# EECS 545 – Machine Learning - Homework #3

Due: 5:00PM 10/18/2018

Homework submission via Gradescope as usual.

1) **Linear Regression (5 pts).**

Download the file `bodyfat_data.mat` from Canvas. This contains variables `X` and `y` for a regression problem. The input variables correspond to abdomen circumference and hip circumference (in centimeters), and `y` corresponds to % body fat.

Use the first 150 examples for training, and the remainder for estimating the mean squared error.

Using regularized least squares regression with $\lambda = 10$, report your estimated parameters, test error (mean squared error), and the predicted response at the input $\boldsymbol{x} = [100\ 100]$ (viewing the feature vectors as row vectors).

You do not need to submit your code for this problem.

2) **Robust Regression (20 pts).**

In this problem you will compare least-squares and robust regression on a synthetic data set. We provide a code skeleton for you below, in Matlab and Python. The first part creates a data set for linear regression. Most of the data come from an upward sloping line plus noise, but a fraction (thought of as outliers) come from a downward sloping line.

Please recall that you are not allowed to use built in Matlab/Python commands that solve regression problems directly. It will be helpful to write a helper function to solve weighted least squares regression.

Here is the Matlab code:

```
function [] = main()
clear all; close all;

n = 200;
rng(0); % seed random number generator
x = rand(n,1);
z = zeros(n,1); k = n*0.4; rp = randperm(n); ...
    outlier_subset = rp(1:k); z(outlier_subset)=1; % outliers
y = (1-z).*(10*x + 5 + randn(n,1)) + z.*(20 - 20*x + 10*randn(n,1));

% plot data and true line
scatter(x,y,'b')
hold on
t = 0:0.01:1;
plot(t,10*t+5,'k')
```

```
% add your code for ordinary least squares below


plot(t, w_ols*t + b_ols, 'g--');

% add your code for the robust regression MM algorithm below


plot(t, w_rob*t + b_rob, 'r:');
legend('data','true line','least squares','robust')

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [w,b] = wls(x,y,c);
% a helper function to solve weighted least squares
```

Here is the Python code:

```
import numpy as np
import matplotlib.pyplot as plt
import random
import pylab as pl

n = 200
np.random.seed(0) #Seed the random number generator
x = np.random.rand(n,1)
z = np.zeros([n,1])
k = n*0.4
rp = np.random.permutation(n)
outlier_subset = rp[1:k]
z[outlier_subset] = 1 #outliers
y = (1-z)*(10*x + 5 + np.random.randn(n,1)) + z * (20 - 20*x +
10*np.random.randn(n,1))

#Plot data and true line

plt.scatter(x, y,label='data')

t = pl.frange(0,1,0.01)
plt.plot(t, 10*t+5, 'k-',label='true line')

# Add your code for ordinary least squares below


plt.plot(t, w_ols*t+b_ols, 'g--',label = 'least squares')

# Add your code for robust regression MM algorithm below


plt.plot(t, w_rob*t+b_rob, 'r:','robust')

legend = plt.legend(loc='upper right', shadow=True)
plt.show()
```

```
###############################################################################

def wls(x,y,c):

# helper function to solve weighted least squares
    #add your code here

    return w, b
```

**(a)** (5 pts) Argue that for general $\rho$ satisfying the lemma in the notes on robust regression, the majorize-minimize algorithm can be understood as "iteratively reweighted least squares." Explain how the algorithm achieves robustness by considering how weights are assigned to outliers vs. inliers in comparison with the least squares loss.

**(b)** (5 pts) Consider the robust loss

$$\rho(r) = \sqrt{1 + r^2} - 1.$$

Implement the MM algorithm, and report the parameters of the linear function you estimated. For comparison, also report the parameters of ordinary least squares on this data.

**(c)** (5 pts) Generate a single plot that shows the data, the true line, the OLS estimate, and the robust estimate. Create a legend and use different line styles.

**(d)** (5 pts) Submit your Matlab or Python code.

3) **Logistic Regression as ERM (5 pts).**

Consider training data $(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_n, y_n)$ for binary classification, and assume $y_i \in \{-1, 1\}$.

Show that if $L(y, t) = \log(1 + \exp(-yt))$, then

$$\frac{1}{n} \sum_{i=1}^{n} L(y_i, \mathbf{w}^T \boldsymbol{x}_i + b)$$

is proportional to the negative log likelihood for logistic regression. Therefore ERM with the logistic loss is equivalent to the maximum likelihood approach to logistic regression.

*To be clear:* In the above expression, $y$ is assumed to be $-1$ or 1. In the logistic regression notes we had $y = 0$ or 1. So all you need to do is rewrite the negative log-likelihood for LR using the $\pm 1$ label convention, and simply that formula until it looks like the formula above.

4) **Subgradient methods for the optimal soft margin hyperplane (25 pts).**

In this problem you will implement the subgradient and stochastic subgradient methods for minimizing the convex but nondifferentiable function

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

where $L(y, t) = \max\{0, 1 - yt\}$ is the hinge loss. As we saw in class, this corresponds to the optimal soft margin hyperplane.

**(a)** (5 pts) Determine $J_i(\mathbf{w}, b)$ such that

$$J(\mathbf{w}, b) = \sum_{i=1}^{n} J_i(\mathbf{w}, b).$$

Determine a subgradient $\mathbf{u}_i$ of each $J_i$ with respect to the variable $\boldsymbol{\theta} = [b\ \mathbf{w}^T]^T$. A subgradient of $J$ is then $\sum_i \mathbf{u}_i$.

*Note:* Recall that if $f(\mathbf{z}) = g(h(\mathbf{z}))$ where $g : \mathbb{R} \to \mathbb{R}$ and $h : \mathbb{R}^n \to \mathbb{R}$, and both $g$ and $h$ are differentiable, then

$$\nabla f(\mathbf{z}) = \nabla h(\mathbf{z}) \cdot g'(h(\mathbf{z})).$$

If $g$ is convex and $h$ is differentiable, the same formula gives a subgradient of $f$ at $\mathbf{z}$, where $g'(h(\mathbf{z}))$ is replaced by a subgradient of $g$ at $h(\mathbf{z})$.

Download the file `nuclear.mat` from Canvas. The variables `x` and `y` contain training data for a binary classification problem. The variables correspond to the total energy and tail energy of waveforms produced by a nuclear particle detector. The classes correspond to neutrons and gamma rays. Neutrons have a slightly larger tail energy for the same total energy relative to gamma rays, which allows the two particle types to be distinguished. This is a somewhat large data set ($n = 20,000$), and subgradient methods are appropriate given their scalability.

**(b)** (6 pts) Implement the subgradient method for minimizing $J$ and apply it to the nuclear data. Submit two figures: One showing the data and the learned line, the other showing $J$ as a function of iteration number. Also report the estimated hyperplane parameters and the minimum achieved value of the objective function.

*Comments:*

- Please execute the line

  ```
  rng(0); \% in Matlab
  ```

  or

  ```
  np.random.seed(0) \# in Python
  ```

  at the beginning of your code to seed the random number generator.
- Use $\lambda = 0.001$. Since this is a linear problem in a low dimension, we don't need much regularization.
- Use a step-size of $\alpha_j = 100/j$, where $j$ is the iteration number.
- To compute the subgradient of $J$, write a subroutine to find the subgradient of $J_i$, and then sum those results.
- Since the objective will not be monotone decreasing, determining a good stopping rule can be tricky. Just look at the graph of the objective function and "eyeball it" to decide when the algorithm has converged.
- Debugging goes faster if you just look at a subsample of the data. To debug in Matlab, the command `keyboard` is very helpful. Just type `help keyboard` and also look up related commands. For something analagous in Python, try `http://stackoverflow.com/questions/13432717/enter-interactive-mode-in-python`.

**(c)** (6 pts) Now implement the stochastic subgradient method, which is like the subgradient method, except that your step direction is a subgradient of a random $J_i$, not $J$. Be sure to cycle through all data points before starting a new loop through the data. Report/hand in the same items as for part **(b)**.

*More comments:*

- Use the same $\lambda$, stopping strategy, and $\alpha_j$ as in part **(b)**. Here $j$ indexes the number of times you have cycled (randomly) through the data.
- Your plot of $J$ versus iteration number will have roughly $n$ times as many points as in part **(b)** since you have $n$ updates for every one update of the full subgradient method.
- To generate a random permutation use

  ```
  randperm
  ```

  in Matlab, or

  ```
  import numpy as np
  np.random.permutation
  ```

  in Python.
- Please reseed the random number generator as decribed previously.

**(d)** (3 pts) Comment on the (empirical) rate of convergence of the stochastic subgradient method relative to the subgradient method. Explain your finding.

**(e)** (5 pts) Submit your code.

5) **ERM Losses (10 pts).**

An alternate way to extend the max-margin hyperplane to nonseparable data is to solve the following quadratic program:

$$\begin{aligned} \min_{\mathbf{w},b,\boldsymbol{\xi}} \quad & \frac{1}{2}\|\mathbf{w}\|^2 + \frac{C}{n}\sum_{i=1}^{n}\xi_i^2 \\ \text{s.t.} \quad & y_i(\mathbf{w}^T\boldsymbol{x}_i + b) \geq 1 - \xi_i \quad \forall i \\ & \xi_i \geq 0 \quad \forall i \end{aligned}$$

The only difference with respect to the OSM hyperplane is that we are now squaring the slack variables. This assigns a stronger penalty to data points that violate the margin.

**(a)** (4 pts) Which loss is associated with the above quadratic program? In other words, show that learning a hyperplane by the above QP is equivalent to regularized ERM with a certain loss.

**(b)** (4 pts) Argue that the second set of constraints can be dropped without changing the solution.

**(c)** (2 pts) Identify an advantage and a disadvantage of this loss compared to the hinge loss.