# Introduction to Machine Learning
# Homework 6: Support Vector Machines

Jack Langerman

1. Consider the data set for four points with features $\mathbf{x}_i = (x_{i1}, x_{i2})$ and binary class labels $y_i = \pm 1$.

| | | | | |
|---|---|---|---|---|
| $x_{i1}$ | 0 | 1 | 1 | 2 |
| $x_{i2}$ | 0 | 0.3 | 0.7 | 1 |
| $y_i$ | -1 | -1 | 1 | 1 |

   (a) Find a linear classifier that separates the two classes. Your classifier should be of the form

$$\hat{y} = \begin{cases} 1 & \text{if } b + w_1 x_1 + w_2 x_2 > 0 \\ -1 & \text{if } b + w_1 x_1 + w_2 x_2 < 0 \end{cases}$$

   State the intercept $b$ and weights $w_1$ and $w_2$ for your classifier. Note there is no unique answer as there are multiple linear classifiers that could separate the classes.

   line at $x_{i2} = 0.5$ for all $x_{i1}$:

   $b = -0.5, w_1 = 0, w_2 = 1$

$$\hat{y} = \begin{cases} 1 & \text{if } b + w_1 x_1 + w_2 x_2 > 0 \\ -1 & \text{if } b + w_1 x_1 + w_2 x_2 < 0 \end{cases}$$

   (b) Find the maximum $\gamma$ such that $y_i(b + w_1 x_{i1} + w_{i2} x_{i2}) \geq \gamma$, for all $i$, for the classifier in part (a)?

$$\gamma = 0.5$$

   (c) Compute the margin of the classifier $m = \frac{\gamma}{\|\mathbf{w}\|}, \quad \|\mathbf{w}\| = \sqrt{w_1^2 + w_2^2}$.

$$\|\mathbf{w}\| = \sqrt{w_1^2 + w_2^2} = \sqrt{0^2 + 1^2} = 1$$

$$m = \frac{\gamma}{\|\mathbf{w}\|} = \frac{0.5}{1} = 0.5$$

   (d) Which samples $i$ are on the margin for your classifier?
   Assuming $i = 0..3$ then samples $x_i, i = \{1, 2\}$ are in the margin.

$$(1)(0) + (0.3)(1) - 0.5 = -0.2$$

$$|-0.2| < 0.5$$

2. Consider the data set with scalar features $x_i$ and binary class labels $y_i = \pm 1$.

| $x_i$ | 0 | 1.3 | 2.1 | 2.8 | 4.2 | 5.7 |
|---|---|---|---|---|---|---|
| $y_i$ | -1 | -1 | -1 | 1 | -1 | 1 |

Consider a linear classifier for this data of the form,

$$\hat{y} = \begin{cases} 1 & z > 0 \\ -1 & z < 0, \end{cases} \qquad z = x - t,$$

where $t$ is a threshold. For each threshold $t$, let $J(t)$ denote the sum hinge loss,

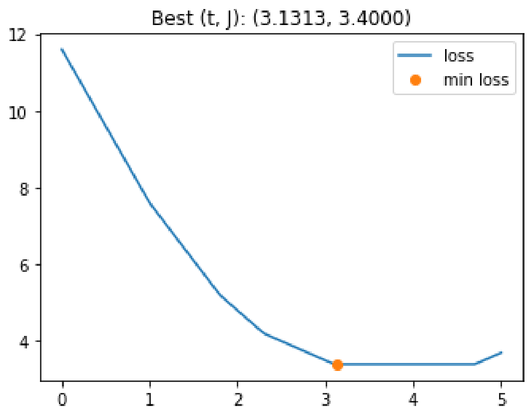
$$J(t) = \sum_i \epsilon_i, \quad \epsilon_i = \max(0, 1 - y_i z_i).$$

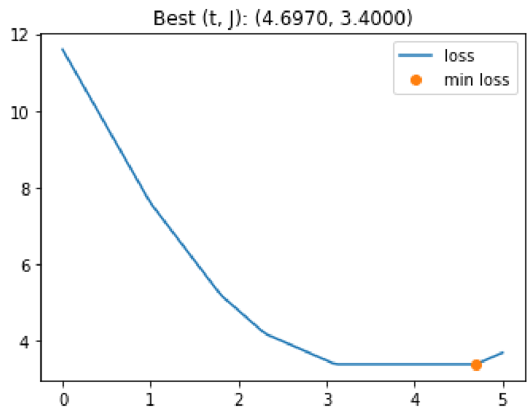
(a) Write a short python program to plot $J(t)$ vs. $t$ for 100 values of $t$ in the interval $t \in [0, 5]$.

```
x = np.array([0, 1.3, 2.1, 2.8, 4.2, 5.7])
y = np.array([-1, -1, -1, 1, -1, 1 ])

def predict(x, t):
    z = x-t
    # 1 if z>0, -1 if z<0, break if z==0
    return 1 if z > 0 else (-1 if z < 0 else float('NaN'))

def hindgeLoss(x, y, t):
    z = x-t
    epsilon = np.maximum(0, 1 - y*z)
    return np.sum(epsilon)
```

```
plt.figure(figsize=(12,4))
for i, cmp in enumerate([lambda a, b: a<b, lambda a, b: a<=b]):
    plt.subplot(1,2,i+1)
    hist = {}
    minLoss = float('Inf')
    t_best = -1
    hist['t'] = []
    hist['J'] = []

    for t in np.linspace(0, 5, 100):
        J = hindgeLoss(x, y, t)

        if cmp(J, minLoss):
            minLoss = J
            t_best = t

        hist['t'].append(t)
```

```python
        hist['J'].append(J)

    plt.title("Best (t, J): ({:.4f}, {:.4f})".format(t_best, minLoss))
    plt.plot(hist['t'], hist['J'], label="loss")
    plt.plot(t_best, minLoss, 'o', label="min loss")
    plt.legend()
```




(b) Based on the plot, what is one value of $t$ that minimizes $J(t)$.

$$t = 4.5$$

(c) For the value of $t$ in part (b), find the corresponding slack variables $\epsilon_i$.

```python
t = 4.5
z = x-t
epsilon = np.maximum(0, 1 - y*z)

print(epsilon)
```

$$[\quad 0 \quad 0 \quad 0 \quad 2.7 \quad 0.7 \quad 0 \quad]$$

(d) Which samples $i$ violate the margin ($\epsilon_i > 0$) and which samples $i$ are misclassified ($\epsilon_i > 1$).

| x | y | y_hat | correct | slack |
|---|---|---|---|---|
| 0.0 | -1 | -1 | True | 0.0 |
| 1.3 | -1 | -1 | True | 0.0 |
| 2.1 | -1 | -1 | True | 0.0 |
| 2.8 | 1 | -1 | False | 2.7 |
| 4.2 | -1 | -1 | True | 0.7 |
| 5.7 | 1 | 1 | True | 0.0 |

3. Consider an image recognition problem, where an image $\mathbf{X}$ and filter $\mathbf{W}$ are $4 \times 4$ matrices:

$$\mathbf{X} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}, \quad \mathbf{W} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}.$$

(a) Recall that in linear classification, the $4 \times 4$ image matrices $\mathbf{X}$ and $\mathbf{W}$ can be represented as 16-dimensional vectors, $\mathbf{x} = \text{vec}(\mathbf{X})$ and $\mathbf{w} = \text{vec}(\mathbf{W})$ by stacking the columns of the matrices vertically. What are $\mathbf{x}$ and $\mathbf{w}$ for the matrices above.

$$\mathbf{x} = [0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0]^\mathsf{T}$$

$$\mathbf{w} = [0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0]^\mathsf{T}$$

(b) What is the inner product $z = \mathbf{w}^\mathsf{T}\mathbf{x}$.

$$\mathbf{w}^\mathsf{T}\mathbf{x} = 2$$

(c) What is the inner product $z = \mathbf{w}^\mathsf{T}\mathbf{x}_{\text{right}}$ where $\mathbf{x}_{\text{right}}$ is the vector corresponding to the matrix $\mathbf{X}$ right shifted by one pixel with the left column filled with zeros.

$$\mathbf{w}^\mathsf{T}\mathbf{x}_{\text{right}} = 0$$

(d) What is the inner product $z = \mathbf{w}^\mathsf{T}\mathbf{x}_{\text{left}}$ where $\mathbf{x}_{\text{left}}$ is the vector corresponding to the matrix $\mathbf{X}$ left shifted by one pixel with the right column filled with zeros.

$$\mathbf{w}^\mathsf{T}\mathbf{x}_{\text{right}} = 2$$

(e) Write the python command that can covert a $4 \times 4$ image matrix, `Xmat` to the 16-dimensional vector, `x`. What is the python command to go from `x` to `Xmat`.

```
def vec(A):
    B = []
    [B.extend(A[:, i]) for i in range(A.shape[1])]
    return np.array(B)

X = [
 [0, 0, 0,  0],
 [0, 0, 1,  0],
 [0, 0, 1,  0],
 [0, 0, 1,  0]
]

x = vec(X)
print("X =",x)
```

```
x = [0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0]
```

```
print(x.reshape(4,4, order='F'))
```

```
[[0 0 0 0]
 [0 0 1 0]
 [0 0 1 0]
 [0 0 1 0]]
```

4. Consider the data set with scalar features $x_i$ and binary class labels $y_i = \pm 1$.

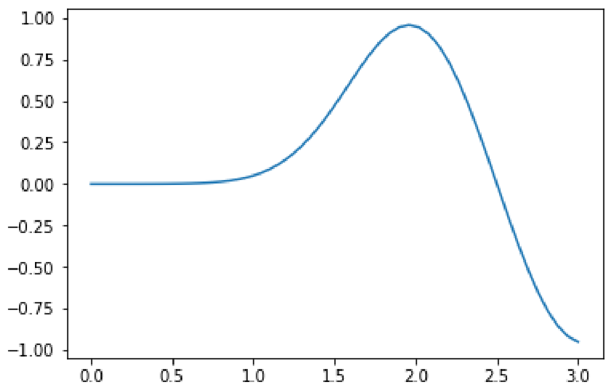
| $x_i$ | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| $y_i$ | 1 | -1 | 1 | -1 |

A support vector classifier is of the form

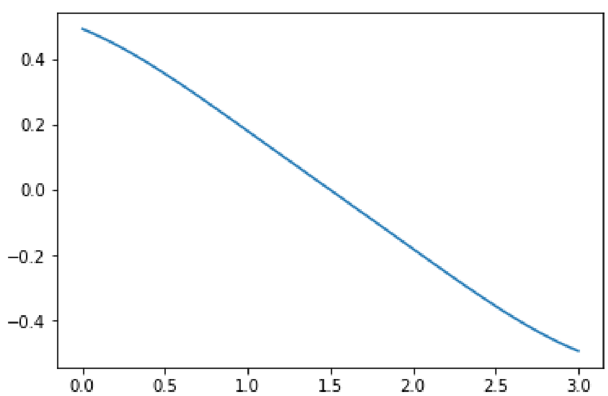
$$\hat{y} = \begin{cases} 1 & z > 0 \\ -1 & z < 0, \end{cases} \qquad z = \sum_i \alpha_i y_i K(x_i, x),$$

where $K(x, x')$ is the radial basis function, $K(x, x') = e^{-\gamma(x-x')^2}$, and $\gamma > 0$ and $\boldsymbol{\alpha} = [\alpha_1, \ldots, \alpha_4]$ are parameters of the classifier.

(a) Use python to plot $z$ vs. $x$ and $\hat{y}$ vs. $x$ when $\gamma = 3$ and $\boldsymbol{\alpha} = [0, 0, 1, 1]$.




(b) Repeat (a) with $\gamma = 0.3$ and $\boldsymbol{\alpha} = [1, 1, 1, 1]$.




(c) Which classifier makes more errors on the training data.

The classifier in part (b) makes more errors on the training set

# HW 6 notebook

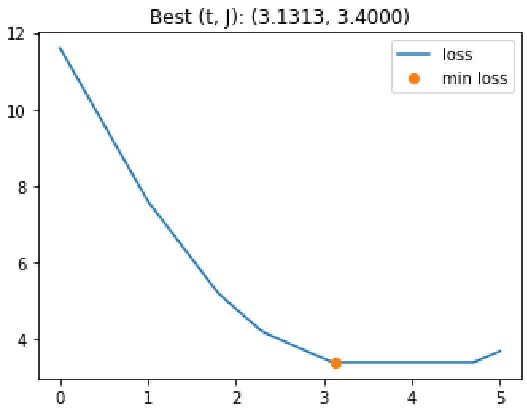
November 19, 2017

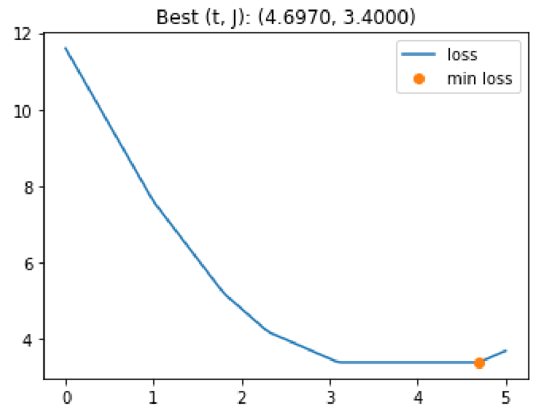
```
In [19]: import numpy as np
         import matplotlib
         import matplotlib.pyplot as plt

         %matplotlib inline

In [20]: x = np.array([0, 1.3, 2.1, 2.8, 4.2, 5.7])
         y = np.array([-1, -1, -1, 1, -1, 1 ])

In [84]: def predict(x, t):
             z = x-t
             return 1 if z > 0 else (-1 if z < 0 else float('NaN')) # 1 if z>0, -1 if z<0, break
             return res

In [85]: def hindgeLoss(x, y, t):
             z = x-t
             epsilon = np.maximum(0, 1 - y*z)
             return np.sum(epsilon)
```

### 0.0.1 Part 2 (a)

```
In [86]: plt.figure(figsize=(12,4))
         for i, cmp in enumerate([lambda a, b: a<b, lambda a, b: a<=b]):
             plt.subplot(1,2,i+1)
             hist = {}
             minLoss = float('Inf')
             t_best = -1
             hist['t'] = []
             hist['J'] = []

             for t in np.linspace(0, 5, 100):
                 J = hindgeLoss(x, y, t)

                 if cmp(J, minLoss):
                     minLoss = J
                     t_best = t

                 hist['t'].append(t)
```

```
        hist['J'].append(J)

    plt.title("Best (t, J): ({:.4f}, {:.4f})".format(t_best, minLoss))
    plt.plot(hist['t'], hist['J'], label="loss")
    plt.plot(t_best, minLoss, 'o', label="min loss")
    plt.legend()
```



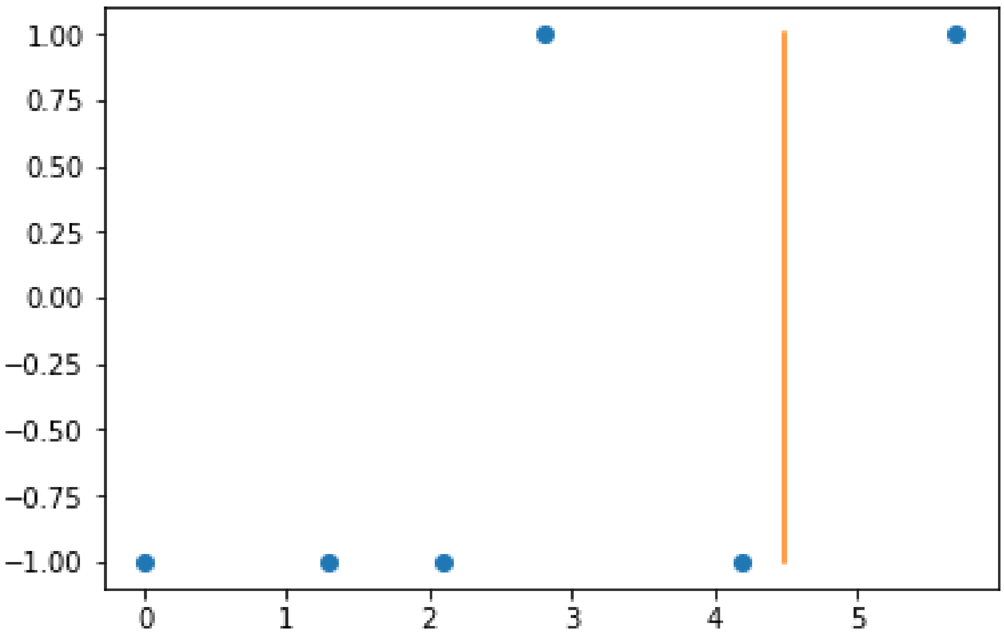

### 0.0.2 Part 2 (c)

```
In [111]: t = 4.5
          z = x-t
          epsilon = np.maximum(0, 1 - y*z)

          print(epsilon)

[ 0.   0.   0.   2.7  0.7  0. ]
```

### 0.0.3 Part 2 (d)

```
In [112]: def predict(x, t):
              z = x-t
              res = np.zeros_like(z)
              res[np.greater(z, 0)] = 1
              res[np.less(z, 0)] = -1
              res[np.equal(z, 0)] = float('NaN')
              return res

In [120]: plt.plot(x, y, 'o')
          plt.plot([t,t], [-1, 1])

          print(" x\ty\ty_hat\tcorrect\tslack")
          print("", *["{}\t{}\t{}\t{}\t{}\n".format(*u) for u in zip(x, y, predict(x, t), predic
```

```
 x         y         y_hat        correct        slack
 0.0        -1         -1.0         True        0.0
 1.3        -1         -1.0         True        0.0
 2.1        -1         -1.0         True        0.0
 2.8        1         -1.0         False        2.7
 4.2        -1         -1.0         True        0.7000000000000002
 5.7        1         1.0         True        0.0
```




```
In [122]: print(" x & y & y_hat & correct & slack \\hline")
          print("", *["{} & {} & {} & {} & {} \\\\\n".format(*u) for u in zip(x, y, predict(x, t
```

```
 x & y & y_hat & correct & slack \hline
 0.0 & -1 & -1.0 & True & 0.0 \\
 1.3 & -1 & -1.0 & True & 0.0 \\
 2.1 & -1 & -1.0 & True & 0.0 \\
 2.8 & 1 & -1.0 & False & 2.7 \\
 4.2 & -1 & -1.0 & True & 0.7000000000000002 \\
 5.7 & 1 & 1.0 & True & 0.0 \\
```

### 0.0.4 Part 3 (a)

```
In [158]: X = [
            [0, 0, 0,  0],
```

```
     [0, 0, 1,  0],
     [0, 0, 1,  0],
     [0, 0, 1,  0]
    ]

    W = [
     [0, 0, 0,  0],
     [0, 1, 1,  0],
     [0, 1, 1,  0],
     [0, 0, 0,  0]
    ]

    X, W = np.array(X), np.array(W)


    def vec(A):
        B = []
        [B.extend(A[:, i]) for i in range(A.shape[1])]
        return np.array(B)

    x = vec(X)
    w = vec(W)

    print("X =",x)
    print("W =",w)
    print("x \\dot w =", np.dot(x, w))
```

```
X = [0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0]
W = [0 0 0 0 0 1 1 0 0 1 1 0 0 0 0 0]
x \dot w = 2
```

### 0.0.5 Part 3 (e)

```
In [159]: print(x.reshape(4,4, order='F'))
```

```
[[0 0 0 0]
 [0 0 1 0]
 [0 0 1 0]
 [0 0 1 0]]
```

### 0.0.6 Part 4 (a)

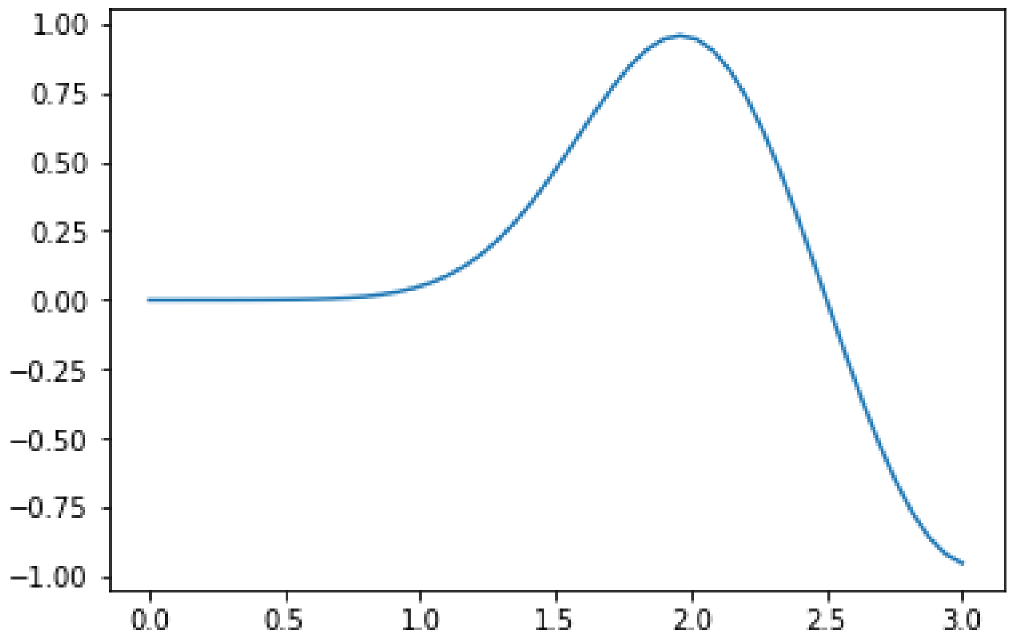
```
In [275]: # data
          x = np.array([0, 1, 2, 3])
          y = np.array([1, -1, 1, -1])

          # params
```

```
alpha = np.array([0, 0, 1, 1]) # dual vector
gamma = 3                      # param for rbf

def rbf(a, b):
    return np.exp(-gamma * (a-b)**2)

def predict(x, alpha, gamma, K):
    # data
    data_x = np.array([0, 1, 2, 3])
    data_y = np.array([1, -1, 1, -1])

    # score
    z = np.matmul(K(x, data_x), alpha*data_y)

    # prediction
    y_hat = np.zeros_like(z)
    y_hat[np.greater(z, 0)] = 1
    y_hat[np.less(z, 0)] = -1

    return y_hat
```

```
In [276]: # data
data_x = np.array([0, 1, 2, 3])
data_y = np.array([1, -1, 1, -1])

# params
alpha = np.array([0, 0, 1, 1]) # dual vector
gamma = 3                      # param for rbf

# kernal function
def K(a, b):
    return np.exp(-gamma * (a-b)**2)

# points to classify
x = np.linspace(0,3)[:, None]

# interior score
z = np.matmul(K(x, data_x), alpha*data_y)

# prediction
y_hat = predict(data_x[:,None], alpha, gamma, K)

# accuracy
acc_a = np.mean(np.equal(y_hat, data_y))
print("\n\t{}% accuracy\n".format(100*acc_a))

# plot
plt.plot(x, z)
```

```
75.0% accuracy
```

```
Out[276]: [<matplotlib.lines.Line2D at 0x109eb5e48>]
```



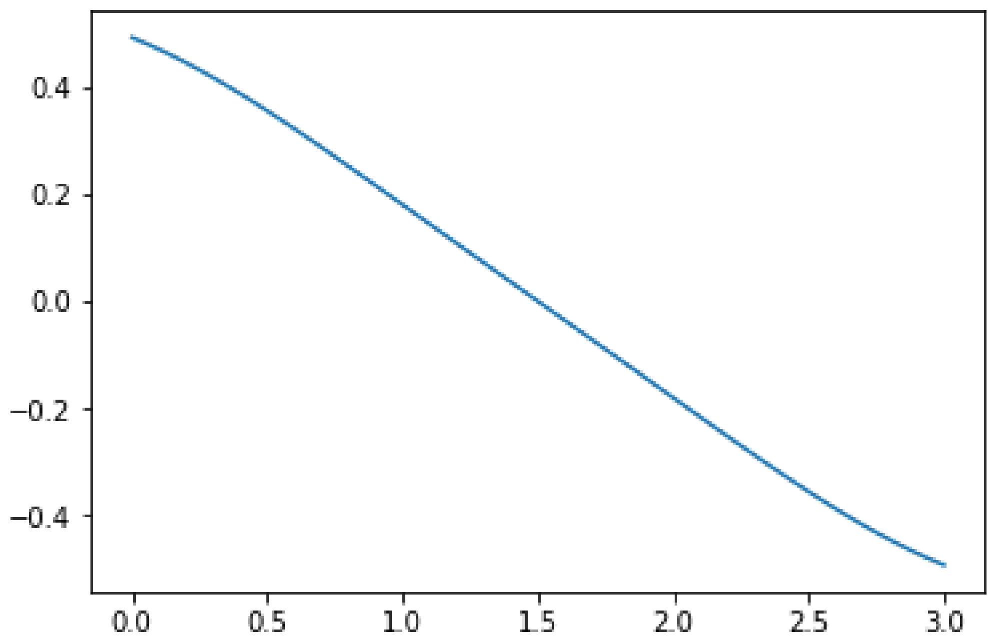

### 0.0.7 Part 4 (b)

```
In [277]: # data
          data_x = np.array([0, 1, 2, 3])
          data_y = np.array([1, -1, 1, -1])

          # params
          alpha = np.array([1, 1, 1, 1]) # dual vector
          gamma = 0.3                     # param for rbf

          # kernal function
          def K(a, b):
              return np.exp(-gamma * (a-b)**2)

          # points to classify
          x = np.linspace(0,3)[:, None]

          # interior score
          z = np.matmul(K(x, data_x), alpha*data_y)
```

```
# prediction
y_hat = predict(data_x[:,None], alpha, gamma, K)

# accuracy
acc_b = np.mean(np.equal(y_hat, data_y))
print("\n\t{}% accuracy\n".format(100*acc_b))

# plot
plt.plot(x, z)
```

```
50.0% accuracy
```

```
Out[277]: [<matplotlib.lines.Line2D at 0x109fbceb8>]
```




### 0.0.8 Part 4 (c)

```
In [281]: print("\n\ta) {}% accuracy".format(100*acc_a))
          print("\tb) {}% accuracy\n".format(100*acc_b))
          print("\tThe settings in part (a) yield higher accuracy")
```

```
a) 75.0% accuracy
```

b) 50.0% accuracy

The settings in part (a) yield higher accuracy

```
In [ ]:
```