Question 1. Large Margin Classification

In this exercise you will write Matlab/Octave functions to implement soft margin support vector machines, both in primal and dual formulations. Then train and test them on simulated data. You will need to be familiar with **quadratic programming** solver in Matlab or Octave. (In Matlab, you can use `quadprog` to solve quadratic programs. In Octave, you can use `qp` to solve quadratic programs.

Part 1: Implement primal.

(a) (1%) Write a function for max margin classification as below:

```
function [yhat] = margin_classify(Xtest, w, b)
% Receives:
% Xtest: (te x n)  test data matrix,
% w: (n x 1) vector of weights,
% b: a scalar,
%
% Returns:
% yhat: (te x 1) vector of predicted target classes on the test
    patterns.
```

The prediction for a test example is given by $\hat{y} = \text{sign}(x^\top w + b)$.

(b) (1%) Write a function to perform training for soft margin SVM in primal form.

```
function [w, b] =  primal_softmargin(X, y, beta)
% Receives:
% X a (t x n) matrix
% y a (t  x 1) vector of training labels
% beta  a scalar regularization parameter beta
%s
% Returns:
% n x 1 vector of weights w
% b a scalar offset
end
```

to implement the soft margin classifier. The (regularized) soft -margin classifier is defined by a weight vector `w`, and offset `b` that minimizes

$$\frac{\beta}{2}\|\mathbf{w}\|_2^2 + \sum_{i=1}^{t} \max\Big(0, 1 - y_i(X_{i:}\mathbf{w} + b)\Big).$$

The solution can be computed by a quadratic program that produces a weight vector `w`, offset `b` and slack vector $\boldsymbol{\delta}$ that minimizes

$$\frac{\beta}{2}\|\mathbf{w}\|_2^2 + \mathbf{1}^\top\boldsymbol{\delta}$$

subject to the constraints

$$\boldsymbol{\delta} \geq \mathbf{1} - \Delta(\mathbf{y})\big(X\mathbf{w} + b\mathbf{1}\big) \quad \text{and} \quad \boldsymbol{\delta} \geq \mathbf{0}.$$

Here $\Delta(\mathbf{y})$ denotes putting `y` on the main diagonal of a square matrix (the `diag` function in Matlab/Octave). Your function must be able to handle arbitrary n and t.

Part 2: Implement dual.

Now, you will implement the dual formulation of the soft margin classifiers.

(c) (1.5%) Write a function to perform training:

```
[lambda, b] = dual_softmargin(K, y, beta)
% Inputs:
% K: t x t kernel matrix
% y: t x 1 vector of Lagrange multipliers
% beta: scalar regularization parameter
% Returns:
% a t x 1 vector of Lagrange multipliers lambda
% a scalar offset b
```

for the soft margin problem in Lagrange dual form.

The Lagrange multipliers $\boldsymbol{\lambda}$ can be found by solving the quadratic program

$$\max_{\boldsymbol{\lambda}} L(\boldsymbol{\lambda}) = \max_{\boldsymbol{\lambda}} \boldsymbol{\lambda}^\top \mathbf{1} - \frac{1}{2\beta}\boldsymbol{\lambda}^\top \Delta(\mathbf{y}) K \Delta(\mathbf{y}) \boldsymbol{\lambda}$$

subject to

$$\boldsymbol{\lambda}^\top \mathbf{y} = 0, \quad 1 \geq \boldsymbol{\lambda} \geq \mathbf{0}.$$

After recovering the optimal $\boldsymbol{\lambda}$, you need to recover b. To do so, just solve for b in the following equation for any one ℓ such that

$$0 < \lambda_\ell < 1: \quad y_\ell \big(K_{\ell:} \Delta(\mathbf{y}) \boldsymbol{\lambda} \frac{1}{\beta} + b \big) = 1.$$

(d) (1%) Write a function to perform prediction in dual form:

```
function [yhat] = dual_classify(Ktest, lambda, b, y, beta}
% Inputs:
% Ktest: te x t matrix of test kernel matrix
% lambda: t x 1 vector of Lagrange multipliers
% b: a scalar offset
% y: t x 1 vector of training labels
% beta: a scalar regularization parameter
% Output:
% yhat: te x 1 vector of class labels
end
```

The vector of predictions for test examples expressed in matrix Ktest is given by

$$\hat{\mathbf{y}} = \text{sign}(Ktest \Delta(y) \boldsymbol{\lambda} \frac{1}{\beta} + b\mathbf{1})$$

for the soft-margin classifier.

Part 3: Implement kernel functions.
(1.5%)

(e) Write three Matlab/Octave functions that receives matrices $X1$ and $X2$ and returns a linear, polynomial, and Gaussian Kernel matrix K respectively. K_{ij} is the similarity between the i th example of `X1` and j th example of `X2` measured with a kernel function.

Linear Kernel

```
function [K] = linear_kernel(X1,X2)
%linear kernel
%Input:
%X1 is a t1xn matrix of examples
%X2 is a t2xn matrix of examples
%Output:
% K is the t1xt2 kernel matrix.
```

Polynomial Kernel

```
function [K] = poly_kernel(X1,X2,d)
% degree d polynomial kernel
%Input:
% X1 is a t1xn matrix of data examples
% X2 is a t2xn matrix of data examples
%Output:
% K is the t1xt2 kernel matrix.
```

Gaussian Kernel (Radial Basis Functions)

```
function [K] = gauss_kernel(X1,X2,sigma)
% Gaussian kernel with sigma parameter
% Input:
% X1 is a t1xn matrix of examples
% X2 is a t2xn matrix of examples
% Output:
% K is the t1xt2 kernel matrix.
```

To form the training kernel matrix `K` for training X1 = X2 = X. To form the test kernel matrix `Ktest`, similarity between examples of X and $Xtest$ is computed.
The definition of similarity function for each kernel, (when x_1, x_2 are *single examples (hence column vectors)*) is shown-up below.

- Linear kernel $k(x_1, x_2) = x_1^\top x_2$
- Polynomial kernel with degree d $k(x_1, x_2) = (1 + x_1^\top x_2)^d$
- Gaussian kernel $k(x_1, x_2) = \exp(-\frac{||x_1 - x_2||_2^2}{2\sigma^2})$

Sanity check

(0%)

Verify that your primal and dual representation gives the same objective valuet on simulated data. (Update: No need to deliver this part!) You can use data generation models below, for this purpose.

Data generation

```
n = 2 % dimension
t = 10 % training size
u = 2*ones(n,1); % target weights (models 1 & 2)
v = 0.8*n % target offset (models 1 & 2)
p_pos = 0.5 % prob of positive example
mu_pos = ones(n,1) % mean loc for pos (model 3)
mu_neg = zeros(n,1) % mean loc for neg (model 3)
```

Generative model 1 : target linear discriminant

```
X = rand(t,n)
y = sign(X*u-v)
```

Generative model 2: target sine nonlinear discriminant

```
X = rand(t,n)
y = sign(sin(X)*u-v)
```

Generative model 3: noisy linear discriminant

```
X = rand(t,n)
y = 2 * (rand(t,1) < p_pos) - 1
pos = find(y > 0)
neg = find(y < 0)
X(pos,1) = X(pos,1) + mu_pos(1); X(pos,2) = X(pos,2) + mu_pos(2)
X(neg,1) = X(neg,1) + mu_neg(1); X(neg,2) = X(neg,2) + mu_neg(2)
```

Experiments

(2%)

In this question, you will run experiments on a real dataset. The problem is to distinguish handwritten '7's and '8's. First download the file data3.mat.

When you load it in Matlab/Octave, you will load training data `X`, training label `y`, test data matrix `xtest` and test labels `ytest`. Each instance is 256 dimensional vector that is a 16×16 grayscale image of a handwritten digit, The images are if handwritten '7'. Thus the main goal of the problem is to learn a function which can accurately distinguish imges of '7's from omamges of '8's. here, functions will be learned on the training data and then tested on the separate test data.

In order to view images use:

```
imagesc(reshape(X(i,:)), 16,16)')
```

Announce the training and test misclassification errors for each of the following classifiers:

1. Primal SVM.
2. Dual SVM with linear kernel.

3. Dual SVM with Gaussian kernel
4. Dual SVM with polynomial kernel

Use parameters `beta=1, sigma =0.01, d=2.`

Question 2. Data Mining using WEKA

(1 %)

This question shows you how to apply ready to use machine learning packages on your data.

In this excercise you learn to work with a useful tool for applying machine learning on different tasks.

"Weka is a collection of machine learning algorithms for data mining tasks. The algorithms can either be applied directly to a dataset or called from your own Java code. Weka contains tools for data pre-processing, classification, regression, clustering, association rules, and visualization."

Find Weka at: `http://www.cs.waikato.ac.nz/ml/weka/`

First, download Weka and start using it and getting comfortable with it. You could easily open, view data, perform cross-validation and then learning and evaluation.

Next, download the data provided in supplementary files and experiment on that/ Announce the best the classification performance you could get on this data. (See it as a contest) Deliver a table of methods you have checked. In particular, announce what is the best accuracy you have been able to get using **SMO** (which is an effective implementation if **SVM**). Check different kernels. Select parameters using **cross validation on training data**. After selecting your parameters and kernels, train on complete training data and test on the provided test data and announce the result.