Essential imports

```
import cvxpy as cp
import numpy as np
import pandas as pd
```

Preprocessing Data

```
data = pd.read_csv("/content/drive/MyDrive/DM/letter-recognition.csv")
data.columns = ['letter', 'xbox', 'ybox', 'width', 'height', 'onpix', 'xbar','ybar', 'x2bar', 'y2bar', 'xybar', 'x2ybar', 'xy2bar', 'xedge','xed
print(data)
```

```
       letter  xbox  ybox  width  height  ...  xy2bar  xedge  xedgey  yedge  yedgex
0           I     5    12      3       7  ...       9      2       8      4      10
1           D     4    11      6       8  ...       7      3       7      3       9
2           N     7    11      6       6  ...      10      6      10      2       8
3           G     2     1      3       1  ...       9      1       7      5      10
4           S     4    11      5       8  ...       6      0       8      9       7
...       ...   ...   ...    ...     ...  ...     ...    ...     ...    ...     ...
19994       D     2     2      3       3  ...       4      2       8      3       7
19995       C     7    10      8       8  ...      13      2       9      3       7
19996       T     6     9      6       7  ...       5      2      12      2       4
19997       S     2     3      4       2  ...       8      1       9      5       8
19998       A     4     9      6       6  ...       8      2       7      2       8

[19999 rows x 17 columns]
```

Normalization

```
for i in range(1,17):
  col = data.columns[i]
  mx = max(data[col])
  mn = min(data[col])
  d = mx - mn
  data[col] -= mn
  data[col] /= d

print(data)
```

```
       letter      xbox      ybox  ...    xedgey     yedge    yedgex
0           I  0.333333  0.800000  ...  0.533333  0.266667  0.666667
1           D  0.266667  0.733333  ...  0.466667  0.200000  0.600000
2           N  0.466667  0.733333  ...  0.666667  0.133333  0.533333
3           G  0.133333  0.066667  ...  0.466667  0.333333  0.666667
4           S  0.266667  0.733333  ...  0.533333  0.600000  0.466667
...       ...       ...       ...  ...       ...       ...       ...
19994       D  0.133333  0.133333  ...  0.533333  0.200000  0.466667
19995       C  0.466667  0.666667  ...  0.600000  0.200000  0.466667
```

```
19996      T  0.400000  0.600000  ...  0.800000  0.133333  0.266667
19997      S  0.133333  0.200000  ...  0.600000  0.333333  0.533333
19998      A  0.266667  0.600000  ...  0.466667  0.133333  0.533333

[19999 rows x 17 columns]
```

```
onlyA_df = data.loc[data['letter']=='A']
total = len(onlyA_df)
train = int(0.8*total) # number of A samples in training
test = total -train # number of A samples in testing
onlyA = []

for i in range(total):
  temp = []
  for j in range(1,17,1):
    temp.append(onlyA_df.iloc[i,j])
  onlyA.append(temp)

onlyA = np.array(onlyA)
```

Train and Test split

```
train_X = onlyA[0:train]
print(train_X,train_X.shape)
```

```
[[0.06666667 0.06666667 0.2        ... 0.4        0.13333333 0.46666667]
 [0.2        0.46666667 0.33333333 ... 0.4        0.2        0.53333333]
 [0.2        0.53333333 0.33333333 ... 0.4        0.2        0.46666667]
 ...
 [0.2        0.53333333 0.33333333 ... 0.46666667 0.06666667 0.53333333]
 [0.06666667 0.         0.13333333 ... 0.46666667 0.06666667 0.53333333]
 [0.13333333 0.2        0.2        ... 0.4        0.06666667 0.53333333]] (631, 16)
```

```
test_X = onlyA[train:]
test_Y = [1]*test

# including 5 samples each of remaining alphabets
for i in range(1,26,1):
  c = chr(i+ord('A'))
  c_df = data.loc[data['letter']==c][0:5]
  del c_df['letter']
  c_df = c_df.to_numpy()
  for j in c_df:
    test_X = np.vstack((test_X,j))
    test_Y.append(-1)

test_Y = np.array(test_Y)
print(test_X,test_X.shape)
print(test_Y,test_Y.shape)
```

```
[[0.4        0.66666667 0.6        ... 0.53333333 0.53333333 0.26666667]
 [0.2        0.6        0.4        ... 0.46666667 0.26666667 0.33333333]
 [0.2        0.4        0.33333333 ... 0.4        0.13333333 0.53333333]
 ...
 [0.2        0.53333333 0.26666667 ... 0.53333333 0.53333333 0.53333333]
 [0.06666667 0.         0.06666667 ... 0.53333333 0.4        0.53333333]
 [0.4        0.66666667 0.53333333 ... 0.46666667 0.4        0.6       ]] (283, 16)
[ 1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1
  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1
  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1
  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1
  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1
  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1
  1  1  1  1  1  1  1  1  1  1  1  1  1  1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1] (283,)
```

Defining kernels

```
def linear_kernel(x1, x2):

  return np.dot(x1, x2)

def polynomial_kernel(x, y, p=3):

  return (1 + np.dot(x, y)) ** p

def gaussian_kernel(x, y, sigma=5.0):

  return np.exp(-np.linalg.norm(x-y)**2 / (2 * (sigma ** 2)))

def hellinger_kernel(X1, X2):

  X1,X2 = np.sqrt(X1),np.sqrt(X2)

  return X1 @ X2

def chi_square_kernel(x,y):

  sum = 0.0

  for i in range(len(x)):

    if (x[i]+y[i]) != 0:
      sum += (2*x[i]*y[i])/(x[i]+y[i])

  return sum
```

```
def intersection_kernel(x,y):

  sum = 0.0

  for i in range(len(x)):

    sum += min(x[i],y[i])

  return sum
```

```
# x =
# hellinger_kernel(np.array([1,2,3]),np.array([3,4,5]))
np.sqrt([2,4,4])
```

```
array([1.41421356, 2.        , 2.        ])
```

Calculating kernel matrix

```
def kernel_matrix(X,kernel=linear_kernel):
  m = X.shape[0]
  K = np.zeros((m,m))
  for i in range(m):
    for j in range(m):
      K[i,j] = kernel(X[i], X[j])

  return K
```

```
kernel_matrix(train_X,hellinger_kernel)
```

```
array([[1.18133634, 1.33998342, 1.25077754, ..., 1.22565175, 1.17378779,
        1.25077754],
       [1.33998342, 1.6       , 1.51143787, ..., 1.4727148 , 1.30809446,
        1.44067885],
       [1.25077754, 1.51143787, 1.44683563, ..., 1.41264134, 1.21106014,
        1.34329611],
       ...,
       [1.22565175, 1.4727148 , 1.41264134, ..., 1.43604395, 1.21837779,
        1.31993266],
       [1.17378779, 1.30809446, 1.21106014, ..., 1.21837779, 1.19629056,
        1.24543611],
       [1.25077754, 1.44067885, 1.34329611, ..., 1.31993266, 1.24543611,
        1.33998342]])
```

Parameters

```
m = len(train_X)
v1,v2 = 0.9,0.9
e = 2/3
```

```
c1,c2 = 1/(v1*m),e/(v2*m)
```

Solving Optimization problem

```
def optimize(train_X,c1,c2,e,kernel=linear_kernel):

  m = len(train_X) # number of samples
  n = len(train_X[0])  # number of features in one samples

  alpha = cp.Variable(m)
  alpha1 = cp.Variable(m)

  A1 = np.ones((1,m))
  b1 = np.array([1])
  b2 = np.array([e])

  G = np.eye(m)
  h = np.full((m,),0)
  h1 = np.full((m,),c1)
  h2 = np.full((m,),c2)
  G1 = -np.eye(m)

  K = kernel_matrix(train_X,kernel)
  # print(K)

  prob = cp.Problem(cp.Minimize((1/2)*cp.quad_form((alpha-alpha1),K)),
                    [A1 @ alpha == b1,
                    A1 @ alpha1 == b2,
                    G @ alpha <= h1,
                    G @ alpha1 <=h2,
                    G1 @ alpha <= h,
                    G1 @ alpha1 <= h])
  prob.solve()
  print(prob.status+" Solution found")

  # print("\nThe optimal value is", prob.value)
  # print("A solution for dual variables is")
  alpha = alpha.value
  alpha1 = alpha1.value
  # print(alpha1)
  # print(alpha)

  return alpha,alpha1
```

Calculating offsets/bias

```
def calculate_bias(alpha,alpha1,c1,c2,X,kernel=linear_kernel):

  m = X.shape[0] # number of samples

  n = 0 # number of support vectors
  sum = 0

  for i in range(m):
    if (alpha[i]>0 and alpha[i]<c1):
      n+=1
      for j in range(m):
        sum += ((alpha[j]-alpha1[j])*kernel(X[i],X[j]))

  sum = sum/n;
  # print(n,' out of ',m)
  p1 = sum

  n = 0 # number of support vectors
  sum = 0.0

  for i in range(m):
    if (alpha1[i]>0 and alpha1[i]<c2):
      n+=1
      for j in range(m):
        sum += ((alpha[j]-alpha1[j])*kernel(X[i],X[j]))

  sum = sum/n;
  # print(n,' out of ',m)

  p2=sum

  return p1,p2
```

Calculating svm score

```
def svm_score(x,train_X,alpha,alpha1,kernel=linear_kernel):

  m = train_X.shape[0] # number of samples
  score = 0.0

  for i in range(m):
    score += (alpha[i]-alpha1[i])*kernel(x,train_X[i])

  return score
```

Prediction function

```
def predict(x,train_X,p1,p2,alpha,alpha1,kernel=linear_kernel):

  score = svm_score(x,train_X,alpha,alpha1,kernel)
  return np.sign((score-p1)*(p2-score))
```

Using Linear Kernel

```
# first calculating biases
from sklearn.metrics import matthews_corrcoef

alpha,alpha1 = optimize(train_X,c1,c2,e,linear_kernel)
p1,p2 = calculate_bias(alpha,alpha1,c1,c2,train_X,linear_kernel)

pred_Y = []
total_test = len(test_Y)
correct = 0
for i in range(total_test):

  res = predict(test_X[i],train_X,p1,p2,alpha,alpha1,linear_kernel)
  pred_Y.append(int(res))
  # print(res)

print("matthews correlation coefficient: ",matthews_corrcoef(test_Y, pred_Y))
```

```
optimal Solution found
matthews correlation coefficient:  0.1654787361413416
```

Using Polynomial Kernel

```
# first calculating biases
from sklearn.metrics import matthews_corrcoef

alpha,alpha1 = optimize(train_X,c1,c2,e,polynomial_kernel)
p1,p2 = calculate_bias(alpha,alpha1,c1,c2,train_X,polynomial_kernel)

pred_Y = []
total_test = len(test_Y)
correct = 0
for i in range(total_test):

  res = predict(test_X[i],train_X,p1,p2,alpha,alpha1,polynomial_kernel)
```

```
  pred_Y.append(int(res))
  # print(res)

print("matthews correlation coefficient: ",matthews_corrcoef(test_Y, pred_Y))
```

```
optimal Solution found
matthews correlation coefficient:  0.10884988311277556
```

Using Gaussian Kernel

```
# first calculating biases
from sklearn.metrics import matthews_corrcoef

alpha,alpha1 = optimize(train_X,c1,c2,e,gaussian_kernel)
p1,p2 = calculate_bias(alpha,alpha1,c1,c2,train_X,gaussian_kernel)

pred_Y = []
total_test = len(test_Y)
correct = 0
for i in range(total_test):

  res = predict(test_X[i],train_X,p1,p2,alpha,alpha1,gaussian_kernel)
  pred_Y.append(int(res))
  # print(res)

print("matthews correlation coefficient: ",matthews_corrcoef(test_Y, pred_Y))
```

```
optimal Solution found
matthews correlation coefficient:  -0.1873314482020479
```

Using Hellinger Kernel

```
# first calculating biases
from sklearn.metrics import matthews_corrcoef

alpha,alpha1 = optimize(train_X,c1,c2,e,hellinger_kernel)
p1,p2 = calculate_bias(alpha,alpha1,c1,c2,train_X,hellinger_kernel)

pred_Y = []
total_test = len(test_Y)
correct = 0
for i in range(total_test):

  res = predict(test_X[i],train_X,p1,p2,alpha,alpha1,hellinger_kernel)
  pred_Y.append(int(res))
  # print(res)
```

```
print("matthews correlation coefficient: ",matthews_corrcoef(test_Y, pred_Y))
```

```
optimal Solution found
matthews correlation coefficient:  0.21491918681694266
```

Using Chi square Kernel

```
# first calculating biases
from sklearn.metrics import matthews_corrcoef

alpha,alpha1 = optimize(train_X,c1,c2,e,chi_square_kernel)
p1,p2 = calculate_bias(alpha,alpha1,c1,c2,train_X,chi_square_kernel)

pred_Y = []
total_test = len(test_Y)
correct = 0
for i in range(total_test):

  res = predict(test_X[i],train_X,p1,p2,alpha,alpha1,chi_square_kernel)
  pred_Y.append(int(res))
  # print(res)

print("matthews correlation coefficient: ",matthews_corrcoef(test_Y, pred_Y))
```

```
optimal Solution found
matthews correlation coefficient:  0.24173674600428716
```

Using Intersection Kernel

```
# first calculating biases
from sklearn.metrics import matthews_corrcoef

alpha,alpha1 = optimize(train_X,c1,c2,e,intersection_kernel)
p1,p2 = calculate_bias(alpha,alpha1,c1,c2,train_X,intersection_kernel)

pred_Y = []
total_test = len(test_Y)
correct = 0
for i in range(total_test):

  res = predict(test_X[i],train_X,p1,p2,alpha,alpha1,intersection_kernel)
  pred_Y.append(int(res))
  # print(res)

print("matthews correlation coefficient: ",matthews_corrcoef(test_Y, pred_Y))
```

```
optimal Solution found
matthews correlation coefficient:  0.18253628081942064
```