HW5 for Machine Learning

Hsiu Hsuan Yeh

1st June 2023

1 Multiple Choice

1.1 1. (d)

$\xi_n^* > 1$ means the sample is misclassified while $\xi_n^* = 0$ means the sample is classified correctly. Hence the number of misclassified examples will less than $\Sigma_{n=1}^N \xi_n^*$. Among all choices, only $\Sigma_{n=1}^N \frac{\xi_n^*}{2}$ is not the upper bound since $1 < \xi_n^* < 2$ will be underestimated.

1.2 2. (c)

Thrrough the inner-primal feasible condition, we have the equation: $\alpha_n(1 - \xi_n - y_n(w^T z_n + b)) = 0$. Then considering the constraint: $\xi_n \geq 0$, and for all bounded support vector, $\alpha_n = c$.

$$\Rightarrow 1 - y_n(w^T z + n + b) \geq 0$$
$$\begin{cases} y_n = 1, 1 - w^T z + n + b \geq 0, \Rightarrow b \leq w^T z_n + b \\ y_n = -1, 1 + w^T z + n + b \geq 0, \Rightarrow b \geq w^T z_n + b \end{cases}$$
$$\Rightarrow \min_n b_n = \min_{y_n=-1}(y_n - w^T z_n) = \min_{n:y_n=-1}(-1 - \Sigma_{m=1}^N y_m \alpha_m^* K(x_n, x_m))$$

1.3 3. (a)

$$L(w, b, \xi) = \frac{1}{2} w^T w + C\Sigma_{n=1}^N \xi_n^2 + \Sigma_{n=1}^N \alpha_n(1 - \xi_n - y_n(w^T \phi(x_n) + b))$$
$$\frac{\partial}{\partial \xi_n} = 2C\xi_n - \alpha_n = 0, \Rightarrow \xi_n^* = \frac{1}{2C}\alpha_n^*$$

1.4 4. (a)

The maximum of $K_{ds}(x, x')$ is 2d(R-L). It occurs when x = x' which means for all s, d and θ, g(x) == g(x'). Howerver, if g(x_i)g(x_i') = -1, 2d(R-1) should be minused by ($\frac{||x_i - x_i'||_1}{2} * 2 = ||x_i - x_i||_1$). $\frac{||x_i - x_i'||_1}{2}$ and 2 represent the number of θ and s which will let g(x_i)g(x_i') = -1 respectively.

1.5 5. (a)

let's assume the "true" function form is f(x)

$$
\begin{aligned}
E_{out}(G) &= \frac{1}{N}\Sigma_{i=1}^{N}[G(x_i) \neq f(x_i)] && (1)\\
&= \frac{1}{N}\Sigma_{i=1}^{N}[sign(\Sigma_{t=1}^{2M+1}g_t(x_i)) \neq f(x_i)] && (2)\\
&= \frac{1}{N}\Sigma_{i=1}^{N}[sign(\Sigma_{t=1}^{2M+1}f(x_i)g_t(x_i)) \leq 0] && (3)\\
&= \frac{1}{N}\Sigma_{i=1}^{N}[sign(\frac{1}{k}\Sigma_{t=1}^{2M+1}f(x_i)g_t(x_i)) \leq 0], \forall 0 < k \leq 2M+1 && (4)
\end{aligned}
$$

let's denote $m_i^t = f(x_i)g_t(x_i), F(m) = [m < 0]$

$$\Rightarrow E_{out}(G) = \frac{1}{N}\Sigma_{i=1}^{N}F(\frac{1}{k}\Sigma_{t=1}^{2M+1}m_i^t)$$

$$
\begin{cases}
|\{m_i^t : m_i^t = -1\}| > |\{m_i^t : m_i^t = 1\}|, \Rightarrow F(\frac{1}{k}\Sigma_{t=1}^{2M+1}m_i^t) = 1 \leq \frac{1}{m+1}\Sigma_{t=1}^{2M+1}F(m_i^t)\\
|\{m_i^t : m_i^t = -1\}| < |\{m_i^t : m_i^t = 1\}|, \Rightarrow F(\frac{1}{k}\Sigma_{t=1}^{2M+1}m_i^t) = 0 \leq \frac{1}{k}\Sigma_{t=1}^{2M+1}F(m_i^t), \forall 0 < k \leq 2M+1
\end{cases}
$$

$$\Rightarrow E_{out}(G) = \frac{1}{N}\Sigma_{i=1}^{N}F(\frac{1}{k}\Sigma_{t=1}^{2M+1}m_i^t) \leq \frac{1}{m+1}\Sigma_{t=1}^{2M+1}\frac{1}{N}\Sigma_{i=1}^{N}F(m_i^t) = \Sigma_{t=1}^{2M+1}\frac{1}{N}e_t$$

1.6 6. (b)

$$1 - \frac{C_{N'}^1 127 * N'}{1127^{N'}} \geq 0.75, \Rightarrow whenN = 56, \frac{C_{N'}^1 127 * N'}{1127^{N'}} \approx 0.24$$

1.7 7. (c)

$$
\begin{cases}
w = \frac{\Sigma_{i=1}^{N}u_n y_n x_n}{\Sigma_{i=1}^{N}x_n^T x_n}\\
\tilde(w) = \frac{\Sigma_{i=1}^{N}\tilde{y_n}\tilde{x_n}}{\Sigma_{i=1}^{N}\tilde{x_n}^T\tilde{x_n}}
\end{cases}
$$

$$\Rightarrow \tilde{x_n} = \sqrt{u_n}x_n, \tilde{y_n} = \sqrt{u_n}y_n$$

1.8 8. (c)

change of impurity(calculated by gini index)

- (a): 0.375 -> 0, 0.5, increase by 0.25
- (b): 0.3375 -> 0.32, 0.375, increase by 0.02
- (c): 0.4662 -> 0.42, , 0, decrease by 0.5124
- (d): 0.3848 -> 0.18, 0.18, decrease by 0.4096
- (e): 0.2952 -> 0.32, 0.18, decrease by 0.0904

1.9 9. (d)

$$\epsilon_T = \frac{\Sigma_{n=1}^N u_n^T [y_n \neq g(x_n)]}{\Sigma_{n=1}^N u_n^T}$$

$$\begin{aligned}
\Sigma_{n=1}^N u_n^{T+1} &= \sqrt{\frac{1-\epsilon_T}{\epsilon_T}} \Sigma_{n=1}^N u_n^T [y_n \neq g(x_n)] + \sqrt{\frac{\epsilon_T}{1-\epsilon_T}} \Sigma_{n=1}^N u_n^T [y_n = g(x_n)] && (5)\\
&= \sqrt{\frac{1-\epsilon_T}{\epsilon_T}} \epsilon_T \Sigma_{n=1}^N u_n^T + \sqrt{\frac{\epsilon_T}{1-\epsilon_T}} (1-\epsilon_T) Sigma_{n=1}^N u_n^T && (6)\\
&= 2\sqrt{\epsilon_T(1-\epsilon_T)} \Sigma_{n=1}^N u_n^T && (7)\\
&= 2^T \Pi_{t=1}^T \sqrt{\epsilon_t(1-\epsilon_t)} && (8)
\end{aligned}$$

1.10 10. (b)

$$\begin{aligned}
s_n &= s_n + \alpha_t g_t, \alpha_t = \frac{y_n - s_n}{g_t} && (9)\\
&= s_n + g(y_n - s_n) && (10)
\end{aligned}$$

2 Coding

```
features, labels = read_data("../train.txt")
test_features, test_labels = read_data("../test.txt")
```

2.1 11. (c)

```
new_labels =  [i == 1. ? 1. : -1. for i = labels]
model = train(new_labels, features; param="-t 0 -c 1 -q")
coef = get_coefficient(model)

@printf "||w|| = %.5f\n" norm(coef)
```

```
||w|| = 6.30872
```

2.2 12. (b)

```
param = "-t 1 -d 2 -g 1 -r 1 -q"

new_labels = [i == 2. ? 1. : -1. for i = labels]
model2 = train(new_labels, features; param=param)
@printf "Ein of model2: %5f\n" binary_error(model2; y=new_labels, x=features)

new_labels = [i == 3. ? 1. : -1. for i = labels]
model3 = train(new_labels, features; param=param)
@printf "Ein of model3: %.5f\n" binary_error(model3; y=new_labels, x=features)

new_labels = [i == 4. ? 1. : -1. for i = labels]
model4 = train(new_labels, features; param=param)
@printf "Ein of model4: %.5f\n" binary_error(model4; y=new_labels, x=features)

new_labels = [i == 5. ? 1. : -1. for i = labels]
model5 = train(new_labels, features; param=param)
@printf "Ein of model5: %.5f\n" binary_error(model5; y=new_labels, x=features)

new_labels = [i == 6. ? 1. : -1. for i = labels]
model6 = train(new_labels, features; param=param)
@printf "Ein of model6: %.5f\n" binary_error(model6; y=new_labels, x=features)
```

```
Ein of model2: 0.011048
Ein of model3: 0.00648
Ein of model4: 0.00914
Ein of model5: 0.01476
Ein of model6: 0.01105
```

2.3 13 (b)

```
@printf "number of SV of model2: %.5f\n" model2.get_nr_sv()
@printf "number of SV of model3: %.5f\n" model3.get_nr_sv()
@printf "number of SV of model4: %.5f\n" model4.get_nr_sv()
@printf "number of SV of model5: %.5f\n" model5.get_nr_sv()
@printf "number of SV of model6: %.5f\n" model6.get_nr_sv()
```

```
number of SV of model2: 587.00000
number of SV of model3: 373.00000
number of SV of model4: 481.00000
number of SV of model5: 640.00000
number of SV of model6: 499.00000
```

2.4 14. (d)

```
new_labels = [i == 7. ? 1. : -1. for i = test_labels]

model = train(
    [i == 7. ? 1. : -1. for i = labels],
    features;
    param="-t 2 -g 1 -c 0.01 -q"
)
@printf "Eout when c=0.01: %.5f:\n" binary_error(model; y=new_labels, x=test_features)

model = train(
    [i == 7. ? 1. : -1. for i = labels],
    features;
    param="-t 2 -g 1 -c 0.1 -q"
)
@printf "Eout when c=0.1: %.5f:\n" binary_error(model; y=new_labels, x=test_features)

model = train(
    [i == 7. ? 1. : -1. for i = labels],
    features;
    param="-t 2 -g 1 -c 1 -q"
)
@printf "Eout when c=1: %.5f:\n" binary_error(model; y=new_labels, x=test_features)

model = train(
    [i == 7. ? 1. : -1. for i = labels],
    features;
    param="-t 2 -g 1 -c 10 -q"
)
@printf "Eout when c=10: %.5f:\n" binary_error(model; y=new_labels, x=test_features)

model = train(
    [i == 7. ? 1. : -1. for i = labels],
    features;
    param="-t 2 -g 1 -c 100 -q"
)
@printf "Eout when c=100: %.5f:\n" binary_error(model; y=new_labels, x=test_features)
```

```
Eout when c=0.01: 0.04520:
Eout when c=0.1: 0.04520:
Eout when c=1: 0.01420:
Eout when c=10: 0.00400:
Eout when c=100: 0.00540:
```

2.5 15. (c)

```
model = train(
    [i == 7. ? 1. : -1. for i = labels],
    features;
    param="-t 2 -g 0.1 -c 0.1 -q"
)
@printf "Eout when g=0.1: %.5f:\n" binary_error(model; y=new_labels, x=test_features)

model = train(
    [i == 7. ? 1. : -1. for i = labels],
    features;
    param="-t 2 -g 1 -c 0.1 -q"
)
@printf "Eout when g=1: %.5f:\n" binary_error(model; y=new_labels, x=test_features)

model = train(
    [i == 7. ? 1. : -1. for i = labels],
    features;
    param="-t 2 -g 10 -c 0.1 -q"
)
@printf "Eout when g=10: %.5f:\n" binary_error(model; y=new_labels, x=test_features)

model = train(
    [i == 7. ? 1. : -1. for i = labels],
    features;
    param="-t 2 -g 100 -c 0.1 -q"
)
@printf "Eout when g=100: %.5f:\n" binary_error(model; y=new_labels, x=test_features)

model = train(
    [i == 7. ? 1. : -1. for i = labels],
    features;
    param="-t 2 -g 1000 -c 0.1 -q"
)
@printf "Eout when g=1000: %.5f:\n" binary_error(model; y=new_labels, x=test_features)
```

```
Eout when g=0.1: 0.04520:
Eout when g=1: 0.04520:
Eout when g=10: 0.04020:
Eout when g=100: 0.04520:
Eout when g=1000: 0.04520:
```

2.6 16. (a)

```
new_labels = [i == 7. ? 1 : -1 for i = labels]
gamma = [0.1, 1, 10, 100, 1000]
params = [
    "-t 2 -g $g -c 0.1 -q"
    for g in gamma
]
score = zeros(length(params))

p = Progress(500, desc="Trials: ", color=:white, barlen=30)
for _ = 1:500
    res = pick_param_val(new_labels, features, param=params)
    score[res.idx] += 1
    next!(p)
end
print(score)
```

```
[327.0, 0.0, 173.0, 0.0, 0.0]
```

2.7 17. (a)

```
idx = findall(x->(x == 11 || x == 26), labels)
new_features = features[idx, :]
new_labels = [i == 11 ? 1. : -1. for i = labels[idx]]

idx = findall(x->(x == 11 || x == 26), test_labels)
new_test_features = test_features[idx, :]
new_test_labels = [i == 11 ? 1. : -1. for i = test_labels[idx]]

pred, min_loss, max_loss, param = ada_boosting(new_features, new_labels)
@printf "min Ein_g: %.5f" min_loss
```

```
min Ein_g: 0.09847
```

2.8 18. (c)

```
@printf "max Ein_g: %.5f" max_loss
```

```
max Ein_g: 0.57161
```

2.9 19 (a)

```
@printf "Ein_G: %.5f" mean(pred .!= new_labels)
```

```
Ein_G: 0.00000
```

2.10 20 (a)

```
@printf "Eout_G: %.5f" mean(predict(param, new_test_features) .!= new_test_labels)
```

```
Eout_G: 0.00279
```

3 Code Reference

```
using Printf, LinearAlgebra

import DelimitedFiles: readdlm
import FLoops: @floop
import PyCall: pyimport
import InvertedIndices: Not
import ProgressMeter: Progress, next!
using Distributions



function read_data(path)
    data = readdlm(path, ' ')[:, begin:end-1]
    for j = axes(data, 2)
        for i in axes(data, 1)
            elem = data[i, j]
            try
                start = collect(findfirst(":", elem))[1]+1
                data[i, j] = parse(Float64, elem[start:end])
            catch
            end

        end
    end

    features = Matrix{Float64}(hcat(ones(size(data, 1)), data[:, begin+1:end]))
    label = Vector{Float64}(data[:, begin])

    return features, label
end


global libsvm = pyimport("libsvm.svmutil")


function train(y, x; param)
    param = libsvm.svm_parameter(param)
    prob = libsvm.svm_problem(y, x)
    model = libsvm.svm_train(prob, param)

    return model
end


function get_coefficient(model)
    sv_dict = model.get_SV()
    sv = Matrix{Float64}(undef, length(sv_dict), length(sv_dict[1]))
    for i = axes(sv, 1), j = axes(sv, 2)
        sv[i, j] = sv_dict[i][j]
    end
    sv_coef = [i[1] for i in model.get_sv_coef()]

    coef = sv' * sv_coef

    return coef
end
```

```julia
function binary_error(model; y, x)
    _, p_acc, _ = libsvm.svm_predict(y, x, model, "-q")
    err = 1 - (p_acc[1]/100)

    return err
end


struct ValResult{T}
    idx::Int64
    errs::Vector{Float64}
    models::T
end

function pick_param(param, train_y, train_x, eval_y, eval_x)
    models = [
        train(train_y, train_x; param=i)
        for i in param
    ]
    err  = binary_error.(models; y=eval_y, x=eval_x)
    idx = argmin(err)
    @inbounds model = models[idx]

    return ValResult(idx, err, models)
end
function pick_param_val(y, x; param)
    N = size(x, 1)
    pos = sample(1:N, 2000, replace=false)
    @inbounds train_x, eval_x = x[Not(pos), :], x[pos, :]
    @inbounds train_y, eval_y = y[Not(pos)], y[pos]
    res = pick_param(param, train_y, train_x, eval_y, eval_x)

    return res, θ
end


check_sign(a) = a == 0. ? -1. : sign(a)
predict(s::Real, x::Real, θ::Real) = s * check_sign(x - θ)
predict(s::Real, x::AbstractVector, θ::Real) = predict.(Ref(s), x, Ref(θ))
function decision_stump(features, labels)
    n, k = size(features)
    temp = vcat(  # (n+1) * k
        reshape(fill(-Inf, k), 1, k),
        reduce(hcat, sort.(eachcol(features)))
    )
    @inbounds θ = (temp[begin+1:end, :] + temp[begin:end-1, :]) / 2

    pred = Array{Float64}(undef, n, n*k, 2)
    for (κ, s) = enumerate([-1, 1])
        @floop for (j, θ_j) = enumerate(θ)
            d = ceil(Int64, j/n)
            pred[:, j, κ] = predict(s, features[:, d], θ_j)
        end
    end
    incorrect_ptr = pred .!= labels

    return incorrect_ptr, pred, θ
end
```

```julia
function ada_boosting(features, labels; T=1000)
    incorrect_ptr, pred, θ = decision_stump(features, labels)

    # model(decision stump) parameters
    s = Vector{Float64}(undef, T)
    theta = Vector{Float64}(undef, T)
    d = Vector{Int64}(undef, T)

    # aggegation weights for each sub-model(decision stump)
    alpha = Vector{Float64}(undef, T)

    n = size(pred, 1)
    loss = Vector{Float64}(undef, T)
    best_pred = Matrix{Float64}(undef, n, T)  # best prediction for each iterations
    u_t = fill(1/n, n)  # scaling factor for each iterations

    p = Progress(T, desc="Boosting: ", color=:white, barlen=30)
    @inbounds for t = eachindex(s, d, theta, alpha)
        loss_t = dropdims(mean(incorrect_ptr, dims=1); dims=1)
        weighted_loss_t = dropdims(mean(u_t  .* incorrect_ptr, dims=1); dims=1)

        idx = argmin(weighted_loss_t)
        loss[t] = loss_t[idx]
        best_pred[:, t] = pred[:, idx]

        d[t] = ceil(Int64, idx[1]/n)
        theta[t] = θ[idx[1]]
        s[t] = [-1, 1][idx[2]]

        opt_correct_ptr = best_pred[:, t] .== labels
        opt_incorrect_ptr = best_pred[:, t] .!= labels

        ϵ_t = (opt_incorrect_ptr' * u_t) / sum(u_t)
        diamond_t = sqrt((1-ϵ_t)/ϵ_t)
        alpha[t] = log(diamond_t)

        u_t = (opt_incorrect_ptr .* u_t) * diamond_t +
             (opt_correct_ptr .* u_t) / diamond_t
        next!(p)
    end
    param = (s=s, d=d, theta=theta, alpha=alpha)

    return check_sign.(best_pred*alpha), minimum(loss),  maximum(loss), param
end
```

```julia
function predict(param::NamedTuple, features)
    weighted_pred = zeros(size(features, 1))
    for (s, d, theta, alpha) in zip(values(param)...)
        weighted_pred += alpha * predict(s, features[:, d], theta)
    end

    return check_sign.(weighted_pred)
end
```