ECE 532 THEORY AND APPLICATIONS OF PATTERN RECOGNITION
FINAL PROJECT: LAB REPORT

The Reweighted Lasso and Application to Neuroimaging Data

Qihong Lu [*], Jiaan Wang, Zhenyu Zhang

Supervised by Dr. Robert D. Nowak & Urvashi K. Oswal

December 5, 2015

[*]Correspondence concerning this document should be addressed to Qihong Lu, Psychology Department, University of Wisconsin-Madison, Madison, WI 53715. E-mail: qihong.lu@wisc.edu

Identify the functional role of different brain regions is important to understand how the brain works. Imagine we have a neuroimaging data recorded while a person was viewing different kind of images. Suppose you can guess if this person was viewing a picture of a human face based on the activation values of a brain region, what would you conclude?

A typical "brain decoding" problem might look like the following. First, we have the neuroimaging data, $X \in \mathbb{R}^{mxn}$, where n is much larger than m. Each row is a picture that the participant looked and each column is a tiny chuck of the brain, called voxel. Second, we have the labels, $y \in \mathbb{R}^{mx1}$, indicating what kinds of images the participant was viewing. The goal is to find a subset of voxels that are important in encoding a given type of images.

For this feature selection problem, we prefer a sparse solution. Suppose we want to predict if a participant was looking at a picture of face or not, ridge regression would assign small weights for all voxels. This is saying all voxels are important, which is not particularly interesting: we know the entire brain is important for about all cognitive functions. Instead, being able to make claims about the relative importance of different brain regions is more informative. Therefore, we hope to minimize the number of voxels with non-zero coefficient.

MOTIVATION: REWEIGHTED LASSO AS A PROXY FOR THE ℓ_0 PROBLEM

Recall we learned lasso as a relaxation of the "ℓ_0 problem", which is to minimize the number of features with non-zero coefficient. Since the ℓ_0 problem is non-convex, we approximated it by lasso, which is convex.

More formally, this is our goal, the ℓ_0 problem

$$\min_{\beta} \|y - X\beta\|_2^2 + \lambda \,\mathrm{nnz}(\beta) \qquad \lambda > 0$$

Here, X represents a m by n design matrix and y represents the responses. Lambda is the regularization parameter indicating how much do we care about the sparsity. And nnz is the cardinality of non-zero elements in β. The goal is to find a β that minimizes the objective. The ℓ_0 penalty imposed on the regularization term makes the problem non-convex, so we approximate it by lasso:

$$\min_{\beta} \|y - X\beta\|_2^2 + \lambda \|\beta\|_1 \qquad \lambda > 0$$

In this lab, I will show you that **the reweighted lasso can do better** in terms of approximating ℓ_0-like solution. Now, I will explain the intuition behind it. Later, we will implement the

reweighted lasso to study its behaviors.

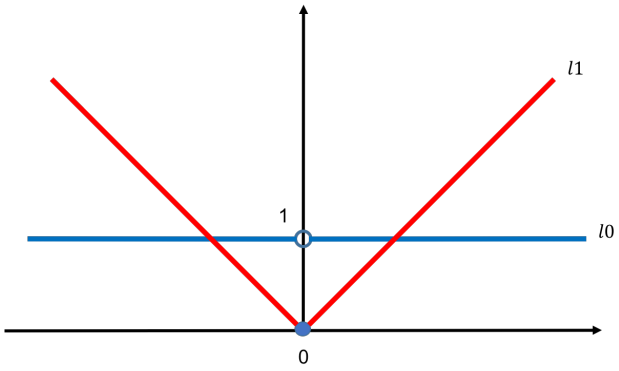
INTUITION: FROM LASSO TO THE REWEIGHTED LASSO

The underlying idea of the reweighted lasso is very simple. The reweighted lasso is actually solving a sequence of lasso problems with some weights multiplied on the regularization terms. The question is how to identify an appropriate set of weights so the following minimization problem would produce a ℓ_0-like solution.

$$\min_{\beta} \|y - X\beta\|_2^2 + \|W\beta\|_1$$

Here, X represents a m by n design matrix and y represents the responses. The weights, denoted by W, is a n by n diagonal matrix. The i^{th} diagonal entry of W, denoted by w_i, is the weight for the i^{th} element of β.

To see what W we need, we need to understand the difference between ℓ_0 and ℓ_1 penalty. Recall that one major difference between ℓ_1 and ℓ_0 penalty can be seen from their loss functions, visualized below. The ℓ_0 penalty has zero loss for features with zero coefficient, and loss of one otherwise. On the other hand, the ℓ_1 penalty is linearly proportional to the feature magnitude.



Reweighting tries to push the ℓ_1 penalty towards ℓ_0 penalty by removing the magnitude dependency of ℓ_1 penalty. For the regularization term, if we assign big weight to features with high magnitude, these features will be penalized more when regularizing. In essence, reweighted lasso is regularizing different feature by different amount, instead of using a single lambda value that regularize every features by the same amount. But how do we know which feature has high magnitude and which feature has small magnitude (Exercise 3)?

ALGORITHM: MODIFIED REWEIGHTED LASSO

Here is the algorithm of the reweighted lasso method, modified from Candès et al. (2008).
1. Initialize the weights to be all one.
2. At the k^{th} iteration, solve the weighted ℓ_1 problem.

$$\beta^{(k)} = \arg\min_{\beta} \| y - X\beta \|_2^2 + \| W^{(k)} \beta \|_1$$

3. Update the weight w_i for all i.

$$w_i^{(k+1)} = \frac{1}{|\beta_i^{(k)}| + e} \qquad (e > 0)$$

4. Alternate between step 2 and 3. Terminate when a stopping criterion is met.

In step 3, e is a parameter of your choice to ensure we do not divide by zero. This parameter has direct influences on the weights. Finding an appropriate e is actually non-trivial and you might need to experiment with it.
Note: This algorithm is not exactly what Candès et al.(2008) proposed, but it gives qualitatively similar results and I feel it can give you some good intuitions about the relation between the reweighted lasso and regular lasso. I will explain the "real" algorithm later.

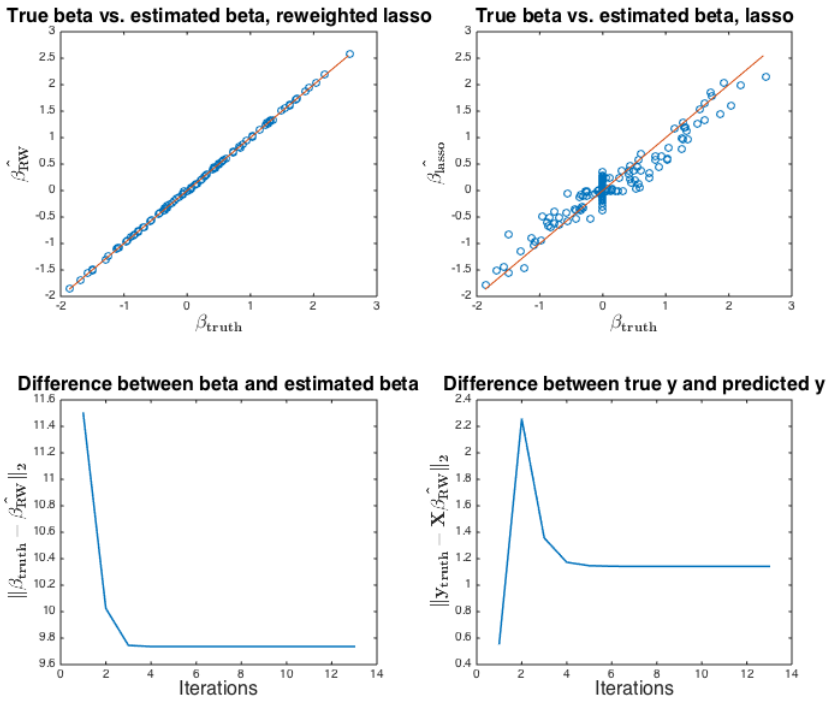
Warn up exercises
As you can see, to solve the reweighted lasso problem, we just need to solve a sequence of lasso-like problems. The following exercises would give you a sense of how the reweighted lasso and the regular lasso relate to each other.
1. Write down the iterative soft thresholding update for lasso.
2. Write down the reweighted lasso update step.
Hint: you just need to modify the lasso ISTA update.
3. In the 3^{rd} step in the reweighted lasso algorihtm, why it makes sense to update the weights to be inversely proportional to the β estimates?

SIMULATION: EXPLORE THE BEHAVIOR OF THE REWEIGHTED LASSO

Now, I want to show you that **the reweighted lasso can recover sparse solution accurately**. In particular, we are going to compare the reweighted lasso and the regular lasso in a simulation, when the ground truth is known.

First, I generated a standard normal random matrix X, with 256 rows and 512 columns and a standard normal random 512 dimensional β vector with 100 non-zero elements. Then I generated the response, $y = X\beta$. No noise was imposed. Now, we know that the true β is sparse, so I want to seek a sparse solution with the underdetermined system formed by X and y.

With X and y, I "guessed" the β values with lasso and the reweighted lasso. The plot below compares their deviation from the truth. On the upper left, I am plotting the β estimated by the reweighted lasso against the true β, which is aligned on 45 degree line indicating closeness. Also, the reweighted lasso solution had 96 non-zero elements, all of which are indeed non-zero. The reweighted lasso also assigned zero coefficients to four non-zero features, since their true values were too small. Here, the reweighted lasso did a great job at recovering the true β, compared to lasso on the upper right. For example, lasso solution had 262 non-zero elements. (The lambda I used for the regular lasso was 1. Increasing this lambda value will increase its sparsity for sure, but I found the estimated β was even worse.)



Reweighted lasso is an iterative algorithm. The two plots on the bottom are showing the progress of the reweighted lasso overtime. On the bottom left, I am plotting the deviation of the estimated β and the true β over time, which decreased and converged and the end. On

the bottom right, I plotted the deviation of predicted y and the true y for the reweighted lasso. As you can see, the final predicted y value of the reweighted lasso is worse than regular lasso. Do you have any idea about why is this the case (Exercise 7)?

In this particular simulation scenario, the reweighted lasso was accurate at reconstructing the true β : all β elements identified as being non-zero (by the reweighted lasso) were truly non-zero, and it was not saying any zero feature as non-zero. Nevertheless, the reweighted lasso is not necessarily better at fitting the responses, y.

Simulation Exercises

4. Implement the regular lasso. I used the iterative soft thresholding (ISTA) with landweber iteration (instead of gradient decent) because I do not want to worry about the learning rate. Recall that for ISTA, a step size satisfing $0 < \tau \leq \frac{1}{\|X\|_2^2}$ will ensure the approximation never become worse.

5. Implement reweighted lasso. If you implemented lasso correctly, you can implement the reweighted lasso with slight modifications. Empirically, set e to be .1 seems to work well. Also, to get more insights about this reweighting algorithm, you might want to use the debugging mode to step though how the estimated β and the weights interact.

6. Conduct a simulation to compare the regular lasso and the reweighted lasso. In my simulation, I did not add any noise. Describe what you found with different levels of noise. According to my experience, the performance of the reweighted lasso is not very stable, so you might want to run the simulation several times.

7. Do you have any hypothesis about why reweighted lasso was not so good at predicting y? This is a open question that I am also not exactly sure.

8. I was lying to you. The algorithm I provided is different from what was proposed (Candès et al., 2008). In particular, the step 2 is actually:

$$\min_{\beta} \|W\beta\|_1 \qquad \text{subject to} \quad y = X\beta$$

This is a constrained optimization problem: it is minimizing the lasso penalty, *subject to an equality constraint.* Since constrained optimization is not covered in class, I attached my implementation in Appendix 1. Feel free to play with it. Can you see some similarities and

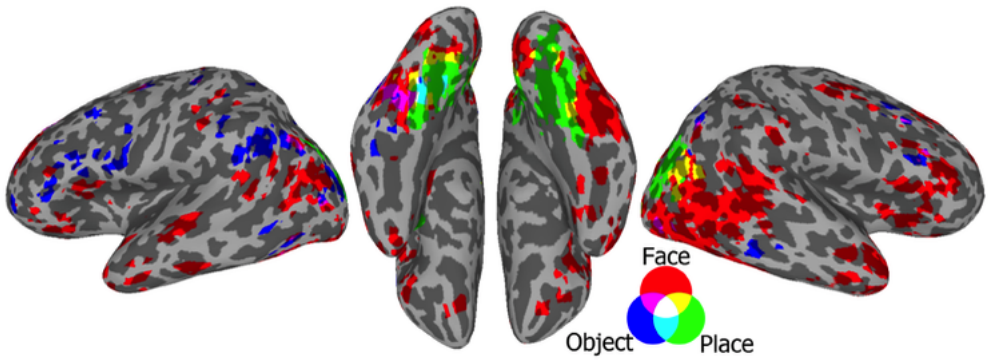
differences between what I wrote previously versus this constrained optimization?

BRAIN DECODING: APPLICATION TO NEUROIMAGING DATA

We are going to have a peek at a fMRI data set from Lewis-Peacock and Postle (2008) in our psychology department. In their neuroimaging experiment, the participants were viewing three kinds of images: pictures of faces, familiar places or common objects. The goal is to identify brain regions that are important for, say, face recognition. How do we know which region is important? One answer is classification accuracy, if there is a brain region that allows a classifier to guess if the participant was seeing a face or not with high accuracy, this brain region is probably relevant to face recognition.

In our fMRI data, each training example is a stimulus (a picture, in our case). Each feature is a voxel, which is a small chunk of the cortex. A typical fMRI data is underdetermined: we usually have many more voxels than stimuli.

Previously, a classifier was implemented using a modified lasso method, which was able to guess the kind of images the participant was viewing with high accuracy (Cox, Lu & Rogers, 2015). On following inflated brain models, the colored regions are showing voxels with non-zero weights (Cox, Lu & Rogers, 2015). For example, the red regions are showing voxels with non-zero weights in face vs. non-face classification.



It turns out that these regions have functional-neuroanatomical meanings. For example, across many participants, lasso consistently identified the the fusiform face area (FFA) as being important for face vs. non-face classification (Cox, Lu & Rogers, 2015). We know that FFA is relevant to face recognition. In some clinical cases, patents with FFA damage cannot recognize faces: a symptom called prosopagnosia (Barton, et al., 2002), but as always, the

story is more complicated (Steeves, 2006).

ALGORITHM FOR THE REWEIGHTED LASSO (CANDÈS ET AL., 2008)

To work with real data, we need to modify the reweighted lasso algorithm. We need to change the equality constraint $y = X\beta$ to $\|y - X\beta\|_2 \leq \delta$ (Candès et al., 2008). Do you understand why?

To summarize, the algorithm for the reweighted lasso is the following:
1. Initialize the weights to be all one.
2. At the $(k+1)^{th}$ iteration, solve the constrained ℓ_1 problem:

$$\beta^{(k)} = \arg\min_{\beta} \|W^{(k)}\beta\|_1 \quad \text{subject to} \quad \|y - X\beta\|_2 \leq \delta$$

3. Update the weight w_i for all i.

$$w_i^{(k+1)} = \frac{1}{|\beta_i^{(k)}| + e} \qquad (e > 0)$$

4. Alternate between step 2 and 3. Terminate when a stopping criterion is met.

Now, δ is a parameter that you need to experiment with. If you set δ to be very tiny, then it is close to the equality constraint. In practice, strict equality constraint is not going to work, since equality rarely exists in nature. More generally, a δ that is too small would induce overfitting to the training set and poor generalization to unseen data.

Exercise 9: "brain decoding"

Load the `jlpdata.mat`. Use the classifiers you built previously to guess whether the participant was viewing a face or not (the code template in Appendix 2 explains what is in the data). One suggestion is to compare the cross-validated accuracy and sparsity of lasso and the reweighted lasso, but you can also explore anything you want to see.

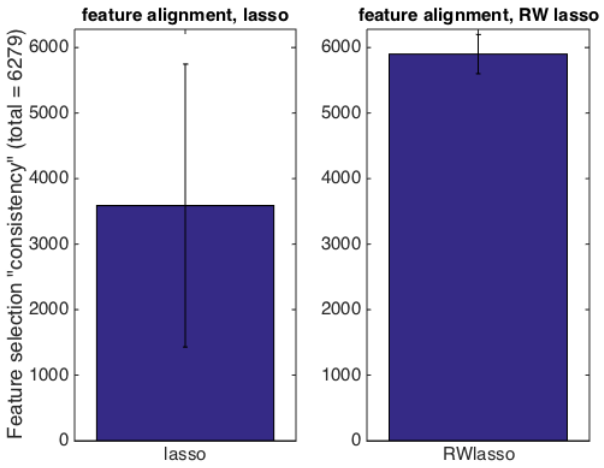
Feel free to use the reweighted lasso with constrained optimization I provided. Make sure you modify the equality constraint in the code. If you are not sure how to modify it, I have a hint under Appendix 1.

Since the data set is large, you do not need to run the full K-folds validations. The test set should be selected randomly though, since the fMRI data is a time series.

THE REWEIGHTED LASSO & THE NORMALIZATION ISSUE IN NEUROIMAGING DATA

The initial objective of this project is to alleviate the normalization issue. In neuroimaging analysis, normalizing the features often changes the feature selection result. In particular, the features selected by lasso using the raw data is different from features selected after feature normalization, which leads to problems when interpreting the results.

The reweighted lasso should be more resilient to the normalization influence, because it has less magnitude dependency (compared to lasso). The plot below is showing some preliminary results suggesting when comparing solutions estimated from the raw data versus the normalized data, the reweighted lasso solutions had better “alignment” than the regular lasso. In other words, which features did the reweighted lasso picked were less affected by feature normalization.



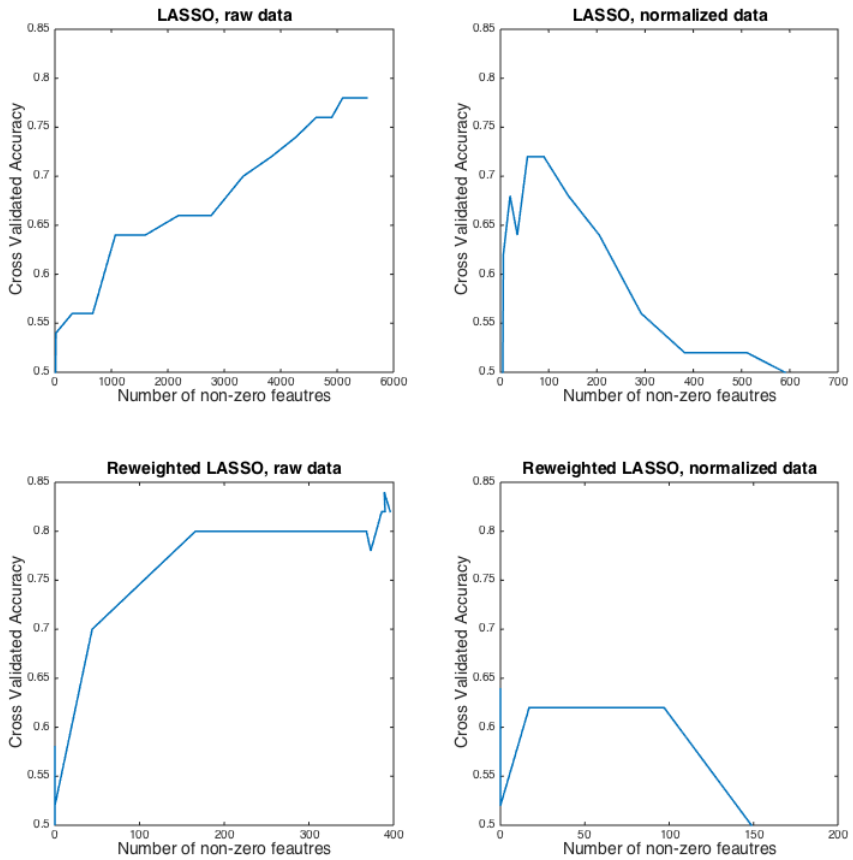
More specifically, here is what I did. First, I estimated $\hat{\beta}$ by using the two methods, for both raw data and normalized data. Then, I converted $\hat{\beta}$ to logical vectors, where "1" represents non-zero elements in the $\hat{\beta}$, and "0" represents zero-valued $\hat{\beta}$. The resulting logical vector

indicates which subset of features were selected (non-zero).

For each parameter value (such as lambda), I have the $\hat{\beta}$ vectors estimated from the raw data and the normalized data. I compared the "alignment" by counting how many elements in the two corresponding logical vectors agree. And I did this for both lasso and reweighted lasso.

To summarize, the plot above is showing the "feature alignment counts" for the two methods. The error bar indicates one sample standard deviation. The interpretation is that almost 6000 features (out of 6279) selected by reweighted lasso are aligned between raw data and normalized data, averaged across different parameter values. Compared to the regular lasso, the reweighted lasso solution had higher alignment and smaller variance.

However, further research is needed. The following plots show the accuracy and sparsity trade off.



The classification accuracy of the reweighted lasso and the regular lasso on the raw data are similar. However, the reweighted lasso used a solution with higher sparsity. In particular, the reweighted lasso used about 200 voxels to achieve a performance that is similar to lasso. Therefore, I would prefer the reweighted lasso in this case.

However, on the normalized data set, both algorithm are worse at classificaion. Importantly, the reweighted lasso worse than the regular lasso method. This is a caveat should be taken into account when thinking about the feature alignment results. Finally, the data that I plotted here came from a single subject. I will try to replicate this results on other nine subjects to establish a statistics to draw more definitive conclusion.

DISCUSSION & FUTURE DIRECTIONS

We have seen that the reweighted lasso can accurately reconstruct the true β, and it can yield solution with higher sparsity, compared to the regular lasso. It also has some weaknesses. Empirically, I found that the performance of the reweighted lasso is not very stable. In the simulation, occasionally, the reweighted lasso failed to reconstruct the true β vector. I would like to understand what properties in the data is affecting the performance of the reweighted lasso.

One possibility is multicollinearity, which is also a problem in fMRI data. Neuroimaging data typically have highly correlated features and examples. That is, many stimuli elicit similar neural responses and there are many voxels that behave similarly.

ACKNOWLEDGEMENTS

I would like to thank Professor Robert Nowak, Professor Laurent Lessard and Urvashi Oswal for their insightful suggestions about how to conduct simulations and how to understand the behavior of a given algorithm. I want to thank Professor Jarrod Lewis-Peacock and Professor Brad Postle for generously providing the data. I also want to thank you for the time you spent reading this document. I hope you found it to be interesting. Finally, I appreciate any question, comment and suggestion. Feel free to contact me via this email: `qihong.lu@wisc.edu`

References

Barton J.J., Press D.Z., Keenan J.P. & O'Connor M. (2002). Lesions of the fusiform face area impair perception of facial configuration in prosopagnosia. *Neurology, 58*(1):71-8.

Candès, E. J., Wakin, M. B., & Boyd, S. P. (2008). Enhancing Sparsity by Reweighted ℓ_1 Minimization. *Journal of Fourier Analysis and Applications, 14*(5-6), 877-905.

Cox, C. R., Lu, Q., & Rogers, T. T. (2015). Iterative Lasso: An even-handed approach to whole brain multivariate pattern analysis. *Poster presented at the 22nd Cognitive Neuroscience Society annual conference*, San Francisco, CA.

Lewis-Peacock, J. A., & Postle, B. R. (2008). Temporary Activation of Long-Term Memory Supports Working Memory. *Journal of Neuroscience, 28*(35), 8765-8771.

Figueiredo, M. A. T., Bioucas-Dias, J. M., & Nowak, R. D. (2007). Majorization & Minimization Algorithms for Wavelet-Based Image Restoration. *IEEE Transactions on Image Processing, 16*(12), 2980-2991.

Steeves, J. K. E., Culham, J. C., Duchaine, B. C., Pratesi, C. C., Valyear, K. F., Schindler, I., et al. (2006). The fusiform face area is not sufficient for face recognition: evidence from a patient with dense prosopagnosia and no occipital face area. *Neuropsychologia, 44*, 594-609.

APPENDIX 1: REWEIGHTED LASSO WITH CONSTRAINED OPTIMIZATION

I implemented the Reweighted Lasso algorithm with the CVX package. To use my code, you need to download the CVX package.

1. Download the CVX from here: `http://cvxr.com/cvx/download/`
2. Add CVX to your matlab search path (set path).
3. Type `cvx_setup` in your matlab console.

For more information, please refers to `http://cvxr.com/cvx/doc/install.html`

```
%% Solve Reweighted lasso algorithm with constrained optimization
% Note: this function has dependency on the CVX package
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Input:    X   m by n design matrix
%           y   m by 1 response
% Output:   beta    the coefficients for the features
%           history the beta and weights over iterations
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [beta, history] = reweightedLasso_cvx(X, y)
% constants
MAX_ITER = 50;
TOLERANCE = 1e-6;

% other parameters
e = .1;
n = size(X,2);
weights = ones(n,1);

%% iteratively solving the proxy of L0 minimization
for i = 1 : MAX_ITER
    % solve contrained l1 minimization
    cvx_begin quiet
        variable beta_hat(n)
        minimize( norm( weights .* beta_hat ,1) )
        subject to
            X * beta_hat == y
    cvx_end

    % update weights
    weights = 1 ./ (abs(beta_hat) + e);
```

```
    % save beta and weights
    history.beta(:,i) = beta_hat;
    history.weights (:,i) = weights;

    if i > 1
        diff = norm(history.beta(:,i) - history.beta(:,i-1),2);
        fprintf('Iteration: %4d \t betaChange: %12f \n', i, diff)
        % stopping criterion
        if diff < TOLERANCE
            break;
        end
    end
end
% extract final beta_estimates
beta = history.beta(:,end);
end
```

Note: to work with real data or simulation with noise, you might want to change the equality constraint "`X * beta_hat == y`" to "`norm(X * beta_hat - y) <= delta`". Forcing equality would leads to overfitting to the training set. Also, equality is very unlikely in practice.

APPENDIX 2: CODE TEMPLATE FOR LOADING THE NEUROIMAGING DATA

This is intended to be a code template for the 9^{th} exercise. It is loading a truncated data of one subject from the Lewis-Peacock and Postle (2008) data.

You will have the data for one of the subject in the Lewis-Peacock and Postle (2008) study. I truncated the number of columns by a half to make the size more manageable. Also, I am not making the data fully publically available because of privacy.

```
% load the fMRI data
clear variables;close all;clc
load('jlpData.mat')
% The data is already formatted into a matrix form
% X is a m by n matrix, where m corresponds to the number of picture
% and n corresponds to the number of voxels.
% y is a m by 1 vector consisting 1 and -1. 1 indicates that the
% participants was viewing a picture of face, -1 corresponds to a
% non-face stimulus.

%% set up cross validation blocks

%% fit lasso

%% fit reweighted lasso

%% compare lasso vs. reweighted lasso
```