1.

$z_1 = (-2, -2)$ -1

$z_2 = (4, -5)$ -1

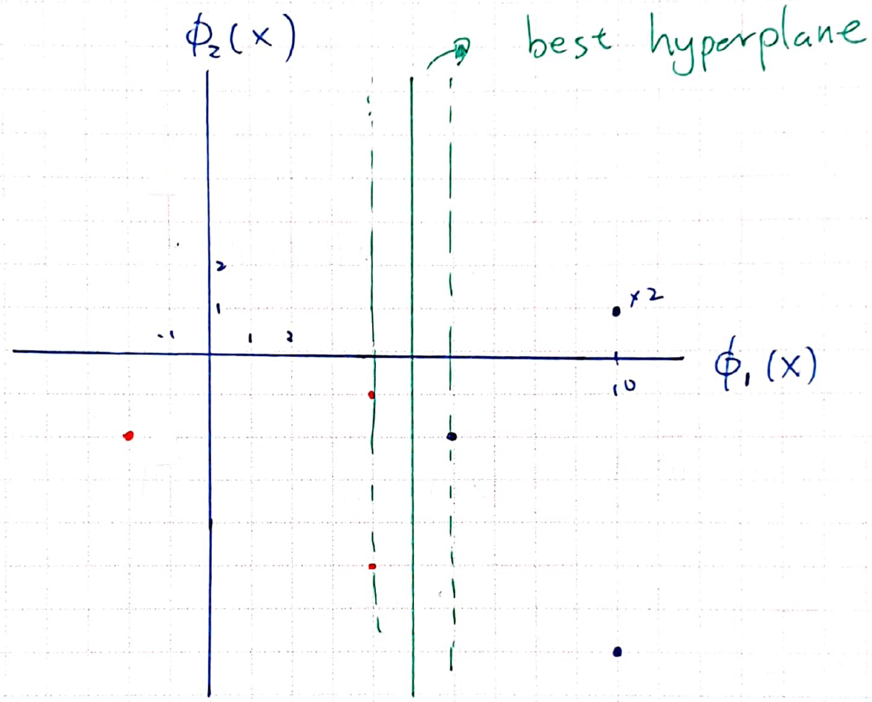
$z_3 = (4, -1)$ -1

$z_4 = (6, -2)$ 1

$z_5 = (10, -7)$ 1

$z_6 = (10, 1)$ 1

$z_7 = (10, 1)$ 1



$\phi_1(x) = 5$ is the best hyperplane since it expands the thickest margin.

Q2&Q3

March 26, 2019

```
In [17]: import numpy as np
         from cvxopt import matrix
         from cvxopt.solvers import qp
         from sklearn.preprocessing import PolynomialFeatures

In [18]: poly = PolynomialFeatures(interaction_only=False)

In [29]: X = np.array([[1, 0], [0, 1], [0, -1], [-1, 0], [0, 2], [0, -2], [-2, 0]])
         Y = np.array([-1, -1, -1, 1, 1, 1, 1])
         Xt=poly.fit_transform(X)
         print(Xt)
```

```
[[ 1.  1.  0.  1.  0.  0.]
 [ 1.  0.  1.  0.  0.  1.]
 [ 1.  0. -1.  0. -0.  1.]
 [ 1. -1.  0.  1. -0.  0.]
 [ 1.  0.  2.  0.  0.  4.]
 [ 1.  0. -2.  0. -0.  4.]
 [ 1. -2.  0.  4. -0.  0.]]
```

```
In [20]: A = matrix(Y,(1,7),'d')
         b =matrix(0,(1,1),'d')
         h = matrix(0,(7,1),'d')
         G = matrix(-np.eye(7),(7,7),'d')
         p = matrix(-1,(7,1),'d')
         XX = Xt@Xt.T
         YY = Y.reshape(7,1)@Y.reshape(1,7)
         Q = matrix(XX*YY,(7,7),'d')

In [21]: sol=qp(Q,p,G, h,A,b)
```

```
     pcost       dcost       gap    pres   dres
 0: -2.1712e+00 -5.0654e+00  2e+01  3e+00  2e+00
 1: -3.8978e+00 -5.7620e+00  6e+00  1e+00  7e-01
 2: -1.7493e+00 -2.7818e+00  1e+00  5e-16  6e-15
 3: -1.9825e+00 -2.0130e+00  3e-02  4e-16  1e-15
 4: -1.9997e+00 -2.0001e+00  4e-04  7e-16  2e-15
```

```
 5: -2.0000e+00 -2.0000e+00  4e-06  3e-16  1e-15
 6: -2.0000e+00 -2.0000e+00  4e-08  3e-16  1e-15
Optimal solution found.
```

```
In [25]: alpha = np.array(sol['x'])
         print(alpha)
```

```
[[3.75668650e-08]
 [9.99999978e-01]
 [9.99999977e-01]
 [1.33333334e+00]
 [3.33333329e-01]
 [3.33333328e-01]
 [5.23032669e-10]]
```

Q2: 由上可知道 2,3,4,5,6 F support vectores

```
In [59]: SV = [[0, 1], [0, -1], [-1, 0], [0, 2], [0, -2]]
```

```
In [30]: w = 0
         for i in range(len(alpha)):
             w = w +Xt[i]*alpha[i]*Y[i]
         print(w)
```

```
[ 5.18168890e-17 -1.33333337e+00 -2.22044605e-16  1.33333330e+00
  0.00000000e+00  6.66666670e-01]
```

```
In [58]: b=Y[1] -Xt[1]@w
         print(b)
```

```
-1.6666666700348152
```

Q3: 由上可知 $-4x_1 + 4x_1^2 + 2x_2^2 = 5$ 是 linear curve

Q4: the kernel in question 2 and 4 are different space, so they cannot be the same. (one's dimmension is 2, the other is 6)

5. $$\exp(-x^2) = \frac{1}{\exp(x^2)} \stackrel{?}{=} \frac{1}{\|\tilde{\Phi}(x)\|}$$

show $$\exp(x^2) = \|\tilde{\Phi}(x)\|$$

$$\|\tilde{\Phi}(x)\| = \sqrt{1^2 + \left(\sqrt{\tfrac{2}{1!}}x\right)^2 + \left(\sqrt{\tfrac{2^2}{2!}}x^2\right)^2 + \cdots}$$

$$\Rightarrow \|\tilde{\Phi}(x)\|^2 = 1 + \frac{2}{1!}x^2 + \frac{2^2}{2!}x^4 + \cdots$$

$$= 1 + \frac{2x^2}{1!} + \frac{(2x^2)^2}{2!} + \cdots$$

$$= \exp(2x^2) = \left[\exp(x^2)\right]^2$$

$$\Rightarrow \|\tilde{\Phi}(x)\| = \exp(x^2)$$

6.

6.

$$\cos(x, x') = \frac{x^T x'}{\|x\| \cdot \|x'\|}$$

Mercer condition

let $K_{ij} = \cos(x_i, x_j)$

$$K = \begin{bmatrix} \frac{x_1^T x_1}{\|x_1\| \|x_1\|} & \frac{x_1^T x_2}{\|x_1\| \|x_2\|} & \cdots & \frac{x_1^T x_N}{\|x_1\| \|x_N\|} \\ \vdots & & \ddots & \vdots \\ \frac{x_N^T x_1}{\|x_N\| \|x_1\|} & & & \frac{x_N^T x_N}{\|x_N\| \|x_N\|} \end{bmatrix}$$

$$= \begin{bmatrix} \frac{x_1}{\|x_1\|} & \frac{x_2}{\|x_2\|} & \cdots & \frac{x_N}{\|x_N\|} \end{bmatrix}^T \begin{bmatrix} \frac{x_1}{\|x_1\|} & \frac{x_2}{\|x_2\|} & \cdots & \frac{x_N}{\|x_N\|} \end{bmatrix}$$

$$= ZZ^T$$

this is always PS.D and symmetric

let

$$Z = \begin{bmatrix} \frac{x_1^T}{\|x_1\|} \\ \frac{x_2^T}{\|x_2\|} \\ \vdots \\ \frac{x_N^T}{\|x_2\|} \end{bmatrix}$$

$\cos(x, x')$ is a valid kernel

7.

$$\mathcal{L}(R, c, \lambda)$$

$$= R^2 + \sum_{n=1}^{N} \lambda_n \left(\|z_n - c\|^2 - R^2 \right)$$

8. KKT, conditions

primimal feasible. $\|z_n - c\|^2 \le R^2, \ \forall n$

dual-feasible $\lambda_n \ge 0 \quad \forall n$

and

$\lambda_n \left(\|z_n - c\|^2 - R^2 \right) = 0 \quad \forall n.$

And (D)

$$\max_{\alpha_n \ge 0} \left(\min_{R, c} \mathcal{L}(R, c, \lambda) \right)$$

inner optimal.

$$\frac{\partial L}{\partial R} = 0 \Rightarrow 2R - 2R \sum_{n=1}^{N} \lambda_n = 0$$

$$\Rightarrow R\left(1 - \sum_{n=1}^{N} \lambda_n\right) = 0 \quad \text{— (a)}$$

$$\frac{\partial L}{\partial c_i} = 0 \Rightarrow -2 \sum_{n=1}^{N} \lambda_n \left(z_n(i) - c_i \right) = 0$$

$$\Rightarrow c_i \sum_{n=1}^{N} \lambda_n = \sum_{n=1}^{N} \lambda_n z_n(i)$$

$$\Rightarrow c = \frac{\sum^{N} \lambda_i z_n}{\sum^{N} \lambda_i} \quad \left(\text{if } \sum_{n=1}^{N} \lambda_i \ne 0 \right) \quad \text{— (b)}$$

9. $R > 0$. by (a) $\Rightarrow \sum_{n=1}^{N} \lambda_i = 1$

$$\Rightarrow C = \sum_{n=1}^{N} \lambda_n z_n$$

then $L(R, C, \lambda) = \sum^{N} \lambda_n \| z_n - C \|^2$

(D')

$$= \sum_{n=1}^{N} \lambda_n \| z_n - \sum_{m=1}^{N} \lambda_m z_m \|^2$$

$$\text{s.t. } \sum_{i=1}^{N} \lambda_i = 1$$

10.

$$L(R, C, \lambda) = \sum_{n=1}^{N} \lambda_n \left(z_n - \sum_{m=1}^{N} \lambda_m z_m \right)^T \left(z_n - \sum_{m=1}^{N} \lambda_m z_m \right)$$

$$= \sum_{n=1}^{N} \lambda_n z_n^T z_n - 2 \sum_{n=1}^{N} \sum_{m=1}^{N} \lambda_n \lambda_m z_m^T z_n + \sum_{n=1}^{N} \lambda_n \sum_{m=1}^{N} \sum_{j=1}^{N} \lambda_m \lambda_j z_m^T z_j$$

$$= \sum_{n=1}^{N} \lambda_n K(x_n, x_n) - 2 \sum_{n=1}^{N} \sum_{m=1}^{N} \lambda_n \lambda_m K(x_m, x_n) + \sum_{n=1}^{N} \lambda_n \left[\sum_{m=1}^{N} \sum_{j=1}^{N} \lambda_m \lambda_j K(x_m, x_j) \right]$$

$$= \sum_{n=1}^{N} \lambda_n K(x_n, x_n) - \sum_{n=1}^{N} \sum_{m=1}^{N} \lambda_n \lambda_m K(x_m, x_n)$$

$\rightarrow$ ~~In~~ to get minimized R^2.

We take any $\lambda_i = 0$. then

$$R = \sqrt{\| z_i - C \|} = \sqrt{K(z_i, z_i) - 2 \sum_{m=1}^{N} \lambda_m K(x_m, x_i) + \sum_{n=1}^{N} \sum_{m=1}^{N} \lambda_n \lambda_m K(x_n, x_m)}$$

11. In Soft-margin SVM

$\beta_n = C - \alpha_n \rightarrow$ multiplier for $(-\xi)$

If $C \geq \max \alpha_n^*$, α_n^* in hard-margin.

this implie all $\beta_n \geq 0$

$\Rightarrow$ all $\xi_i = 0$, this makes soft-margin SVM the same as hard-margin SVM.

12. Let original SVM with $K(x, x')$ is

$$g_{SVM}(x) = \text{sign}\left(\sum_{SV_n} \alpha_n y_n K(X_n, X) + \left(y_s - \sum \alpha_n y_n K(X_n, X_s)\right)\right)$$

New $\tilde{g}_{SVM} = \text{sign}\left(\sum_{SV} \tilde{\alpha}_n y_n p K(x_n, x) + \left(y_s - \sum \tilde{\alpha}_n y_n p(x_n, x_s)\right)\right)$,

with $\tilde{K}(x, x') = p\, K(x, x')$

if let $\tilde{\alpha}_n = \frac{\alpha_n}{p}$, then $g_{SVM} = \tilde{g}_{SVM}$

and to make sure bounded SV and SV are the same

for $\Rightarrow \tilde{\alpha}_n = \frac{\alpha_n}{p} = \boxed{\frac{C}{p} = \tilde{C}}$

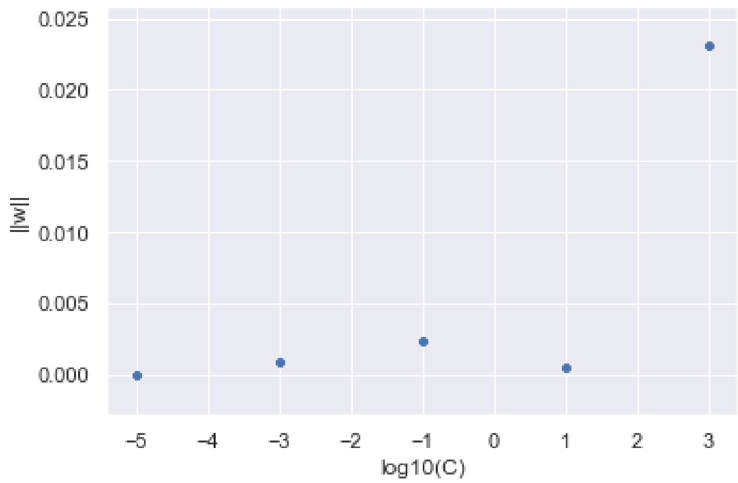
In [1]:
```python
import numpy as np
from sklearn import svm
import matplotlib.pyplot as plt
import pandas as pd
```

In [2]:
```python
data_train = pd.read_csv("data_train.csv")
data_test = pd.read_csv("data_test.csv")
X = data_train[['intensity','symmetry']]
```

In [3]:
```python
y2 = np.where(data_train["digit"] ==2, 1 ,-1)
Clist = [-5,-3,-1,1,3]
```

In [4]:
```python
w = []
for c in Clist:
    result = svm.SVC(C = 10**c, kernel = "linear").fit(X,y2)
    w = w +[np.linalg.norm(result.coef_)]
```

In [5]:
```python
import seaborn as sns; sns.set()
import matplotlib.pyplot as plt
df=pd.DataFrame({'log10(C)': Clist, '||w||': w})
ax = sns.scatterplot(x='log10(C)', y='||w||',data=df)
```



In [6]:
```python
w
```

Out[6]:
```
[1.1763105414828839e-05,
 0.0009147796947899397,
 0.002331791308596588,
 0.000513661010677975,
 0.023067043402405518]
```

Q13: The norm of the weight is not always increasing with C

In [1]:
```python
import numpy as np
from svm import*
from svmutil import *
import matplotlib.pyplot as plt
import pandas as pd
```

In [2]:
```python
data_train = pd.read_csv("data_train.csv")
data_test = pd.read_csv("data_test.csv")
X = data_train[['intensity','symmetry']]
X_test = data_test[['intensity','symmetry']]
```

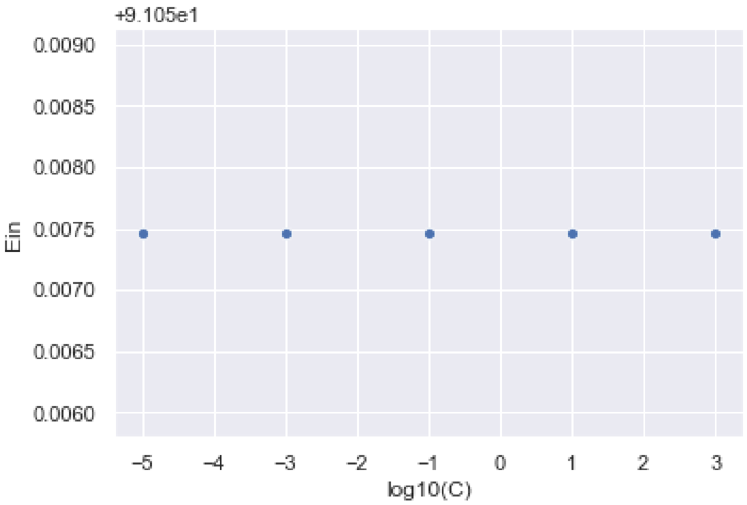
In [3]:
```python
y4 = np.where(data_train["digit"] ==4, 1 ,-1)
y4_test = np.where(data_test["digit"] ==4, 1 ,-1)
Clist = [-5,-3,-1,1,3]
```

In [4]:
```python
prob = svm_problem(y4,X.values)
param = svm_parameter()
param.kernel_type = POLY
param.coef0 =1
param.degree = 2
model = svm_train(prob, param)
```

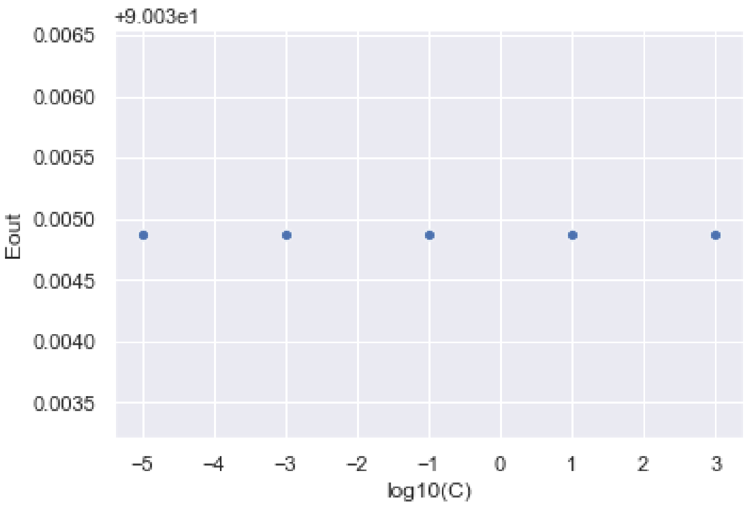
In [5]:
```python
Ein = []
Eout = []
for c in Clist:
    param.C = 10**c
    model = svm_train(prob, param)
    Ein = Ein +[svm_predict(y4,X.values,model)[1][0]]
    Eout = Eout +[svm_predict(y4_test,X_test.values,model)[1][0]]
```

```
Accuracy = 91.0575% (6639/7291) (classification)
Accuracy = 90.0349% (1807/2007) (classification)
Accuracy = 91.0575% (6639/7291) (classification)
Accuracy = 90.0349% (1807/2007) (classification)
Accuracy = 91.0575% (6639/7291) (classification)
Accuracy = 90.0349% (1807/2007) (classification)
Accuracy = 91.0575% (6639/7291) (classification)
Accuracy = 90.0349% (1807/2007) (classification)
Accuracy = 91.0575% (6639/7291) (classification)
Accuracy = 90.0349% (1807/2007) (classification)
```

In [6]:
```
import seaborn as sns; sns.set()
import matplotlib.pyplot as plt
df=pd.DataFrame({'log10(C)': Clist, 'Ein': Ein,'Eout':Eout})
ax = sns.scatterplot(x='log10(C)', y='Ein',data=df)
```



In [7]:
```
ax = sns.scatterplot(x='log10(C)', y='Eout',data=df)
```



Q14: No matter how C changes, Ein and Eout remains the same. This suggest that Hard-margin SVM does a good job.

In [ ]:

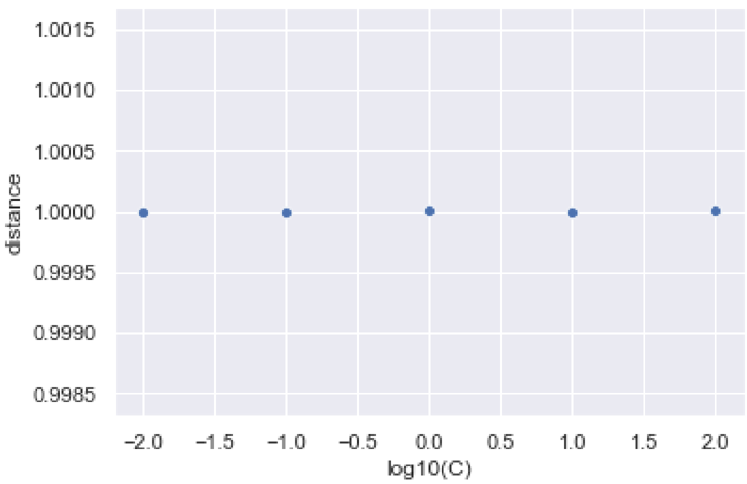
In [1]:
```
import numpy as np
from sklearn import svm
import matplotlib.pyplot as plt
import pandas as pd
```

In [2]:
```
data_train = pd.read_csv("data_train.csv")
data_test = pd.read_csv("data_test.csv")
X = data_train[['intensity','symmetry']]
```

In [3]:
```
y0 = np.where(data_train["digit"] ==0, 1 ,-1)
Clist = [-2,-1,0,1,2]
```

In [4]:
```
d = []
for c in Clist:
    result = svm.SVC(C = 10**c, kernel = "rbf", gamma = 80).fit(X,y0)
    K = [ind for ind, coef in  enumerate(abs(result.dual_coef_[0])) if coef > 0 and
    d = d + [1/abs(result.decision_function(result.support_vectors_[K])).mean()]
```

In [5]:
```
import seaborn as sns; sns.set()
import matplotlib.pyplot as plt
df=pd.DataFrame({'log10(C)': Clist, 'distance': d})
ax = sns.scatterplot(x='log10(C)', y='distance',data=df)
```



In [6]:
```
print(d)
```

```
[0.9999999999909325, 0.9999857911256719, 1.0000119984722506, 0.9999920492162087, 1.
000000485322164]
```

```
Q15: The distance to the hyperplane always is 1.
```

In [1]:
```
import numpy as np
from sklearn import svm
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.model_selection import train_test_split
```

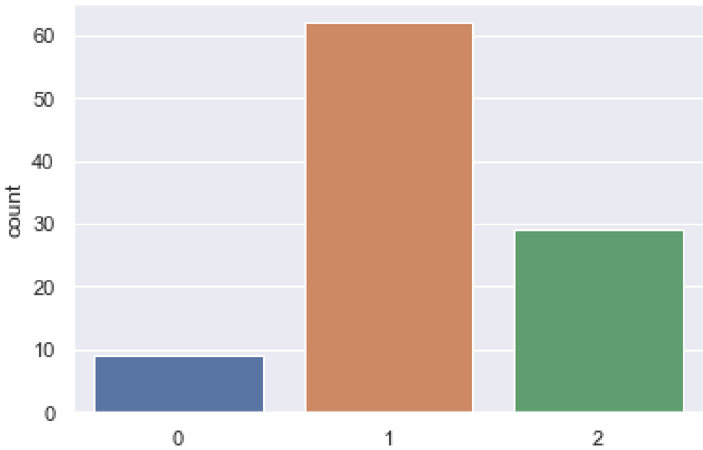
In [3]:
```
data_train = pd.read_csv("data_train.csv")
data_test = pd.read_csv("data_test.csv")

X = data_train[['intensity','symmetry']]
```

In [24]:
```
y0 = np.where(data_train["digit"] ==0, 1 ,-1)
Gamma = [-2,-1,0,1,2]
```

In [29]:
```
BestGamma = []
for i in range(100):
    X_train, X_test, y_train, y_test = train_test_split(X, y0, test_size=1000)
    best_gamma = -10
    Prec = 0
    for gamma in Gamma:
        result = svm.SVC(C = 0.1, kernel = "rbf", gamma = 10**gamma).fit(X_train,y_tı
        Eval = sum(result.predict(X_test) == y_test)
        if Eval > Prec:
            Prec = Eval
            best_gamma = gamma
    BestGamma = BestGamma + [best_gamma]
```

In [43]:
```
import seaborn as sns; sns.set()
sns.countplot(BestGamma)
```

Out[43]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1d4ceaa84a8>
```



Q16: only the gamma of 1,10,100 could be the candidate for best gamma. This may result from smaller gamma could eliminate the effect of higher polynomials.

17.

Consider the primal question

$$\min_{b,w} \frac{1}{2} w^T w$$

$$\text{s.t. } y_n(w^T \underbrace{z_n}_{\phi(x_n)} + b) \geq 1$$

~~Suppose~~ Let w_i is the weight for constant feature. (z_i)

suppose $w_i^* \neq 0$ is the optimal solution. b^*

we can consider another $\tilde{w}^*$ with $w_i = 0$

$\tilde{b} = b^* + \Sigma w_i z_i$

the $\frac{1}{2}\tilde{w}^T \tilde{w} < \frac{1}{2} w^{*T} w^*$

and every constraint $y_n(\tilde{w} z_n + \tilde{b}) \geq 0$ still holds

$\Rightarrow$ w^* is $w_i \neq 0$ is not an optimal

$\Rightarrow$ In optimal w, w_i for constant features must be 0.

18 Suppose $g_{SVM}(x) = \text{sign}\left(\sum_{SV_n} \alpha_n y_n K(x_n, x) + y_s - \sum_{SV_n} \alpha_n y_n K(x_n, x_s)\right)$

new $\tilde{g}_{SVM}(x) = \text{sign}\left(\sum_{SV_n} \alpha_n y_n K(x_n, x) + y_s - \sum \alpha_n y_n (\cancel{K(x_n, x_s + q)}) + \cancel{\sum_{SV_n} \alpha_n y_n} q\right)$

$= \text{sign}\left(\Sigma \alpha_n y_n K(x_n, x) + y_s - \sum_{SV_n} \alpha_n y_n K(x_n, x_s)\right)$

$= g_{SVM}(x) \Rightarrow$ equivalent!