Shirui Ye
PS4
Programming Report

Code:

```
%% SHIRUI YE
%% Nonlinear Kernel Function with ONE-ONE & ONE-
REST & DAG multiclass SVM classifier
implementation for MNIST dataset
clear; clc;
load('MNIST_data.mat')
polynomial_deg = 4;

%% Train Classifier with one-versus-one: design a
SVM between any two samples, so k classes has k(k-
1)/2 SVM. There are 10 classes, so the number of
SVM is 45n The goal is to train 45 different
classifier
disp('1-1')
round = 0;
votes = zeros(size(test_samples_labels,1),10);
for m = 0 : 8
    for n = m + 1 : 9
        round = round + 1;
        [data_matrix, label_vector] =
cluster(train_samples,train_samples_labels,m,n);
        % Langrangian optimization dual form
        a_vector = findAlpha(data_matrix,
label_vector, polynomial_deg);
        pred_vec =
predict_class(a_vector,data_matrix,label_vector,te
st_samples, polynomial_deg);
        m_class = pred_vec > 0;
        pred_vec(m_class) = m;
        pred_vec(~m_class) = n;
        for i = 1:size(pred_vec,1)
            votes(i, pred_vec(i) + 1) = votes(i,
pred_vec(i) + 1) + 1;
        end
    end
end
```

```
[confusion_matrix_1_1,accuracy] =
computeConfusionMatrix(votes, test_samples_labels);
disp(confusion_matrix_1_1)
disp(accuracy)

disp('1-rest')
round = 0;
votes = zeros(size(test_samples,1),10);
for m = 0 : 9
    round = round + 1;
    data_matrix = train_samples;
    label_vector = train_samples_labels;
    m_class = label_vector == m;
    label_vector(m_class) = 1;
    label_vector(~m_class) = -1/9;
    % Langrangian optimization dual form
    a_vector = findAlpha(data_matrix, label_vector,
polynomial_deg);
    votes(:, m + 1) =
predict_class(a_vector,data_matrix,label_vector,te
st_samples, polynomial_deg);
end
[confusion_matrix_1_rest,accuracy] =
computeConfusionMatrix(votes ,test_samples_labels);
disp(confusion_matrix_1_rest)
disp(accuracy)

votes = ones(size(test_samples_labels,1),10);
disp('DAGSVM')
for times = 1 : 9
    for m = 0 : times - 1
        n = m + (10 - times);
        [data_matrix, label_vector] =
cluster(train_samples,train_samples_labels,m,n);
        a_vector = findAlpha(data_matrix,
label_vector, polynomial_deg);
        pred_vec =
predict_class(a_vector,data_matrix,label_vector,te
st_samples,polynomial_deg);
        m_class = pred_vec > 0;
        votes(m_class, n + 1) = 0;
```

```
            votes(~m_class, m + 1) = 0;
        end
    end
    [confusion_matrix_DAG,accuracy] =
    computeConfusionMatrix(votes, test_samples_labels);
    disp(confusion_matrix_DAG)
    disp(accuracy)
```

```
function [data_matrix, label_vector] =
cluster(data, label, m, n)
data_matrix = [];
label_vector = [];
for i = 1:size(data,1)
    if label(i) == m
        label_vector = [label_vector; 1]; %Extend
the vector
    elseif label(i) == n
        label_vector = [label_vector; -1];%Extend
the vector
    else
        continue
    end
    data_matrix = [data_matrix; data(i,:)];%Extend
the matrix
end
end

%According to the different constraints, the
quadratic programming can be divided into the
equality constrained quadratic programming problem
and the inequality constrained quadratic
programming problem. The equality-constrained
quadratic programming problem only contains
equality constraints. The common solutions are
direct elimination method, generalized elimination
```

```
method and Lagrange method. For the inequality
constrained quadratic programming problem, the
basic idea is to impose inequality constraints. It
is transformed into an equality constraint and
solved. The common solution has an active set
method. The effective set method takes the
effective constraint as an equality constraint in
each iteration, and then can be solved by the
Lagrangian method and repeated until the most
Excellent solution.
function a_vector = findAlpha(data_matrix,
label_vector, polynomial_deg)
% MATLAB Function: x = quadprog(H,f,A,b,Aeq,beq):
% minimize 0.5 * x'Hx - f'x where x is variable,
A*x <= b, Aeq * x = beq
    N = size(label_vector,1); % N data points
    H = ((data_matrix *
data_matrix').^polynomial_deg) .* (label_vector *
label_vector');
    f = -ones(N,1);  %- f
    A = -eye(N);
    b = zeros(N,1);
    Aeq = [label_vector'; zeros(N-1,N)]; % A zero
matrix where 1st row contains y
    beq = zeros(N,1); % such that effectively
label_vector' * a_vector = 0
%Display is set to 'off', indicating that the
optimization process does not display information
about the optimization process (in contrast to
'iter', 'iter-detailed', 'notify', 'notify-
detailed', 'final', 'final -detailed' and other
options, please refer to the documentation for the
specific meaning; Algorithm is set to 'sqp', which
means to select the Sequential Quadratic
Programming algorithm. If you want to know more
about the algorithm, some basic introductions are
provided in the documentation. You can know the
basic principles and general characteristics of
various algorithms, and if you want to go deeper,
you need to refer to other specialized documents.
```

```
    options =
optimoptions('quadprog','Algorithm','interior-
point-convex','Display','off');
    a_vector = quadprog(H, f, A, b, Aeq, beq,
[],[],[], options);
end

%%Nonlinear

function prediction_vector =
predict_class(a_vector, data_matrix, label_vector,
test_data, polynomial_deg)
        support_index = a_vector > 0.0001;
        support_matrix_x =
data_matrix(support_index,:);
        support_vector_y =
label_vector(support_index);
        support_alpha = a_vector(support_index);

        M = size(support_vector_y,1); % size of
support vectors
        b = 1/M * sum(support_vector_y -
((support_matrix_x *
support_matrix_x').^polynomial_deg *
(support_vector_y .* support_alpha)));
        prediction_vector = (test_data *
support_matrix_x').^polynomial_deg *
(support_vector_y .* support_alpha) + b;

end

%% Compute confusion matrix and post-processing
%% test sample labels are 1000*1 vectors,votes is
a 1000*10 matrix
function [confusion_matrix,accuracy] =
computeConfusionMatrix(votes, test_samples_labels)

confusion_matrix = zeros(10,10);
[max_counts, max_index] = max(votes,[],2);
for i = 1:size(max_index, 1)
    confusion_matrix(test_samples_labels(i) + 1,
max_index(i))=
```

```
confusion_matrix(test_samples_labels(i) + 1,
max_index(i)) + 1;
end
accuracy = trace(confusion_matrix) /
size(test_samples_labels,1);
end
```

Report:
One-versus-rest (OVR SVMs)

In the training, the samples of a certain category are classified into one class, and the other remaining samples are classified into another class, so that the samples of the k categories construct k kVMs. When classifying, the unknown sample is classified as the one with the largest classification function value.

If I have four categories to divide (that is, 4 Labels), they are A, B, C, and D.

So when I extracted the training set, I extracted it separately.

(1) The vector corresponding to A is used as the positive set, and the vector corresponding to B, C, and D is used as the negative set;

(2) The vector corresponding to B is used as the positive set, and the vector corresponding to A, C, and D is used as the negative set;

(3) The vector corresponding to C is used as the positive set, and the vector corresponding to A, B, and D is used as the negative set;

(4) The vector corresponding to D is used as the positive set, and the vector corresponding to A, B, and C is used as the negative set;

Each of the four training sets is used for training, and then four training result files are obtained.

At the time of testing, the corresponding test vectors are respectively tested using the four training result files.

Finally, each test has a result of f1(x), f2(x), f3(x), and f4(x).

The end result is the largest of the four values as a classification result.

Evaluation:

This method has a drawback because the training set is 1:M, which is biased in this case. It is not very practical. You can extract one third from the complete negative set as the training negative set when extracting the data set.

One-versus-one (OVO SVMs or pairwise)

The approach is to design an SVM between any two types of samples, so k samples need to design k(k-1)/2 SVMs.

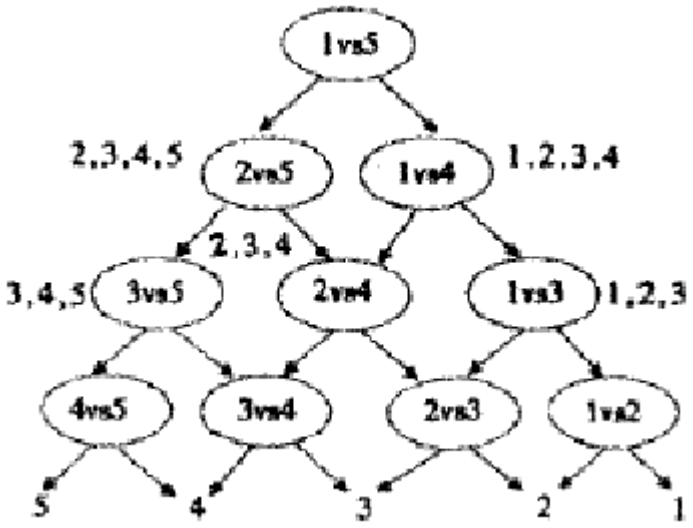
When classifying an unknown sample, the category with the most votes last is the category of the unknown sample.

The multi-class classification in Libsvm is implemented according to this method.

Suppose there are four types of four classes A, B, C, and D. During training, I choose A, B; A, C; A, D; B, C; B, D; C, D corresponding vector as a training set, and then get six training results, at the time of testing, The corresponding vector tests the six results separately, then takes the voting form, and finally gets a set of results.

The vote is like this:
A=B=C=D=0;
(A,B)-classifier If it is A win, then A=A+1; otherwise, B=B+1;
(A, C)-classifier If it is A win, then A=A+1; otherwise, C=C+1;
...
(C, D)-classifier If it is A win, then C=C+1; otherwise, D=D+1;
The decision is the Max (A, B, C, D)

Evaluation: Although this method is good, when there are many categories, the number of models is n*(n-1)/2, and the cost is still quite large.



DAG SVM

So when sorting, we can first ask the classifier "1 to 5" (meaning it can answer "is the first or fifth class"), if it answers 5, we go left and ask "2" For the 5" classifier, if it is still said to be "5", we will continue to the left, so that we can continue to ask, you can get the classification results. Where are the benefits? We actually only call 4 classifiers (if the number of categories is k, only k-1 are called), the classification speed is fast, and there is no classification overlap and unclassifiable phenomenon.

In the field of machine learning, the confusion matrix is also called the probability table or the error matrix. It is a specific matrix used to visualize the performance of the algorithm, usually supervised learning (unsupervised learning, usually with matching matrix: matching matrix). Each column represents a predicted value, and each row represents the actual category. The name comes from the fact that it can be very easy to indicate whether multiple categories are confusing (that is, one class is predicted to be another class).

Output:
1-1

84	0	0	0	0	1	1	0	0	0
0	121	0	0	0	0	0	0	1	0
0	0	110	0	0	0	0	1	2	0
0	0	2	105	0	4	0	2	1	1
0	0	1	0	103	0	1	0	0	3
0	0	1	2	1	87	0	0	1	0
2	0	0	0	1	2	81	0	1	0
0	0	0	1	4	0	0	94	0	0
1	0	1	1	1	0	0	1	80	1
0	0	0	0	1	0	0	0	3	88

0.9530

1-rest

85	0	0	0	0	0	1	0	0	0
0	121	0	0	0	0	0	0	1	0
0	0	109	0	0	0	0	1	3	0
0	0	0	110	0	2	0	1	1	1
0	0	1	0	102	0	1	0	1	3
1	0	0	2	1	86	1	0	1	0
2	0	0	0	0	2	82	0	1	0
0	1	0	1	1	0	0	96	0	0
1	0	1	2	1	0	0	0	80	1
0	0	0	0	1	0	0	0	3	88

0.9590

DAGSVM

84	0	0	0	0	1	1	0	0	0
0	121	0	0	0	0	0	0	1	0
0	0	110	0	0	0	0	1	2	0
2	0	0	105	0	4	0	2	1	1
0	0	1	0	103	0	1	0	0	3
1	0	0	2	1	87	0	0	1	0
2	0	0	0	1	2	81	0	1	0
1	0	0	1	3	0	0	94	0	0
1	0	1	1	1	0	0	1	80	1
0	0	0	0	1	0	0	0	3	88

0.9530