Related Formulas:

Dual Form:

$$\widetilde{L}(\mathbf{a}) = \sum_{n=1}^{N} a_n - \frac{1}{2}\sum_{n=1}^{N}\sum_{m=1}^{N} a_n a_m t_n t_m k(\mathbf{x}_n, \mathbf{x}_m)$$

such that

$$0 \leqslant a_n \leqslant C$$

$$\sum_{n=1}^{N} a_n t_n = 0$$

Score:

$$y(\mathbf{x}) = \sum_{n=1}^{N} a_n t_n k(\mathbf{x}, \mathbf{x}_n) + b.$$

Quadratic Programming:

$$\min_{x} \frac{1}{2} x^T H x + f^T x \text{ such that } \begin{cases} A \cdot x \leq b, \\ Aeq \cdot x = beq, \\ lb \leq x \leq ub. \end{cases}$$

Voting Scheme:

(1) One-vs-one classification:

First, create 4 empty 10*10 cells (alpha, bias, X and Y) to store N(N-1)/2 = 45 sets of data. Then train the dataset using one-vs-one classification and store the results into these 4 cells. When testing, for each test-sample, it will be tested N(N-1)/2 = 45 times so that every two different comparisons will be covered, and each time one of the two comparisons will win (score > 0), then its votes will plus one. After 45 times, one of the 10 comparisons (from 0 to 9) will have the most votes, and that one will be the winner. Then, we predict that particular test-sample as the number of the winner. And so on so forth for 1000 times (number of test-samples).

(2) One-vs-the-rest classification:

Similarly, we create 4 empty cells and train the dataset using one-vs-the-rest classification and store the results into these 4 cells. When testing, for each test-sample, it will be tested 10 times, each time we get a score (formula see above), and after that, the one who gets the highest score wins. Then winner will be the candidate of the prediction this time. And so on so forth for 1000 times.

Performance of Each Classification:

(1) One-vs-one classification:

For one-vs-one, the overall accuracy is **0.952**
Running time: 11.516 s

(2) One-vs-the-rest classification:

For one-vs-the-rest, the accuracy is **0.949**
Running time: 67.499 s

(3) DAGSVM classification:

For DAGSVM, the accuracy is **0.955** (which is > 0.95)
Running time: 12.125 s

Conclusion:

In this MNIST dataset, the one-vs-one classification is faster and more accurate than the one-vs-rest classification. While the DAGSVM is even more accurate than the regular one-vs-one classification.

Strengths and Weaknesses:

(1) One-vs-one classification:

Strength: Suitable for larger train-samples, since the train-samples will be broken down into small pieces for classifier.
Weakness: Not suitable for too many classes, since classification time N(N-1)/2 will explode if N is too large.

(2) One-vs-the-rest classification:

Strength: Suitable for larger classes, since classification time is linear which will not explode even if N is very large.
Weakness: Not suitable for larger train-samples, because each training time, the classifier has to use the whole train-samples.

(3) DAGSVM classification:

Similar with the regular one-vs-one classification, but this one is more accurate than that. Although, the running time is slightly more than the regular one-vs-one classification.

Code

one_vs_one.m:

```
clear all;
clc;
importfile('/Users/qiujingye/Downloads/ps4_2019_files/MNIST_data.mat')
C = 10;

disp('Start one-vs-one classification:')
% Create cells to store results
a_cell = cell(10);
b_cell = cell(10);
X_new_cell = cell(10);
Y_new_cell = cell(10);
for i=1:10
    for j=i+1:10
        disp(['Training "',num2str(i-1),'" VS "',num2str(j-1),'"'])
        l1 = find(train_samples_labels == i-1);
        l2 = find(train_samples_labels == j-1);
        t1 = find(test_samples_labels == i-1);
        t2 = find(test_samples_labels == j-1);

        x1 = train_samples(l1,:)';
        y1 = ones(1,length(l1));
        x2 = train_samples(l2,:)';
        y2 = -ones(1,length(l2));

        X = [x1,x2];
        Y = [y1,y2];
        svm = svmTrain(X,Y,C);
        temp = (svm.a'.*svm.Ysv)*kernel(svm.Xsv,svm.Xsv);
        total_b = svm.Ysv-temp;
        b = mean(total_b);
        % Store results into cells
        a_cell{i, j} = svm.a;
        b_cell{i, j} = b;
        X_new_cell{i, j} = svm.Xsv;
        Y_new_cell{i, j} = svm.Ysv;
    end
end

disp('Testing...');
pred = zeros(size(test_samples, 1), 1);
for i = 1:size(test_samples, 1)
    vote = zeros(10, 1);
    for j = 1:10
        for k = j+1:10
            a = a_cell{j,k};
            b = b_cell{j,k};
            X_new = X_new_cell{j,k};
            Y_new = Y_new_cell{j,k};
            out = (a' .* Y_new) * kernel(X_new, test_samples(i, :)') + b;
            % The one that has positive outcome gets a vote
            if out > 0
                vote(j,1) = vote(j,1) + 1;
            else
                vote(k,1) = vote(k,1) + 1;
            end
        end
    end
    % The one gets the most votes is the winner
    [x,I] = max(vote);
    pred(i) = I-1;
end
accuracy =
length(find(pred==test_samples_labels))/length(test_samples_labels);
disp(['For one-vs-one, the accuracy is ',num2str(accuracy)])
disp(' ')
```

one_vs_the_rest.m:

```
clear all;
clc;
importfile('/Users/qiujingye/Downloads/ps4_2019_files/MNIST_data.mat')
C = 10;

disp('Start one-vs-the-rest classification:')
% Create cells to store results
a_cell = cell(10);
b_cell = cell(10);
X_new_cell = cell(10);
Y_new_cell = cell(10);
for k=0:9
    disp(['Training "',num2str(k),'" VS the rest'])
    l1 = find(train_samples_labels == k);
    l2 = find(train_samples_labels(:,1) ~= k);
    t1 = find(test_samples_labels == k);
    t2 = find(test_samples_labels(:,1) ~= k);

    x1 = train_samples(l1,:)';
    y1 = ones(1,length(l1));
    x2 = train_samples(l2,:)';
    y2 = -ones(1,length(l2));

    X = [x1,x2];
    Y = [y1,y2];
    svm = svmTrain(X,Y,C);
    temp = (svm.a'.*svm.Ysv)*kernel(svm.Xsv,svm.Xsv);
    total_b = svm.Ysv-temp;
    b = mean(total_b);
    % Store results to cells
    a_cell{k+1, 1} = svm.a;
    b_cell{k+1, 1} = b;
    X_new_cell{k+1, 1} = svm.Xsv;
    Y_new_cell{k+1, 1} = svm.Ysv;
end

disp('Testing...');
pred = zeros(size(test_samples, 1), 1);
for i = 1:size(test_samples, 1)
```

```
    score = zeros(10, 1);
    for j = 0:9
            a = a_cell{j+1, 1};
            b = b_cell{j+1, 1};
            X_new = X_new_cell{j+1, 1};
            Y_new = Y_new_cell{j+1, 1};
            out = (a' .* Y_new) * kernel(X_new, test_samples(i, :)') + b;
            score(j+1) = out;
    end
    % The one that gets the highest score wins
    [x, I] = max(score);
    pred(i) = I-1;
end

accuracy =
length(find(pred==test_samples_labels))/length(test_samples_labels);
disp(['For one-vs-the-rest, the accuracy is ',num2str(accuracy)])
disp(' ')
```

DAGSVM.m:

```
clear all;
clc;
importfile('/Users/qiujingye/Downloads/ps4_2019_files/MNIST_data.mat')
C = 10;

disp('Start DAGSVM classification:')
% Create cells to store results
a_cell = cell(10);
b_cell = cell(10);
X_new_cell = cell(10);
Y_new_cell = cell(10);
classes = unique(train_samples_labels);
for i=1:10
    for j=i+1:10
        disp(['Training "',num2str(i-1),'" VS "',num2str(j-1),'"'])

        l1 = find(train_samples_labels == i-1);
        l2 = find(train_samples_labels == j-1);
        t1 = find(test_samples_labels == i-1);
        t2 = find(test_samples_labels == j-1);

        x1 = train_samples(l1,:)';
        y1 = ones(1,length(l1));
        x2 = train_samples(l2,:)';
        y2 = -ones(1,length(l2));

        X = [x1,x2];
        Y = [y1,y2];
        svm = svmTrain(X,Y,C);
        temp = (svm.a'.*svm.Ysv)*kernel(svm.Xsv,svm.Xsv);
        total_b = svm.Ysv-temp;
        b = mean(total_b);
        % Store results to cells
        a_cell{i, j} = svm.a;
        b_cell{i, j} = b;
        X_new_cell{i, j} = svm.Xsv;
        Y_new_cell{i, j} = svm.Ysv;
    end
end
```

```
disp('Testing...');
pred = zeros(size(test_samples, 1), 1);
for i = 1:size(test_samples, 1)
    remain = classes; % From 0 to 9
    while length(remain) > 1
        a = a_cell{remain(1) + 1, remain(end) + 1};
        b = b_cell{remain(1) + 1, remain(end) + 1};
        X_new = X_new_cell{remain(1) + 1, remain(end) + 1};
        Y_new = Y_new_cell{remain(1) + 1, remain(end) + 1};
        out = (a' .* Y_new) * kernel(X_new, test_samples(i, :)') + b;
        if out > 0
            remain = remain(1:end-1); % Remove the tail No. of the remain
        else
            remain = remain(2:end); % Remove the head No. of the remain
        end
    end
    % What is left wins
    pred(i) = remain;
end
accuracy =
length(find(pred==test_samples_labels))/length(test_samples_labels);
disp(['For DAGSVM, the accuracy is ',num2str(accuracy,10)])
disp(' ')
```

svmTrain.m:

```
function svm = svmTrain(X,Y,C)

options = optimset;
options.LargeScale = 'off';
options.Display = 'off';

% Quadratic programming
n = length(Y);
H = (Y'*Y).*kernel(X,X);
f = -ones(n,1);
A = [];
b = [];
Aeq = Y;
beq = 0;
lb = zeros(n,1);
ub = C*ones(n,1);
a0 = zeros(n,1);
a = quadprog(H,f,A,b,Aeq,beq,lb,ub,a0,options);
epsilon = 1e-6;
sv_label = find(abs(a)>epsilon);
svm.a = a(sv_label);
svm.Xsv = X(:,sv_label);
svm.Ysv = Y(sv_label);

end
```

kernel.m:

```
function K = kernel(X,Y)

K = (X'*Y+1).^6;

end
```