Homework7 solution

November 8, 2021

0.0.1 10.16

(a) $\mu = (1/n)A^T\mathbf{1}$.

This follows from $\mu_i = \left(\mathbf{1}^T a_i\right)/n = \left(a_i^T \mathbf{1}\right)/n$

(b) $\tilde{A} = A - 1\mu^T$.

The i th column of $\tilde{A}$ is $\tilde{a}_i = a_i - \mu_i \mathbf{1}$, so

$$\begin{aligned}
\tilde{A} &= \left[\begin{array}{lll} a_1 & \cdots & a_k \end{array}\right] - \left[\begin{array}{lll} \mu_1 \mathbf{1} & \cdots & \mu_k \mathbf{1} \end{array}\right] \\
&= \left[\begin{array}{lll} a_1 & \cdots & a_k \end{array}\right] - \mathbf{1}\left[\begin{array}{lll} \mu_1 & \cdots & \mu_k \end{array}\right] \\
&= A - \mathbf{1}\mu^T
\end{aligned}$$

(c) The diagonal entries of $\sum_{ii}$ are

$$\Sigma_{ii} = \frac{1}{N}\left(\tilde{A}^T \tilde{A}\right)_{ii} = \frac{1}{N}\tilde{a}_i \tilde{a}_i^T = \frac{1}{N}\left\|\tilde{a}_i\right\|^2 = \mathbf{std}\left(a_i\right)^2$$

since the standard deviation is defined as $\mathbf{std}\left(a_i\right) = \left\|\tilde{a}_i\right\|/\sqrt{N}$.

The off-diagonal entry Σ_{ij} is zero if $\tilde{a}_i = 0$ or $\tilde{a}_j = 0$. Otherwise,

$$\Sigma_{ij} = \frac{1}{N}\left(\tilde{A}^T \tilde{A}\right)_{ij} = \frac{1}{N}\tilde{a}_i \tilde{a}_j^T = \rho_{ij}\mathbf{std}\left(a_i\right)\mathbf{std}\left(a_j\right)$$

since the correlation coefficient is defined as

$$\rho_{ij} = \frac{1}{N}\frac{\tilde{a}_i^T \tilde{a}_j}{\mathbf{std}\left(a_i\right)\mathbf{std}\left(a_j\right)}$$

(d) $Z = \left(A - 1\mu^T\right)\mathbf{diag}\left(1/\mathbf{std}\left(a_1\right), \ldots, 1/\mathbf{std}\left(a_k\right)\right)$. The standardized vectors are $z_i = \tilde{a}_i/\mathbf{std}\left(a_i\right)$. Therefore

$$\begin{aligned}
Z &= \tilde{A}\mathbf{diag}\left(1/\mathbf{std}\left(a_1\right), \ldots, 1/\mathbf{std}\left(a_k\right)\right) \\
&= \left(A - \mathbf{1}\mu^T\right)\mathbf{diag}\left(1/\mathbf{std}\left(a_1\right), \ldots, 1/\mathbf{std}\left(a_k\right)\right).
\end{aligned}$$

0.0.2 14.3

[1]:
```julia
using LinearAlgebra
include("iris_flower_data.jl");
```

```julia
include("iris_multiclass_helpers.jl");

function least_sqaure_classifier(X, y, k)
    sz = size(X)[2]
    one = ones(sz)
    A = [X' one]
    newy = 2(y .== k) .- 1
    theta = pinv(A)*newy

    return theta, error_rate(X, y, k, theta, sz)
end

function error_rate(X, y, k, theta, sz)
    one = ones(sz)
    A = [X' one]
    y_hat = sign.(A*theta)

    newy = 2(y .== k) .- 1
    correct = sum(y_hat.==newy)
    return 1-correct/sz
end

X_train = X[:,1:100];
X_test = X[:,101:150];
y_train = y[1:100];
y_test = y[101:150];
```

(a)

[2]:
```julia
theta1, er_train1 = least_sqaure_classifier(X_train, y_train, 1);
theta2, er_train2 = least_sqaure_classifier(X_train, y_train, 2);
theta3, er_train3 = least_sqaure_classifier(X_train, y_train, 3);
er_test1 = error_rate(X_test, y_test, 1, theta1, 50);
er_test2 = error_rate(X_test, y_test, 2, theta2, 50);
er_test3 = error_rate(X_test, y_test, 3, theta3, 50);
println("Class\t\tError rate on train\tError rate on␣
↪test\nsetosa\t\t",er_train1,"\t\t\t", er_test1,␣
↪"\nversicolor\t",er_train2,"\t\t\t", er_test2,␣
↪"\nvirginica\t",er_train3,"\t", er_test3);
```

```
Class           Error rate on train      Error rate on test
setosa          0.0                      0.0
versicolor      0.28                     0.24
virginica       0.09999999999999998      0.020000000000000018
```

(b)

[3]:
```julia
one_train = ones(size(X_train)[2]);
one_test = ones(size(X_test)[2]);
```

```
A_train = [X_train' one_train];
A_test = [X_test' one_test];
theta = [theta1 theta2 theta3];
yy_train = argmax_by_row(A_train*theta);
yy_test = argmax_by_row(A_test*theta);
C_train = confusion_matrix(yy_train, y_train);
C_test = confusion_matrix(yy_test, y_test);

println("Comfusion matrix on train set")
C_train
```

```
Comfusion matrix on train set
```

[3]:
```
3×3 Matrix{Float64}:
 29.0   0.0   0.0
  0.0  24.0   5.0
  0.0  11.0  31.0
```

[4]:
```
println("Comfusion matrix on test set")
C_test
```

```
Comfusion matrix on test set
```

[4]:
```
3×3 Matrix{Float64}:
 20.0  0.0   0.0
  1.0  9.0   0.0
  0.0  6.0  14.0
```

0.0.3 15.2

[1]:
```
using LinearAlgebra
using Plots
include("lsq_classifier_data.jl");
```

(a)

[2]:
```
function error_rate(X, y, theta)
    sz = size(X)[2]
    one = ones(sz)
    A = [X' one]
    y_hat = sign.(A*theta)
    correct = sum(y_hat.==y)
    return 1-correct/sz
end

sz = size(X)[2];
one = ones(sz);
A = [X' one];
theta = pinv(A)*y;
```

```
er_train = error_rate(X, y, theta);
er_test = error_rate(X_test, y_test, theta);

println("Classification error on training: ", er_train, "\nClassification error␣
 ↪on training: ", er_test)
```

```
Classification error on training: 0.19999999999999996
Classification error on training: 0.28
```

(b)

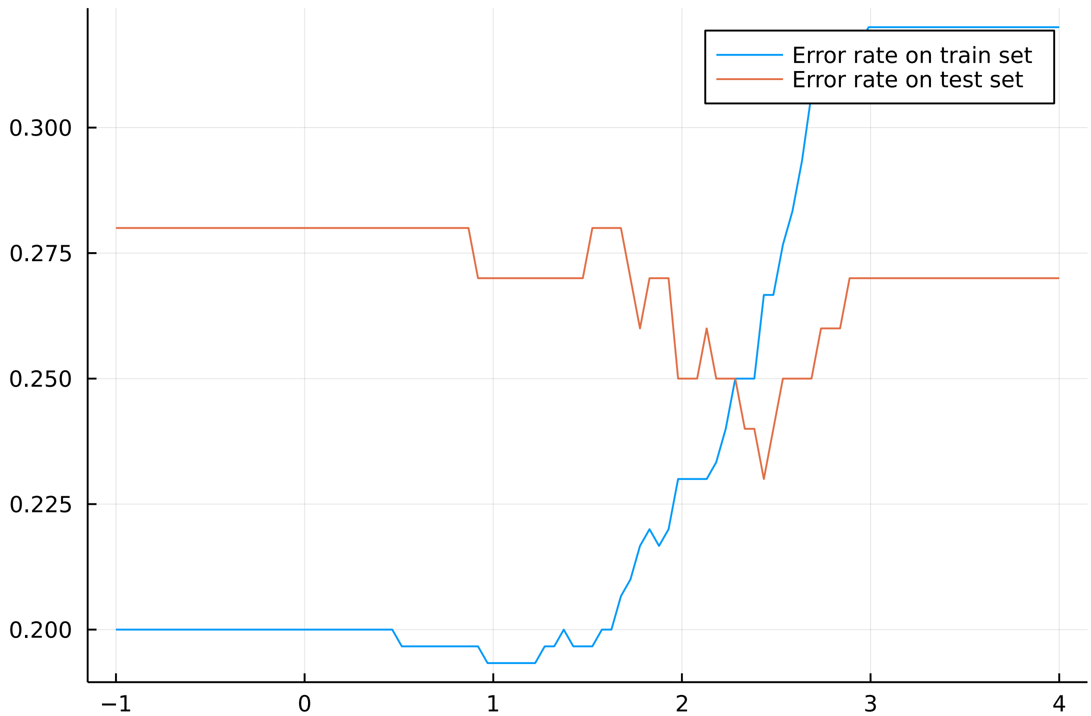
[3]:
```
function regu(X, y, lambda)
    A = [X' ones(size(X)[2])]
    I = Array(Diagonal(ones(size(X)[1])))
    newX = [A;sqrt(lambda)*I zeros(size(X)[1])]
    newy = [y;zeros(size(X)[1])]
    theta = newX\newy
    return theta
end

lambda = [10^i for i in range(-1,stop=4,length=100)];
xaxis = log10.(lambda);
error_train = zeros(100);
error_test = zeros(100);
for i in 1:100
    theta = regu(X, y, lambda[i]);
    error_train[i] = error_rate(X, y, theta);
    error_test[i] = error_rate(X_test, y_test, theta);
end
index = findmin(error_test)[2];
println("Reasonable lambda = ", lambda[index]);

plot(xaxis,error_train, label ="Error rate on train set")
plot!(xaxis,error_test, label ="Error rate on test set")
```

```
Reasonable lambda = 271.85882427329403
```

[3]:



0.0.4 15.3

[1]:
```
using LinearAlgebra
include("price_elasticity.jl")
delta_p = (Prices .- p_nom) ./ p_nom;
delta_d = (Demands .- d_nom) ./ d_nom;
delta_p_train = delta_p[:,1:50];
delta_d_train = delta_d[:,1:50];
delta_p_test = delta_p[:,51:N];
delta_d_test = delta_d[:,51:N];

# Regression without regularization
E_hat = delta_d_train*pinv(delta_p_train)
```

[1]:
```
5×5 Matrix{Float64}:
 -0.962267    -0.0833953  -0.520435   -0.648362  -0.387259
  0.107803    -0.30387    -0.0194254  -0.454458  -0.69952
 -0.245873     0.0569617  -0.546831   -0.11042   -0.140529
 -0.0147948   -0.298046    0.298149   -1.37377    0.794056
  0.360962    -0.912985   -1.39774     0.68058   -1.1817
```

Using regularization will avoid over-fitting problem.

[2]:
```
# Regression with regularization
E_hat_regu = inv(delta_p_train*delta_p_train' + 0.
↪001*Array(Diagonal(ones(size(delta_p_train)[1]))))delta_p_train*delta_d_train'
```

[2]:
```
5×5 Matrix{Float64}:
 -0.957856    0.106472   -0.244633   -0.0163845   0.358508
 -0.0826826  -0.300272    0.0564209  -0.29443    -0.907341
 -0.517686   -0.0195579  -0.543665    0.295951   -1.38978
 -0.645188   -0.451344   -0.110264   -1.36318     0.676573
 -0.385549   -0.695741   -0.139906    0.788808   -1.17367
```

[3]:
```
test = norm(E_hat*delta_p_test-delta_d_test)/sqrt(n*25)
test_regu = norm(E_hat_regu*delta_p_test-delta_d_test)/sqrt(n*25)
println("RMS without regularization: ", test, "\nRMS using regularization: ",␣
↪test_regu)
```

```
RMS without regularization: 0.22099654989476042
RMS using regularization: 0.22175995430670603
```