HOMEWORK 4:
SVMS AND KERNELS

CMU 10601: MACHINE LEARNING (FALL 2016)
OUT: Sep. 26, 2016
DUE: 5:30 pm, Oct. 05, 2016
TAs: Simon Shaolei Du, Tianshu Ren, Hsiao-Yu Fish Tung

Instructions

- **Homework Submission:** Submit your answers and results to Gradescope. You will use Autolab to submit your code in Problem 2.2. For Gradescope, you will need to specify which pages go with which question. We provide a LaTeX template which you can use to type up your solutions. Please check Piazza for updates about the homework.
- **Collaboration policy**: The purpose of student collaboration is to facilitate learning, not to circumvent it. Studying the material in groups is strongly encouraged. It is also allowed to seek help from other students in understanding the material needed to solve a particular homework problem, provided no written notes (including code) are shared, or are taken at that time, and provided learning is facilitated, not circumvented. The actual solution must be done by each student alone. The presence or absence of any form of help or collaboration, whether given or received, must be explicitly stated and disclosed in full by all involved. Please refer to the course webpage.

1 Support Vector Machine (50 pts + 10 pts Extra Credits)

Suppose we have the following data

$$\mathcal{D} = (\mathbf{X}, \mathbf{y})$$

where $\mathbf{X} \in \mathbb{R}^{d \times n}$, the i-th column x_i are the features of the i-th training sample and y_i is the label of the i-th training sample. $y \in \{-1, 1\}^n$ if this is a classification problem and $y \in \mathbb{R}^n$ if this is a regression problem.

1.1 SVM for Classification (Hard Margin)

In this problem, you will derive SVM objective from large margin principle. We use linear discriminant function

$$f(x) = w^\top x$$

for classification: we classify x into class -1 if $f(x) < 0$ and into class 1 otherwise. If the data is linearly separable, and f is the target function, then $y \cdot f(x) > 0$. This fact allows us to define

$$\gamma = \frac{y \cdot f(x)}{\|w\|}.$$

as distance of x from the decision boundary, i.e., margin.

1. (20 pts) We would like to make this margin γ as large as possible, i.e. maximize the perpendicular distance to the closest point. Thus our objective function becomes

$$\max_w \min_{i=1}^{n} \frac{y_i f(x_i)}{\|w\|}. \tag{1}$$

   (Think about why we use this function). Show that (1) is equivalent to the following problem:

$$\min_w \frac{1}{2}\|w\|_2^2$$
$$\text{such that } y_i \cdot \left(w^\top x_i\right) \geq 1, i = 1, \cdots n.$$

   (Hint: Does it matter if we rescale $w \to kw$?)

   Note that if we scale $w \to kw$, $\frac{y_i w^\top x_i}{\|w\|} = \frac{y_i k w^\top x_i}{\|kw\|}$. Further, we have $y_i f(x_i) > 0$. Thus, for simplicity, we can let the point to have minimum distance from the decision boundary to satisfy $y_s f(x_s) = 1$. And the original problem can be reformulated as

$$\max_w \frac{1}{\|w\|} \tag{2}$$
$$\text{such that } y_i \cdot (w^\top x_i) \geq 1, i = 1, \cdots n. \tag{3}$$

   Maximizing $\frac{1}{\|w\|}$ is equivalent to minimizing $\frac{1}{2}$ of the square of its reciprocal. Thus we have shown the equivalence.

2. (10 pts) In the linearly separable case if one of the training samples is removed, will the decision boundary shift toward the point removed or shift away from the point remove or remain the same? Justify your answer.

   If the removed point is at the boundary, then the boundary will shift toward the point. Otherwise, it remains the same.

1.2 SVM for Classification (Soft Margin)

Recall from the lecture notes that if we allow some misclassification in the training data, the primal optimization of SVM is given by

$$\begin{aligned} \underset{w,\xi_i}{\text{minimize}} \quad & \frac{1}{2}||w||_2^2 + C\sum_{i=1}^{n} \xi_i & \\ \text{subject to} \quad & y_i(w^T x_i) \geq 1 - \xi_i & \forall i = 1, \ldots, n \\ & \xi_i \geq 0 & \forall i = 1, \ldots, n \end{aligned}$$

Recall from the lecture notes $\xi_1, \ldots, \xi_n$ are called slack variables. The optimal slack variables have intuitive geometric interpretation as shown in Fig. 1. Basically, when $\xi_i = 0$, the corresponding feature vector x_i is correctly classified and it will either lie on the margin of the separator or on the correct side of the margin. Feature vector with $0 < \xi_i \leq 1$ lies within the margin but is still be correctly classified. When $\xi_i > 1$, the corresponding feature vector is misclassified. Support vectors correspond to the instances with $\xi_i > 0$ or instances that lie on the margin.

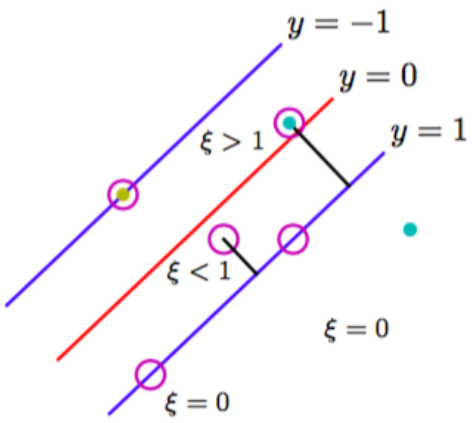

Figure 1: The relationship between the optimal slack variables and the optimal linear separator in the feature space. Support vectors are surrounded with circles.

1. (5 pts) Suppose the optimal $\xi_1, \ldots, \xi_n$ have been computed. Use the ξ_i to obtain an upper bound on the number of misclassified instances.

   $\sum_i \xi_i$.

2. (15 pts) In the primal optimization of SVM, what's the role of the coefficient C? Briefly explain your answer by considering two extreme cases, i.e., $C \to 0$ and $C \to \infty$.

   C works as a trade-off between the classification accuracy on the training data set and the margin of the linear separator. When $C \to \infty$, all training points are forced to be correctly classified, corresponding to more complex models. When $C \to 0$, no mis-classification error will be penalized, which encourages simpler models with larger margins.

1.3 SVM for Regression (10 pts Extra Credits)

Let $x \in \mathbb{R}^d$ be the feature vector and $y \in \mathbb{R}$ be the label. In this question, we use a linear predictor for the label, i.e. given the feature vector, we predict the label by

$$\hat{y}(x) = w^\top x$$

where $w \in \mathbb{R}^d$ is the linear coefficient vector. In this question, we consider the **epsilon insensitive loss function**, defined by

$$L_\epsilon(y, \hat{y}) = \begin{cases} 0 & \text{if } |y - \hat{y}| < \epsilon \\ |y - \hat{y}| - \epsilon & \text{otherwise.} \end{cases}$$

where ϵ is a tuning parameter. To obtain a good w, we would like to solve the following optimization:

$$J(w) = \frac{1}{n} \sum_{i=1}^{n} L_\epsilon(y, \hat{y}(x_i)) + \lambda \|w\|_2^2. \tag{4}$$

1. (2 pts) What kind of loss is L_ϵ if $\epsilon = 0$? When $\epsilon = 0$, $L(y, \hat{y}) = |y - \hat{y}|$, which is a L-1 loss on the difference between y and $\hat{y}$.

2. (3 pts) When it is beneficial to use L_ϵ for $\epsilon > 0$? Give a real world example. (Hint: think about the name of this loss function.) In tasks like stock prediction, we do not care about small prediction error (within ϵ) because small error is often considered as noise.

3. (5 pts) Notice that (4) is not a differentiable. Show how to convert (4) into a optimization problem whose objective is differentiable and constraints are linear by introducing slack slack variables.

   Note that we can represent $L_\epsilon(y, \hat{y})$ as $\max(y - \hat{y} - \epsilon, 0) + \max(\hat{y} - y - \epsilon, 0)$. Thus we can introduce non-negative slack variables $\xi = y - \hat{y} - \epsilon$ and $\hat{\xi} = \hat{y} - y - \epsilon$ and reformulate the original objective into the following optimization form:

$$\min \frac{1}{n} \sum_{i=1}^{n} (\xi_i + \hat{\xi}_i) + \lambda \|w\|_2^2 \tag{5}$$

$$\text{such that } y_i - w^\top x_i - \xi_i - \epsilon \leq 0 \tag{6}$$

$$w^\top x_i - y_i - \hat{\xi}_i - \epsilon \leq 0 \tag{7}$$

$$\xi_i \geq 0 \tag{8}$$

$$\hat{\xi}_i \geq 0 \tag{9}$$

2 Kernels (50 pts)

2.1 Kernel Basics (10 pts)

1. (5 pts) Suppose the input space is three-dimensional $\mathbf{x} = (x_1, x_2, x_3)^T$. Define the feature mapping $\phi(\mathbf{x}) = (x_1^2, x_2^2, x_3^2, \sqrt{2}x_1x_2, \sqrt{2}x_1x_3, \sqrt{2}x_2x_3)^T$. What is the corresponding kernel function, i.e. $K(\mathbf{x}, \mathbf{z})$? Express your answer in terms of $x_1, x_2, x_3, z_1, z_2, z_3$.

$$K(\mathbf{x}, \mathbf{z}) = \phi(\mathbf{x})^\top \phi(\mathbf{z}) = (x_1z_1)^2 + (x_2z_2)^2 + (x_3z_3)^2 + 2x_1x_2z_1z_2 + 2x_1x_3z_1z_3 + 2x_2x_3z_2z_3 \tag{10}$$

$$= (\mathbf{x}^\top \mathbf{z})^2 \tag{11}$$

2. (5 pts) With the same feature map in the previous question, suppose we want to compute the value of kernel function $K(\mathbf{x}, \mathbf{z})$ on two vectors $\mathbf{x}, \mathbf{z} \in \mathbb{R}^3$. How many additions and multiplications are needed if you

   (a) map the input vector to the feature space and then perform dot product on the mapped features? Mapping each input vector to feature space takes 9 multiplications. Performing dot product takes 6 multiplications and 5 additions. Total number of multiplication is 24 and total number of additions is 5.

   (b) compute through kernel function you derived in part (a)? Computing the inner product in the original space takes 3 multiplications and 2 additions. Taking the square of that result takes 1 multiplication. Total 4 multiplications and 2 additions.

2.2 Programming Kernel Perceptron (40 pts)

In this section you implement the kernel perceptron algorithm to between handwritten digits 4 and 7 of MNIST dataset. Also you will compare kernel perceptron with linear perceptron on a synthetic dataset.

For your convenience, we have provided full code for linear perceptron in `perceptron_run.m` and starter codes for kernel perceptron in the handout, which you may download from Autolab. You may also test the correctness of your program with the offline dataset we provided, however you will be evaluated on a separate dataset on Autolab. For problem 2.1 - 2.4, you are expected to experiment on `perceptron_train.mat` and `perceptron_test.mat`. For problem 2.5, you are expected to experiment on `synthetic_train.mat` and `synthetic_test.mat`.

To submit the code, make sure you put all your files (excluding data) in a folder called **hw4** and run the following command in its top directory: `tar -cvf hw4.tar hw4`. Then submit the tarball created to Autolab.

In order to create faster code we approach the kernel perceptron by first building a kernel matrix, K, and then querying that kernel matrix every time we are interested in using some kernel evaluation $k(x_i, x_j)$.

The kernel matrix K stores the evaluations of the kernel $X1 \in R^{d\times n}$ and $X2 \in R^{d\times m}$ where $X1$ is usually the training set and $X2$ the testing set. Thus, $K \in R^{n\times m}$ has dimensions $n \times m$.

Recall the kernel perceptron algorithm we learned in class for $x_i \in R^d$ and $y_i \in \{-1, 1\}$

- `Initialize the counting vector` $\alpha \in R^n$
- `Iterate until convergence:`
  - `For` $i = 1 \dots n$`:`
    - $\hat{y}_i = \varphi(\sum_{r=1}^{n} \alpha_r y_r K_{i,r})$
    - `If:` $\hat{y}_i \neq y_i$`; Then:` $\alpha_i = \alpha_i + 1$

where

$$\varphi(z) = \begin{cases} -1 & \texttt{if } z \leq 0 \\ 1 & \texttt{otherwise} \end{cases}.$$

Note that in the code we use a instead of α.

1. (5 pts) Complete `polykernel.m`: the function `K = polykernel(X1,X2)` computes a kernel matrix of a non-homogeneous polynomial kernel of degree-3 with an offset of 1. The kernel matrix should be constructed such that if $X1 \in R^{d\times n}$ and $X2 \in R^{d\times m}$, then $K \in R^{n\times m}$.

   The polynomial kernel of degree-3 with an offset of 1 is defined as:

   $$k(x_i, x_j) = (x_i^T x_j + 1)^3$$

```
function K = polykernel(X1,X2)
% Homogeneous polynomial kernel of degree-3 with an offset of 1
% X1 is a matrix
% X2 is a matrix

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Your code goes here
l = 1;
d = 3;
K = (X1' * X2 + l).^d;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

2. (10 pts) Complete `kernel_perceptron_pred.m`: the function `kernel_perceptron_pred(a, Y, K, i)` takes in a counting vector, a, class labels, Y, the kernel matrix K, and the index of the sample we are interested in evaluating, i. The output should be the predicted label for x_i.

```
function pred = kernel_perceptron_pred(a, Y, K, i)
%PERCEPTRON_PRED: Make prediction using weight vector w and feature vector
%   a is counting vector (1 * n)
%   X is feature values of training examples (d * n)
%   Y is labels of training examples (1 * n)
%   xi is the current x of interest

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Your code goes here
input = sum(a.*Y.*K(i,:));
if input <= 0; pred = -1;
else pred = 1; end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

3. (10 pts) Complete `w = kernel_perceptron.m`: the function `w = kernel_perceptron(a0, X, Y)` runs the kernel perception training algorithm taking in an initial counting vector $a0$, a matrix of features X, and vector of labels Y. It outputs a learned weight vector.

```
function w = kernel_perceptron(w0, X, Y)
%Kernel perceptron algorithm
%   w0 is the initial weight vector (1 * n)
%   X is feature values of training examples (d * n)
%   Y is labels of training examples (1 * n)

[d,n] = size(X);
maxiter = 100;
stop_criteria = 1;

% WEIGHTING vector "w" initialized to zeros. Use this in your algorithm!
w = w0;

% PREDICTION vector "y_hat" initialized to zeros. Use this in your algorithm!
y_hat = zeros(1,n);
y_hat_prev = y_hat;

% The Gram matrix is constructed here.
K = polykernel(X,X);

for k = 1:maxiter

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Your code goes here
for i=1:n
y_hat(i) = kernel_perceptron_pred(w, Y, K, i);
if y_hat(i) ~= Y(i);
w(i) = w(i) + 1;
end
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

eps = sum(y_hat == y_hat_prev) / length(y_hat);
fprintf('%21d %20g\n', k, eps);
if eps >= stop_criteria
break;
end
y_hat_prev = y_hat;

end
```

4. (10 pts) Run `kernel_perceptron_run.m` on `perceptron_train.mat` and `perceptronb_test.mat` to perform training and see the performance on training and test sets. Report your accuracy on both datasets.

```
clear variables;
load perceptron_train.mat; load perceptron_test.mat;

% Visualize the data
display_network(train.X(:,1:100));

% Normalize and center training data
train.X = standardize(train.X);
test.X = standardize(test.X);
```

```
% Add a row of 1's as x0 to introduce intercept
train.X = [ones(1, size(train.X, 2)); train.X];
test.X = [ones(1, size(test.X, 2)); test.X];

% Initialize
w0 = zeros(1, size(train.X, 2));

% Train
w = kernel_perceptron(w0, train.X, train.y);

% Print out training accuracy
K = polykernel(train.X,train.X);
for i=1:size(train.X,2); train_pred(i) = kernel_perceptron_pred(w,  train.y, K, i);
trainnacc = train_pred == train.y;
trainacc = sum(trainnacc) / length(train.y);

fprintf('Your training accuracy is:%6.4f%%\n', 100 * trainacc);

% Print out cross-validation accuracy
K = polykernel(test.X,train.X);
for i=1:size(test.X,2); test_pred(i) = kernel_perceptron_pred(w,  train.y, K, i); er
testnacc = test_pred == test.y;
testacc = sum(testnacc) / length(test.y);

fprintf('Your test accuracy is:%6.4f%%\n', 100 * testacc);
```

5. (5 pts) We have provided you linear perceptron codes and another data set: `synthetic_train.mat` and `synthetic_test.mat`. Please run `perceptron_run.m` and report the accuracy of linear perceptron. Next test your kernel perceptron on this new data set. Compare the classification accuracy of these two methods. Which one is better? Why is this the case?

   The accuracy of linear perceptron is about 35% on test dataset. The accuracy of kernel perceptron is about 99% on test dataset. The kernel perceptron is better. This is because the data was not linearly separable in the original space, but becomes separable in the implicit mapped feature space of the kernel.

3 Perceptron and Subgradient (10 pts Extra Credits)

In this problem you will find the connection between Perceptron algorithm and (stochastic) subgradient descent. First, please look at the following two definitions:

Definition 1 (Hing Loss)**.** For $y, y' \in \mathbb{R}$, the Hinge loss is defined by:

$$L(y, y') = \max(0, 1 - y \cdot y').$$

Definition 2 (Subgradient)**.** Given a function $F(x)$, which may be non-differentiable, $v(x)$ is a subgradient of $F(x)$ if and only if for any x', the following holds:

$$F(x') \geq F(x) + v(x)^\top (x' - x)\,.$$

1. (5 pts) Let (x, y) be a sample from the training data where $x \in \mathbb{R}^d$ and $y \in \mathbb{R}$. Define $F(w) = L(y, w^\top x)$. What is the subgradient of $F(w)$? (hint: there are 3 cases).

$$\partial F(w) = \begin{cases} 0 & \text{if } y(w^\top x) > 1 \\ -yx & \text{if } y(w^\top x) < 1 \\ [-yx, 0] & \text{if } y(w^\top x) = 1. \end{cases}$$

2. (5 pts) For a given function $F(w) = \frac{1}{n}\sum_{i=1}^{n} F_i(w)$, Cyclic Subgradient Descent Algorithm is following procedure

   - `Initialize the variable` $w \in R^d$.
   - `Iterate until convergence:`
     - `For` $i = 1 \ldots n$`:`
       * Find a subgradient of $v_i(w)$ of $F_i(w)$.
       * $w \leftarrow w - \eta v_i(w)$ where $\eta \in \mathbb{R}^+$ is the step size.

   Show why perceptron is a special case of Cyclic Subgradient Descent Algorithm.

Consider the loss function

$$F(w) = \frac{1}{n}\sum_{i=1}^{n} L(y_i, w^\top x_i). \tag{12}$$

, where $L(y, y') = \max(0, -y \cdot y')$ Note that the formula for perceptron is, at every iteration

$$w \leftarrow w - \left(-\mathbf{1}\left(y_i\left(w^\top x_i\right) \le 0\right) yx\right)$$

where $\mathbf{1}\left(y_i\left(w^\top x_i\right)\right)$ is indication function. Note

$$\left(-\mathbf{1}\left(y_i\left(w^\top x_i\right) \le 0\right) yx\right)$$

is a subgradient of $F_i(w)$. Thus perpectron is a cyclic subgradient descent algorithm minimizing (12).