1 Problem 1

1.1 Part a

Supposing \( w^T y x > w^T  y x + 1 \). In this case our loss function is defined to be 0, therefore we know that any partial derivative of the function will also be 0 so:

\[
\frac{\partial L((w_1, ..., w_k), (x, y))}{\partial w_{j,l}} = 0
\]

1.2 Part b

Supposing \( w^T y x < w^T  y x + 1 \) and \( j = y \). In this case we can write relevant piece of the loss function as:

\[
(max_{y' \neq j} w^T_y x) + 1 - w^T_j x
\]

\[
\equiv (max_{y' \neq j} w^T_y x) + 1 - \sum_{i=1}^{D} w_{j,i} x_i
\]

Taking the derivative with respect to \( w_{j,l} \) we get:

\[
\frac{\partial L((w_1, ..., w_k), (x, y))}{\partial w_{j,l}} = -x_l
\]

1.3 Part c

Supposing \( w^T y x < w^T  y x + 1 \) and \( j = \hat{y} \), where \( \hat{y} = argmax_{y' \neq y} w^T_y x \). In this case we can write relevant piece of the loss function as:

\[
(max_{y' \neq j} w^T_{y'} x) + 1 - w^T_{\hat{y}} x
\]

\[
\equiv w^T_{\hat{y}} x + 1 - w^T_{y} x
\]

\[
\equiv \sum_{i=1}^{D} w_{j,i} x_i + 1 - w^T_{y} x
\]
Taking the derivative with respect to $w_{j,l}$ we get:

$$\frac{\partial L((w_1, \ldots, w_k), (x, y))}{\partial w_{j,l}} = x_l$$

### 1.4 Part d

Supposing $w_y^T x < w_y^T x + 1$ and $j \neq \hat{y}$ and $j \neq y$, where $\hat{y} = \operatorname{argmax}_{y \neq y} w_y^T x$.

In this case we can write relevant piece of the loss function as:

$$\equiv w_y^T x + 1 - w_y^T x$$

Noting that this expression does not include $w_j$, we get that the partial derivative with respect to $w_{j,l}$ is 0:

$$\frac{\partial L((w_1, \ldots, w_k), (x, y))}{\partial w_{j,l}} = 0$$

### 2 Problem 2

#### 2.1 Part a

Recall that the decision function for the multiclass SVM is:

$$y_i = \operatorname{argmax}_y w_y^T x_i$$

For the 2-class (0 or 1) case, this is:

$$y_i = \operatorname{argmax}(w_0^T x_i, w_1^T x_i)$$

We can equivalently write this as:

$$y_i = 1 \iff w_1^T x_i > w_0^T x_i$$

Rearranging our condition we get:

$$w_1^T x_i - w_0^T x_i > 0$$

$$(w_1 - w_0)^T x_i > 0$$

Note that this is equivalent to a decision function of the form:

$$y_i = 1 \iff w^T x_i > 0$$

Which is a linear perceptron. Therefore we see that our 2-class multiclass SVM with parameters $w_0$ and $w_1$ is equivalent to a perceptron with parameter:

$$w = (w_1 - w_0)$$
2.2 Part b

Now we consider the generalized case, where again our decision function is defined as:

\[ y_i = \operatorname{argmax}_y w_y^T x_i \]

Consider the decision boundary around a particular class \( k \) from our set of \( K \) classes. We see that:

\[ y_i = k \iff w_k^T x_i > w_j^T x_i \quad \forall j \neq k \]

Equivalently:

\[ (w_k - w_j)^T x_i > 0 \quad \forall j \neq k \]

We see then that the decision region for class \( k \) is formed by the intersection of the positive regions for \( K - 1 \) implicitly defined linear perceptrons, where each separates \( k \) from another class \( j \). Therefore the boundary around class \( k \) is formed by the intersection of up to \( K - 1 \) linear decision boundaries.

Accounting for all of the classes, the multiclass SVM’s decision function is defined by \( \binom{K}{2} \) implicit perceptrons. So all of the boundaries defined by the model will have a piecewise linear form. For the \( K = 3 \) case, we can illustrate the decision boundaries like this:

3 Problem 3

See the svmObj function in the code section.
4 Problem 4

4.1 Part a
See the gradOpt function in the code section.

4.2 Part b
See code section for the implementation. The plot resulting from training with $C = 0.01$ is shown below.

![Objective vs. iterations](image)

4.3 Part c
Training and testing with 7 different $C$ values, we get the following results:

<table>
<thead>
<tr>
<th>Test accuracy for each $C$ value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$C$</td>
</tr>
<tr>
<td>-----</td>
</tr>
<tr>
<td>0.0001</td>
</tr>
<tr>
<td>0.001</td>
</tr>
<tr>
<td><strong>0.01</strong></td>
</tr>
<tr>
<td>0.1</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>10</td>
</tr>
<tr>
<td>100</td>
</tr>
</tbody>
</table>
The best accuracy (with $C = 0.01$) was: **87.28%**. The corresponding confusion matrix is shown below.

![Confusion Matrix](image)
4.4 Part d

Plot of the vector \((w_k)\) for each class:
5 Code
5.1 Code

function [X, Y] = setupData(varargin)
% Preprocess a dataset into and appropriate form for
% our SVM training/prediction code.
%
% varargin: A set of (N' x d) matrices where each
%    matrix contains all (N') observations for
%    a given class.
%
% X: Combined matrix of all observations (d x N)
% Y: Vector of labels for each observation (1 x N)

% Concatenate the 'X' matrices into one training
set
X = double(vertcat(varargin{:}));

% Add the extra dimension
X = [X; ones(1, size(X, 2))];

% Create a vector with the label for each
observation
Y = zeros(1, size(X, 2));
start = 1;
for label = 1:nargin
    numlabel = size(varargin{label}, 1);
    Y(start:(start + numlabel - 1)) = label;
    start = start + numlabel;
end
end
function [obj, grad] = svmObj(w, C, X, Y)
% Computes the SVM objective and gradient.
%
% w: Current model weights (d x k)
% C: Regularization tradeoff hyperparameter
% X: Input observations (d x N)
% Y: Input labels (1 x N)
%
% obj: Objective value
% grad: Gradient matrix

% 'Score' of each class for each observation
wx = w' * X;

% Index of the true class' score for each obs.
yind = sub2ind(size(wx), Y, 1:size(wx, 2));

% Get the true class scores, then 'remove' them
wyx = wx(yind);
wx(yind) = -inf;

% Max scores after removing the true classes
[wyhatx, ~] = max(wx, [], 1);

% Loss for each observation
Loss = max(0, wyhatx + 1 - wyx);

% Objective as given in the notes
obj = 0.5 * (w(:)' * w(:)) + C * sum(Loss);

% If needed compute the gradient
if nargout > 1
% Create a #classes x #obs. matrix, where
% entry (i,j) indicates how obs. 'j' should
% contribute to the gradient for class 'i'
% [bsxfun applies a given operation row-by-row]
    cm = double(bsxfun(@eq, wx, wyhatx));
    cm(yind) = -1;
    cm = bsxfun(@times, cm, Loss > 0);

% Compute grad by multiplying our contrib. % matrix by our obs. matrix and adding the % regularization term
    grad = w + C * (cm * X')';
end
end
function [ w, traceout ] = gradOpt(func, w0, r0, T, varargin)

%Generic gradient descent with line search.

% func: Function to compute objective and gradient
% w0: Initial values of parameters to optimize
% r0: Base learning rate
% T: # of iterations for gradient descent
% varargin: Additional arguments passed to 'func'

% w: Final trained weights
% trace: Objective value at each iteration

%Initialize the weights and objective trace
trace = zeros(T,1);
w = w0;

%Run for T iterations
for iter = 1:T
    [obj, grad] = func(w, varargin{:});
    trace(iter) = obj;

    %Line search to find optimal step size
    r = r0;
    for lsiter = 1:20
        wnew = w - r * grad;
        r = r * 0.5;

        if func(wnew, varargin{:}) < obj
            w = wnew;
            break;
        end
    end
end

if nargout > 1
    traceout = trace;
end
end
function [ w, trace ] = svmTrain( X, Y, C, T, r )
%Trains a support vector machine given a dataset.
% X: Input observations (d x N)
% Y: Input labels (1 x N)
% C: Regularization tradeoff hyperparamter
% T: # of iterations for gradient descent
% r: Base learning rate for gradient descent
% w: Final trained model weights (d x k)
% trace: Objective value at each iteration

%Defaults for the hyperparameters
if nargin < 5
    r = 0.5;
end
if nargin < 4
    T = 100;
end
if nargin < 3
    C = 1;
end

%Randomly initialize the model
nlabels = numel(unique(Y));
dim = size(X, 1);
w0 = randn(dim, nlabels);

%Run gradient descent (w/ or w/o the plot data)
if nargout == 1
    w = gradOpt(@svmObj, w0, r, T, C, X, Y);
else
    [w,trace]=gradOpt(@svmObj, w0, r, T, C, X, Y);
end
function [ Y ] = svmPredict( w, X )
% Makes predictions for a set of observations given
% a trained model.
% w: Trained model weights (d x k)
% X: Input observations (d x N)
% Y: Output labels (1 x N)

wx = w' * X;
[~, Y] = max(wx, [], 1);
end

function [ acc ] = score( Ytrue, Ypred )
% Computes the accuracy of a set of predictions
% Ytrue: True labels of data (1 x N)
% Ypred: Predicted labels of data (1 x N)
% acc: Accuracy

acc = sum(double(Ytrue == Ypred)) / numel(Ytrue);
end

function [ mat ] = confusion( Ytrue, Ypred )
% Generate a confusion matrix for a set of predictions.
% Ytrue: True labels of data (1 x N)
% Ypred: Predicted labels of data (1 x N)
% mat: Confusion matrix

labels = unique([Ytrue, Ypred]);
nlabels = numel(labels);
mat = zeros(nlabels);
for l1 = 1:nlabels
    for l2 = 1:nlabels
        mat(l1, l2) = sum(double(Ytrue == l1 & Ypred == l2));
    end
end
end
%hw5.mat
%Runs all of the experiments for homework 5

%Load and combine the training and test data matrices
load digits.mat
[Xtrain, Ytrain] = setupData(train0, train1, train2, train3, train4, train5, train6, train7, train8, train9);
[Xtest, Ytest] = setupData(test0, test1, test2, test3, test4, test5, test6, test7, test8, test9);

%Variables to save our best result
bestC = 0;
bestTrace = [];
bestPred = [];
bestW = [];
bestAcc = 0;

%Train and evaluate the SVM for a range of C values
C = [0.01, 0.1, 1, 10, 100];
[w, trace] = svmTrain(Xtrain, Ytrain, C);
Ypred = svmPredict(w, Xtest);
acc = score(Ytest, Ypred);

%Save the best result
if acc > bestAcc
    bestC = C;
    bestTrace = trace;
    bestPred = Ypred;
    bestW = w;
    bestAcc = acc;
end

%Report the test accuracy and confusion matrix
%(Alternatively, use Matlab's plotconfusion func.)
bestAcc
confusionMat = confusion(Ytest, bestPred)

%Plot the convergence of gradient descent
figure
plot(bestTrace)
title('Objective vs. Iterations')
xlabel('Gradient descent iterations')
ylabel('SVM objective')

% Visualize the weights for the best classifier
figure
title('Best Classifier Visualization')
for digit = 1:10
    img = bestW(1:(end-1), digit);
    subplot(2, 5, digit);
    imagesc(reshape(img, 28, 28))
    axis off
end