of hyper-parameter c, the linear kernel svm gives 98.56% accuracy on train set and 96% accuracy on test set

- Polynomial kernel with grid search c (Penalty), g(gamma), d(degree)

GridSearch C (penalty), g (gamma), d (degree) for Polynomial kernel

```
def gridSearch_linear(train_y,train_x):
    print ("RUNNING gridSearch for SVC Polynomial Kernel")
    prob  = svm_problem(train_y, train_x)
    bestc=-1
    bestg=-1
    bestd=-1
    bestCrossVal=-1
    grid=[]
    for d in range (1,5,1) :
        for g in range (-2,2,1):
            for c in range (-8,1,1):
                print('c:',2**c,'g:',2**g,'d:',d)
                param = svm_parameter("-q -t 1 -v 3 -c " + str(2**c) +" -g " + str (2**g) + " -d " + str (d) )
                m = svm_train(prob,param)
                grid.append([2**c,2**g,d,m])
                if (m > bestCrossVal):
                    bestCrossVal=m
                    bestc=2**c
                    bestg=2**g
                    bestd=d
    print()
    print()
    print("Parameters c, g, d  Vs Accuracy:")
    for C in grid:
        print(C)

    return bestc,bestg,bestd
```

```
bestc,bestg,bestd = gridSearch_linear(train_y,train_x)
```

```
Cross Validation Accuracy = 97.28%
c: 0.0078125 g: 2 d: 3
Cross Validation Accuracy = 97.56%
c: 0.015625 g: 2 d: 3
Cross Validation Accuracy = 97.52%
c: 0.03125 g: 2 d: 3
Cross Validation Accuracy = 97.4%
c: 0.0625 g: 2 d: 3
Cross Validation Accuracy = 97.8%
c: 0.125 g: 2 d: 3
Cross Validation Accuracy = 97.7%
c: 0.25 g: 2 d: 3
Cross Validation Accuracy = 97.34%
c: 0.5 g: 2 d: 3
```

```
# run on the best c
prob  = svm_problem(train_y, train_x)
param = svm_parameter("-s 0 -q -t 1 -c " + str(bestc)  +" -g " + str (bestg) + " -d " + str (bestd) )
m = svm_train(prob, param)
print("Prediction on training data:")
p_label, p_acc, p_val = svm_predict(train_y, train_x, m)
print("Prediction on test data:")
p_label, p_acc, p_val = svm_predict(test_y, test_x, m)
```

```
Prediction on training data:
Accuracy = 100% (5000/5000) (classification)
Prediction on test data:
Accuracy = 97.68% (2442/2500) (classification)
```

- c=0.0625, g=2, d=3 gives the best performance 97.8% on cross validation
- Using these values of hyper-parameters c, g, d, the polynomial kernel svm gives 100% accuracy on train set and 97.8% accuracy on test set

- RBF with grid search c (Penalty), g(gamma)

GridSearch C (penalty), g (gamma), for RBF kernel

```
def gridSearch_linear(train_y,train_x):
    print ("RUNNING gridSearch for SVC RBF Kernel")
    prob  = svm_problem(train_y, train_x)
    bestc=-1
    bestg=-1
    bestCrossVal=-1
    grid=[]
    for g in range (-4,2,1):
        for c in range (-8,1,1):
            print('c:',2**c,'g:',2**g)
            param = svm_parameter("-q -t 2 -v 3 -c " + str(2**c) +" -g " + str (2**g) )
            m = svm_train(prob,param)
            grid.append([2**c,2**g,m])
            if (m > bestCrossVal):
                bestCrossVal=m
                bestc=2**c
                bestg=2**g

    print()
    print()
    print("Parameters c, g, Vs Accuracy:")
    for C in grid:
        print(C)

    return bestc,bestg
```

```
bestc,bestg= gridSearch_linear(train_y,train_x)
```

```
c: 0.125 g: 0.0625
Cross Validation Accuracy = 84.58%
c: 0.25 g: 0.0625
Cross Validation Accuracy = 93%
c: 0.5 g: 0.0625
Cross Validation Accuracy = 96.76%
c: 1 g: 0.0625
Cross Validation Accuracy = 97.84%
```

```
# run on the best c
prob  = svm_problem(train_y, train_x)
param = svm_parameter("-s 0 -q -t 2 -c " + str(bestc)  +" -g " + str (bestg) )
m = svm_train(prob, param)
print("Prediction on training data:")
p_label, p_acc, p_val = svm_predict(train_y, train_x, m)
print("Prediction on test data:")
p_label, p_acc, p_val = svm_predict(test_y, test_x, m)
```

```
Prediction on training data:
Accuracy = 100% (5000/5000) (classification)
Prediction on test data:
Accuracy = 97.36% (2434/2500) (classification)
```

- c=1, g=0.0625, the best performance 97.84% on cross validation
- Using these values of hyper-parameters c, g, the RBF svm gives 100% accuracy on train set and 97.36% accuracy on test set\

- Comparison between grid search kernels

	Linear	Polynomial	RBF
Accuracy on test	96%	97.68%	97.36%

3. User-defined Linear+RBF Kernel

User-defined Linear+RBF kernel

```
def user_defined_kernel(train_y,train_x,test_y,test_x):

    #define linear kernal using linear kernel formula K(x,x')=x.T.dot(x')
    g = 0.125
    print("Computing Linear Kernel")
    LK_Train = train_x.dot(train_x.T)
    idx1 = range(1,len(train_x)+1)
    LK_Train = np.column_stack([idx1, LK_Train])
    LK_Train = (LK_Train).tolist()

    LK_Test = test_x.dot(train_x.T)
    idx2 = range (1,len(test_x)+1)
    LK_Test = np.column_stack([idx2, LK_Test])
    LK_Test = (LK_Test).tolist()

    #define RBF kernel using RBF kernel formula e^gamma||xi-xj||
    print("Computing RBF Kernel")
    dist= np.zeros((len(train_x),len(train_x)))

    for i in range (len(train_x)):
        for j in range (i,len(train_x),1):
            dist[i,j] = np.linalg.norm(train_x[i]-train_x[j])
            dist[j,i] = dist[i,j]

    RBK_Train = np.zeros((len(train_x),len(train_x)))

    for i in range (len(train_x)):
        for j in range (i,len(train_x),1):
            RBK_Train[i,j] =  exp((-g)*dist[i,j])
            RBK_Train[j,i] =  RBK_Train[i,j]


    idx3 = range(1,len(train_x)+1)
    RBK_Train = np.column_stack([idx3, RBK_Train])
    RBK_Train = (RBK_Train).tolist()

    distTest= np.zeros((len(train_x),len(train_x)))

    for i in range (len(test_x)):
        for j in range (len(train_x)):
            distTest[i,j] = np.linalg.norm(test_x[i]-train_x[j])

```

```
    RBK_Test = np.zeros((len(test_x),len(train_x)))

    for i in range (len(test_x)):
        for j in range (len(train_x)):
            RBK_Test[i,j] = exp((-g) *distTest[i,j])

    idx4 = range (1,len(test_x)+1)
    RBK_Test = np.column_stack([idx4, RBK_Test])
    RBK_Test = (RBK_Test).tolist()

    #combine Linear and RBF Kerel; Linear+RBF Kernel
    print("Computing Linear+RBF Kernel")
    RBL_Train = np.zeros((len(train_x),len(train_x)))

    for i in range (len(train_x)):
        for j in range (i,len(train_x),1):
            RBL_Train[i,j] = LK_Train[i][j+1] + RBK_Train[i][j+1]
            RBL_Train[j,i] =  RBL_Train[i,j]
    idx5 = range (1,len(train_x)+1)
    RBL_Train = np.column_stack([idx5, RBL_Train])
    RBL_Train = (RBL_Train).tolist()

    RBL_Test = np.zeros((len(test_x),len(train_x)))
    for i in range (len(test_x)):
        for j in range (1,len(train_x),1):
            RBL_Test[i,j] = LK_Test[i][j+1] + RBK_Test[i][j+1]
    idx6 = range(1,len(test_x)+1)
    RBL_Test = np.column_stack([idx6, RBL_Test])
    RBL_Test = (RBL_Test).tolist()

    print("Training and testing uisng User Defined LINEAR + RBF ")
    prob  = svm_problem(train_y, RBL_Train, isKernel=True)
    param = svm_parameter('-t 4 -c 0.0156 -q')
    m = svm_train(prob, param)
    p_label, p_acc, p_val = svm_predict(test_y, RBL_Test, m)
    sv_idx_linear_rbf = m.get_sv_indices()

    return p_label, sv_idx_linear_rbf
```

```
_=user_defined_kernel(train_y,train_x,test_y,test_x)
```

```
Computing Linear Kernel
Computing RBF Kernel
Computing Linear+RBF Kernel
Training and testing uisng User Defined LINEAR + RBF
Accuracy = 95.96% (2399/2500) (classification)
```

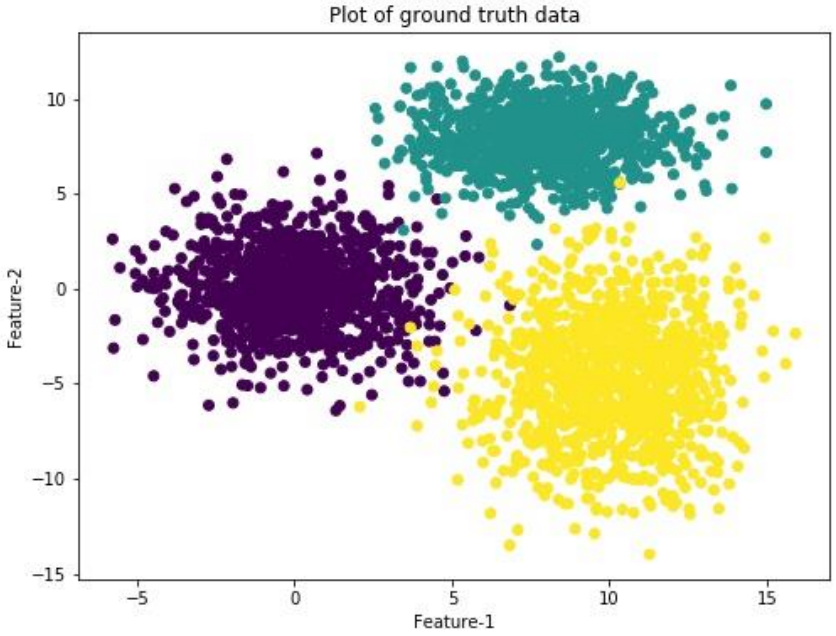
- Comparison of accuracy on test set with default kernels and grid-search kernels

Kernel	Linear	Polynomial	RBF	Linear	Polynomial	RBF	Linear +RBF
Accuracy	95.08%	34.68%	95.32%	96%	97.68%	97.36%	95.96

The comparison shows polynomial kernel with grid search hyper-parameters performs better than Linear+RBF kernel on test set.

- **Find out support vectors of 2D data**

    - Ground truth



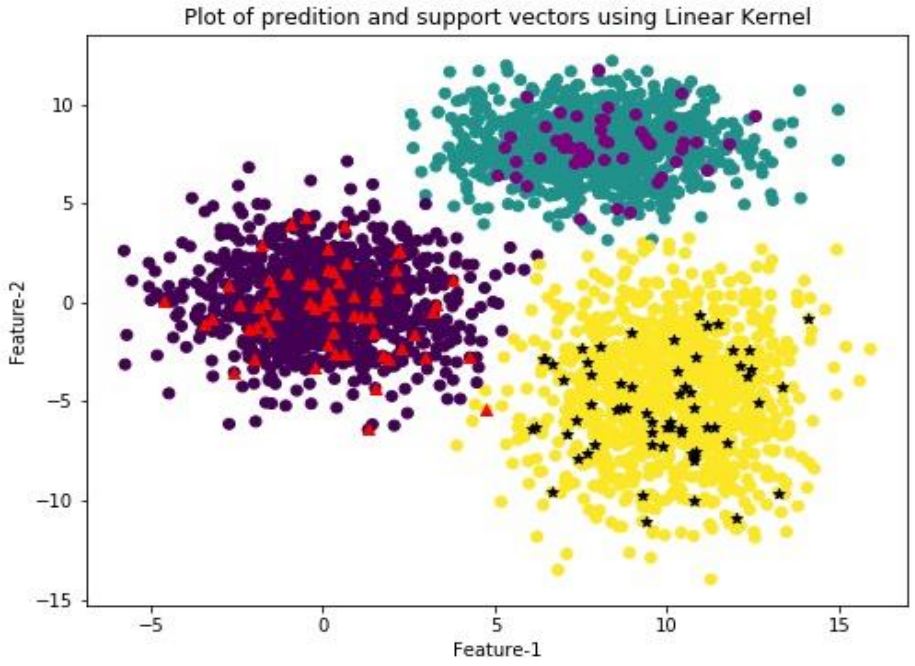
    - Linear Kernel

Find support vectors using Linear Kernel

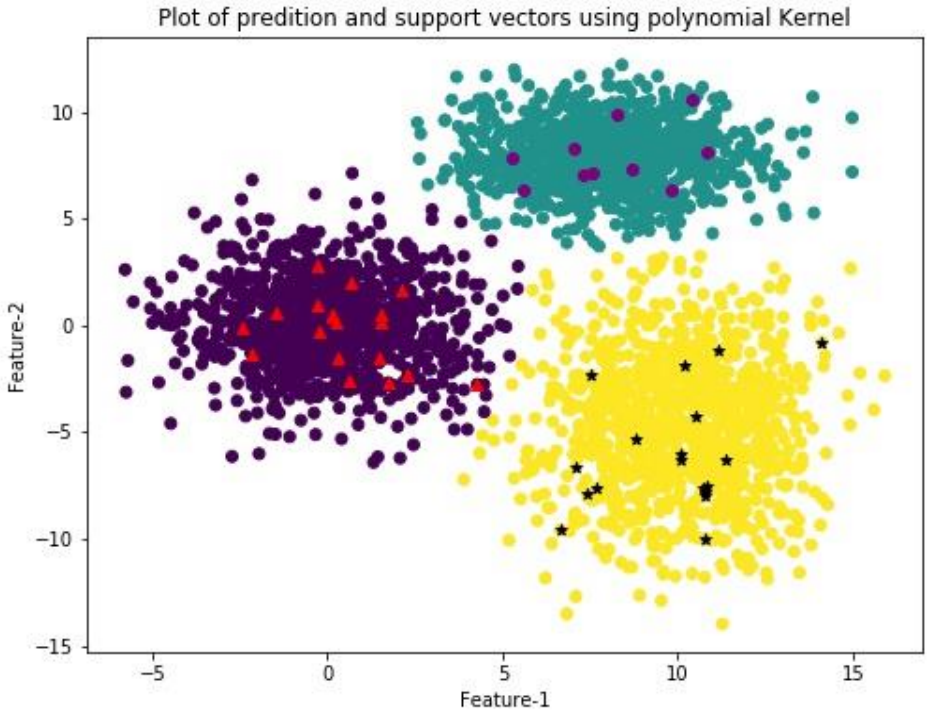
```
:  1  bestc=0.0078125
   2  print("Default Linear Kernel:")
   3  # run on the best c
   4  prob  = svm_problem(plot_y, plot_x)
   5  param = svm_parameter("-s 0 -t 0 -c " + str(bestc) )
   6  m = svm_train(prob, param)
   7  print("Prediction on training data:")
   8  plot_yhatl, p_acc, p_val = svm_predict(plot_y, plot_x, m)
   9  sv_idx_linear = m.get_sv_indices()
```

```
Default Linear Kernel:
Prediction on training data:
Accuracy = 99.3667% (2981/3000) (classification)
```

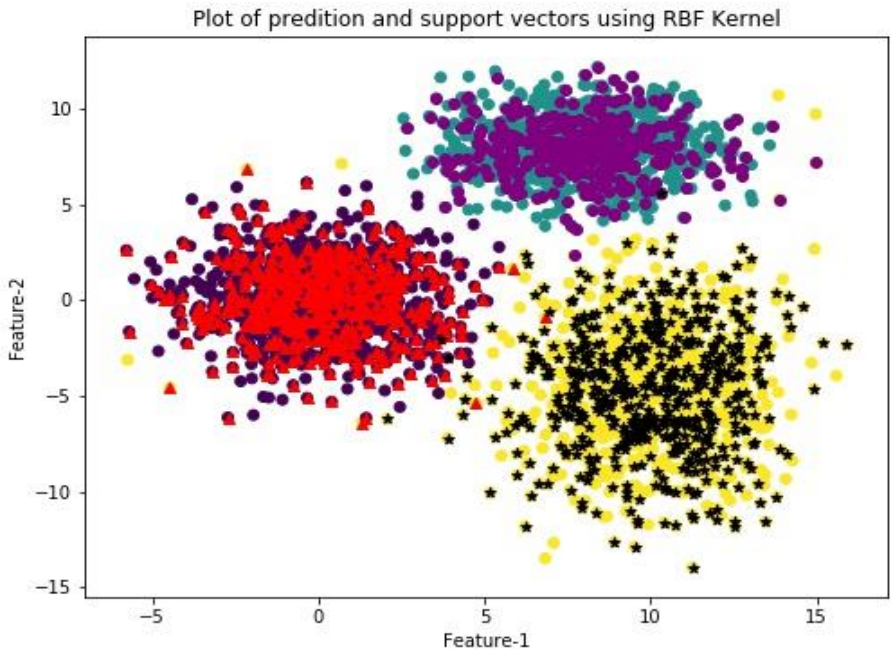
```python
#plot predictions on training data and support of vectors of linear kernel
sv_x0 =[]
sv_x1 =[]
sv_x2 =[]
sv_y0 =[]
sv_y1 =[]
sv_y2 =[]

for indx in sv_idx_linear:
    if plot_y[indx]==0:
        sv_x0.append(plot_x[indx])
        sv_y0.append(plot_y[indx])
    elif plot_y[indx]==1:
        sv_x1.append(plot_x[indx])
        sv_y1.append(plot_y[indx])
    else:
        sv_x2.append(plot_x[indx])
        sv_y2.append(plot_y[indx])

sv_x0=np.array(sv_x0)
sv_x1=np.array(sv_x1)
sv_x2=np.array(sv_x2)

plt.figure(figsize=(8,6))
plt.scatter(plot_x[:,0],plot_x[:,1],c=plot_yhatl)
plt.scatter(sv_x0[:,0],sv_x0[:,1],marker='^', c='red')
plt.scatter(sv_x1[:,0],sv_x1[:,1],marker='o', c='Purple')
plt.scatter(sv_x2[:,0],sv_x2[:,1],marker='*', c='black')
plt.xlabel('Feature-1')
plt.ylabel('Feature-2')
plt.title('Plot of predition and support vectors using Linear Kernel')
plt.savefig(r'C:\Users\Rashid Ali\Desktop\MAchine Learning\Homework-5\sv_linear.jpg')
```



- Polynomial Kernel



- RBF Kernel



- User Defined Linear+RBF kernel

Find support vectors using user defined Linear+RBF kernel

```
def user_defined_kernel(train_y,train_x):

    #define linear kernal using linear kernel formula K(x,x')=x.T.dot(x')
    g = 0.125
    print("Computing Linear Kernel")
    LK_Train = train_x.dot(train_x.T)
    idx1 = range(1,len(train_x)+1)
    LK_Train = np.column_stack([idx1, LK_Train])
    LK_Train = (LK_Train).tolist()

    #define RBF kernel using RBF kernel formula e^gamma||xi-xj||
    print("Computing RBF Kernel")
    dist= np.zeros((len(train_x),len(train_x)))

    for i in range (len(train_x)):
        for j in range (i,len(train_x),1):
            dist[i,j] = np.linalg.norm(train_x[i]-train_x[j])
            dist[j,i] = dist[i,j]

    RBK_Train = np.zeros((len(train_x),len(train_x)))

    for i in range (len(train_x)):
        for j in range (i,len(train_x),1):
            RBK_Train[i,j] =  exp((-g)*dist[i,j])
            RBK_Train[j,i] =  RBK_Train[i,j]

    idx3 = range(1,len(train_x)+1)
    RBK_Train = np.column_stack([idx3, RBK_Train])
    RBK_Train = (RBK_Train).tolist()

    #combine Linear and RBF Kerel; Linear+RBF Kernel
    print("Computing Linear+RBF Kernel")
    RBL_Train = np.zeros((len(train_x),len(train_x)))

    for i in range (len(train_x)):
        for j in range (i,len(train_x),1):
            RBL_Train[i,j] = LK_Train[i][j+1] + RBK_Train[i][j+1]
            RBL_Train[j,i] =  RBL_Train[i,j]

    idx5 = range (1,len(train_x)+1)
    RBL_Train = np.column_stack([idx5, RBL_Train])
    RBL_Train = (RBL_Train).tolist()
```

```
    print("Prediction on training data:")
    prob  = svm_problem(train_y, RBL_Train, isKernel=True)
    param = svm_parameter('-t 4 -c 0.0156 -q')
    m = svm_train(prob, param)
    p_label, p_acc, p_val = svm_predict(train_y, RBL_Train, m)
    sv_idx_linear_rbf = m.get_sv_indices()

    return p_label, sv_idx_linear_rbf
```

```
plot_yhatu,sv_idx_linear_rbf = user_defined_kernel(plot_y,plot_x)
```

```
Computing Linear Kernel
Computing RBF Kernel
Computing Linear+RBF Kernel
Prediction on training data:
Accuracy = 99.4% (2982/3000) (classification)
```