HW3_212699581_211709597

June 5, 2022

[1]:
```
import matplotlib.pyplot as plt
import numpy as np
import numpy.linalg as lg
import numpy.random as rnd
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
```

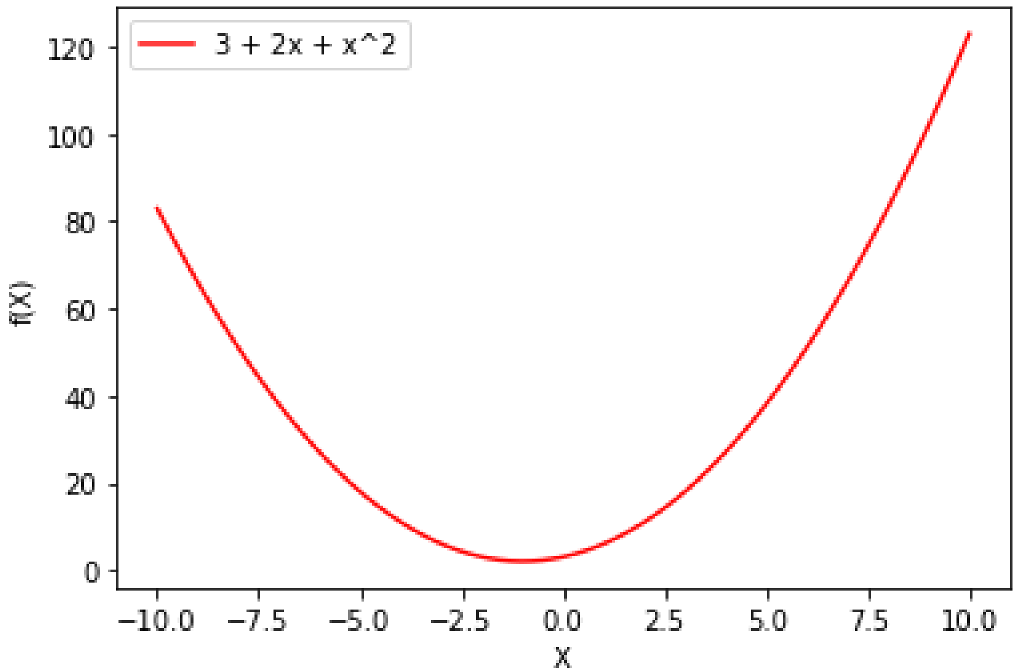
Question 1:

A.

[2]:
```
f = lambda x: 3 + 2 * x + x**2

lim = (-10, 10)
```

[3]:
```
X = np.linspace(lim[0], lim[1], 1000)
Y = f(X)
```

[4]:
```
plt.plot(X, Y, label='3 + 2x + x^2', color='red')
plt.xlabel('X')
plt.ylabel('f(X)')
plt.legend()
plt.show()
```



B.

```
[5]: grad_f = lambda x: 2 + 2 * x
```

C.

$ f=2+2x \ $ $ f=0 2+2x=0 x=-1 $

D.

```
[6]: def grad_update(grad, x, n=0.01, ep = 10**-10):
         xt_1 = x
         xt = xt_1 - n * grad(x)

         X = [xt_1, xt]

         while abs(xt - xt_1) >= ep:
             xt_1 = xt
             xt -= n * grad(xt_1)
             X.append(xt)
         return xt, np.array(X)
```

E.

```
[7]: print(grad_update(grad_f, 0, 0.01)[0])
```

```
-0.999999995188065
```

F.

[8]:
```
print(len(grad_update(grad_f, 0, 0.01)[1]))
print(len(grad_update(grad_f, 0, 0.1)[1]))
print(len(grad_update(grad_f, -0.5, 0.01)[1]))
print(len(grad_update(grad_f, -0.5, 0.1)[1]))

print(len(grad_update(grad_f, 5, 0.01, 10**-6)[1]))
print(len(grad_update(grad_f, 5, 0.1, 10**-6)[1]))
```

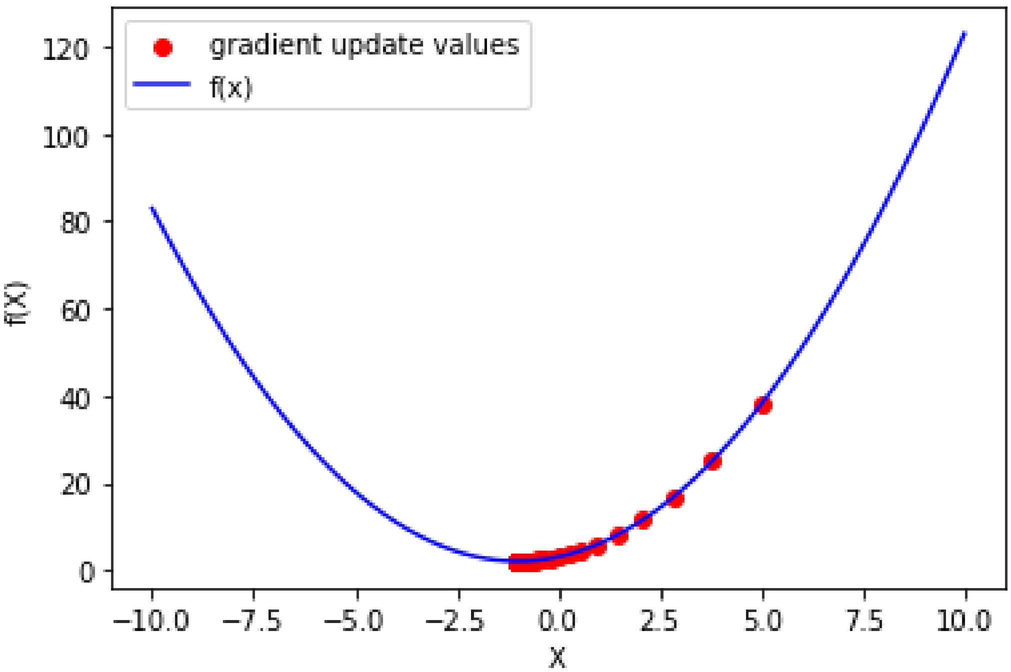
```
949
98
914
95
581
65
```

[9]:
```
print(grad_update(grad_f, 5, 0.1, 10**-6)[0])
```

```
-0.9999962337389587
```

G.

[10]:
```
XT = grad_update(grad_f, 5, 0.1, 10**-6)[1]
fXT = f(XT)

plt.scatter(XT, fXT, color='red', label='gradient update values')
plt.plot(X, Y, color='blue', label='f(x)')
plt.xlabel('X')
plt.ylabel('f(X)')
plt.legend()
plt.show()
```



Question 2:

A:

As we've seen in lectures $\lambda\|w\|^2$ is 2λ-strongly convex
And we've seen that $max\{0, 1 - y_i(< w, x_i > +b)\}$ is convex, and from a theorem from algebra if f and g is convex them f+g is convex as well and so $\frac{1}{m}\sum_{i=1}^{m} max\{0, 1 - y_i(< w, x_i > +b)\} + \lambda\|w\|^2$ is convex

B:

we need to proof that $|l(w_1, x_i, y_i) - l(w_2, x_i, y_i)| < R\|w_1 - w_2\|$

- $|l(w_1, x_i, y_i) - l(w_2, x_i, y_i)| =$
  $|max\{0, 1 - y_i < w_1, x_i >\} - max\{0, 1 - y_i < w_2, x_i >\}| =$
  $i.\ 1 - y_i < w_1, x_i > -\ 0 \leq 1 - y_i < w_1, x_i > - (1 - y_i < w_2, x_i >) = max\{0, 1 - y_i < w_1, x_i > - (1 - y_i < w_1, x_i >)\}$
  $ii.\ 1 - y_i < w_1, x_i > - (1 - y_i < w_1, x_i >) = max\{0, 1 - y_i < w_1, x_i > - (1 - y_i < w_1, x_i >)\}$
  $iii.\ 0 - (1 - y_i < w_2, x_i >) \leq 0 = max\{0, 1 - y_i < w_1, x_i > - (1 - y_i < w_1, x_i >)\}$
  $iv.\ 0 - 0 \leq max\{0, 1 - y_i < w_1, x_i > - (1 - y_i < w_1, x_i >)\}$

** $i. \max_k \|x_k\|\|w_2 - w_1\| \geq \|x_i\|\|w_2 - w_1\| \underset{\text{cauchy schwarz inequality}}{\geq}$
$< x_i, w_2 - w_1 > = < x_i, w_2 > - < x_i, w_1 > =$
$-1 + < x_i, w_2 > + 1 - < x_i, w_1 > =$
$a. 1 - y_i < x_i, w_1 > - (1 - y_i < x_i, w_2 >)$, for $y_i = 1$

b. for $y_i = -1$ start with $\|w_1 - w_2\|$ and we'll get the same inequality
$ii. \|x_i\| \|w_2 - w_1\| \geq 0$
$\Rightarrow \underset{k}{max} \|x_k\| \|w_2 - w_1\| \geq max\{0, 1 - y_i < w_1, x_i > - (1 - y_i < w_2, x_i >)\}$

combine * and ** and we get $|l(w_1, x_i, y_i) - l(w_2, x_i, y_i)| < R\|w_1 - w_2\|$

C:

Denote $\mathcal{L}(w, b) = max\{0, 1 - y_i(< w, x_i > +b)\} + \lambda\|w\|^2$

$$\frac{\partial \mathcal{L}}{\partial w} = \begin{cases} 2\lambda w, & \text{if } 1 - y_i(< w, x_i > +b) \leq 0 \\ -y_i x_i + 2\lambda w, & o.w. \end{cases}$$

$$\frac{\partial \mathcal{L}}{\partial b} = \begin{cases} 0, & \text{if } 1 - y_i(< w, x_i > +b) \leq 0 \\ -y_i, & o.w. \end{cases}$$

D:

[11]:
```python
def sub_grad(w, b, x, y, lam, d, m):
    return 2*lam*w if (1 - y * (w@x + b)) <= 0 else -y*x+2*lam*w, 0 if (1 - y *␣
 ↪(w@x + b)) <= 0 else -y


def pract_sgd(X, y, lam, epochs, l_rate):
    m = len(X)
    d = len(X[0])

    w = rnd.uniform(0, 1, d)
    b = rnd.uniform(0, 1)

    perm = np.arange(m)

    for i in range(epochs):
#         print(f'epoch {i}. {float(i)/epochs:.2f}%')
        rnd.shuffle(perm)
        for i in perm:
            subGrad = sub_grad(w, b, X[i], y[i], lam, d, m)
            w -= l_rate * subGrad[0]
            b -= l_rate * subGrad[1]

    return w, b



def theory_sgd(X, y, lam, epochs, l_rate):
    m = len(X)
    d = len(X[0])
```

```
    w = rnd.uniform(0, 1, d)
    b = rnd.uniform(0, 1, 1)

    W = []
    B = []

    W.append(w)
    B.append(b)

    for i in range(m*epochs):
#         print(f'epoch {i}. {float(i)/(m*epochs):.2f}%')
        index = rnd.randint(0, m)
        subGrad = sub_grad(w, b, X[index], y[index], lam, d, m)
        w -= l_rate * subGrad[0]
        b -= l_rate * subGrad[1]
        W.append(w)
        B.append(b)


    return sum(W) / (m * epochs), sum(b) / (m * epochs)




def svm_with_sgd(X, y, lam=0, epochs=1000, l_rate=0.01, sgd_type="practical"):
    rnd.seed(2)
    if sgd_type == "practical":
        return pract_sgd(X, y, lam, epochs, l_rate)
    return theory_sgd(X, y, lam, epochs, l_rate)
```

E:

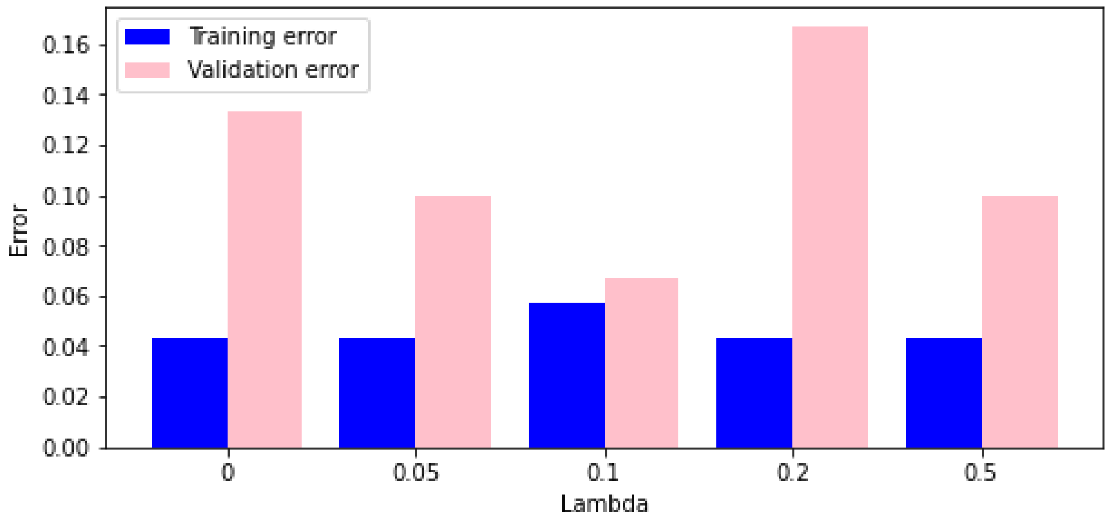
[12]:
```
sign = lambda x: 1 if x >= 0 else -1

def calculate_error(w, b, X, y):
    c = 0
    for x_i, y_i in zip(X, y):
        if sign(w@x_i + b) != y_i:
            c += 1
    return c / len(y)
```

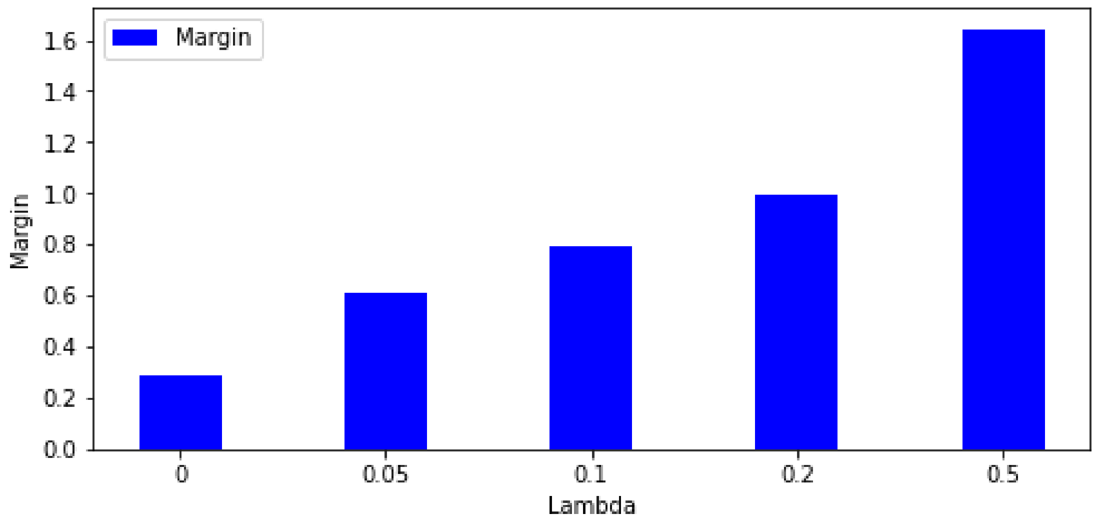
```
```

F:

[13]:
```
X, y = load_iris(return_X_y=True)
X = X[y != 0]
y = y[y != 0]
y[y == 2] = -1
X = X[:, 2:4]

X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.3,␣
 ↪random_state=0)

Lam = [0, 0.05, 0.1, 0.2, 0.5]

train_err = list()
val_err = list()
margin = list()

for l in Lam:
    w, b = svm_with_sgd(X_train, y_train, l)
    train_err.append(calculate_error(w, b, X_train, y_train))
    val_err.append(calculate_error(w, b, X_val, y_val))
    margin.append(1/lg.norm(w))
```

[14]:
```
fig, ax = plt.subplots(2, 1)

fig.set_size_inches(8, 8)

x = np.arange(len(Lam))

ax[0].bar(x-0.4/2, train_err, 0.4, color="blue", label="Training error")
ax[0].bar(x+0.4/2, val_err, 0.4, color="pink", label="Validation error")
ax[0].set_xticks(x)
ax[0].set_xticklabels(Lam)
ax[0].set_xlabel("Lambda")
ax[0].set_ylabel("Error")
ax[0].legend()

ax[1].bar(x, margin, 0.4, color="blue", label="Margin")
ax[1].set_xticks(x)
ax[1].set_xticklabels(Lam)
ax[1].set_xlabel("Lambda")
ax[1].set_ylabel("Margin")
ax[1].legend()

plt.show()
```





We see that with $\lambda = 0.1$ we get the best validation error

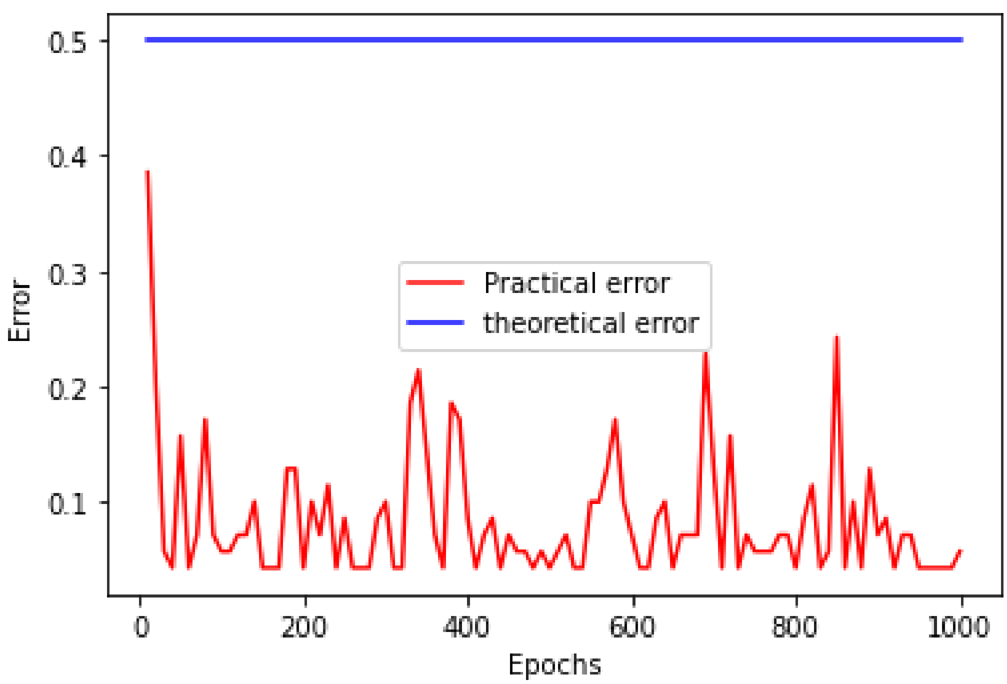
G:

[20]:
```
Epochs = np.arange(10, 1001, 10)
practical_train_error = list()
theoretical_train_error = list()

practical_validation_error = list()
theoretical_validation_error = list()

for epoch in Epochs:
    if epoch % 200 == 0:
        print(f'epoch {epoch}')
```

```
    w1, b1 = svm_with_sgd(X_train, y_train, lam=0.1, epochs=epoch,␣
 ↪sgd_type="practical")
    w2, b2 = svm_with_sgd(X_train, y_train, lam=0.1, epochs=epoch,␣
 ↪sgd_type="theory")
    practical_train_error.append(calculate_error(w1, b1, X_train, y_train))
    theoretical_train_error.append(calculate_error(w2, b2, X_train, y_train))

    practical_validation_error.append(calculate_error(w1, b1, X_val, y_val))
    theoretical_validation_error.append(calculate_error(w2, b2, X_val, y_val))
```

[18]:
```
plt.plot(Epochs, practical_train_error, color="red", label="Practical error")
plt.plot(Epochs, theoretical_train_error, color="blue", label="theoretical␣
 ↪error")
plt.xlabel("Epochs")
plt.ylabel("Error")
plt.legend()
plt.plot()
```

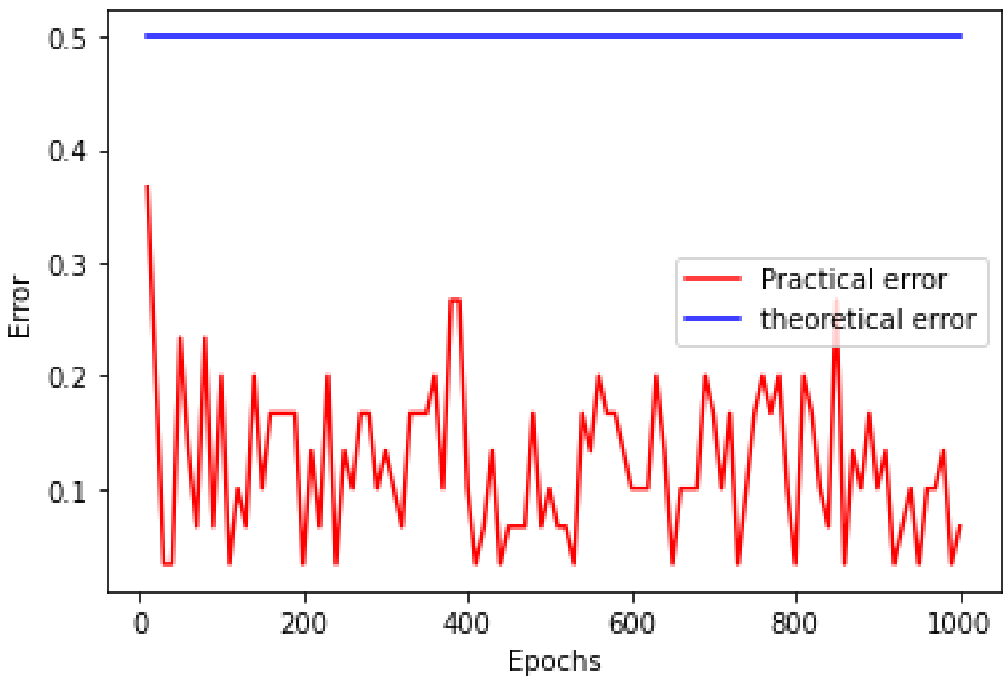
[18]: []



[19]:
```
plt.plot(Epochs, practical_validation_error, color="red", label="Practical␣
 ↪error")
```

```
plt.plot(Epochs, theoretical_validation_error, color="blue", label="theoretical␣
↪error")
plt.xlabel("Epochs")
plt.ylabel("Error")
plt.legend()
plt.plot()
```

[19]: []



שאלה 3:

א-

```python
#Q3_A:
import numpy as np
#folds=k
def cross_validation_error(X, y, model, folds):
 X_k=np.array_split(X, folds)
 Y_k=np.array_split(y, folds)
 train_error=[]
 val_error=[]
 for fold_1 in range(folds):
      X_val_K=X_k[fold_1]
      Y_val_K=Y_k[fold_1]
      X_train_K=np.array([element for fold_2 in range(folds) if fold_2!=fold_1 for element in X_k[fold_1]])
      Y_train_K=np.array([element for fold_2 in range(folds) if fold_2!=fold_1 for element in Y_k[fold_1]])
      model.fit(X_train_K,Y_train_K)
      train_error.append(model.score(X_train_K,Y_train_K))
      val_error.append(model.score(X_val_K,Y_val_K))
 return(1-(np.array(train_error).mean()),1-(np.array(val_error).mean()))
```

ב-

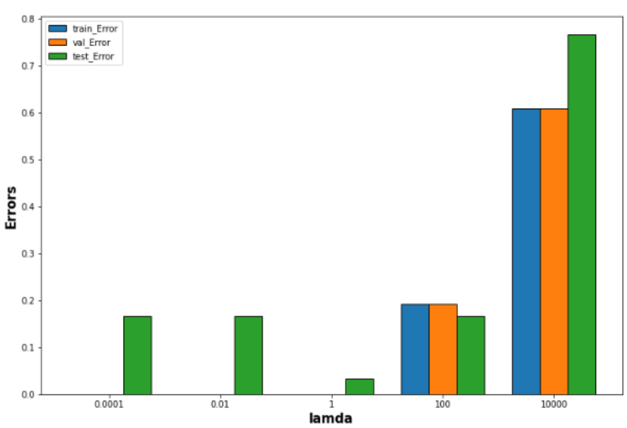
[2]

```python
#Q3_B:
from sklearn.svm import SVC
def svm_results(X_train, y_train, X_test, y_test):
 folds=5
 lamda=[0.0001, 0.01, 1, 100, 10000 ]
 output={}
 for lamda in lamda:
   model=SVC(kernel='linear',C=1/lamda)
   train_error,val_error=cross_validation_error(X_train, y_train, model, folds)
   model.fit(X_train,y_train)
   test_error= (model.predict(X_test)!=y_test).mean()
   output[f'svm_lambda_{lamda}']=(train_error,val_error,test_error)
 return output
```

ג-

```python
#Q3_C:
from sklearn.datasets import load_iris
iris_data = load_iris()
X, y = iris_data['data'], iris_data['target']

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=7)
output=svm_results(X_train, y_train, X_test, y_test)
```

```python
import numpy as np
import matplotlib.pyplot as plt

# set width of bar
barWidth = 0.25
fig = plt.subplots(figsize =(12,8))

# set height of bar
train=[]
val=[]
test=[]
for i in range(len(list(output.values()))):
    train.append(list(output.values())[i][0])
    test.append(list(output.values())[i][2])
    val.append(list(output.values())[i][1])

# Set position of bar on X axis
br1 = np.arange(len(train))
br2 = [x + barWidth for x in br1]
br3 = [x + barWidth for x in br2]

# Make the plot
plt.bar(br1, train, width = barWidth,
        edgecolor ='black', label ='train_Error')
plt.bar(br2, val, width = barWidth,
        edgecolor ='black', label ='val_Error')
plt.bar(br3, test,  width = barWidth,
        edgecolor ='black', label ='test_Error')


# Adding Xticks
plt.xlabel('lamda', fontweight ='bold', fontsize = 15)
plt.ylabel('Errors', fontweight ='bold', fontsize = 15)
plt.xticks([r + barWidth for r in range(len(train))],
        [0.0001, 0.01, 1, 100, 10000 ])

plt.legend()
plt.show()
```



הסבר :

המודל הטוב ביותר לפי שיטת CV על מדגם המבחן
הינו עבור $\lambda=1$ כי השגיאה (train error) הינה מינימלית.
בדרך כלל
אנו מחפשים המודל עם ה val error & test error
וזה מתקיים עבור $\lambda=1$ ולכן עבור ערך ה lamda
הזה מתקבל המודל הטוב ביותר לפי שיטת CV .

נשים לב שעבור ערכי λ קטנים ($\lambda<1$) יש נטייה
גבוהה ל overfitting כי בערכים האלה train error
גבוהה מאוד ביחס ל val error .
במקביל ערכי λ הגדולים הם גרועים ב train/test/val
כלומר יש נטייה ל underfitting .

ולכן בשיטה האחרונה המודל הטוב ביותר הינו
של $\lambda=1$.

שאלה 4:

$\forall i\in\{1,..,r\}$ $\quad g_i: R^d \to R$

כך שהפונקציה קמורה וגזירה בכל R^d.

$$g(w)=\max_{i\in[r]} g_i(w)$$

נוכיח עבור w מסוים כך ש

$$j\in \max_i g_i(w)$$

ש $\nabla g_j(w)$ תת-גרדיאנט של הפונקציה g בנקודה w.

פתרון:

לכל $u\in R^d$ מתקיים:

$$g(w) + \langle u-w, \nabla g_j(w)\rangle \underset{\substack{\text{מהגדרת} \\ g}}{=} g_j(w) + \langle u-w, \nabla g_j(w)\rangle$$

נתון ש g_j פונקציה קמורה וגזירה בכל R^d ובפרט בנקודה w ולכן לפי אי שוויון הגרדיאנט:

$$\forall u\in R^d: \quad g_j(u) \ge g_j(w) + \langle u-w, \nabla g_j(w)\rangle$$

ושוב לפי הגדרת הפונקציה g:

$$g_j(u) \le \max_{i\in[r]} g_i(u) = g(u) \quad \blacksquare$$