CS480/680: Introduction to Machine Learning

Homework 1

Due: 11:59 pm, September 27, 2018, submit on LEARN.

Include your name and student number!

Name: Sidong Wei Student No. 20779006

All The instruction for my codes are written in the "readme" file.

Exercise 1: Perceptron (40 pts)

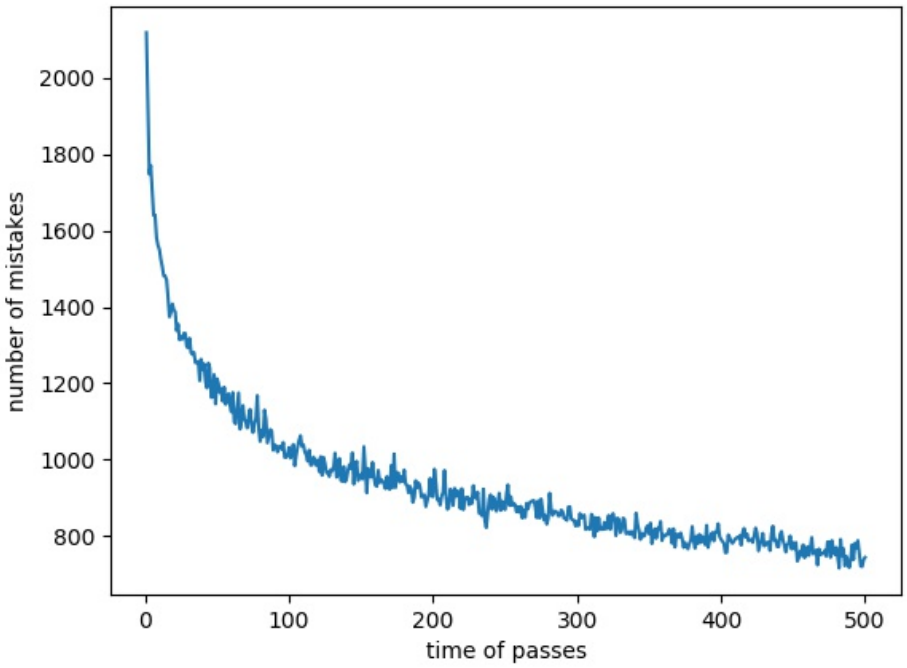
Convention: All algebraic operations, when applied to a vector or matrix, are understood to be element-wise (unless otherwise stated).

Algorithm 1: The perceptron algorithm.

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

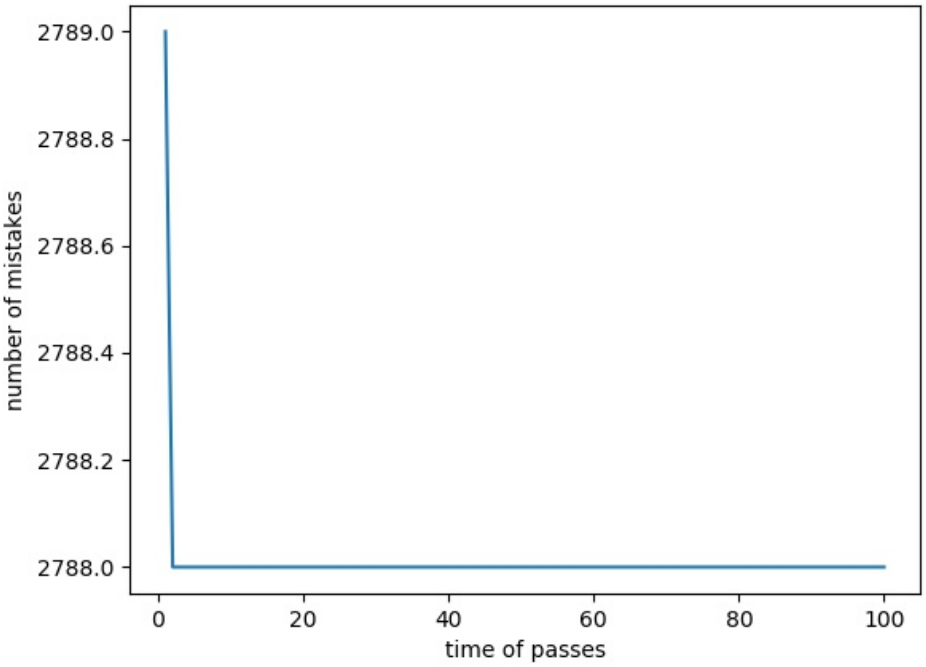
```
for t = 1, 2, ..., max_pass do
    mistake(t) ← 0
    for i = 1, 2, ..., n do
        if y_i(⟨x_i, w⟩ + b) ≤ 0 then
            w ← w + y_i x_i                    // x_i is the i-th column of X
            b ← b + y_i
            mistake(t) ← mistake(t) + 1
```

1. (15 pts) Implement the perceptron in Algorithm 1. Your implementation should take input as $X = [\mathbf{x}_1, \ldots, \mathbf{x}_n] \in \mathbb{R}^{d\times n}$, $\mathbf{y} \in \{-1,1\}^n$, 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).



I set max pass as 500 and plot the number of mistakes w.r.t. number of passes as above. The number of mistakes has a overall trend to decrease with a decreasing speed, which seems to converge to some value. To add some more information, the minimal number of mistakes within 500 passes is 716, the number of mistakes after 500 passes is 744, and the number of mistakes after 10000 passes is 510.

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.



After the first pass, the number of mistakes seems to stop changing, and this may due to the fact that we update the parameters every iteration, and it turns into a situation that the sum of all changes of parameters in one pass adds up to 0, which makes it doing nothing.

3. (10 pts) Prove the following claim: If there exist $\mathbf{w}^*$ and b^* such that for all i,

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

then there exist $\mathbf{w}^\star$ and $b^\star$ such that for all i,

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

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

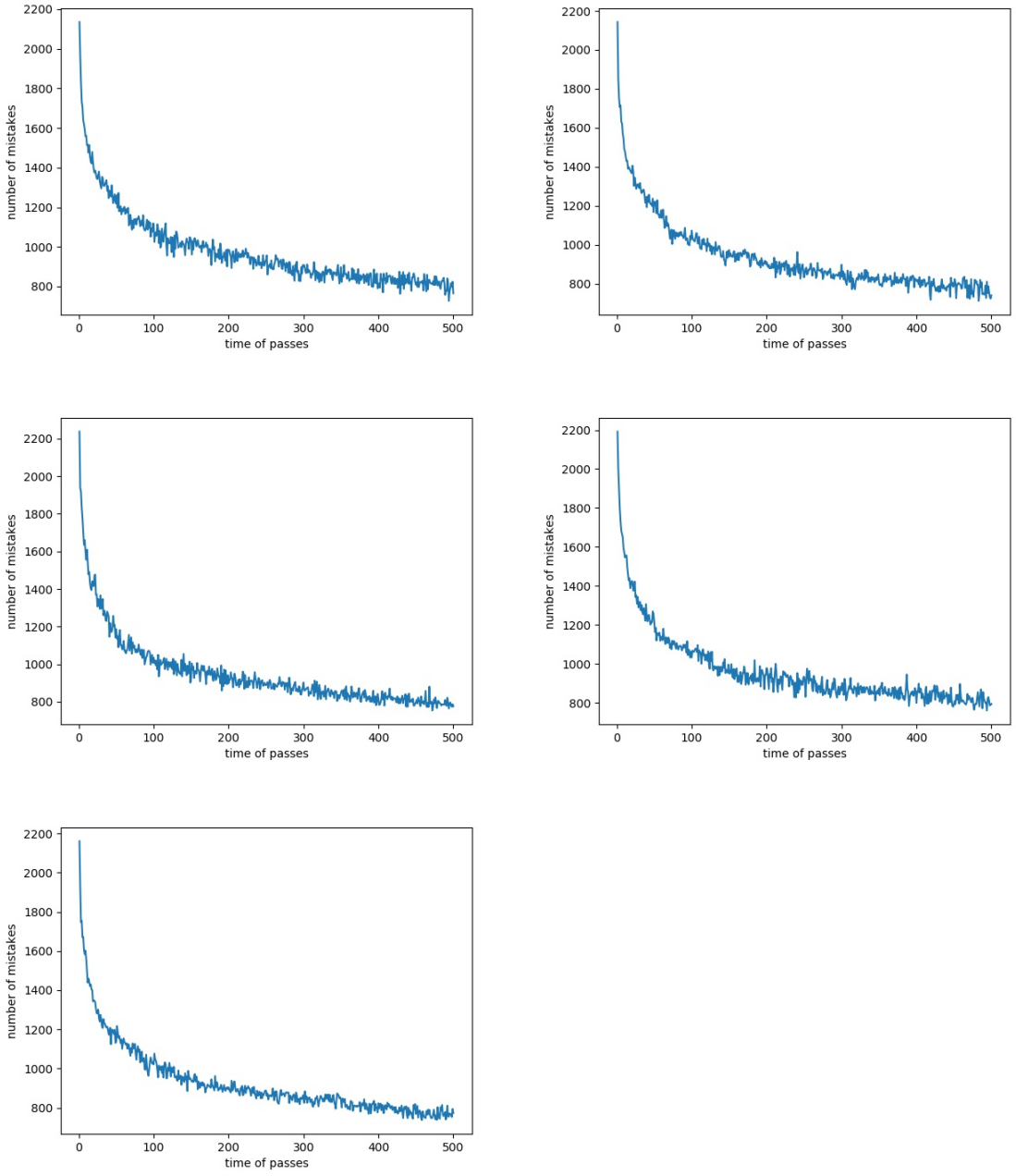
Proof. First we should assume that the number of data is finite, and thus countable, because the result may not be true otherwise. For each of the data point x_i that satisfy $\langle \mathbf{x}_i, \mathbf{w}^* \rangle + b^* < 0$, there should be a corresponding positive constant b_i that satisfy $\langle \mathbf{x}_i, \mathbf{w}^* \rangle + b^* + b_i = 0$.

As the $\{x_i\}$ being countable, the set of all b_i, denoted as $\{b_i\}$, should also be countable. Thus we can find a minimum value of $\{b_i\}$, denoted as b_{min}, being strictly positive.

We can choose $b^\star = b^* + \frac{1}{2}b_{min}, \mathbf{w}^\star = \mathbf{w}^*$, and we can see that $\langle \mathbf{x}_i, \mathbf{w}^\star \rangle + b^\star = \langle \mathbf{x}_i, \mathbf{w}^* \rangle + b^* + \frac{1}{2}b_{min} < 0$ still holds for all i that $y_i = -1$. On the other handfor any x_i such that $\langle \mathbf{x}_i, \mathbf{w}^* \rangle + b^* \geq 0$, we can see that $\langle \mathbf{x}_i, \mathbf{w}^\star \rangle + b^\star = \langle \mathbf{x}_i, \mathbf{w}^* \rangle + b^* + \frac{1}{2}b_{min}$ cannot be zero any more, so it is strictly greater than zero.

So we proved that such $b^\star$ and $\mathbf{w}^\star$ always exist.

4. (10 pts) Randomly permute the columns in X and run perceptron again. Plot the number of mistakes w.r.t. the number of passes. Repeat this for 5 times. Is there any difference as compared to Ex1.1? Does perceptron "converge" to the same solution eventually?



From the figures we can see that even if we permute the columns, the overall speed of convergence and the result after 500 passes are almost the same, which means they finally converge to a same result. There are only some differences in the detail of the figure, like how often and how strong the fluctuation is.

Exercise 2: Linear Regression and Regularization (60 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 2: Alternating minimization for Lasso.

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

1. (15 pts) Implement ridge regression that we discussed in class:

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

   Your implementation should take input as $X \in \mathbb{R}^{d\times n}$, $\mathbf{y} \in \mathbb{R}^n$, $\lambda \geq 0$, 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 ($\frac{1}{n}\|X^\top \mathbf{w} - \mathbf{y}\|_2^2$), 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).

lamda	training set error	validation set error	test set error	nonzero ratio
0.0	9.69429863891	12.1623051818	370.222957298	1.0
10.0	10.9180093732	11.9955025595	111.949038842	1.0
20.0	11.902947124	13.551547082	139.584110477	1.0
30.0	13.0391071363	14.3492681473	156.241815994	1.0
40.0	14.2120561764	16.2918417744	164.365930527	1.0
50.0	15.3634658704	17.0850447294	167.22994633	1.0
60.0	16.4657619384	18.9847711073	167.018217922	1.0
70.0	17.5072497209	19.7373929023	165.060403113	1.0
80.0	18.4844749871	21.0488578468	162.152045087	1.0
90.0	19.3982318643	22.4154219127	158.768991099	1.0
100.0	20.2514295412	23.2466758959	155.196223188	1.0

   The sheet contains the result I got. It worth mentioning that all the column of error is the mean square error on each set, and also I carry out the 10-fold cross-validation by splitting into 9 even part and 1 part that is a little bit fewer. Since there are other ways to split the data, I found it necessary to make it clear how I did it.

   We can see that when $\lambda = 10$, we get the least mean error on validation set, and that means we should choose this λ as the parameter for regularization term. Also as we check the column of test set error, we can verify that choosing $\lambda^* = 10$ does give us the best result on test set.

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 ($\frac{1}{n}\|X^\top \mathbf{w} - \mathbf{y}\|_2^2$), validation set (averaged over 10-fold) and test set [different normalization constant n] for each candidate λ. Can you explain the results?

lamda	training set error	validation set error	test set error
0.0	24.3926045798	1.33019364598e+12	3614.08236155
10.0	26.7228469305	1.410456234e+12	1719.46249232
20.0	27.8598717484	1.34393340464e+12	1483.64678142
30.0	29.1126823835	1.41910068316e+12	1314.81170474
40.0	30.3958833132	1.30793325716e+12	1183.906704
50.0	31.6551597821	1.39274804916e+12	1079.2047489
60.0	32.8625225193	1.37235797913e+12	993.626608001
70.0	34.0054441668	1.34734256436e+12	922.455647181
80.0	35.0797197439	1.53203038493e+12	862.389225673
90.0	36.0857555138	1.40772683094e+12	811.053269571
100.0	37.0263264161	1.4797395406e+12	766.697694819

For this question, I did "Multiply x by 10^6 and y by 10^3" at the same time. And the result is interesting, no matter how we choose λ, the mean square error on validation set is alway much higher than previous one, it grows to around 10^{11} times bigger. However, the mean square error on training set and test set change not that much, they grow to about 10 times bigger than the previous results.

In my analysis, the reason for this phenomenon is that if a exaggerated data is in the training set (as what we do to get the test set error column) or in both training and test set (as what we do to get the training set error column), it will not affect that much, since the machine has learned something about this abnormal data, and modify its parameters to somewhere between the normal data and the abnormal data.

But if the abnormal data only appears in the test set (as what we do in one of the cross-validation procedure), the machine had no information about it before, so it will predict it normally and make a really huge mistake. We can see how huge this mistake is, as it is still so large after being averaged.

This fact tells.us we could not let a really abnormal data exists (only) in the test set.

And in this question, it seems $\lambda = 40$ performs the best during cross-validation, but it is kind of meaningless.

3. (10 pts) Add 1000 irrelevant rows 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 ($\frac{1}{n}\|X^\top \mathbf{w} - \mathbf{y}\|_2^2$), 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).

lamda	training set error	validation set error	test set error	nonzero ratio
0.0	6.57746260429e-24	942164.241452	2398279.98831	1.0
10.0	0.0394728139299	54.181240511	136.780871804	1.0
20.0	0.143796894512	54.2240174807	133.527726467	1.0
30.0	0.296501095762	49.7257016479	130.674971697	1.0
40.0	0.485669954766	53.9259050044	128.151589238	1.0
50.0	0.702506505427	50.8797389103	125.902956419	1.0
60.0	0.940402736708	48.4868143896	123.886273216	1.0
70.0	1.19432086733	51.0517705499	122.067458302	1.0
80.0	1.46037107419	46.4079168825	120.418987174	1.0
90.0	1.7355168917	46.6510276452	118.918352085	1.0
100.0	2.01736564154	45.8137552013	117.546942836	1.0

First of all, noticed that this time the column of "nonzero ratio" only reports the nonzero ratio in the last 1000 entries. But the result still is all 1, which is kind of weird. In theory, if we have a completely random or irrelevant feature, the algorithm should give a zero weight on this feature, thus the ratio should be zero or very close to zero, but it says 1, why?

It is actually more like a math problem, it is like we formulate n numbers from a normal distribution function and check the mean of them, which should be zero according to theory, but in fact it can be

any number. The reason for this problem is that we cannot always get the "Expectation" result when we try finite times, it will only always be the "Expectation" if we try infinite times.

Similarly, for our problem, each row we add, has n numbers where n is the scale of the training data set, and the value of w is straightly computed by doing some operations to the sum of product of the rows. In abstract, the result to get a zero entry in w is really similar to get a mean of zero as I mentioned above, so it is also almost impossible (in probability sense). As long as the number of columns is not infinite, each row we add is almost irrelevant, but not completely irrelevant to current rows. Thus why we cannot get a zero in this question, while it should be really close to zero.

Another phenomenon with the result is that if we do not add the regularization term, i.e.$\lambda = 0$, then we will overfit and the error on training set will get extremely small (because we test with set same as training set), but from cross-validation and test on real test set we can see the error is extremely big. That is typically what will happen when an overfit happens, and fortunately, we can see from the result that adding a regularization term can effectively prevent the overfitting from happening, and the results are pretty similar to the ones before we adding these rows. This phenomenon verifies the importance to have a regularization term.

And in this question, $\lambda = 100$, i.e. the largest λ, performs the best during cross-validation, and it also performs the best on test set.

4. (15 pts) The Lasso replaces the (squared) 2-norm penalty in ridge regression with the 1-norm:

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

Implement the alternating minimization algorithm for Lasso (Algorithm 2). You might find the following fact useful:

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

which is known as the soft-thresholding operator. You should try to perform step 3 of Algorithm 2 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 ($\frac{1}{n}\|X^\top \mathbf{w} - \mathbf{y}\|_2^2$), 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).

lamda	training set error	validation set error	test set error	nonzero ratio
0.0	9.71326027548	11.6569176695	341.475144171	1.0
10.0	11.2931212695	12.4944935126	252.458378269	0.857142857143
20.0	11.9310404194	13.1153450333	360.426250317	0.785714285714
30.0	12.4855572788	13.9954944578	410.939135082	0.785714285714
40.0	12.9286017753	13.97192931	413.038919496	0.785714285714
50.0	13.3425709064	14.3080181187	392.423645449	0.785714285714
60.0	13.6657926872	14.5762832926	354.851655777	0.785714285714
70.0	13.9244078198	14.8600479894	310.390588671	0.785714285714
80.0	14.193636452	13.9484230004	267.260841956	0.785714285714
90.0	14.4305680066	12.8601390378	224.191716049	0.785714285714
100.0	11.2523357418	12.3622407351	63.6749574804	0.714285714286

For this question, I would like to present how I carry out the algorithm. First I let vector $A = X_{j:}^\top$, and vector $B = \mathbf{y} - \sum_{k\neq j} X_{k:}^\top \mathbf{w}_k$. From Algorithm 2 step 3 we can rewrite the formula as

$$\operatorname*{argmin}_{z\in\mathbb{R}} \tfrac{1}{2}\|Az - B\|_2^2 + \lambda|z|.$$

Then we can compute the norm and get a better expression as

$$\operatorname*{argmin}_{z\in\mathbb{R}} \tfrac{1}{2}\left(A^\top A z^2 - 2A^\top B z + B^\top B\right) + \lambda|z|.$$

Considering the definition of argmin, we can plus or subtract or multiply or divide the expression by any positive constant without changing the result. Thus we can subtract $B^\top B$ then divide it by $A\top A$, and get

$$\operatorname*{argmin}_{z\in\mathbb{R}} \ \tfrac{1}{2}\left(z^2 - 2\frac{A\top B}{A\top A}z\right) + \frac{\lambda}{A^\top A}|z|.$$

Then we also modify the soft-thresholding operator a little bit, also compute the square and subtract the constant, and we will get

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

Now we just need to assign the parameter in the soft-thresholding operator as $w = \frac{A\top B}{A\top A}, \lambda = \frac{\lambda}{A^\top A}$ (it is a little bit confusing to use the same letter on both side, hope you can get my point). Then we can use the left side to computer the result of $\operatorname*{argmin}_{z\in\mathbb{R}} \ \frac{1}{2}\|Az - B\|_2^2 + \lambda|z|$.

The code just assign A and B as I mentioned, and compute w_{temp} and lam_{temp} as the two parameters in the soft-thresholding operator. One trick here is noticing that $B = \mathbf{y} - \sum_{k\neq j} X_{k:}^\top \mathbf{w}_k$ here seems to require $O(dn)$ operations where d represent the number of attributes. However, we can actually store a value $sum = \sum X_{k:}^\top \mathbf{w}_k$, and each time when we want to use B, we can simply compute $B = \mathbf{y} - sum + X_{j:}^\top \mathbf{w}_j$, and we update sum each time after some $\mathbf{w}_i$ is changed.

By doing this trick, we can check that all the operations in step 3 only needs at most $O(n)$ time and space, and there is only constant number of operations, so it guarantees that the complexity is $O(n)$. Also, the function "lasso" in the file "regression_func.py" gives detail and annotation about this algorithm.

Now let's see the result, and they are overall very close to the result that we got by ridge regression, only this time it seems that $\lambda = 0$ gives the best answer. This does not mean that we should not use regularization, and the fact is that the real best λ may lie between in some intervals like $[0, 10]$ or $[100, \infty)$.

As for $\lambda = 0$, by theory the results produced by lasso and ridge regression should be the same, and actually they are pretty close, but not the same. This is due to how we carry out the algorithm, we directly get the result by solving linear equations in ridge regression, but we use iterations to get close to our goal in lasso algorithm. That means in practice, the result of lasso will continuously getting close to the result of ridge, but since we stop the algorithm when changes become less than a tolerance, so it probably won't reach its destination by the time it stops. That is why the results are close but not the same.

Noticing the nonzero ratio is not always 1 this time, I would like to analyze this phenomenon in the following question.

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 ($\frac{1}{n}\|X^\top \mathbf{w} - \mathbf{y}\|_2^2$), 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.]

lamda	training set error	validation set error	test set error	nonzero ratio
0.0	0.000280578087331	603.614914749	50296.5505236	1.0
10.0	0.266132286111	25.8359427188	96.5994861244	0.279
20.0	0.941550801104	23.5504972366	101.748804038	0.248
30.0	1.86168139681	20.998604616	96.5299601489	0.215
40.0	2.94699744721	19.5968534383	91.4899874298	0.184
50.0	4.06938861665	19.9679548	90.5439135756	0.153
60.0	5.25480550995	20.5064429753	88.8064420284	0.133
70.0	6.48665787969	19.8539277036	87.1735009629	0.113
80.0	7.70300835212	19.0958541852	81.3312975745	0.094
90.0	8.87755982917	19.1825029935	72.4694737385	0.085
100.0	10.0015997274	19.3582378846	63.4176705859	0.069

First let's have a look at the results here. The overall trend is quite alike with the one we get in question 3, still overfitting when no regularization term, and performs fairly when regularization exists. These two results (question 3 and question 5) both proves that if the number of features is very large, or with too many irrelevant features, there is a possibility to have overfitting when performing linear regression, and adding a regularization term can effectively relieve this problem, which makes is very necessary.

As for the best λ, this time seems that when $\lambda = 80$, the error during cross-validation is the minimum. However, this time it does not performs the best result on test set. This is also normal because the "best λ" is only in theoretical sense, and it does not guarantee to be the real best one.

At last, let's talk about the nonzero ratio in lasso algorithm. From question 4 we can see there are some entries in $\mathbf{w}$ that get zero, and in question 5 there are even more. This phenomenon kind of fits the reality that some features are irrelevant, but why this time $\mathbf{w}$ can have zeros but ridge regression cannot?

The answer is simple, just take a look at the soft-thresholding operator, we can see on the left side there is a threshold. And this means it will return a constant 0 for a certain range of input, and that is why we will get zero in lasso. If the output should be very small, and it will probably fall into such range and get a zero rather than get a real small but nonzero number. Imagine we have totally random input, if we have a strictly monotonous function, the probability of returning a certain value is zero, but if we have a threshold function(e.g. sign function), the probability of returning some value (e.g. returning "1") may not be zero. This is just like what happened when we compare ridge regression and lasso's algorithm.

We may also noticed that when $\lambda = 0$, lasso also has no zeros, and this is because the threshold closes when $\lambda = 0$, which means it only produces $|w| - \lambda$ rather than 0 (this λ is the not the λ in the first sentence, it refers to the λ in the soft-thresholding operator, but anyway they are all zero because they only matters a factor of a constant $A^\top A$, hope I did not confuse you).