Essential imports

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

Preprocessing Data

```
data = pd.read_csv("/content/drive/MyDrive/DM/Image-pixels.csv")
```

```
header = []
for i in range(22501):
  header.append(str(i))
data.columns = header
```

```
print(data)
```

```
          0      1      2      3      4  ...  22496  22497  22498  22499  22500
0     118.0  119.0  118.0  118.0  117.0  ...  102.0  103.0  105.0  105.0    0.0
1     213.0  214.0  214.0  212.0  213.0  ...  148.0  154.0  163.0  158.0    0.0
2     129.0  127.0  128.0  129.0  127.0  ...  129.0  124.0  123.0  132.0    0.0
3     110.0  109.0  108.0  107.0  109.0  ...   81.0   77.0   82.0   79.0    0.0
4     255.0  255.0  255.0  255.0  255.0  ...  171.0  121.0  119.0  126.0    0.0
...     ...    ...    ...    ...    ...  ...    ...    ...    ...    ...    ...
4377  240.0  240.0  240.0  241.0  239.0  ...   98.0  100.0   99.0   79.0    6.0
4378  184.0  207.0  203.0  204.0  190.0  ...  173.0  107.0  119.0  193.0    6.0
4379   27.0   75.0   39.0   29.0   28.0  ...  205.0  198.0  199.0  212.0    6.0
4380  255.0  254.0  254.0  251.0  253.0  ...   83.0   81.0   58.0   81.0    6.0
4381   54.0   51.0   54.0   48.0   47.0  ...  120.0  120.0  121.0  119.0    6.0

[4382 rows x 22501 columns]
```

Normalization

```
for i in range(0,22500):
  col = data.columns[i]
  mx = max(data[col])
  mn = min(data[col])
  d = mx - mn
  data[col] -= mn
  data[col] /= d

print(data)
```

```
             0          1          2  ...      22498      22499  22500
```

```
0     0.462745  0.466667  0.462745  ...  0.411765  0.411765    0.0
1     0.835294  0.839216  0.839216  ...  0.639216  0.619608    0.0
2     0.505882  0.498039  0.501961  ...  0.482353  0.517647    0.0
3     0.431373  0.427451  0.423529  ...  0.321569  0.309804    0.0
4     1.000000  1.000000  1.000000  ...  0.466667  0.494118    0.0
...        ...       ...       ...  ...       ...       ...    ...
4377  0.941176  0.941176  0.941176  ...  0.388235  0.309804    6.0
4378  0.721569  0.811765  0.796078  ...  0.466667  0.756863    6.0
4379  0.105882  0.294118  0.152941  ...  0.780392  0.831373    6.0
4380  1.000000  0.996078  0.996078  ...  0.227451  0.317647    6.0
4381  0.211765  0.200000  0.211765  ...  0.474510  0.466667    6.0

[4382 rows x 22501 columns]
```

```
only_zero_df = data.loc[data['22500']==0]
total = len(only_zero_df)
train = int(0.8*total) # number of A samples in training
test = total -train # number of A samples in testing

only_zero = only_zero_df.to_numpy()
only_zero = np.delete(only_zero, -1, axis=1)
print(only_zero,only_zero.shape)
```

```
[[0.4627451  0.46666667 0.4627451  ... 0.40392157 0.41176471 0.41176471]
 [0.83529412 0.83921569 0.83921569 ... 0.60392157 0.63921569 0.61960784]
 [0.50588235 0.49803922 0.50196078 ... 0.48627451 0.48235294 0.51764706]
 ...
 [0.36078431 0.38039216 0.36862745 ... 0.37254902 0.36470588 0.37647059]
 [0.81960784 0.80392157 0.79215686 ... 0.3372549  0.3372549  0.34901961]
 [0.35686275 0.36078431 0.35686275 ... 0.38039216 0.37647059 0.37254902]] (670, 22500)
```

Train and Test split

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

```
[[0.4627451  0.46666667 0.4627451  ... 0.40392157 0.41176471 0.41176471]
 [0.83529412 0.83921569 0.83921569 ... 0.60392157 0.63921569 0.61960784]
 [0.50588235 0.49803922 0.50196078 ... 0.48627451 0.48235294 0.51764706]
 ...
 [0.61568627 0.61568627 0.61568627 ... 0.4745098  0.47843137 0.47843137]
 [0.43921569 0.45490196 0.4627451  ... 0.58823529 0.58431373 0.57254902]
 [0.5254902  0.52156863 0.52156863 ... 0.70588235 0.70588235 0.70588235]] (536, 22500)
```

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

for i in range(1,7,1):
  temp = data.loc[data['22500']==i][0:5]
  temp = temp.to_numpy()
  temp = np.delete(temp, -1, axis=1)
  test_X = np.vstack((test_X,temp))
  tempy = [-1]*5
  test_Y.extend(tempy)
```

```
  test_Y.extend(tempy)

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

```
[[0.45882353 0.47058824 0.48235294 ... 0.05098039 0.03921569 0.05098039]
 [0.7372549  0.74117647 0.74509804 ... 0.64313725 0.64705882 0.65098039]
 [0.76470588 0.76470588 0.76862745 ... 0.64705882 0.64705882 0.64313725]
 ...
 [0.95294118 0.95686275 0.96078431 ... 0.05490196 0.0745098  0.09411765]
 [0.45098039 0.45098039 0.45098039 ... 0.3372549  0.32941176 0.27843137]
 [0.47058824 0.48235294 0.48235294 ... 0.80784314 0.87843137 0.30196078]] (164, 22500)
[ 1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1
  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1
  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1
  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1
  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1
  1  1  1  1  1  1  1  1  1  1  1  1  1  1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1] (164,)
```

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
```

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
```

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:  -0.054659483230836456
```

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))
```

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:  0.0
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:900: RuntimeWarning: invalid value encountered in double_scalars
  mcc = cov_ytyp / np.sqrt(cov_ytyt * cov_ypyp)
```

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:  -0.15337059307783316
```

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:  -0.054659483230836456
```

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:  -0.1694709615988283
```