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).

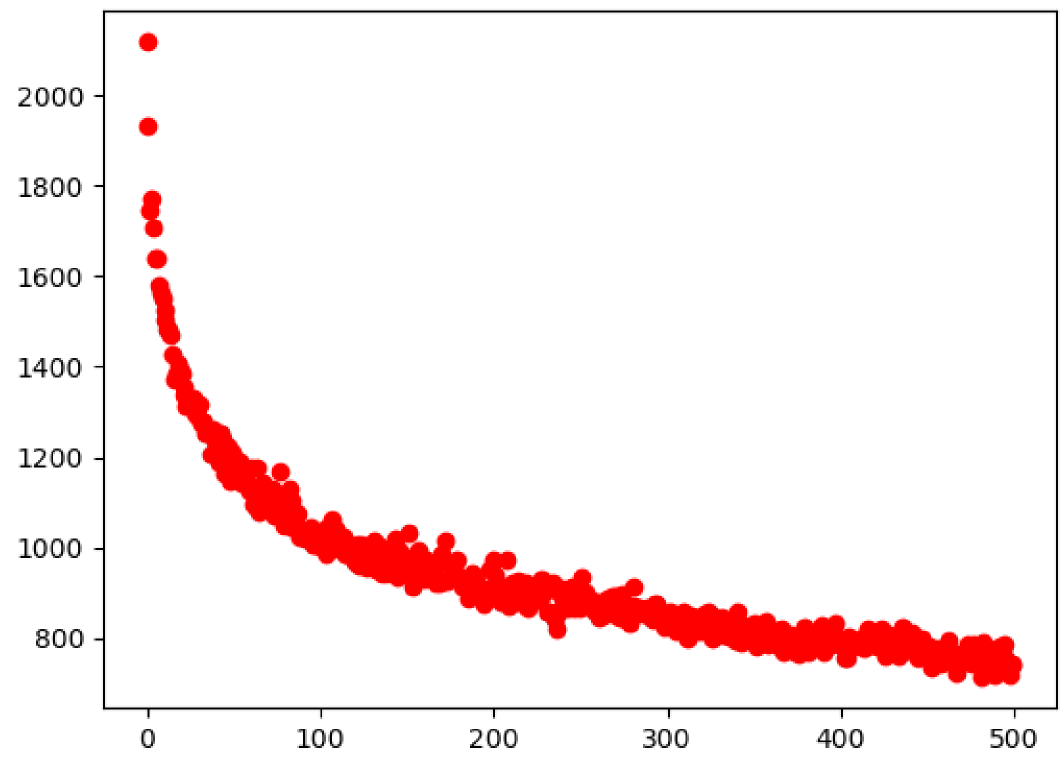
Algorithm 1: The perceptron algorithm.

```
Input: $X \in \mathbb{R}^{n\times d}$, $\mathbf{y} \in \{-1,1\}^n$, $\mathbf{w} = \mathbf{0}_d$, $b = 0$, max_pass $\in \mathbb{N}$
Output: $\mathbf{w}, b, mistake$
for $t = 1, 2, \ldots,$ 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 $X$
            $b \leftarrow b + y_i$
            $mistake(t) \leftarrow mistake(t) + 1$
```

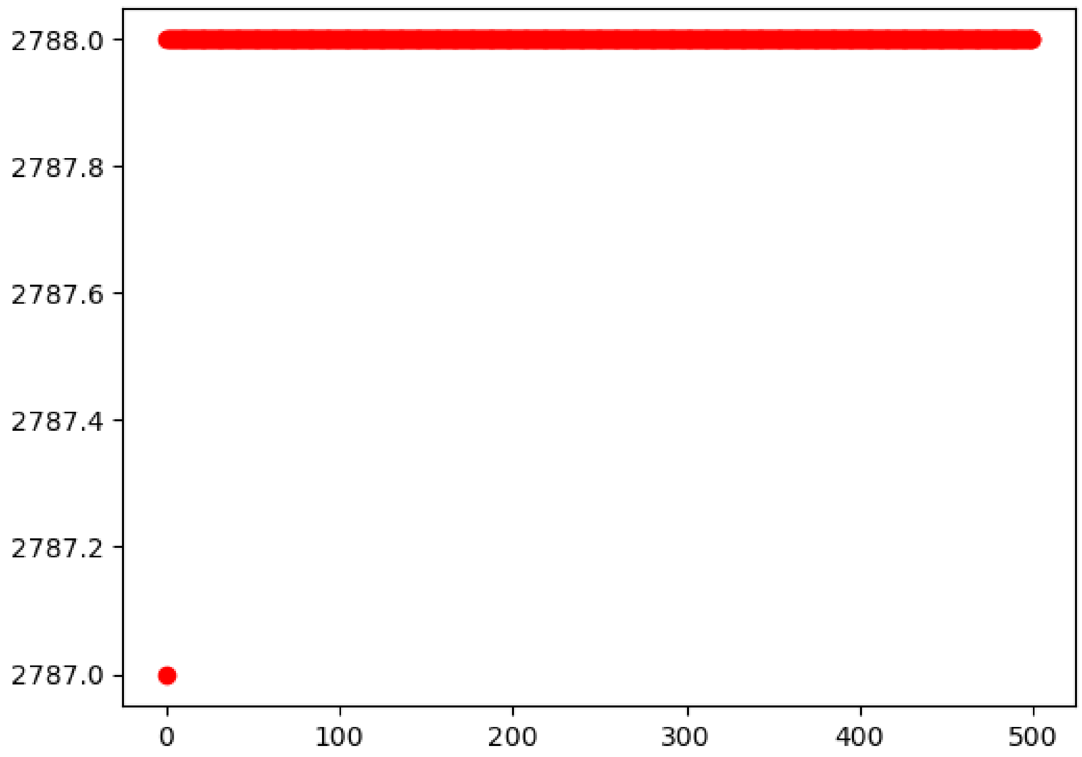
Algorithm 2: The winnow algorithm.

```
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 $\eta > 0$, max_pass $\in \mathbb{N}$
Output: $\mathbf{w}, b, mistake$
for $t = 1, 2, \ldots,$ 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 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).



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.



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

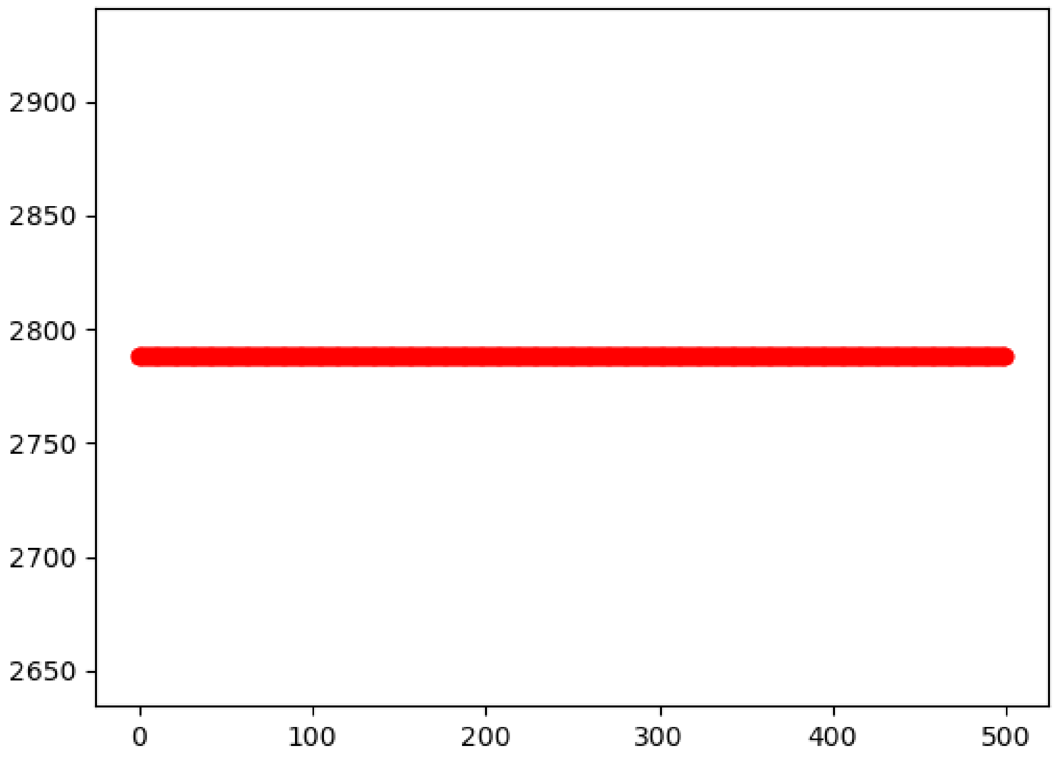
$$\begin{cases} \langle \mathbf{x}_i, \mathbf{w}^\star \rangle + b^\star > 0, & \text{if } y_i = 1 \\ \langle \mathbf{x}_i, \mathbf{w}^\star \rangle + b^\star < 0, & \text{if } y_i = -1 \end{cases}. \tag{2}$$

Answer: If that exists $\mathbf{w}^*$ and b^* such that such that $\langle \mathbf{x}_i, \mathbf{w}^* \rangle + b^* < 0$, if $y_i = -1$ then there $\exists\ a^* > 0$ such that $\langle \mathbf{x}_i, \mathbf{w}^* \rangle + b^* + a^* < 0$, if $y_i = -1$
since $a^* > 0$ and when $\langle \mathbf{x}_i, \mathbf{w}^* \rangle + b^* \geq 0$ we could have $\langle \mathbf{x}_i, \mathbf{w}^* \rangle + b^* + a^* > 0$ So by letting $b^* = b^* + a^*$ we could have there exist $\mathbf{w}^\star$ and $b^\star$ such that

$$\begin{cases} \langle \mathbf{x}_i, \mathbf{w}^\star \rangle + b^\star > 0, & \text{if } y_i = 1 \\ \langle \mathbf{x}_i, \mathbf{w}^\star \rangle + b^\star < 0, & \text{if } y_i = -1 \end{cases}. \tag{3}$$

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

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.]



Answer: The step size is irrelevant to the result, since all training data we have in spambasex.csv are all positive, the weight vector w and bias b starts with all non-negative elements $(1/(d+1))$. And exp(nyx) and exp(ny) are all always positive. The weight vector w and bias b will never be negative, which means $(\langle \mathbf{x}_i, \mathbf{w}\rangle + b)$ will always be non-negative. Then when $\mathbf{y}_i$ is negative, the $y_i(\langle \mathbf{x}_i, \mathbf{w}\rangle + b) \leq 0$ and this will be consider as a mistake. In another word, all negative result will be consider as a mistake based on given winnow algorithm (number of mistakes = number of negative $\mathbf{y}_i$) regardless of step size. As you can see the plot number are all (x, 2788) and the number of negative $\mathbf{y}_i$ is 2788.

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.]

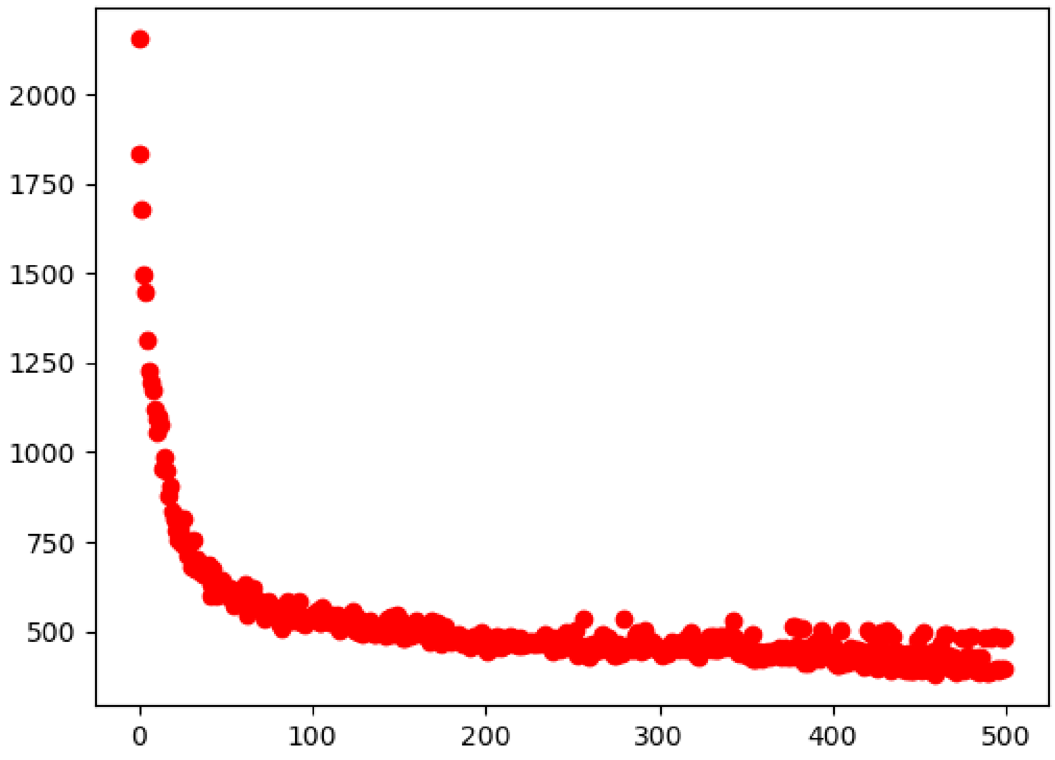
Answer:

Transformation: Data set $(X, \mathbf{y}) -> ((X, 1, -X, -1), \mathbf{y})$ eg(for each row ($\mathbf{x}_1$,$\mathbf{x}_2$,$\mathbf{x}_3$,...) link a row which have all reversed corresponding element to it. $->$ ($\mathbf{x}_1$,$\mathbf{x}_2$,$\mathbf{x}_3$,...,1,$-\mathbf{x}_1$,$-\mathbf{x}_2$,$-\mathbf{x}_3$,...,-1)
Proof: There exist $\mathbf{w}^*$ and b^* so that $y_i(\langle \mathbf{x}_i, \mathbf{w}^*\rangle + b^*) > 0$ for all i,
Let data set $(X, \mathbf{y}) = ((X, 1, -X, -1), \mathbf{y})$

First, link $\mathbf{b}^*$ to the end of $\mathbf{w}^*$
Extend $\mathbf{w}^*$ to 2(d+1) column $\mathbf{w}^{\star\star}$ and if $\mathbf{w}^*i >= 0$ then $\mathbf{w}^{\star\star}i = \mathbf{w}^*i$ and $\mathbf{w}^{**}(i+d+1) = 0$ (including the $\mathbf{b}^*$ we just added) by doing: if $b >= 0$ then $\mathbf{w}^{\star\star}d = d$ and $\mathbf{w}^{**}(d+d+1) = 0$
if $\mathbf{w}^{**}i < 0$ then $\mathbf{w}^{\star\star}i = 0$ and $\mathbf{w}^{\star\star}(i+d+1) = -\mathbf{w}^*i$ and same for d
By doing so we could have a weight vector $\mathbf{w}^{\star\star}$ and bias which has all non-negative elements.
When we use the new non-negative $\mathbf{w}^{\star\star}$ and bias to do the calculation, we could simplify the $(\langle \mathbf{x}'_i$(after transfer)$, \mathbf{w}^{\star\star}\rangle + b^{\star\star})$ to the original $\langle \mathbf{x}_i, \mathbf{w}^*\rangle + b^*$ by removing all $(\mathbf{x}_i)$ with zero weight.(easy to prove). Then we have proved 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 and the nonnegativity assumption in winnow will not affect the result.

Note: It is same to transfer $(X, \mathbf{y})$ to $((X, -X), \mathbf{y})$ at the very beginning since we could pad b to w and 1(-1) to X later. The difference only could be noticed on the plot. The PNG I attached displays the result for transfer $(X, \mathbf{y})$ to $((X, 1, -X, -1), \mathbf{y})$ and it approach around 300 - 400 mistake when max pass is 500 while for the transfer $(X, \mathbf{y})$ to $((X, -X), \mathbf{y})$ it only converge to 500 mistake when max pass is 500.

After tune the step size, I choose the step size as 0.0013(without normalization)



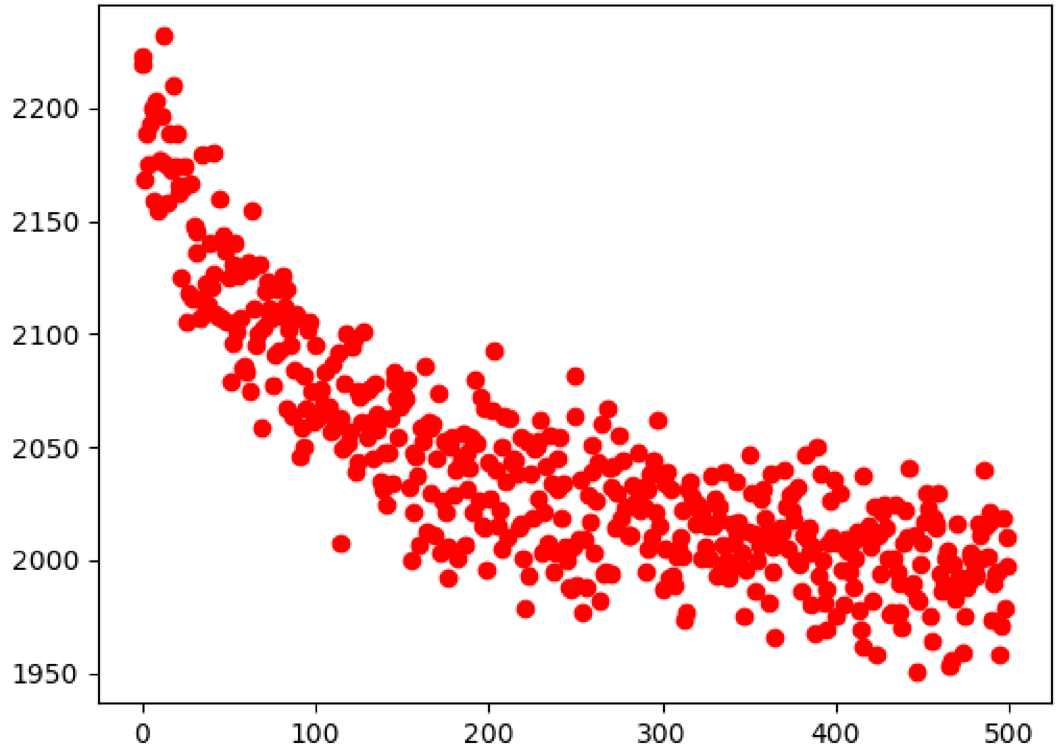
6. (10 pts) Compare what you learned about perceptron with the above result of winnow. What are the respective pros and cons of the two algorithms? [Recall that the margin γ in perceptron can be written as: $\gamma = \max_{b^2+\sum_i w_i^2=1} \min_i \frac{y_i(\langle \mathbf{x}_i, \mathbf{w}\rangle + b)}{\max_i \sqrt{1+\sum_j |X_{ij}|^2}}$, 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).

Answer:

Winnow VS Perceptron:
Data requirement: Perceptron alg will accept all type of data(regardless of positive or negative)
Winnow alg will not work when all training data X are positive and it require extra transformation mentioned in Q5.
Number of mistakes: Perceptron alg should be able to limit the number of mistake to $O(KN)$ where N is the available features while Winnow is able to limit the number of mistakes to $O(K\log(N))$ which would be faster than Perceptron when handling large dimensional data sets
Cons of Two algorithms: Only work with linearly separable data and not really efficient.

Random Plot test:
For Winnow:
The step size has been tuned to 5.4. (from A1E1Q6-winnow-tunestep.py)

For Perceptron:



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:

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

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).

Answer:

on $\lambda = 0$, training set error: 9.796602, valid set error 12.818509, test set error: 424.186499, percentage of nonzeros in w 1.000000
on $\lambda = 10$, training set error: 10.469373, valid set error 13.514303, test set error: 96.379610, percentage of nonzeros in w 1.000000
on $\lambda = 20$, training set error: 10.901473, valid set error 13.924012, test set error: 111.732408, percentage of nonzeros in w 1.000000
on $\lambda = 30$, training set error: 11.436421, valid set error 14.543849, test set error: 127.677207, percentage of nonzeros in w 1.000000
on $\lambda = 40$, training set error: 12.037752, valid set error 15.289777, test set error: 140.707331, percentage of nonzeros in w 1.000000
on $\lambda = 50$, training set error: 12.675279, valid set error 16.105799, test set error: 150.585340, percentage of nonzeros in w 1.000000
on $\lambda = 60$, training set error: 13.328494, valid set error 16.955855, test set error: 157.736070, percentage of nonzeros in w 1.000000
on $\lambda = 70$, training set error: 13.983780, valid set error 17.816542, test set error: 162.688632, percentage of nonzeros in w 1.000000
on $\lambda = 80$, training set error: 14.632175, valid set error 18.672614, test set error: 165.922744, percentage of nonzeros in w 1.000000
on $\lambda = 90$, training set error: 15.267858, valid set error 19.514203, test set error: 167.833844, percentage of nonzeros in w 1.000000
on $\lambda = 100$, training set error: 15.887146, valid set error 20.335036, test set error: 168.735451, percentage of nonzeros in w 1.000000
As we can see from the result the best $\lambda = 10$, since the trainning set, valiadation set and test set error are all lowest

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.

Answer:

only update trainning set x
on $\lambda = 0$,training set error: 39.993866, valid set mean error 1214085415189.452637, test set error:

```
18858.785539
on λ = 10 ,training set error: 44.541755, valid set mean error 1125508812191.328613, test set error:
10897.214417
on λ = 20 ,training set error: 46.272693, valid set mean error 1160399777331.587158, test set error:
8639.283710
on λ = 30 ,training set error: 48.130849, valid set mean error 1188802853036.177246, test set error:
7052.497290
on λ = 40 ,training set error: 50.075577, valid set mean error 1211582987560.106934, test set error:
5864.277430
on λ = 50 ,training set error: 52.033643, valid set mean error 1230236842697.206299, test set error:
4948.446426
on λ = 60 ,training set error: 53.957917, valid set mean error 1245853627528.854980, test set error:
4228.218822
on λ = 70 ,training set error: 55.821255, valid set mean error 1259187760514.459961, test set error:
3652.689648
on λ = 80 ,training set error: 57.609371, valid set mean error 1270767598312.810059, test set error:
3186.540427
on λ = 90 ,training set error: 59.316260, valid set mean error 1280971168700.681885, test set error:
2804.484768
on λ = 100 ,training set error: 60.940448, valid set mean error 1290075008473.671631, test set error:
2488.096048
only update trainning set y
on λ = 0 ,training set error: 722435.471435, valid set mean error 828973.127484, test set error: 18081876.378962
on λ = 10 ,training set error: 728368.927254, valid set mean error 798033.174558, test set error:
5637145.548998
on λ = 20 ,training set error: 729364.046339, valid set mean error 793358.913872, test set error:
4030194.789249
on λ = 30 ,training set error: 730115.783813, valid set mean error 790961.882103, test set error:
3072029.737023
on λ = 40 ,training set error: 730754.352330, valid set mean error 789583.427561, test set error:
2426070.231241
on λ = 50 ,training set error: 731307.958647, valid set mean error 788723.206898, test set error:
1965219.985178
on λ = 60 ,training set error: 731791.935225, valid set mean error 788150.483635, test set error:
1623843.303425
on λ = 70 ,training set error: 732217.823870, valid set mean error 787747.717619, test set error:
1363682.963211
on λ = 80 ,training set error: 732594.934403, valid set mean error 787450.475816, test set error:
1160847.674540
on λ = 90 ,training set error: 732930.865914, valid set mean error 787221.380520, test set error:
999687.401667
on λ = 100 ,training set error: 733231.838195, valid set mean error 787037.749007, test set error:
869568.488925
will update both trainning set x and y
on λ = 0 ,training set error: 39.946023, valid set mean error 1212154201829.447510, test set error:
18824.701622
on λ = 10 ,training set error: 44.485005, valid set mean error 1123649409678.693115, test set error:
10878.089864
on λ = 20 ,training set error: 46.213755, valid set mean error 1158511762065.786377, test set error:
8623.963951
on λ = 30 ,training set error: 48.069897, valid set mean error 1186891861659.682617, test set error:
7039.855382
on λ = 40 ,training set error: 50.012646, valid set mean error 1209653766291.754395, test set error:
5853.659561
on λ = 50 ,training set error: 51.968792, valid set mean error 1228292820860.039551, test set error:
4939.405809
```

on $\lambda = 60$,training set error: 53.891232, valid set mean error 1243897300908.141357, test set error: 4220.432667
on $\lambda = 70$,training set error: 55.752836, valid set mean error 1257220988560.163574, test set error: 3645.917268
on $\lambda = 80$,training set error: 57.539319, valid set mean error 1268791800040.486084, test set error: 3180.598311
on $\lambda = 90$,training set error: 59.244676, valid set mean error 1278987450917.225586, test set error: 2799.230527
on $\lambda = 100$,training set error: 60.867430, valid set mean error 1288084251320.191895, test set error: 2483.417494

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).

Answer:
on $\lambda = 0$,training set error: 0.000000,validation set mean error 105829997.271467, test set error: 3320239.078323, percentage of nonzeros in w 1.000000
on $\lambda = 10$,training set error: 0.007508,validation set mean error 50.161516, test set error: 115.054477, percentage of nonzeros in w 1.000000
on $\lambda = 20$,training set error: 0.028847,validation set mean error 50.016992, test set error: 114.441902, percentage of nonzeros in w 1.000000
on $\lambda = 30$,training set error: 0.062420,validation set mean error 49.885559, test set error: 113.860815, percentage of nonzeros in w 1.000000
on $\lambda = 40$,training set error: 0.106843,validation set mean error 49.765875, test set error: 113.308568, percentage of nonzeros in w 1.000000
on $\lambda = 50$,training set error: 0.160910,validation set mean error 49.656770, test set error: 112.782828, percentage of nonzeros in w 1.000000
on $\lambda = 60$,training set error: 0.223568,validation set mean error 49.557224, test set error: 112.281537, percentage of nonzeros in w 1.000000
on $\lambda = 70$,training set error: 0.293890,validation set mean error 49.466336, test set error: 111.802861, percentage of nonzeros in w 1.000000
on $\lambda = 80$,training set error: 0.371060,validation set mean error 49.383316, test set error: 111.345168, percentage of nonzeros in w 1.000000
on $\lambda = 90$,training set error: 0.454357,validation set mean error 49.307462, test set error: 110.906991, percentage of nonzeros in w 1.000000
on $\lambda = 100$,training set error: 0.543140,validation set mean error 49.238152, test set error: 110.487012, percentage of nonzeros in w 1.000000

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

$$\min_{\mathbf{w}\in\mathbb{R}^d} \tfrac{1}{2}\|X\mathbf{w} - \mathbf{y}\|_2^2 + \lambda\|\mathbf{w}\|_1. \tag{5}$$

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

$$\text{sign}(w)\cdot\max\{0, |w| - \lambda\} = \operatorname*{argmin}_{z\in\mathbb{R}} \tfrac{1}{2}(z - w)^2 + \lambda|z|, \tag{6}$$

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).

Answer:
on $\lambda = 0$, training set error: 9.797682, valid set mean error 12.810455, test set error: 431.604329, percentage of nonzeros in w 1.000000
on $\lambda = 10$, training set error: 10.427198, valid set mean error 13.370082, test set error: 59.429890, percentage of nonzeros in w 0.769231
on $\lambda = 20$, training set error: 10.468848, valid set mean error 13.299904, test set error: 60.158306, percentage of nonzeros in w 0.738462
on $\lambda = 30$, training set error: 10.538040, valid set mean error 13.287038, test set error: 60.984587, percentage of nonzeros in w 0.730769
on $\lambda = 40$, training set error: 10.637298, valid set mean error 13.321235, test set error: 62.001633, percentage of nonzeros in w 0.684615
on $\lambda = 50$, training set error: 10.764400, valid set mean error 13.399117, test set error: 63.229312, percentage of nonzeros in w 0.653846
on $\lambda = 60$, training set error: 10.921267, valid set mean error 13.526494, test set error: 62.697247, percentage of nonzeros in w 0.646154
on $\lambda = 70$, training set error: 11.108919, valid set mean error 13.710215, test set error: 61.932790, percentage of nonzeros in w 0.653846
on $\lambda = 80$, training set error: 11.327597, valid set mean error 13.937419, test set error: 61.202367, percentage of nonzeros in w 0.646154
on $\lambda = 90$, training set error: 11.573840, valid set mean error 14.185586, test set error: 60.496832, percentage of nonzeros in w 0.630769
on $\lambda = 100$, training set error: 11.848576, valid set mean error 14.481719, test set error: 59.870484, percentage of nonzeros in w 0.630769
Note: The training error for $\lambda = 0$ is not zero but too small, so the result just printed as 0.

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.]

Answer:

on $\lambda = 0$, training set error: 0.000000, valid set mean error 629.242899, test set error: 51074.514268, percentage of nonzeros in w 1.000000
on $\lambda = 10$, training set error: 0.901929, valid set mean error 21.295962, test set error: 60.348668, percentage of nonzeros in w 0.201900
on $\lambda = 20$, training set error: 2.793751, valid set mean error 17.409816, test set error: 57.036809, percentage of nonzeros in w 0.143100
on $\lambda = 30$, training set error: 4.839759, valid set mean error 15.200029, test set error: 57.855492, percentage of nonzeros in w 0.098700
on $\lambda = 40$, training set error: 6.772430, valid set mean error 14.204094, test set error: 60.010639, percentage of nonzeros in w 0.063500
on $\lambda = 50$, training set error: 8.420902, valid set mean error 13.884545, test set error: 61.058509, percentage of nonzeros in w 0.038600
on $\lambda = 60$, training set error: 9.673447, valid set mean error 13.792015, test set error: 61.168035, percentage of nonzeros in w 0.019200
on $\lambda = 70$, training set error: 10.566657, valid set mean error 13.864160, test set error: 61.045615, percentage of nonzeros in w 0.008700
on $\lambda = 80$, training set error: 11.122557, valid set mean error 14.072609, test set error: 61.007686, percentage of nonzeros in w 0.002900
on $\lambda = 90$, training set error: 11.516065, valid set mean error 14.257103, test set error: 60.529119, percentage of nonzeros in w 0.001300

on $\lambda = 100$, training set error: 11.833156, valid set mean error 14.511382, test set error: 59.842520, percentage of nonzeros in w 0.000200
This script need about 800 - 1000 secs to finish.
Note: The training error for $\lambda = 0$ is not zero but too small, so the result just printed as 0.

Conclusion:
1. The error of validation set and test set of Q5 when $\lambda = 0$ is significantly larger than the one in Q4, which is caused by too many random features.
2. After λ getting bigger the error between with random feature(Q5) is getting closer to without random feature (Q4), which shows that Lasso alg is capable of resisting the affect from random features. Also, when compare the non-zero percentage with Q3(all 100 %), the lasso is able to remove some of random features (by setting the weight to zero.
3. The time Lasso takes is affected by the size of data significantly. For 1013 features, it takes about 800 - 1000 secs. For 13 feature, it takes around 8 secs. (I also tested on 113 features, it takes about 80 secs). So It runs in O(n) time. May have performance bottleneck dealing with large scale of data.