CS489/698: Introduction to Machine Learning

Homework 1

Due: 11:59 pm, September 26, 2017, submit on LEARN.

Include your name, student number and session!

Submit your writeup in pdf and all source code in a zip file (with proper documentation). Write a script for each programming exercise so that the TAs can easily run and verify your results. Make sure your code runs!
[Text in square brackets are hints that can be ignored.]

Exercise 1: Perceptron and Winnow (50 pts)

Convention: All algebraic operations, when applied to a vector or matrix, are understood to be element-wise (unless otherwise stated).

Algorithm 1: The perceptron algorithm.

Input: $X \in \mathbb{R}^{n\times d}$, $\mathbf{y} \in \{-1,1\}^n$, $\mathbf{w} = \mathbf{0}_d$, $b = 0$, $\mathsf{max\_pass} \in \mathbb{N}$
Output: $\mathbf{w}, b, mistake$

for $t = 1, 2, \ldots, \mathsf{max\_pass}$ **do**
  $mistake(t) \leftarrow 0$
  for $i = 1, 2, \ldots, n$ **do**
    if $y_i(\langle \mathbf{x}_i, \mathbf{w}\rangle + b) \le 0$ **then**
      $\mathbf{w} \leftarrow \mathbf{w} + y_i\mathbf{x}_i$ // $\mathbf{x}_i$ is the i-th row of X
      $b \leftarrow b + y_i$
      $mistake(t) \leftarrow mistake(t) + 1$

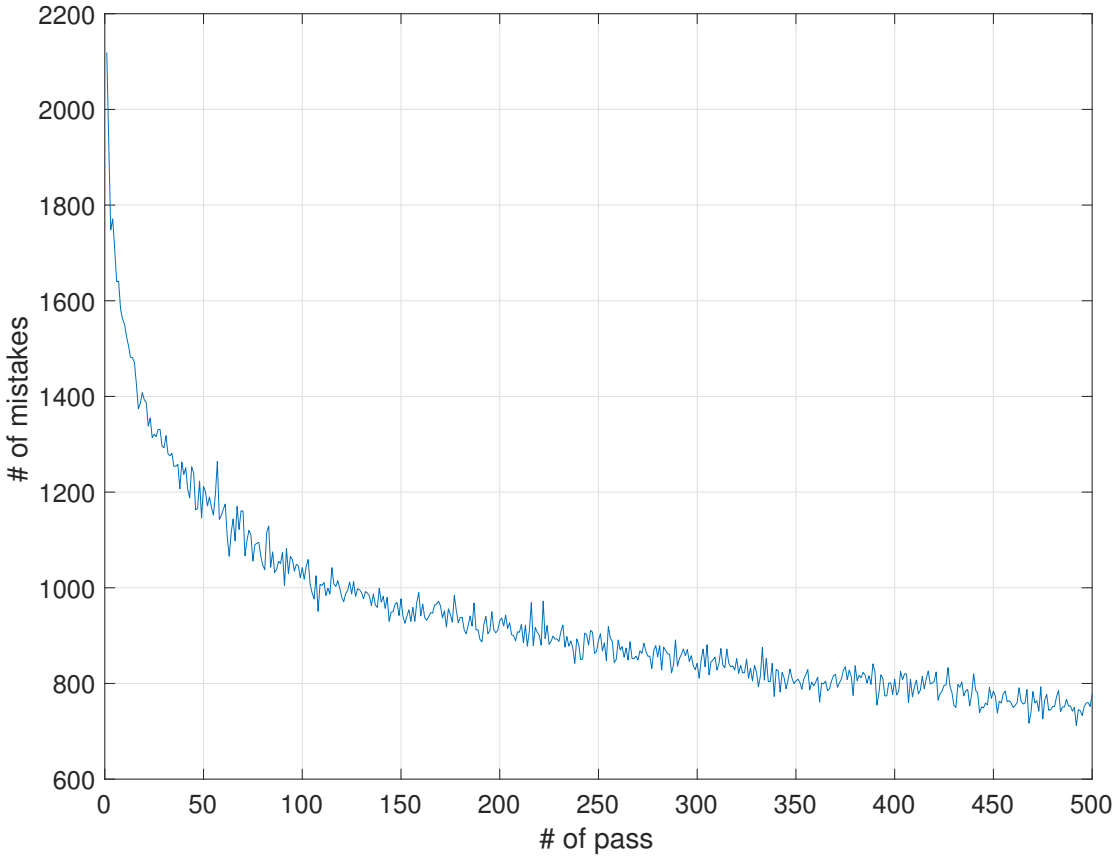
Algorithm 2: The winnow algorithm.

Input: $X \in \mathbb{R}^{n\times d}$, $\mathbf{y} \in \{-1,1\}^n$, $\mathbf{w} = \frac{1}{d+1}\mathbf{1}_d$, $b = \frac{1}{d+1}$, step size $\eta > 0$, $\mathsf{max\_pass} \in \mathbb{N}$
Output: $\mathbf{w}, b, mistake$

for $t = 1, 2, \ldots, \mathsf{max\_pass}$ **do**
  $mistake(t) \leftarrow 0$
  for $i = 1, 2, \ldots, n$ **do**
    if $y_i(\langle \mathbf{x}_i, \mathbf{w}\rangle + b) \le 0$ **then**
      $\mathbf{w} \leftarrow \mathbf{w} \odot \exp(\eta y_i \mathbf{x}_i)$ // $\odot$ is the element-wise product
      $b \leftarrow b \exp(\eta y_i)$ // element-wise exponential
      $s \leftarrow b + \sum_{i=1}^d w_i$ // normalize
      $\mathbf{w} \leftarrow \mathbf{w}/s$
      $b \leftarrow b/s$
      $mistake(t) \leftarrow mistake(t) + 1$

1. (10 pts) Implement the perceptron in Algorithm 1. Your implementation should take input as $X = [\mathbf{x}_1, \ldots, \mathbf{x}_n]^\top \in \mathbb{R}^{n\times d}$, $\mathbf{y} \in \{-1,1\}^n$, an initialization of the hyperplane parameters $\mathbf{w} \in \mathbb{R}^d$ and $b \in \mathbb{R}$, and the maximum number of passes of the training set [suggested $\mathsf{max\_pass} = 500$]. Run your perceptron algorithm on the spambase dataset (available on course website), and plot the number of mistakes (y-axis) w.r.t. the number of passes (x-axis).

   Ans: The number of mistakes should drop as the number of passes of the training data increases.



2. (5 pts) Run your implementation on spambase again, but this time moving the updates (line 5 and line 6 in Algorithm 1) outside of the IF-clause, i.e., we update the weight vectors even when perceptron predicts correctly. We count the number of mistakes as before, i.e., only when perceptron makes a mistake. Plot again the number of mistakes w.r.t. the number of passes.

   Ans: The number of mistakes stays as a constant 2788 (the number of negative instances).

3. (5 pts) Prove the following claim: If there exist $\mathbf{w}^*$ and b^* such that

$$\begin{cases} \langle \mathbf{x}_i, \mathbf{w}^* \rangle + b^* \geq 0, & \text{if } y_i = 1 \\ \langle \mathbf{x}_i, \mathbf{w}^* \rangle + b^* < 0, & \text{if } y_i = -1 \end{cases}, \tag{1}$$

   then there exist $\mathbf{w}^\star$ and $b^\star$ such that

$$\begin{cases} \langle \mathbf{x}_i, \mathbf{w}^\star \rangle + b^\star > 0, & \text{if } y_i = 1 \\ \langle \mathbf{x}_i, \mathbf{w}^\star \rangle + b^\star < 0, & \text{if } y_i = -1 \end{cases}. \tag{2}$$

   This confirms again that we can be indifferent about the value of sign(0).

   Ans: Since we only have a finite training set, let

$$\epsilon = \min_{i:y_i=-1} -\langle \mathbf{x}_i, \mathbf{w}^* \rangle.$$

Then, from our assumption (1) we know $b^* < \epsilon$. Let $\alpha = (\epsilon - b^*)/2 > 0$. We claim that $\mathbf{w}^\star = \mathbf{w}^*$, $b^\star = b^* + \alpha$ satisfy (2). Indeed,

$$\begin{cases} \langle \mathbf{x}_i, \mathbf{w}^\star \rangle + b^\star = \langle \mathbf{x}_i, \mathbf{w}^* \rangle + b^* + \alpha \geq 0 + \alpha = \alpha > 0, & \text{if } y_i = 1 \\ \langle \mathbf{x}_i, \mathbf{w}^\star \rangle + b^\star = \langle \mathbf{x}_i, \mathbf{w}^* \rangle + b^* + \alpha < b^* - \epsilon + \alpha = -\alpha < 0, & \text{if } y_i = -1 \end{cases}.$$

4. (10 pts) Implement winnow (Algorithm 2), the multiplicative version of perceptron. Tune and report a few step sizes η and plot the number of mistakes winnow makes on the spambase dataset (w.r.t. the number of passes for each step size η you tried). Explain the effect of the step size. [A good step size should be inversely proportional to the maximum absolute value in X. You may want to normalize X so that its maximum absolute value is, say, 1.]

   Ans: Winnow will mistake all (2788) negative instances, since both X and $\mathbf{w}$ and b are nonnegative. The step size has no effect here.

5. (10 pts) It is known that if there exist nonnegative $\mathbf{w}^\star$ and $b^\star$ such that $y_i(\langle \mathbf{x}_i, \mathbf{w}^\star \rangle + b^\star) > 0$ for all i, then winnow converges after at most $O(\frac{2\log(d+1)}{\gamma^2})$ mistakes, where

$$\gamma = \max_{\binom{\mathbf{w}}{b} \geq \mathbf{0}, b + \sum_i w_i = 1} \min_i \frac{y_i(\langle \mathbf{x}_i, \mathbf{w} \rangle + b)}{\max_i \max_j \max\{|X_{ij}|, 1\}}.$$

   Recall that in perceptron, there is no nonnegativity assumption. Find a simple transformation of the data $(X, \mathbf{y})$ such that if there exist $\mathbf{w}^*$ and b^* so that $y_i(\langle \mathbf{x}_i, \mathbf{w}^* \rangle + b^*) > 0$ for all i, then there exist nonnegative $\mathbf{w}^* \geq \mathbf{0}$ and $b^* \geq 0$ on the transformed data that separate the two classes. Thus, the additional nonnegativity assumption in winnow is really not a big deal. Plot the number of mistakes that winnow makes on the transformed spambase w.r.t. the number of passes (with a similar step size as in the previous exercise). [Each real number w can be written as the difference of two nonnegative numbers $w_+ := \max\{w, 0\}$ and $w_- := \max\{-w, 0\}$. Consider duplicating each point $\mathbf{x}$ with some version of itself.]

   Ans: As hinted, any real vector $\mathbf{w}$ can be written as the difference of two nonnegative real vectors:

$$\mathbf{w} = \mathbf{w}_+ - \mathbf{w}_-, \text{ where } \mathbf{w}_+ = \max\{\mathbf{w}, \mathbf{0}\}, \ \mathbf{w}_- = \max\{-\mathbf{w}, \mathbf{0}\}.$$

   As a result,

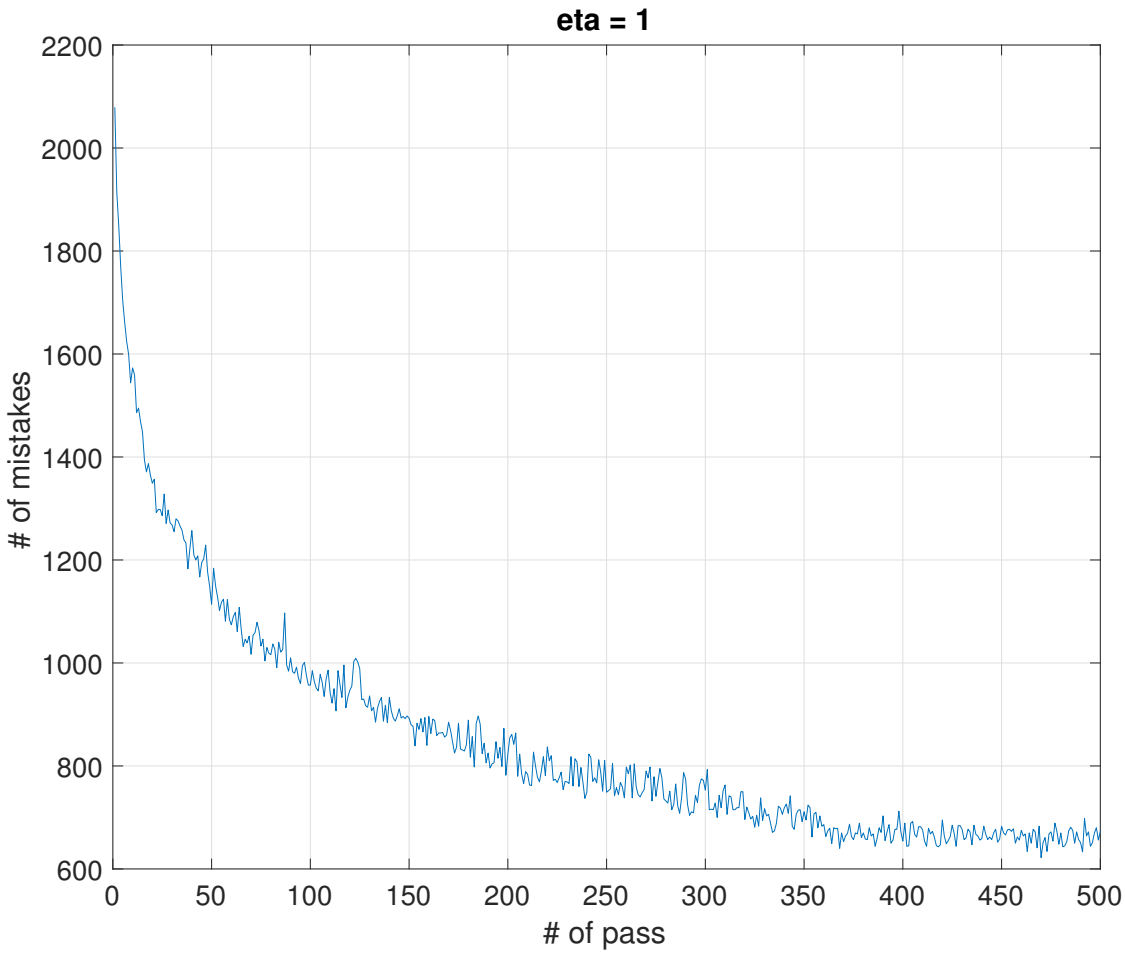
$$\langle \mathbf{x}, \mathbf{w}^\star \rangle + b^\star = \langle \mathbf{x}, \mathbf{w}_+^\star - \mathbf{w}_-^\star \rangle + b_+^\star - b_-^\star = \left\langle \begin{bmatrix} \mathbf{x} \\ 1 \\ -\mathbf{x} \\ -1 \end{bmatrix}, \begin{bmatrix} \mathbf{w}_+^\star \\ b_+^\star \\ \mathbf{w}_-^\star \\ b_-^\star \end{bmatrix} \right\rangle.$$

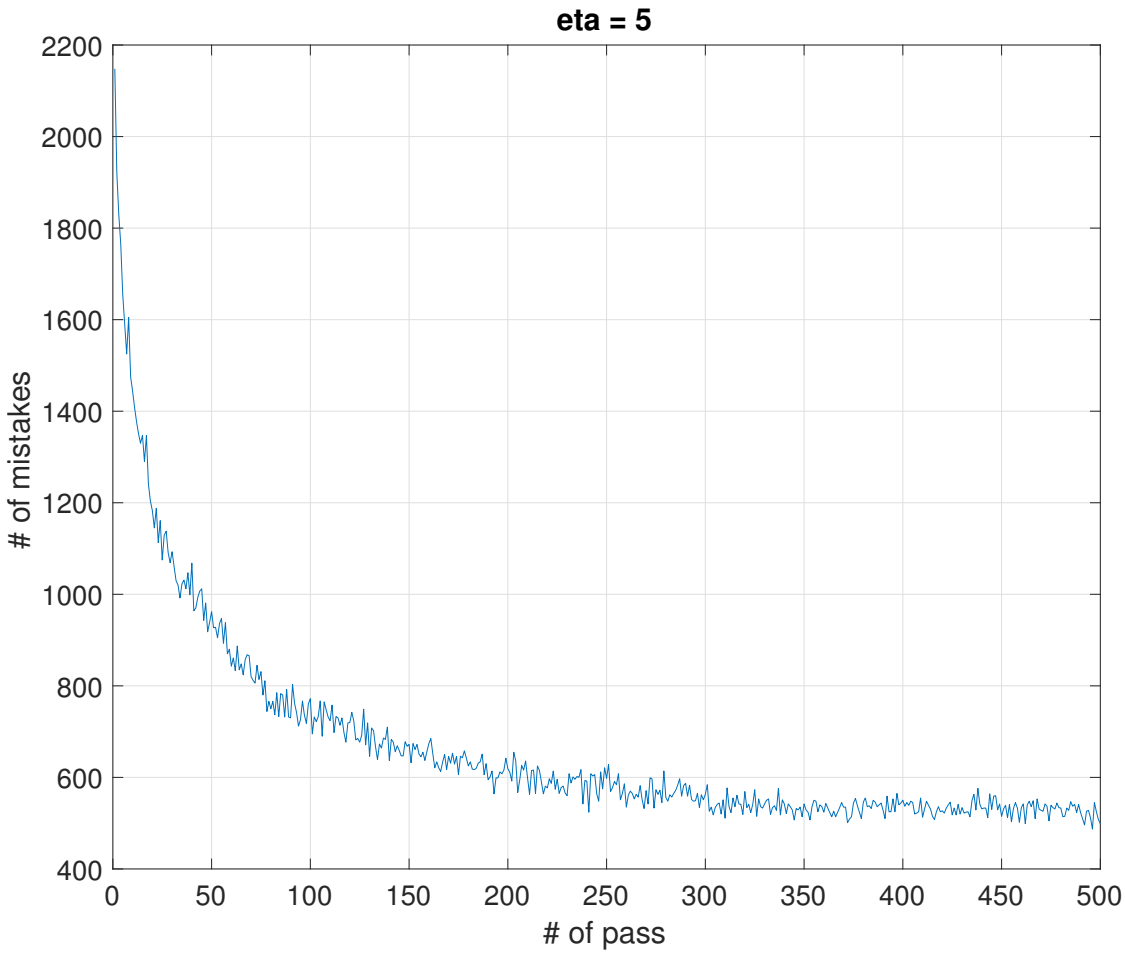
   Therefore, if we define

$$\tilde{\mathbf{x}} = \begin{bmatrix} \mathbf{x} \\ 1 \\ -\mathbf{x} \\ -1 \end{bmatrix}, \ \mathbf{w}^* = \begin{bmatrix} \mathbf{w}_+^\star \\ b_+^\star \\ \mathbf{w}_-^\star \\ b_-^\star \end{bmatrix}, \ b^* = 0,$$

   then $\mathbf{w}^*$ and b^* are nonnegative, and they separate the two classes $(\tilde{\mathbf{x}}, y)$.

   In our implementation of winnow, we normalize the data so that $\|X\|_\infty = 1$.


eta = 1
# of mistakes
# of pass
2200
2000
1800
1600
1400
1200
1000
800
600
0
50
100
150
200
250
300
350
400
450
500



6. (10 pts) Compare what you learned about perceptron with the above result of winnow. What are the respective pros and cons of the two algorithms? [Recall that the margin γ in perceptron can be written as: $\gamma = \max_{b^2+\sum_i w_i^2=1} \min_i \frac{y_i(\langle \mathbf{x}_i, \mathbf{w}\rangle + b)}{\max_i \sqrt{1+\sum_j |X_{ij}|^2}}$, with which the perceptron converges after at most $O(1/\gamma^2)$ mistakes.] Modify the spambase dataset by adding 100 irrelevant features (say, random numbers from the uniform distribution on $[-1, 1]$) [rand in Matlab or Julia, and numpy.random.uniform in python] and run both perceptron and winnow (you may want to use the variant in Ex 1.5). Plot the number of mistakes that each algorithm makes this time (again w.r.t. the number of passes).

   Ans: The number of mistakes made by perceptron and winnow are in general not comparable, because they depend on different definitions of the margin γ. First, both algorithms are online, and they make fewer mistakes if the data is "more" separated. However, we note that the preceptron bound does not have any explicit dependence on the dimension while the winnow bound scales with $\log(d)$. To apply winnow, we usually need to apply the "balancing" trick in Ex 1.5, hence doubling the dimension of the problem, which is another slight disadvantage. On the other hand, if the true separating hyperplane depends only on a few number of features, then the winnow bound is much more appealing (because we are taking the maximum instead of the ℓ_2 norm of all features.)

Exercise 2: Linear Regression and Regularization (50 pts)

Convention: The Matlab notation $X_{:j}$ means the j-th column of X while $X_{i:}$ means the i-th row of X. We

use argmin to denote the set of minimizers of a minimization problem.

Algorithm 3: Alternating minimization for Lasso.

Input: $X \in \mathbb{R}^{n\times d}$, $\mathbf{y} \in \mathbb{R}^n$, $\mathbf{w} = \mathbf{0}_d$, $\lambda \geq 0$
Output: $\mathbf{w}$
1 **repeat**
2 **for** $j = 1, \ldots, d$ **do**
3 $w_j \leftarrow \operatorname{argmin}_{z\in\mathbb{R}} \frac{1}{2}\|X_{:j}z + \sum_{k\neq j} X_{:k}w_k - \mathbf{y}\|_2^2 + \lambda|z|$ `// fix all w but optimize` w_j `only`
4 **until** *convergence*

1. (10 pts) Implement ridge regression that we discussed in class:

$$\min_{\mathbf{w}\in\mathbb{R}^d} \frac{1}{2}\|X\mathbf{w} - \mathbf{y}\|_2^2 + \lambda\|\mathbf{w}\|_2^2. \tag{3}$$

   Your implementation should take input as $X \in \mathbb{R}^{n\times d}$, $\mathbf{y} \in \mathbb{R}^n$, $\lambda \geq 0$, and maybe an initializer for $\mathbf{w} \in \mathbb{R}^d$. [To solve a linear system $A\mathbf{x} = \mathbf{b}$, use $A\backslash\mathbf{b}$ in Matlab or Julia, and numpy.linalg.solve in python.] Test your algorithm on the Boston housing dataset (to predict the median house price, i.e., y). Train and test splits are provided on course website. Find the best λ^* in the range [0, 100] with increment 10 using 10-fold cross validation. Report the mean square error on the training set ($\frac{1}{n}\|X\mathbf{w} - \mathbf{y}\|_2^2$), validation set (averaged over 10-fold) and test set [different normalization constant n] for each candidate λ. Report the percentage of nonzeros in $\mathbf{w}$ (for each λ).

   Ans: Simply solve the linear system $(X^\top X + \lambda I)\mathbf{w} = X^\top \mathbf{y}$.

2. (10 pts) Randomly choose a sample pair $(\mathbf{x}, y)$ from your training set. Multiply $\mathbf{x}$ by 10^6 and/or y by 10^3 and run ridge regression again (with the same cross-validation procedure to choose λ as above). Report the mean square error on the training set ($\frac{1}{n}\|X\mathbf{w} - \mathbf{y}\|_2^2$), validation set (averaged over 10-fold) and test set [different normalization constant n] for each candidate λ.

   Ans: Here the errors are supposedly much larger, due to the one and only extreme outlier.

3. (10 pts) Add 1000 irrelevant columns to X (both training and test splits), with each entry an *iid* sample from the standard normal distribution (randn in Matlab or Julia, and numpy.random.standard_normal in python). Run ridge regression again (with the same cross-validation procedure to choose λ as above). Report the mean square error on the training set ($\frac{1}{n}\|X\mathbf{w} - \mathbf{y}\|_2^2$), validation set (averaged over 10-fold) and test set [different normalization constant n] for each candidate λ. Report the percentage of nonzeros in the *last 1000 entries* in $\mathbf{w}$ (i.e., weights corresponding to the added irrelevant features).

   Ans: Most (and perhaps all) entries of the weight vector should be nonzero.

4. (10 pts) The Lasso replaces the (squared) 2-norm penalty in ridge regression with the 1-norm:

$$\min_{\mathbf{w}\in\mathbb{R}^d} \frac{1}{2}\|X\mathbf{w} - \mathbf{y}\|_2^2 + \lambda\|\mathbf{w}\|_1. \tag{4}$$

   Implement the alternating minimization algorithm for Lasso (Algorithm 3). You might find the following fact useful:

$$\operatorname{sign}(w) \cdot \max\{0, |w| - \lambda\} = \operatorname*{argmin}_{z\in\mathbb{R}} \frac{1}{2}(z - w)^2 + \lambda|z|, \tag{5}$$

   which is known as the soft-thresholding operator. You should try to perform step 3 of Algorithm 3 in $O(n)$ time and space. Stop the algorithm when the change of $\mathbf{w}$ drops below some tolerance tol, say 10^{-3}. [Any idea to make your implementation even more efficient?] Report the mean square error of Lasso on the training set ($\frac{1}{n}\|X\mathbf{w} - \mathbf{y}\|_2^2$), validation set (averaged over 10-fold) and test set [different normalization constant n] for each candidate λ. Report the percentage of nonzeros in $\mathbf{w}$ (for each λ). [A rule of thumb here is that $\lambda = O(\sqrt{\log(d)/n})$.]

Ans: Here the key is to maintain the residual vector explicitly. We start with $\mathbf{r} = X\mathbf{w} - \mathbf{y} = -\mathbf{y}$ (because we initialize with $\mathbf{w} = \mathbf{0}$). In each iteration we do the following:

$$\mathbf{r} \leftarrow \mathbf{r} - X_{:j} w_j \tag{6}$$

$$w_j \leftarrow \operatorname*{argmin}_z \tfrac{1}{2}\|X_{:j}\|_2^2 z^2 + \langle X_{:j}, \mathbf{r}\rangle z + \lambda|z| \tag{7}$$

$$= -\operatorname{sign}(\langle X_{:j}, \mathbf{r}\rangle) \max\{0, |\langle X_{:j}, \mathbf{r}\rangle| - \lambda\}/\|X_{:j}\|_2^2 \tag{8}$$

$$\mathbf{r} \leftarrow \mathbf{r} + X_{:j} w_j \tag{9}$$

[A good practice is to randomly perturb all indices and then go through the indices one by one sequentially.]

Clearly, the above iteration costs $O(n)$. We can further improve the efficiency by pre-compute $\|X_{:j}\|_2$ for all j.

5. (10 pts) Run Lasso on the housing dataset that you modified in Ex2.3, with λ cross-validated as before. Report the mean square error on the training set ($\frac{1}{n}\|X\mathbf{w} - \mathbf{y}\|_2^2$), validation set (averaged over 10-fold) and test set [different normalization constant n] for each candidate λ. Report the percentage of nonzeros in the *last 1000 entries* in $\mathbf{w}$ (i.e., weights corresponding to the added irrelevant features). Comparing with your results in Ex2.3, what can you conclude? [Mean square error, time complexity, sparsity, etc.]

   Ans: Lasso should yield a much sparser weight, hopefully with many (or even most) of the last 1000 entries being 0.