HOMEWORK 3

CMU 10601: MACHINE LEARNING (SPRING 2016)
OUT: Feb. 3, 2016
DUE: 5:30 pm, Feb. 18, 2016
TAs: Tianshu Ren and William Herlands

Instructions

- **Submit your homework** on time **BOTH** electronically by submitting to Autolab **AND** by dropping off a hardcopy in the bin outside Gates 8221 by 5:30 pm, Thursday, February 18, 2016.
- **Include your Andrew ID** on your homework.
- **Late homework policy**: Homework is worth full credit if submitted before the due date, half credit during the next 48 hours, and zero credit after that. Additionally, you are permitted to drop 1 homework.
- **Collaboration policy**: For this homework only, you are welcome to collaborate on any of the questions with anybody you like. However, you *must* write up your own final solution, and you must list the names of anybody you collaborated with on this assignment. The point of this homework is not really for us to evaluate you, but instead for *you* to determine whether you have the right background for this class, and to fill in any gaps you may have.

1 Kernel feature mappings [10 pts]

1. (5 pts) Consider we have a two-dimensional input space such that the input vector is $\mathbf{x} = (x_1, x_2)^T$. Define the feature mapping $\phi(\mathbf{x}) = (x_1^2, \sqrt{2}x_1x_2, x_2^2)^T$. What is the corresponding kernel function, i.e. $K(\mathbf{x}, \mathbf{z})$? Do not leave $\phi(\mathbf{x})$ in your final answer.
Solution: $K(\mathbf{x}, \mathbf{z}) = \phi(\mathbf{x})^T\phi(\mathbf{z}) = (x_1^2z_1^2 + 2x_1x_2z_1z_2 + x_2^2z_2^2) = (x_1z_1 + x_2z_2)^2 = (\mathbf{x}^T\mathbf{z})^2$

2. (5 pts) Suppose we want to compute the value of kernel function $K(\mathbf{x}, \mathbf{z})$ on two vectors $\mathbf{x}, \mathbf{z} \in \mathbb{R}^2$. 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? Solution: 11 multiplications and 2 additions

   (b) compute through kernel function you derived in part (a)? Solution: 3 multiplications and 1 addition

2 Perceptrons [20 pts]

For this section, consider the single layer perceptron network shown in Figure 1 with inputs, $x_i \in \{0, 1\}$, and transfer function

$$y = \varphi\Big(\sum_i x_i * w_i + b\Big) \tag{1}$$

where

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

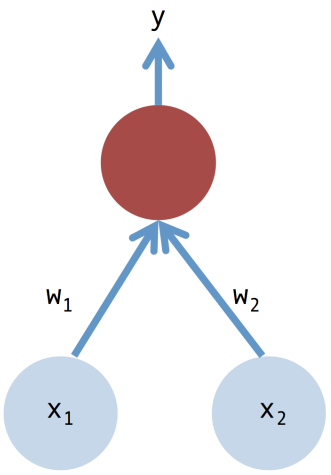

Figure 1: Perceptron for question 2

1. (3 pts) Complete the following truth table for the AND operation

Table 1: AND contingency table

x_1	x_2	AND
0	0	
0	1	
1	0	
1	1	

Solution:
0,0,0,1

2. (3 pts) Provide values for w_1, w_2, and b to use the perceptron for logical AND Solution:
A number of solutions are acceptable. For example:
$w_1 = 1$,$w_2 = 1$,$b = -1.5$
The general rule is that $b < 0$ and $w_1 + w_2 > b$ and $|w_1|, |w_2| < b$

3. (3 pts) Complete the following truth table for the OR operation

Table 2: OR contingency table

x_1	x_2	OR
0	0	
0	1	
1	0	
1	1	

Solution:
0,1,1,1

4. (3 pts) Provide values for w_1, w_2, and b to use the perceptron for logical OR Solution:
A number of solutions are acceptable. For example:
$w_1 = 2$,$w_2 = 2$,$b = -1.5$
The general rule is that $b < 0 <= w_1, w_2$ and $w_1, w_2 > |b|$

5. (3 pts) Complete the following truth table for the XOR operation

Table 3: XOR contingency table

x_1	x_2	XOR
0	0	
0	1	
1	0	
1	1	

Solution:
0,1,1,0

6. (5 pts) Prove that the single layer perceptron depicted in Figure 1 cannot be used to create logical XOR Solution:
Generally, the perceptron creates a linear seperator. However, the XOR function requires two linear

separators. More formally: From the case $x_1 = 0, x_2 = 0$ we know that $b <= 0$
From the case $x_1 = 0, x_2 = 1$ we know that $w_2 > 0, w_2 > |b|$
From the case $x_1 = 1, x_2 = 0$ we know that $w_1 > 0, w_1 > |b|$
For the case $x_1 = 1, x_2 = 1$ we need $w_1 + w_2 + b <= 0$
However, this contradicts our previous requirements. Thus it is impossible to use our model to create a logical XOR.

(Spoiler: to create an XOR function wait for the lecture on neural networks!)

3 Regression theory [30 pts]

3.1 Linear Regression [25pts]

Assume we have n data points $\{x_1, y_1\}, \ldots, \{x_n, y_n\}$, $x \in R^m$, $y \in R^1$ that are sampled iid such that $y_i \sim N(\omega^T x_i, \sigma^2)$ for some known value of $\sigma \geq 0$.

1. Consider linear regression with $f(x) = \omega^T x$. We will derive the gradient descent algorithm using squared loss.

   (a) (3 pts) Define the squared error loss, $J(\omega)$ associated with $f(x)$
   Solution:
   $\sum_i^n (y_i - f(x_i))^2$
   $\sum_i^n (y_i - \omega^T x_i)^2$

   (b) (3 pts) Find the partial derivative of the loss with respect to ω_k.
   Solution:
   $\frac{\delta}{\delta \omega^k} J(\omega) = -2 * \sum_i^n \Big(y_i - f(x_i)\Big) x_i^k$

   (c) (3 pts) Put this together to show that the update rule for each $k = 1 \ldots m$ for batch gradient descent of linear regression is

   $$\omega_{new}^k = \omega^k + \alpha \sum_i^n \Big(y_i - f(x_i)\Big) x_i^k \tag{2}$$

   Solution:
   The general form of stochastic gradient descent is
   $\omega^{k-new} = \omega^k + \alpha \frac{\delta}{\delta \omega_k} J(\omega)$ where α is the step size. It follows from part (b) that
   $\omega^{k-new} = \omega^k + \alpha * 2 * \sum_i^n \Big(y_i - f(x_i)\Big) x_i^k$
   Since the constant factor of 2 is subsumed into the step size tuning parameter we get our desired result.

2. Consider linear regression with $f(x) = \omega^T x$. We will now show the equivalence between the minimizer of the squared error and the maximum likelihood for this problem.

   (a) (3 pts) Derive the conditional likelihood, $L(\omega; x, y)$
   Solution:
   $L(\omega) = \prod_{i=1}^n p(y_i | x_i, \omega)$
   $L(\omega) = \prod_{i=1}^n \frac{1}{\sqrt{2\pi}\sigma} \exp\Big(\frac{-(y_i - \omega^T x_i)^2}{2\sigma^2}\Big)$

(b) (3 pts) Derive the log conditional likelihood, $l(\omega; x, y)$
Solution:
$l(\omega) = \log \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi}\sigma} \exp\left(\frac{-(y_i - \omega^T x_i)^2}{2\sigma^2}\right)$
$l(\omega) = \sum_{i=1}^{n} \log \frac{1}{\sqrt{2\pi}\sigma} \exp\left(\frac{-(y_i - \omega^T x_i)^2}{2\sigma^2}\right)$
$l(\omega) = \sum_{i=1}^{n} \log \frac{1}{\sqrt{2\pi}\sigma} + \sum_{i=1}^{n} \log \exp\left(\frac{-(y_i - \omega^T x_i)^2}{2\sigma^2}\right)$
$l(\omega) = n \log \frac{1}{\sqrt{2\pi}\sigma} - \frac{1}{2\sigma^2} \sum_{i=1}^{n} (y_i - \omega^T x_i)^2$

(c) (3 pts) Show that maximizing the log likelihood is equivalent to minimizing the least square loss from part (1b).
Solution:
$\max_\omega l(\omega) = \max_\omega n \log \frac{1}{\sqrt{2\pi}\sigma} - \frac{1}{2\sigma^2} \sum_{i=1}^{n} (y_j - \omega^T x_j)^2$
Since we are maximizing over ω we can ignore constant terms and scaling terms
$\max_\omega l(\omega) = \max_\omega - \sum_{i=1}^{n} (y_j - \omega^T x_j)^2$
$\max_\omega l(\omega) = \min_\omega \sum_{i=1}^{n} (y_j - \omega^T x_j)^2$

3. In the context of regression, we usually assume X and Y are two random variables such that the relationship between their realizations (x, y) is $y = f(x) + \epsilon$ where ϵ has mean 0 and variance σ^2. The goal is to find a "good" estimator $\hat{f}(x)$ to capture the relationship between X and Y based on training samples $\{(x_1, y_1), ..., (x_n, y_n)\}$.

(a) (3 pts) For a specific $\hat{f}$, state the expected loss under squared-error at some test point x. What is this expectation taken over? (You may also describe the randomness in words)
Solution: Expected loss is $\mathbb{E}[(y - \hat{f}(x))^2]$. The expectation is taken over training samples $(x_1, y_1), ..., (x_n, y_n)$ and ϵ, which determines y. It is okay to describe the randomness as in training samples and y that corresponds to x.

(b) (4 pts) Show that the expected loss under squared-error at point x can be decomposed into $\textit{Bias}^2 + \textit{Variance} + \sigma^2$, where:

$$\textit{Bias} = f(x) - \mathbb{E}[\hat{f}(x)]$$

$$\textit{Variance} = \mathbb{E}[(\mathbb{E}[\hat{f}(x)] - \hat{f}(x))^2]$$

Solution:

$$\begin{aligned}\mathbb{E}[(y - \hat{f}(x))^2] &= \mathbb{E}[(y - \mathbb{E}[\hat{f}(x)] + \mathbb{E}[\hat{f}(x)] - \hat{f}(x))^2] \\ &= \mathbb{E}[(y - \mathbb{E}[\hat{f}(x)])^2] + \mathbb{E}[(\mathbb{E}[\hat{f}(x)] - \hat{f}(x))^2] + 2\,\mathbb{E}[(y - \mathbb{E}[\hat{f}(x)])(\mathbb{E}[\hat{f}(x)] - \hat{f}(x))]\end{aligned}$$

Since $y = f(x) + \epsilon$, $\mathbb{E}[\epsilon] = 0$ and $\text{Var}(\epsilon) = \sigma^2$,

$$\begin{aligned}\mathbb{E}[(y - \mathbb{E}[\hat{f}(x)])^2] &= \mathbb{E}[(f(x) + \epsilon - \mathbb{E}[\hat{f}(x)])^2] \\ &= \mathbb{E}[(f(x) - \mathbb{E}[\hat{f}(x)])^2] + \sigma^2 \\ &= (f(x) - \mathbb{E}[\hat{f}(x)])^2 + \sigma^2\end{aligned}$$

Further,

$$\mathbb{E}[(y - \mathbb{E}[\hat{f}(x)])(\mathbb{E}[\hat{f}(x)] - \hat{f}(x))] = \mathbb{E}[(f(x) - \mathbb{E}[\hat{f}(x)])(\mathbb{E}[\hat{f}(x)] - \hat{f}(x))]$$

Notice that $f(x) - \mathbb{E}[\hat{f}(x)]$ is just a constant. Thus,

$$\begin{aligned}\mathbb{E}[(y - \mathbb{E}[\hat{f}(x)])(\mathbb{E}[\hat{f}(x)] - \hat{f}(x))] &= (f(x) - \mathbb{E}[\hat{f}(x)])\,\mathbb{E}[\mathbb{E}[\hat{f}(x)] - \hat{f}(x)] \\ &= 0\end{aligned}$$

Thus it follows that

$$\begin{aligned}\mathbb{E}[(y - \hat{f}(x))^2] &= (f(x) - \mathbb{E}[\hat{f}(x)])^2 + \mathbb{E}[(\mathbb{E}[\hat{f}(x)] - \hat{f}(x))^2] + \sigma^2 \\ &= \text{Bias}^2 + \text{Variance} + \sigma^2\end{aligned}$$

3.2 Regularization [5pts]

1. Regularization is an important method used in machine learning to avoid overfitting the data. Recall that in Linear Regression, the loss function takes the form $L = \frac{1}{2}\sum_{i=1}^{n}(y_i - \omega^T \mathbf{x}_i)^2$.

   (a) (5pts) Consider adding $\frac{\lambda}{2}\|\omega\|^2$ to the loss function. Derive the gradient of the new loss function with respect to ω^k. Notice that λ controls the tradeoff between training error and weight penalty
   Solution: The new loss function is

   $$L = \frac{1}{2}\sum_{i=1}^{n}(y_i - \omega^T \mathbf{x}_i)^2 + \frac{\lambda}{2}\|w\|^2$$

   Taking the derivative with respect to ω^k:

   $$\begin{aligned}\frac{\partial L}{\partial \omega^k} &= \frac{1}{2}2\sum_{i=1}^{n}(y_i - \omega^T \mathbf{x}_i)(-x_i^k) + 2\frac{\lambda}{2}\omega^k \\ &= -\sum_{i=1}^{n}(y_i - \omega^T \mathbf{x}_i)\mathbf{x}_i^j + \lambda\omega^k\end{aligned}$$

   (b) (EXTRA CREDIT: 5 pts) Consider two update rules of gradient algorithms to solve the linear regression problem with regularization. You may assume $\alpha > 0$ and $0 < \lambda < 1$. For simplicity, assume $\omega^k \geq 0$:

   i. In each iteration, do:

   $$\omega^{k,(t+1)} = \omega^{k,(t)} + \alpha\sum_{i=1}^{n}(y_i - \omega^T \mathbf{x}_i)\mathbf{x}_i^k - \lambda\omega^{k,(t)}$$

   if $\omega^{k,(t+1)} < 0$, set

   $$\omega^{k,(t+1)} = 0$$

   ii. In each iteration, do:

   $$\omega^{k,(t+1)} = \omega^{k,(t)} + \alpha\sum_{i=1}^{n}(y_i - \omega^T \mathbf{x}_i)\mathbf{x}_i^k - \lambda\frac{\omega^{k,(t)}}{|\omega^{k,(t)}|}$$

   if $\omega^{k,(t+1)} < 0$, set

   $$\omega^{k,(t+1)} = 0$$

   (Notice that this is a simplified version of Lasso update rules)

Which of these two algorithms is more likely to give sparse ω, meaning some of the entries ω^k is 0.

Solution: Algorithm ii. Notice that the only difference in the two update rules is in regularization. While the first update rule penalize the weight on feature k by a fraction of its value, the second update rule penalize the weight by a constant λ. Imagine a feature ω^j that has no correlation with the target y. Ideally, ω^j should be zero. Given random initialization, ω_j will be given some positive value. If it is greater than 1, update rule i will decrease ω^j faster at the beginning. However, both update rules will eventually push ω^j below 1. When ω_j is a small number close to 0, then the first update rule will essentially decrease ω^j by a tiny amount, i.e. $\lambda\omega^{j,(t)}$ while the second update rule will decrease ω^j by a constant λ. Thus the first update rule will never drive ω^j to zero, but the second update rule will.

4 Programming logistic regression [40 pts]

In this part, you will be deriving update rules for logistic regression and program to train a logistic regression classifier. The dataset you will be using is MNIST handwritten digits. In 4.1, you will work out the maths behind the model; in 4.2 and 4.3, you will implement logistic regression and regularized logistic regression. The data and starter codes are located in the homework handout folder.

4.1 Logistic Regression [15pts]

Assume we have n data points $\{x^{(1)}, y^{(1)}\}, \ldots, \{x^{(n)}, y^{(n)}\}$, $x \in R^m$, $y \in \{0, 1\}$ that are sampled iid.

1. State the logistic function, $f(x; \omega)$ (no need for derivation, just write the answer).
   Solution:
   $f(x;\omega) = \dfrac{1}{1+\exp(-\omega^T x)}$

2. Derive the conditional likelihood, $L(\omega; x, y)$. You may keep your solution in terms of $f(x; \omega)$.
   Solution:
   Note that $p(y_i|x_i, \omega) = f(x_i;\omega^T)^{y_i}(1 - f(x_i;\omega^T))^{1-y_i}$
   $L(\omega) = \prod_i^n p(y_i|x_i;\omega)$
   $L(\omega) = \prod_i^n f(x_i;\omega^T)^{y_i}(1 - f(x_i;\omega^T))^{1-y_i}$

3. Derive the log conditional likelihood, $l(\omega; x, y)$. You may keep your solution in terms of $f(x; \omega)$.
   Solution:
   $l(\omega) = \log \prod_i^n f(x_i;\omega^T)^{y_i}(1 - f(x_i;\omega^T))^{1-y_i}$
   $l(\omega) = \sum_i^n y_i \log f(x_i;\omega^T) + (1 - y_i)\log(1 - f(x_i;\omega^T))$

4. Derive the derivative of the log likelihood with respect to ω^k. You may keep your solution in terms of $f(x; \omega)$.
   Solution:
   $\frac{\delta}{\delta\omega^k} l(\omega) = \frac{\delta}{\delta\omega^k} \log \sum_i^n y_i \log f(x_i;\omega^T) + (1 - y_i)\log(1 - f(x_i;\omega^T))$
   $\frac{\delta}{\delta\omega^k} l(\omega) = \sum_i^n \left(\dfrac{y_i}{f(x_i;\omega^T)} - \dfrac{1-y_i}{1-f(x_i;\omega^T)}\right) \frac{\delta}{\delta\omega^k} f(x_i;\omega^T)$
   $\frac{\delta}{\delta\omega^k} l(\omega) = \sum_i^n \left(\dfrac{y_i}{f(x_i;\omega^T)} - \dfrac{1-y_i}{1-f(x_i;\omega^T)}\right) f(x_i;\omega^T)(1 - f(x_i;\omega^T))x_i^k$
   $\frac{\delta}{\delta\omega^k} l(\omega) = \sum_i^n \Big(y_i - f(\omega^T x_i)\Big)x_i^k$

5. Put this together to show that the update rule for batch gradient descent, with step size α, of logistic regression is:

$$\omega_{new}^k = \omega^k + \alpha \sum_i^n \Big(y_i - f(\omega^T x_i)\Big) x_i^k \tag{3}$$

Solution:
The general form of stochastic gradient descent is
$\omega_{new}^k = \omega^k + \alpha \frac{\delta}{\delta \omega^k} J(\omega)$ where α is the step size. It follows from part (4) that
$\omega^{k-new} = \omega^k + \alpha \sum_i^n \Big(y_i - f(\omega^T x_i)\Big) x_i^k$

6. Recall in Logistic Regression, one way to define the objective is $\arg\min_\omega -l(\omega; x, y)$, where $l(\omega; x, y)$ is the log-likelihood. Now add the regularization term $\frac{\lambda}{2}\|\omega\|^2$ to the loss function. Derive the gradient of the new loss function with respect to ω^k.
Solution: The new loss function is

$$L = -l(\omega; x, y) + \frac{\lambda}{2}\|\omega\|^2$$

Taking the derivative with respect to ω^k, we have (using the result from 4.1.4):

$$\frac{\partial L}{\partial \omega^k} = -\sum_{i=1}^n (y_i - f(x_i; \omega))\mathbf{x}_i^k + \lambda \omega^k$$

4.2 Coding Logistic regression [15pts]

The goal of this section is to build a binary classifier to differentiate between numbers 4 and 7. Code outline is provided in lr.m. Please complete the steps listed below. You will be graded on testing accuracy on a separate dataset.
Note: you should load the data from `lr_train.mat` as training set and `lr_test.mat` as test set. Each of the two files contains two fields, X and y, where each column of X corresponds to an image vector, and each column of y corresponds to the label of that image vector (0 for digit 4 and 1 for digit 7).

1. run `visualize_data.m`: It is always a good practice to have some intuitive understanding about your dataset before you proceed. This script visualizes the dataset.

2. complete `standardize.m`: The features in a dataset may have various mean and variance. This may adversely impact our learning algorithm, particularly with regularization (you may think why?). Standardize features to mean 0 and variance 1 (approximately) in this function.

3. complete `sigmoid.m`: The function `s = sigmoid(a)` computes the sigmoid of a vector $\mathbf{a}$ and returns the value as a vector $\mathbf{s}$.

4. complete `fv_grad.m`: The function `[f,g] = fv_grad(w, X, y)` returns the objective function value, i.e. negative log-likelohood, as f and its gradient at $\mathbf{w}$ as g .

5. complete `check_grad.m`: Use finite difference test to check that your gradient implementation is correct

6. complete `lr_gd.m`: The function `w = lr_gd(w0, X, y)` updates the weight vector w and returns its value after convergence. You should implement gradient descent using the gradient function you wrote.

7. complete `lr_pred.m`: The function `pred = lr_pred(w, X)` should return a vector with predicted labels based on input X.

8. run `lr.m` to perform training and see the performance on training and test sets. Record your accuracy on both datasets. You may need it for the questions below. (You should achieve about 99% accuracy on training data and 98% accuracy on testing data)

9. Modify `fv_grad.m`: Now add L2 penalty to your objective function by slightly modifing your fv_grad.m file.

4.3 Analysis of results [10 pts]

The goal of this section is to encourage you to interpret your results.

1. Run `lr.m` again and report your training and test accuracy. Do you see much difference from unregularized LR? Is the gap between training test error wider or narrower? Is this what you expected? Please explain your findings.
   Solution: Not much difference. Either expected or unexpected is fine. Either wider or narrower is fine. Since the training data is sufficient and the model is not very complex, the unregularized logistic regression does not overfitting the data. Note that regularization has the effect of reducing model complexity, thus making it faster to converge.

2. Provide a visualization of 16 images that you've incorrectly labeled in the test set. You may use the function `display_network.m` that is provided to you.

   For instance, `display_network(X(:,10:20))` will display the images between column 10 to 20 of matrix X. The number of rows of X needs to be a perfect square - in our case, 784.
   Solution: Below are the figures we generated.

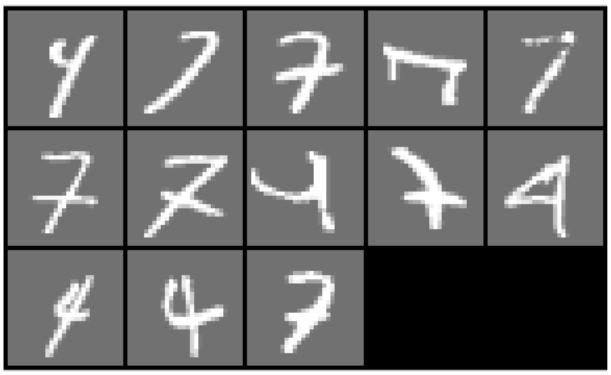

Figure 2: Visualization of errors using regularized LR

5 Programming kernel perceptron [20 pts]

Using the same underlying data, we will now implement the kernel perceptron algorithm for distinguishing between handwritten digits. Note, however, that for perceptrons, it is more natural to use $\{-1, 1\}$ as the label. Thus **PLEASE USE** the $\mathbf{perceptron\_train.mat}$ and $\mathbf{perceptron\_test.mat}$ as your data for this problem.

5.1 Perceptron [10pts]

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

- `Initialize the weight vector` $w \in R^d$
- `Iterate until convergence:`
  - `For` $i = 1 \ldots n$`:`
    - $\hat{y}_i = \varphi(wx_i)$
    - `If:` $\hat{y}_i \neq y_i$`;  Then:` $w = w + y_i x_i$

Where we use a slightly altered definition from Problem 2:

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

Note that this is an iterative algorithm and there are different ways to define "convergence". In this assignment we provide the convergence rule for you. It stops that algorithm when $\hat{y} = \hat{y}^{(prev)}$. Meaning that the algorithm is no longer changing the predicted values of $\hat{y}$.

1. Complete `perceptron_pred.m`: the function `pred = perceptron_pred(w, x)` takes in a weight vector w and a feature vector x, and outputs the predicted label based on $\varphi(z)$.

2. Complete `perceptron.m`: the function `w = perceptron(w0, X, Y)` runs the perceptron training algorithm taking in an initial weight vector $w0$, matrix of covariates X, and vector of labels Y. It outputs a learned weight vector w.

3. Run `perceptron_run.m` to perform training and see the performance on training and test sets. Record your accuracy on both datasets. For reference, we achieved a 97% test accuracy.

5.2 Kernel perceptron [10 pts]

Now we will implement the kernel perceptron algorithm. 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 $X1$ could be the data of a training set and $X2$ could be the data of ta testing set. Thus, $K \in R^{n \times m}$ has dimensions $n \times m$. Be careful about constructing this correctly!

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` $\alpha \in R^d$
- `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:` $\alpha_i = \alpha_i + 1$

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

1. Complete `polykernel.m`: the function `K = polykernel(X1,X2)` computes a kernel matrix of a 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:

   $$k(\boldsymbol{x_i}, \boldsymbol{x_l}) = (\boldsymbol{x_i}^T \boldsymbol{x_l} + 1)^3 \tag{4}$$

2. 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.

3. 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.

4. Run `kernel_perceptron_run.m` to perform training and see the performance on training and test sets. Record your accuracy on both datasets. For reference, we achieved a 98% test accuracy. Compare your results to the perceptron algorithm. Can you explain why they differ or are similar?

6 Submission Instructions

Below are the files you need to submit:

1. `check_grad.m`
2. `fv_grad.m`
3. `lr_gd.m`
4. `lr_pred.m`
5. `sigmoid.m`
6. `standardize.m`
7. `kernel_perceptron_pred.m` (Only if you complete extra credit)
8. `kernel_perceptron.m` (Only if you complete extra credit)
9. `perceptron_pred.m` (Only if you complete extra credit)
10. `perceptron.m` (Only if you complete extra credit)
11. `polykernel.m` (Only if you complete extra credit)

Please put these files, together with your writeup, into a folder called **hw3** and run the following command in the directory that contains the folder:

```
$ tar -cvf hw3.tar hw3
```

Then submit your tarfile.