HTML

Homework 5

B07502166 魏子翔

1. For those correctly classified, $y_i(w^Tx_i+b)>0$, so $0\le\xi_i\le 1$. For those wrongly classified, $y_i(w^Tx_i+b)<0$, $\xi_i>1$.
   The number of misclassified examples $=\sum\limits_{\text{mis}}1+\sum\limits_{\text{correct}}0$, so it can be bounded by:
   $\sum\limits_{n=1}^{N}\xi_i^*$, because the misclassified ones $\xi_i^*>1$, the correct ones $\xi_i^*\ge 0$
   $\sum\limits_{n=1}^{N}\sqrt{\xi_i^*}$, because the misclassified ones $\sqrt{\xi_i^*}>1$, the correct ones $\sqrt{\xi_i^*}\ge 0$
   $\sum\limits_{n=1}^{N}\lfloor\xi_i^*\rfloor$, because the misclassified ones $\lfloor\xi_i^*\rfloor\ge 1$, the correct ones $\lfloor\xi_i^*\rfloor\ge 0$
   $\sum\limits_{n=1}^{N}\log_2(1+\xi_i^*)$, because the misclassified ones $\log_2(1+\xi_i^*)>1$, the correct ones $\log_2(1+\xi_i^*)\ge 0$

2. Follow the complementary slackness of soft-margin SVM, which gives $1-\xi_n-y_n(w^Tz_n+b)=0$ and $\xi_n\ge 0$ since all $\alpha_n=C$. If we take $y_n=-1$, then
   $1+(w^Tz_n+b)=\xi_n\ge 0 \;\Rightarrow\; b\ge -1-w^Tz_n=-1-\sum\limits_{m=1}^{N}y_m\alpha_m^*K(x_n,x_m)$
   So the smallest such $b^*=\max\limits_{n:\,y_n<0}\left(-1-\sum\limits_{m=1}^{N}y_m\alpha_m^*K(x_n,x_m)\right)$

3. Follow the Lagrange function taught in class but change the part $C\sum\limits_{n=1}^{N}\xi_n$ to $C\sum\limits_{n=1}^{N}\xi_n^2$, then $\frac{\partial\mathcal{L}}{\partial\xi_n}=0=2C\xi_n-\alpha_n,\ \xi_n=\frac{1}{2C}\alpha_n$

4. $\phi_{ds}(x)^T\phi_{ds}(x')=\sum\limits_{s,i,\theta}s\cdot\text{sign}(x_i-\theta)\cdot s\cdot\text{sign}(x_i'-\theta)=2\sum\limits_{i,\theta}\text{sign}(x_i-\theta)\cdot\text{sign}(x_i'-\theta)$
   $\text{sign}(x_i-\theta)\cdot\text{sign}(x_i'-\theta)=\begin{cases}-1 & \text{if } \theta\in(\min(x_i,x_i'),\max(x_i,x_i'))\\ 1 & \text{otherwise}\end{cases}$
   So for any d, the number of cases that $\text{sign}(x_i-\theta)\cdot\text{sign}(x_i'-\theta)=-1$ is $\sum\limits_{d}|x_i-x_i'|/2=||x-x'||_1/2$, and the number of cases that $\text{sign}(x_i-\theta)\cdot\text{sign}(x_i'-\theta)=1$ is $2d(2R-2L)/2-||x-x'||_1/2$.
   So the answer is $2d(R-L)-||x-x'||_1$.

5. Assume x is N-dimensional, then $E_{out}(G)=\frac{1}{N}\sum\limits^{N}wrong$. For each wrongly predicted one, there's at least $M+1$ g_ts are wrongly predicted, so $\sum\limits^{N}wrong\cdot(M+1)\le N\cdot\sum\limits_{t=1}^{2M+1}e_t$, where $N\cdot\sum\limits_{t=1}^{2M+1}e_t$ is the summation of those wrongly predicted ones for each g_t.
   So $E_{out}(G)=\frac{1}{N}\sum\limits^{N}wrong\le\frac{1}{M+1}\sum\limits_{t=1}^{2M+1}e_t$

6. Let $P(N)$ be the probability of getting no duplicated N examples in the 1127-sized data set, then $P(N)=\prod\limits_{i=0}^{N-1}\frac{1127-i}{1127}$. So $1-P(N)>0.75\;\Rightarrow\;P(N)<0.25$, then $N=56$ can be calculated in a while loop multiplying and checking whether $P(N)<0.25$.

7. Since the usual $E_{in} = \frac{1}{N}\sum_{n=1}^{N}(y_n - w^T x_n)^2$, so with $u_n \geq 0$, we can multiply $\sqrt{u_n}$ into the square function, which will make $\tilde{x}_n = \sqrt{u_n}x_n,\ \tilde{y}_n = \sqrt{u_n}y_n$

8. (a) split 1: $1 - 1^2 = 0$, split 2: $1 - 0.5^2 - 0.5^2 = 0.5$, total: $\frac{50}{100} \times 0 + \frac{50}{100} \times 0.5 = 0.25$
   (b) split 1: $1 - 0.8^2 - 0.2^2 = 0.32$, split 2: $1 - 0.75^2 - 0.25^2 = 0.375$, total: $\frac{70}{100} \times 0.32 + \frac{30}{100} \times 0.375 = 0.3365$
   (c) split 1: $1 - 0.7^2 - 0.3^2 = 0.42$, split 2: $1 - 1^2 = 0$, total: $\frac{90}{100} \times 0.42 + \frac{10}{100} \times 0 = 0.378$
   (d) split 1: $1 - 0.8^2 - 0.2^2 = 0.32$, split 2: $1 - 0.9^2 - 0.1^2 = 0.18$, total: $\frac{80}{100} \times 0.32 + \frac{20}{100} \times 0.18 = 0.292$
   (e) split 1: $1 - 0.9^2 - 0.1^2 = 0.18$, split 2: $1 - 0.9^2 - 0.1^2 = 0.18$, total: $\frac{80}{100} \times 0.18 + \frac{20}{100} \times 0.18 = 0.18$

9. The recursive function of U_{T+1} is $U_{T+1} = \sum_{n=1}^{N} u_n^{(T+1)} = \epsilon_T \sum_{n=1}^{N} u_n^{(T)} \sqrt{\frac{1-\epsilon_T}{\epsilon_T}} + (1-\epsilon_T) \sum_{n=1}^{N} u_n^{(T)} \sqrt{\frac{\epsilon_T}{1-\epsilon_T}}$
   $= 2\sqrt{\epsilon_T(1-\epsilon_T)} \sum_{n=1}^{N} u_n^{(T)} = 2\sqrt{\epsilon_T(1-\epsilon_T)}U_T$
   $\Rightarrow\ U_{T+1} = 2^T \prod_{t=1}^{T} \sqrt{\epsilon_t(1-\epsilon_t)}U_1 = 2^T \prod_{t=1}^{T} \sqrt{\epsilon_t(1-\epsilon_t)}$

10. Since we are minimizing $\frac{1}{N}\sum_{n=1}^{N}((y_n - s_n) - \eta g_t(x_n))^2$, so after taking the derivative of $\eta = 0$,
    $\eta = \frac{\sum_{n=1}^{N}(y_n - s_n)g_t(x_n)}{\sum_{n=1}^{N} g_t^2(x_n)}$. With $s_n^{(t)} = s_n^{(t-1)} + \alpha_t g_t(x_n)$,
    $\sum_{n=1}^{N}(s_n^{(t)} - y_n)g_t(x_n) = \sum_{n=1}^{N}(s_n^{(t-1)} + \alpha_t g_t(x_n) - y_n)g_t(x_n) = \sum_{n=1}^{N}(s_n^{(t-1)} - y_n)g_t(x_n) + \eta \sum_{n=1}^{N} g_t^2(x_n)$
    $= \sum_{n=1}^{N}(s_n^{(t-1)} - y_n)g_t(x_n) + \sum_{n=1}^{N}(y_n - s_n^{(t-1)})g_t(x_n) = 0$

11.~20.
```python
from libsvm.svmutil import *
import numpy as np

def P11(y, x):
    y_train, x_train = y.copy(), x.copy()
    for i in range(len(y_train)):
        if y_train[i] != 1:
            y_train[i] = -1
    m = svm_train(y_train, x_train, "-t 0 -q")
    sv_coef = np.array(m.get_sv_coef())
    sv_indices = np.array(m.get_sv_indices())

    w = np.zeros(len(x[0]))
    for idx, coef in zip(sv_indices, sv_coef):
        alpha = coef[0]
        for key in x[idx - 1]:
            w[key - 1] = w[key - 1] + (alpha * x[idx - 1][key])

    ans = 0
    for i in range(len(w)):
        ans += w[i] ** 2
    print(np.sqrt(ans))

def P1213(y, x):
    E_in = []
    sv_num = []
    for t in [2, 3, 4, 5, 6]:
        y_train, x_train = y.copy(), x.copy()
        for i in range(len(y_train)):
            if y_train[i] != t:
                y_train[i] = -t
        m = svm_train(y_train, x_train, "-t 1 -d 2 -g 1 -r 1 -q")
        _, acc, _ = svm_predict(y_train, x_train, m)
        E_in.append(1 - acc[0] / 100)
        sv_num.append(len(m.get_SV()))
    print(np.argmax(E_in) + 2)
    print(sv_num)

def P1415(y_train, x_train, y_test, x_test):
    E_out_14 = []
    E_out_15 = []
    y_train, x_train = y_train.copy(), x_train.copy()
    y_test, x_test = y_test.copy(), x_test.copy()
    for i in range(len(y_train)):
        if y_train[i] != 7:
```

```python
            y_train[i] = -7
    for i in range(len(y_test)):
        if y_test[i] != 7:
            y_test[i] = -7
    C = [0.01, 0.1, 1, 10, 100]
    for c in C:
        m = svm_train(y_train, x_train, f"-t 2 -g 1 -c {c} -q")
        _, acc, _ = svm_predict(y_test, x_test, m)
        E_out_14.append(1 - acc[0] / 100)
    print(E_out_14)
    G = [0.1, 1, 10, 100, 1000]
    for g in G:
        m = svm_train(y_train, x_train, f"-t 2 -g {g} -c 0.1 -q")
        _, acc, _ = svm_predict(y_test, x_test, m)
        E_out_15.append(1 - acc[0] / 100)
    print(E_out_15)

def P16(y_train, x_train):
    gamma = [0] * 5
    G = [0.1, 1, 10, 100, 1000]
    y_train, x_train = np.array(y_train.copy()), np.array(x_train.copy())
    for i in range(len(y_train)):
        if y_train[i] != 7:
            y_train[i] = -7
    for i in range(500):
        E_val = []
        rng = np.random.default_rng(i)
        idx = rng.choice(len(y_train), 200)
        y_valid, y_trainn = y_train[idx], np.delete(y_train, idx, axis=0)
        x_valid, x_trainn = x_train[idx], np.delete(x_train, idx, axis=0)
        for g in G:
            m = svm_train(y_trainn, x_trainn, f"-t 2 -g {g} -c 0.1 -q")
            _, acc, _ = svm_predict(y_valid, x_valid, m)
            E_val.append(1 - acc[0] / 100)
        gamma[np.argmin(E_val)] += 1
    print(gamma)

def P1720(y_train, x_train, y_test, x_test):
    y, x = [], []
    for i in range(len(y_train)):
        if y_train[i] == 11:
            y.append(1)
            x.append(list(x_train[i].values()))
        elif y_train[i] == 26:
            y.append(-1)
```

```
            x.append(list(x_train[i].values()))
    y_testt, x_testt = [], []
    for i in range(len(y_test)):
        if y_test[i] == 11:
            y_testt.append(1)
            x_testt.append(list(x_test[i].values()))
        elif y_test[i] == 26:
            y_testt.append(-1)
            x_testt.append(list(x_test[i].values()))
    y, x, y_testt, x_testt = (
        np.array(y),
        np.array(x),
        np.array(y_testt),
        np.array(x_testt),
    )
    y = np.reshape(y, (-1, 1))
    x = np.concatenate((x, y), axis=1)
    U = np.ones(x.shape[0]) / x.shape[0]
    theta = np.zeros((x.shape[1], x.shape[0]))
    for i in range(x.shape[1]):
        theta[i][0] = -1
        x_new = np.array(sorted(x, key=lambda x: x[i]))
        for j in range(1, x_new.shape[0]):
            theta[i][j] = (x_new[j][i] + x_new[j - 1][i]) / 2
    x, y = x[:, :-1], x[:, -1]
    E_in = []
    alpha = []
    g = []
    for _ in range(1000):
        s, idx, best_theta = ada_stump(y, x, theta, U)
        epsilon = U.dot((s * np.sign(x[:, idx] - best_theta)) != y) / np.sum(U)
        k = np.sqrt((1 - epsilon) / epsilon)
        err = 0
        for j in range(x.shape[0]):
            if s * np.sign(x[j][idx] - best_theta) != y[j]:
                U[j] *= k
                err += 1
            else:
                U[j] /= k
        E_in.append(err / x.shape[0])
        alpha.append(np.log(k))
        g.append([s, idx, best_theta])
    print(min(E_in))
    print(max(E_in))
    err = 0
```

```python
    for j in range(x.shape[0]):
        G = 0
        for i in range(1000):
            s, idx, best_theta = g[i]
            G += alpha[i] * (s * np.sign(x[j][idx] - best_theta))
        if np.sign(G) != y[j]:
            err += 1
    print(err / x.shape[0])
    err = 0
    for j in range(x_testt.shape[0]):
        G = 0
        for i in range(1000):
            s, idx, best_theta = g[i]
            G += alpha[i] * (s * np.sign(x_testt[j][idx] - best_theta))
        if np.sign(G) != y_testt[j]:
            err += 1
    print(err / x_testt.shape[0])

def ada_stump(y, x, theta, U):
    best_err = 1
    for i in range(x.shape[1]):
        for t in theta[i]:
            for s in [-1, 1]:
                err = np.sum(U.dot((s * np.sign(x[:, i] - t) != y))) / x.shape[0]
                if err < best_err:
                    best_err = err
                    best_s = s
                    best_idx = i
                    best_theta = t
    return best_s, best_idx, best_theta

if __name__ == "__main__":
    y_train, x_train = svm_read_problem("./letter.scale.tr")
    y_test, x_test = svm_read_problem("./letter.scale.t")
    P11(y_train, x_train)
    P1213(y_train, x_train)
    P1415(y_train, x_train, y_test, x_test)
    P16(y_train, x_train)
    P1720(y_train, x_train, y_test, x_test)
```