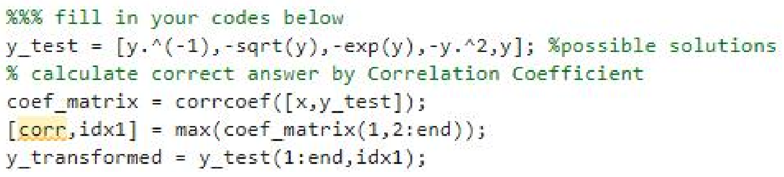
Fundamentals of Machine Learning Coursework

1. Regression

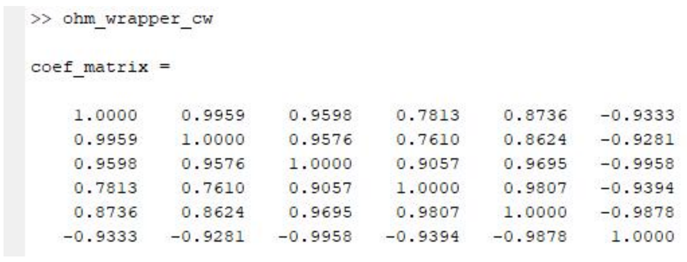
1.(a)

Based on observation and knowledge, the possible distribution function could be negative logarithm, exponent, negative square, reciprocal and square root calculations. The y value is transformed into functions as corresponding vector, which forms a 5*5 matrix. For accurate match, it is appropriate to use correlation coefficient between transformed output and original input. The code, default Pearson coefficient, and its output is placed in Fig. 1.1 and 1.2, respectively. As the maximum of correlation coefficient (0.9959) refers to y^(-1), outputs are transformed into reciprocal value.



```
%%% fill in your codes below
y_test = [y.^(-1),-sqrt(y),-exp(y),-y.^2,y]; %possible solutions
% calculate correct answer by Correlation Coefficient
coef_matrix = corrcoef([x,y_test]);
[corr,idx1] = max(coef_matrix(1,2:end));
y_transformed = y_test(1:end,idx1);
```

Fig. 1.1 Matlab codes for correlation coefficient and transformation



```
>> ohm_wrapper_cw

coef_matrix =

    1.0000    0.9959    0.9598    0.7813    0.8736   -0.9333
    0.9959    1.0000    0.9576    0.7610    0.8624   -0.9281
    0.9598    0.9576    1.0000    0.9057    0.9695   -0.9958
    0.7813    0.7610    0.9057    1.0000    0.9807   -0.9394
    0.8736    0.8624    0.9695    0.9807    1.0000   -0.9878
   -0.9333   -0.9281   -0.9958   -0.9394   -0.9878    1.0000
```

Fig. 1.2 Output of correlation coefficient

To make further clarification, relation between current, resistance, and length of wire could be confirmed by Ohm's law and law of resistance. Corresponding formula is placed in Eq. 1.1 and 1.2, so that Eq. 1.3 could be concluded, and it verifies the linear relation between length of wire and reciprocal of current, if voltage is assumed as constant.

$$I = \frac{U}{R} \qquad \text{Eq. 1.1}$$

$$R = \frac{\rho L}{S} \qquad \text{Eq. 1.2}$$

$$I = \frac{US}{\rho L} \qquad \text{Eq. 1.3}$$

1.(b)

The original formula of $\widetilde{w}$ for Least Squares cost function is placed in Eq. 1.4 and notation is in

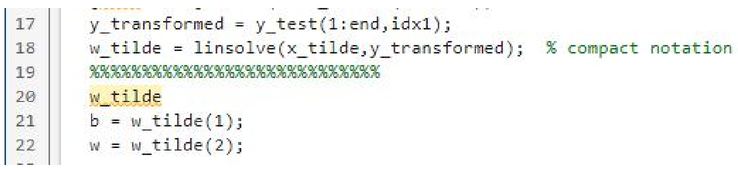
Eq. 1.5. In this case, as the linear relation has been proved and the input is one dimensional, the calculation could be simplified as linear solution of transformed output and original input, as shown in Eq. 1.6.

$$\widetilde{w}^* = (\sum_{p=1}^{P} \tilde{x}_p * \tilde{x}_p^T)^{-1} * \sum_{p=1}^{P} \tilde{x}_p * y_p \qquad \text{Eq. 1.4}$$

$$\widetilde{w}^* = \begin{bmatrix} b^* \\ w^* \end{bmatrix} \qquad \tilde{x}_p = \begin{bmatrix} 1 \\ x_p \end{bmatrix} \qquad \text{Eq. 1.5}$$

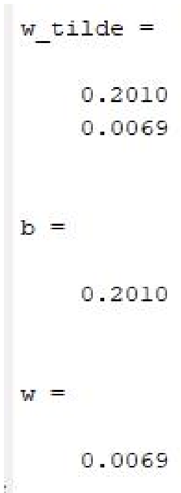
$$X_p \widetilde{w}^* = Y_p \qquad \text{Eq. 1.6}$$

The code is placed in Fig. 1.3 and solution is placed in Fig. 1.4.



```
y_transformed = y_test(1:end,idx1);
w_tilde = linsolve(x_tilde,y_transformed);  % compact notation
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
w_tilde
b = w_tilde(1);
w = w_tilde(2);
```

Fig. 1.3 Matlab codes for linear solution



```
w_tilde =

    0.2010
    0.0069


b =

    0.2010


w =

    0.0069
```

Fig. 1.4 Output of solution

1.(c)

The original and transformed spaces with inputs, corresponding output, and fitting models is illustrated by Fig. 1.5 and Fig. 1.6. In transformed space, bias of fitting model is expected close to 0. The reason is that, according to reciprocal relationship, when input approaches to 0, the output approaches to infinite and transformed output will approach 0. This model fits point well with sensible weights, as the linear relation proved before.

However, in this case, the bias is 0.201, which is not small enough. It may be caused by protection methods in acquisition of original data. For instance, the entire system installs two resistances, only one adjustable, to confirm infinite current will not burn circuit.

The similar situation happens in original space. When input is close to infinite, output present a trend to approach 0. The data proves protection existed in acquisition system. When length=0, current=4.7 instead of infinite. Even such method will cause a few deviations, fitting model could still present the relation.

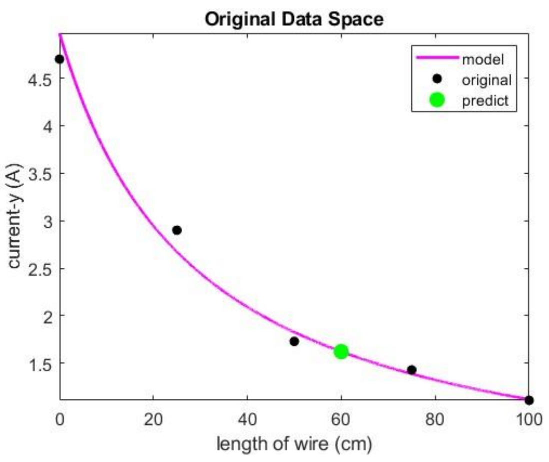


Fig. 1.5 The original data space

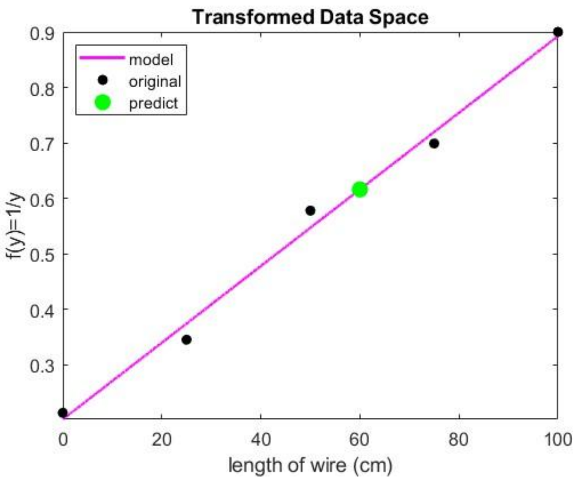
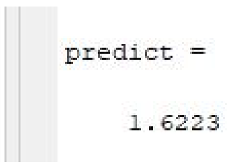


Fig. 1.6 Transformed data space

1.(d)

According to the model before, the predicted value of current is 1.6223 A, as shown in Fig. 1.7. The predict point is marked in both original and transformed data space by green point in Fig. 1.5 and Fig. 1.6.



```
predict =

    1.6223
```

Fig. 1.7 The predicted value

2. The relationship between logistic regression, softmax cost function and Support Vector Machine (SVM) cost function.

Softmax cost function used in perceptron is to address two problems: (a) trivial and undesirable value $b=0$ and $\mathbf{w}=[0]_{N*1}$; (b) cost function $g(b,w)$ is continuous but not everywhere differentiable. It could be used in logistic regression and SVM to provide smooth approximation of original perceptron.

Even the motivations for formally deriving the SVM and logistic regression classifiers differ, if the softmax cost function is employed for SVM, their cost functions can be entirely similar. Eq. 2.1 shows the soft-margin SVM cost function with softmax, and the logistic regression with softmax is shown in Eq. 2.2. Except the ℓ_2 regularized form in SVM, the rest part is apparently similar.

$$g(b,\mathbf{w}) = \sum_{p=1}^{P} \log\left(1+e^{-y_p\left(b+\mathbf{X}_p^T*\mathbf{w}\right)}\right) + \lambda\|\mathbf{w}\|_2^2 \quad \text{Eq. 2.1}$$

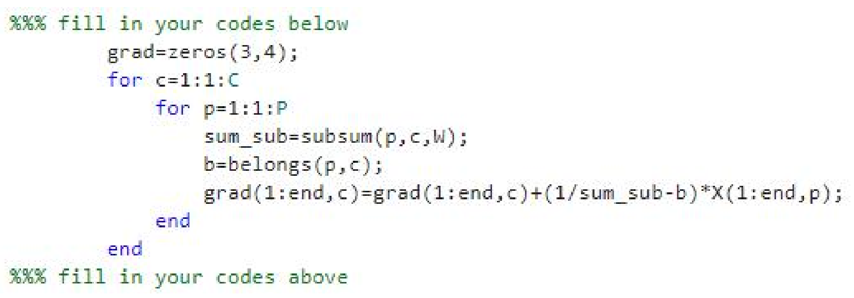
$$\min_{b,\mathbf{w}} \sum_{p=1}^{P} \log\left(1+e^{-y_p\left(b+\mathbf{X}_p^T*\mathbf{w}\right)}\right) \qquad \text{Eq. 2.2}$$

3. Multiclass Softmax Classification

3.(a)

The implementation of gradient 3*4 matrix consists of three parts: two loops for each class and point, the sum of the exponential functions, and judgement of whether a point belongs to calculating class.

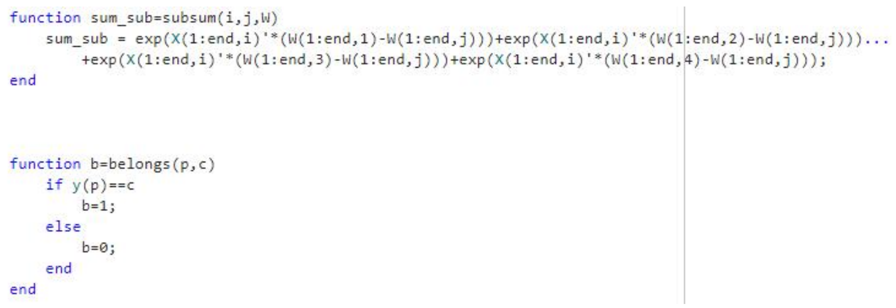
As shown in Fig. 3.1, which is codes for two loops, at the beginning of each iteration, the gradient matrix would be set to zero matrix to make new bias and weights independent from last bias and weights. The calculation of each class, which is placed in corresponding column, is accomplished by loop of coefficient ‘C’. The sum of all points is implemented through loop of coefficient ‘P’.



```
%%% fill in your codes below
        grad=zeros(3,4);
        for c=1:1:C
            for p=1:1:P
                sum_sub=subsum(p,c,W);
                b=belongs(p,c);
                grad(1:end,c)=grad(1:end,c)+(1/sum_sub-b)*X(1:end,p);
            end
        end
%%% fill in your codes above
```

Fig. 3.1 Matlab codes for two loops of each class and point

The sum of the exponential functions and belonging of point are accomplished by two subfunctions, which is shown in Fig. 3.2. Coefficient ‘P’ and ‘C’ is transmitted into to extract correct value. Sum of the exponential functions is calculated directly. Judgement of belonging is accomplished by ‘if’ conditional statement.



```
function sum_sub=subsum(i,j,W)
    sum_sub = exp(X(1:end,i)'*(W(1:end,1)-W(1:end,j)))+exp(X(1:end,i)'*(W(1:end,2)-W(1:end,j)))...
        +exp(X(1:end,i)'*(W(1:end,3)-W(1:end,j)))+exp(X(1:end,i)'*(W(1:end,4)-W(1:end,j)));
end


function b=belongs(p,c)
    if y(p)==c
        b=1;
    else
        b=0;
    end
end
```

Fig. 3.2 Matlab codes for two subfunctions

3.(b)

The output of classification is placed in Fig. 3.3 with distribution of points and separators. After many times operation, the linear separators could be different, but the region classification will keep same. The reason is that, for an particular point, which class it belongs is calculated by set of inequalities that is relative to linear separators. However, even linear separators are different in each operation, it will distribute in a specific range. In this range, it can’t change output of inequalities so that the region classification is always the same.

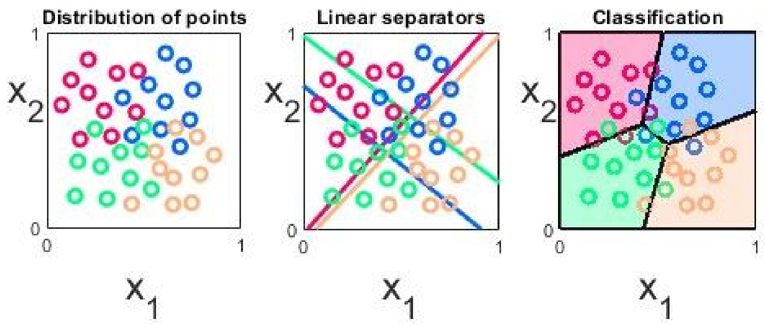
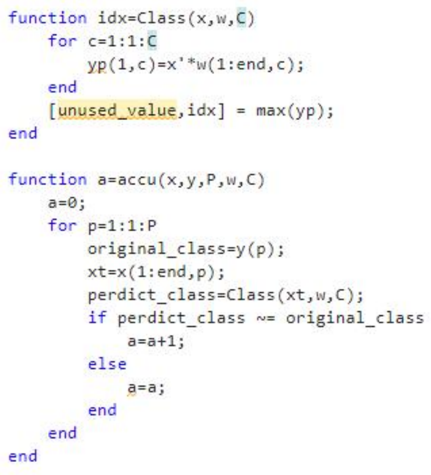


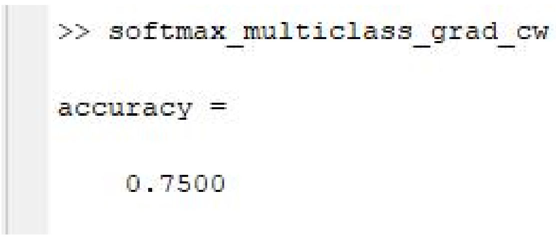
Fig. 3.3 Distribution of points, separators, and output of classification

The calculation of accuracy is arranged as follows. The initial process is to calculate which region a point is placed, which refers to the position of maximum in $\tilde{x}_p^T * \tilde{w}_j\ (j = 1, 2 \ldots C)$. Then, if the class is equal to original label value ‘y’, defined value ‘a’ will keep same. Vice versa, ‘a’ will add 1 if class is different to label value ‘y’. The code is illustrated in Fig. 3.4. Finally, accuracy will be calculated by $1 - a/P$, which is 0.75 in this case. The code is illustrated in Fig. 3.6 and the output of accuracy is shown in Fig. 3.5.



```
function idx=Class(x,w,C)
    for c=1:1:C
        yp(1,c)=x'*w(1:end,c);
    end
    [unused_value,idx] = max(yp);
end

function a=accu(x,y,P,w,C)
    a=0;
    for p=1:1:P
        original_class=y(p);
        xt=x(1:end,p);
        perdict_class=Class(xt,w,C);
        if perdict_class ~= original_class
            a=a+1;
        else
            a=a;
        end
    end
end
```

Fig. 3.4 Matlab codes for class judgment and calculation of accuracy

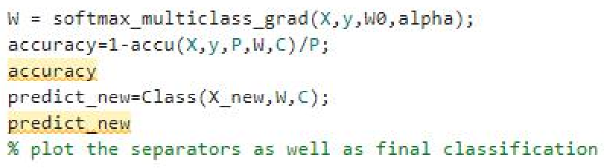


```
>> softmax_multiclass_grad_cw

accuracy =

    0.7500
```

Fig. 3.5 Output of accuracy

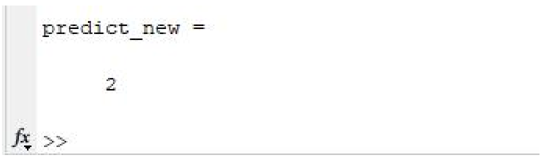
3.(c)

The prediction of new point $[0.5 \quad 0.5]$ could be accomplished by 'Class' subfunction mentioned in previous paragraph. By calculating the position of maximum in $\tilde{x}_{new}^{T} * \widetilde{w}_j \ (j = 1, 2 \dots C)$, the classification for new point is 'Class 2' in this case. The code is illustrated in Fig. 3.6 and the output of accuracy is shown in Fig. 3.7.



```
W = softmax_multiclass_grad(X,y,W0,alpha);
accuracy=1-accu(X,y,P,W,C)/P;
accuracy
predict_new=Class(X_new,W,C);
predict_new
% plot the separators as well as final classification
```

Fig. 3.6 Matlab codes for accuracy and predict value



```
predict_new =

     2

fx >>
```

Fig. 3.7 Output of predict classification for new point