```
import numpy as np
from matplotlib import pyplot as plt
from scipy import misc
%matplotlib inline
```

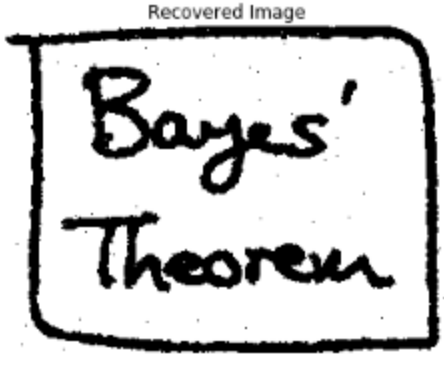
```
original_img = misc.imread('Bayes_noisy.png', mode='L')
img_bw = np.zeros(original_img.shape)
img_bw[original_img > 128] = 1
plt.imshow(img_bw)
plt.set_cmap('gray')
plt.title('Original Image')
a = plt.axis("off")
```


```
y = img_bw
x = y # Initialisation of x

x[x==0] = -1
y[y==0] = -1

# Set parameters
h_ = 0
beta_ = 1
eta_ = 2.1

def local_energy(i, j, x, y, h, beta, eta):
    energy = h * x[i, j]
    s = 0
    for k in [(i-1, j), (i+1, j), (i, j-1), (i, j+1)]:
        if k[0] >= 0 and k[0] <= 459 and k[1] >= 0 and k[1] <= 629:
            s += x[k]
    energy -= eta * x[i, j] * s
    energy -= beta * x[i, j] * y[i, j]
    return energy
```

```
for k in range(2):
    for i in range(460):
        for j in range(630):
            le1 = local_energy(i, j, x, y, h_, beta_, eta_)
            x[i, j] = -x[i, j]
            le2 = local_energy(i, j, x, y, h_, beta_, eta_)
            if le1 < le2:
                x[i, j] = -x[i, j]
    print('Done with epoch', k)
```

```
Done with epoch 0
Done with epoch 1
```

```
x[x==-1] = 0
plt.imshow(x)
plt.set_cmap('gray')
plt.title('Recovered Image')
a = plt.axis("off")
```




```
true_img = misc.imread('Bayes_true.png', mode='L')
true_img_bw = np.zeros(original_img.shape)
true_img_bw[true_img > 128] = 1
plt.imshow(true_img_bw)
plt.set_cmap('gray')
plt.title('Original Image')
a = plt.axis("off")
```

```
numCorrect = 0;

for i in range(460):
    for j in range(630):
        if(x[i,j] == true_img_bw[i,j]):
            numCorrect = numCorrect + 1;

accuracy = numCorrect/(630*460);

print(accuracy)
```

```
0.975728088336784
```

```
import numpy as np
from numpy import linalg
import cvxopt
import cvxopt.solvers
import pandas as pd
```

```
import scipy.io
```

```
temp = scipy.io.loadmat('MNIST_data.mat')

train = temp['train_samples']
test = temp['test_samples']
trainL = temp['train_samples_labels']
testL = temp['test_samples_labels']
```

```
def polynomial_kernel(x, y, p=3):
    return (1 + np.dot(x, y)) ** p
```

```
class SVM():
    def __init__(self, C=1.0):
        self.C = C

    def fit(self, X, y):
        n_samples, n_features = X.shape

        # Gram matrix
        K = np.zeros((n_samples, n_samples))
        for i in range(n_samples):
            for j in range(n_samples):
                K[i,j] = polynomial_kernel(X[i], X[j])

        P = cvxopt.matrix(np.diag(y) * K * np.diag(y))
        q = cvxopt.matrix(np.ones(n_samples) * -1)
        A = cvxopt.matrix(y.reshape(1,-1).astype(float))
        b = cvxopt.matrix(0.0)
```

```
    P = cvxopt.matrix(np.diag(y) * K * np.diag(y))
    q = cvxopt.matrix(np.ones(n_samples) * -1)
    A = cvxopt.matrix(y.reshape(1,-1).astype(float))
    b = cvxopt.matrix(0.0)

    tmp1 = np.diag(np.ones(n_samples) * -1)
    tmp2 = np.identity(n_samples)
    G = cvxopt.matrix(np.vstack((tmp1, tmp2)))
    tmp1 = np.zeros(n_samples)
    tmp2 = np.ones(n_samples) * self.C
    h = cvxopt.matrix(np.hstack((tmp1, tmp2)))

    solution = cvxopt.solvers.qp(P, q, G, h, A, b)
    a = np.ravel(solution['x'])

    # remove nonzero
    sv = a > 1e-5
    ind = np.arange(len(a))[sv]
    self.a = a[sv]
    self.sv = X[sv]
    self.sv_y = y[sv]

    # Intercept
    self.b = 0
    for n in range(len(self.a)):
        self.b += self.sv_y[n]
        self.b -= np.sum(self.a * self.sv_y * K[ind[n],sv])
    self.b /= len(self.a)

def project(self, X):
    y_predict = np.zeros(len(X))
    for i in range(len(X)):
        s = 0
        for a, sv_y, sv in zip(self.a, self.sv_y, self.sv):
            s += a * sv_y * polynomial_kernel(X[i], sv)
        y_predict[i] = s
    return y_predict + self.b

def predict(self, X):
    return np.sign(self.project(X))
```

```
s = pd.Series(trainL.flatten())

df_results = pd.DataFrame()

for counter in range(10):
    v = np.where(s==counter, 1, -1)

    onevsall = SVM()
    onevsall.fit(train, v)

    onevsallpred = onevsall.predict(test)
    onevsallpred = np.where(onevsallpred==1, counter)
    df_results[counter] = onevsallpred
```

```
df_results['pred'] = df_results.max(axis=1)

temp = df_results['pred'].tolist()
v = s.tolist()

numCorr = 0

for i in range(len(temp)):
    if temp[i] == v[i]:
        numCorr +=1

print(numCorr/len(temp))
```

```
0.758
```

```
conf_matr = np.zeros([10,10])

for i in range(len(temp)):
    conf_matr[temp[i], v[i]] +=1

print(conf_matr)
```

```
[[ 81   0   0   0   0   0   4   0   1   0]
 [  0 121   0   0   1   0   0   0   0   0]
 [  2  21  77   2   0   0   2   5   4   0]
 [  0   5   1  90   0   6   6   5   1   1]
 [  1   5   0   0  92   1   7   0   0   2]
 [  7   7   2  20   4  35   6   8   3   0]
 [  5   2   1   0   0   1  78   0   0   0]
 [  2  12   4   0   0   0   0  81   0   0]
 [  3  11   4   6   3   2   4   6  45   2]
 [  0   4   1   3   8   0   0  17   1  58]]
```

```
df_results_ovo = pd.DataFrame()

counter= 0

for i in range(10):
    for j in range(i+1, 10):
        y = np.where(trainL==i)
        x = np.where(trainL==j)
        z = np.append(y[0], x[0])
        selected = train[z]
        selectedL = trainL[z]

        v = np.where(selectedL==i, 1, -1)
        onevsone = SVM()
        onevsone.fit(selected, v)

        pred = onevsone.predict(test)
        pred = np.where(pred==1, i,j)

        df_results_ovo[counter] = pred

        counter+=1
```

```
df_results_ovo['pred'] = df_results_ovo.mode(axis=1)[0]

temp = df_results_ovo['pred'].tolist()
v = s.tolist()

numCorr = 0

for i in range(len(temp)):
    if temp[i] == v[i]:
        numCorr +=1
```

```
df_results_ovo['pred'] = df_results_ovo.mode(axis=1)[0]

temp = df_results_ovo['pred'].tolist()
v = s.tolist()

numCorr = 0

for i in range(len(temp)):
    if temp[i] == v[i]:
        numCorr +=1

print(numCorr/len(temp))
```

```
0.122
```

```
conf_matr = np.zeros([10,10])

for i in range(len(temp)):
    conf_matr[temp[i], v[i]] +=1

print(conf_matr)
```

```
[[  0  86   0   0   0   0   0   0   0   0]
 [  0 122   0   0   0   0   0   0   0   0]
 [  0 113   0   0   0   0   0   0   0   0]
 [  0 115   0   0   0   0   0   0   0   0]
 [  0 108   0   0   0   0   0   0   0   0]
 [  0  92   0   0   0   0   0   0   0   0]
 [  0  87   0   0   0   0   0   0   0   0]
 [  0  99   0   0   0   0   0   0   0   0]
 [  0  86   0   0   0   0   0   0   0   0]
 [  0  92   0   0   0   0   0   0   0   0]]
```