Jacqueline Nugent
AMATH 584, Autumn 2020
Monday, December 14, 2020

Homework 6 (Midterm #2)

GitHub Repository: `https://github.com/jacnugent/amath584/tree/main/hw6_regression_and_sparsity`
(Python code is also reproduced at the end of this document.)

Figures are contained in a separate section at the end of the document.

Problem 1.

Using various $\mathbf{AX} = \mathbf{B}$ solvers, determine a mapping from the image space to the label space.

Solution:

Note that the `python-mnist` package (see here: `https://pypi.org/project/python-mnist/`) was used to read in the MNIST dataset. The code for this was taken from the method described in the first answer of this Stack Exchange discussion.

A total of five different solvers were used:

(a) Moore-Penrose Pseudo-inverse (`np.linalg.pinv`), SVD-based

(b) Lasso with $\lambda = 1.0$ (`sklearn.linear_model.Lasso`), l_1 regularization

(c) Lasso with $\lambda = 0.5$ (`sklearn.linear_model.Lasso(alpha=0.5)`), l_1 regularization

(d) Lasso with $\lambda = 0.1$ (`sklearn.linear_model.Lasso(alpha=0.1)`), l_1 regularization

(e) Ridge (`sklearn.linear_model.Ridge`), linear regression and l_2 regularization

To use these solvers, the MNIST testing labels were first converted into vectors as described in the homework description to get the matrix $\mathbf{B}$. However, $\mathbf{X}$ was taken to be the matrix of images and $\mathbf{A}$ was the matrix of model weights/loadings following the notation in the Neural Networks: 1-Layer Networks lecture from Friday, December 11, 2020.

First, the solvers were applied using the (60,000 x 1) list of labels rather than $\mathbf{B}$ so that only one matrix of weights was computed. These models then show the most important pixels for the digits collectively, which shows the overall structure and general characteristics shared by all digits. However, when applied to the test images, the models are quite inaccurate. Plots of the matrix $\mathbf{A}$ and the values of the loadings are shown in Figures 1 and 2, respectively. The accuracy (number of correct identifications/total number of images) is included in the title.

The Lasso models are also much less accurate than the pseudo-inverse. This makes sense because these models promote sparsity by including a penalty on the l_1 norm while the pseudo-inverse does not. When many of the weights are set to 0, there are few pixels to work with and it is difficult for the models to identify the major features.

The accuracy of these models improves greatly once the $\mathbf{B}$ matrix is used because this results in a (10 x 784) matrix for $\mathbf{A}$. Each row contains the weightings for each pixel for that number alone so the models are able to identify the separate features characteristic of each individual digit. Figure 3 shows the accuracy of each model when the 10 different features are identified for each digit. The pseudo-inverse model is slightly more accurate at 85% compared to 75-78% for the other models, but overall they all have comparable accuracy. This means that the models are able to correctly identify the pixels the majority of the time.

Figures 4-8 show the **A** matrices and the values of the loadings for each digit in each model. There is a clear structure for each digit apparent in the Lasso models, meaning that the model is identifying and weighing more heavily the pixels that show the general features of each digit. This is not the case for the pseudo-inverse and Ridge models, which both weight the pixels towards the edges more heavily.

Problem 2.

By promoting sparsity, determine and rank which pixels in the MNIST set are most informative for correctly labeling the digits. (You'll have to come up with your own heuristics or empirical rules for this. Use pcolor to help you visualize the results from **X**.

Solution:

We know that the models which penalize the l_1 norm will be more sparse than those that only consider the l_2 norm. This is obvious when the **A** matrices from Figures 4-8 are plotted with any zero elements omitted (Figures 9-13). For the pseudo-inverse and Ridge models, only some pixels around the edges and in the corners are equal to zero. However, for the Lasso models, most pixels are equal to zero except those in the center which form the general features for each digit. With decreasing values of λ, the l_1 norm penalty also decreases, so the **A** matrix becomes less and less sparse.

Here, we define the "most important" pixels as those with weights in the top 10%, i.e. $|\mathbf{A}_{i,j}| \geq$ (90th percentile of magnitudes). This metric was chosen because the 90th percentile is approximately where the values of the loadings start to exceed 0 for all digits in the Lasso method (Figure 14).

Problem 3.

Apply your most important pixels to the test data set to see how accurate you are with as few pixels as possible.

Solution:

Figures 15-19 show the values of the weights in the **A** matrices using only the most important pixels for all digits in each model. Even with this small number of pixels, the features for each number are still identified by the Lasso model. For example, the important pixels for 8 resemble a figure 8 and the important pixels for 0 resemble a circle.

The accuracy of each model using only the most important pixels (90th percentile or larger) is shown in Figure 20 (left). The right panel of Figure 20 shows a comparison of the accuracy of the Lasso method $\lambda = 1.0$ using various percentiles as thresholds for the most important pixels. Although higher thresholds use fewer pixels in the model, the accuracy decreases monotonically past the 90th percentile as fewer pixels are used in the identification.

Problem 4.

Redo the analysis with each digit individually to find the most important pixels for each digit.

Solution:

For the individual-digit analysis, the **B** matrix was computed for each digit by setting all elements containing that digit to 1 and all other elements equal to 0 in the original testing labels vector. Based on the results from Problems 2-3, only the Lasso model with $\lambda = 1.0$ is used for the analysis of each

digit for the remainder of this assignment in order to optimize sparsity and accuracy.

When all pixels are used for the matrix **A**, the loadings are equivalent to those shown in Figure 5. However, the accuracy is quite different depending on which metric is used. Here, we consider four metrics: the number of correct labels (e.g. the model correctly labels an image of 1 as 1), the number of incorrect labels (e.g. the model labeled an image of some other digit as 1), the incidence of "false positives" (number of incorrect labels / number of images in the testing data), and the incidence of "true positives" (number of correct labels / total number of images of that digit in the testing data). A summary of these metrics is shown in Figure 21. Table 1 lists the accuracies (#correct / (#incorrect + #correct)), false positive rate, and true positive rate for each digit. The individual-digit models correctly identify each digit almost every time, but the "false positive" rate is quite high because the models falsely identify too many images for each digit. Still, because the analysis is repeated for each digit, the "false positives" are not important. The model for each digit is still able to correctly label images of that digit over 90% of the time, so it is overall very accurate.

The digit-by-digit analysis is much more accurate than the analysis on all digits at once because the pixels used to determine the weights are specific to each number. The small amount of pixels that the model is trained on map to characteristic features that are specific to each digit, so the models are thus able to more accurately identify these characteristics. In contrast, when the models are trained on all digits at once, the limited number of pixels used have to represent the key features of all 10 digits in order to properly identify them.

Table 1: Values of the accuracy, true positive rate, and false positive rate (shown in Figure 21) for the Lasso model ($\lambda = 1.0$) trained on individual digit using all pixels.

Digit	Accuracy (%)	True Positive Rate (%)	False Positive Rate (%)
1	55.1	99.0	9.2
2	16.1	98.9	51.2
3	15.3	99.2	55.6
4	29.0	98.7	23.8
5	35.0	91.5	15.1
6	20.3	99.3	37.3
7	42.2	96.4	13.6
8	10.0	100	87.5
9	16.2	99.2	51.6
0	36.7	99.4	16.8

The most important pixels for each digit were found using the same method as in Problem 2. In this case, the 91st percentile for each digit was used because this was the point at which the loadings exceeded 0 for all digits (Figure 22).

Table 2 lists the accuracies (#correct / (#incorrect + #correct)), false positive rate, and true positive rate for each digit when only these most important pixels are used. Compared to using all pixels, using only the most important pixels results in comparable accuracy (Figure 23). As fewer pixels are used, the model accuracy tends to decrease. There is a slight improvement in accuracy for some digits at higher thresholds because the number of false positives decreases (Figure 23).

Table 2: Values of the accuracy, true positive rate, and false positive rate (shown in Figure 23, upper center panel) for the Lasso model ($\lambda = 1.0$) trained on individual digit using the top 9% of pixel weights.

Digit	Accuracy (%)	True Positive Rate (%)	False Positive Rate (%)
1	55.0	99.0	9.2
2	16.1	98.8	53.1
3	15.8	98.9	53.1
4	28.8	98.7	23.9
5	35.0	91.5	15.1
6	19.9	99.3	38.4
7	38.9	96.6	15.6
8	10.0	100	87.5
9	15.1	99.3	56.2
0	39.3	99.3	15.0

Figures

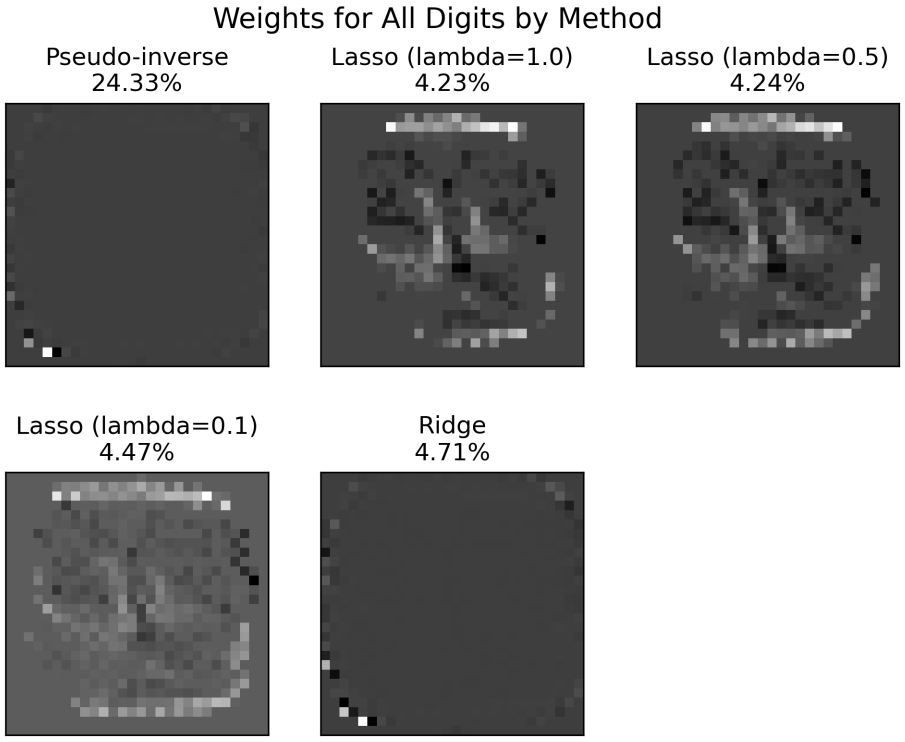


Figure 1: (Problem 1) Image of the single weight/loading matrix **A** for each method for all digits. The percentages indicate the accuracy (# correct/# total).

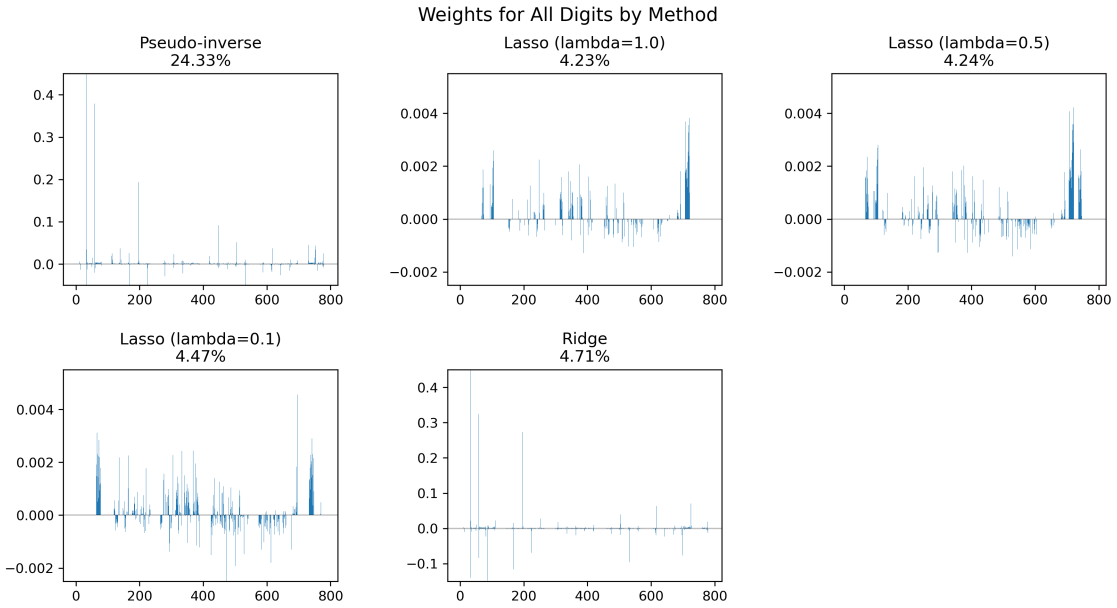


Figure 2: (Problem 1) Values of the weights for each model for all digits. The percentages indicate the accuracy (# correct/# incorrect).

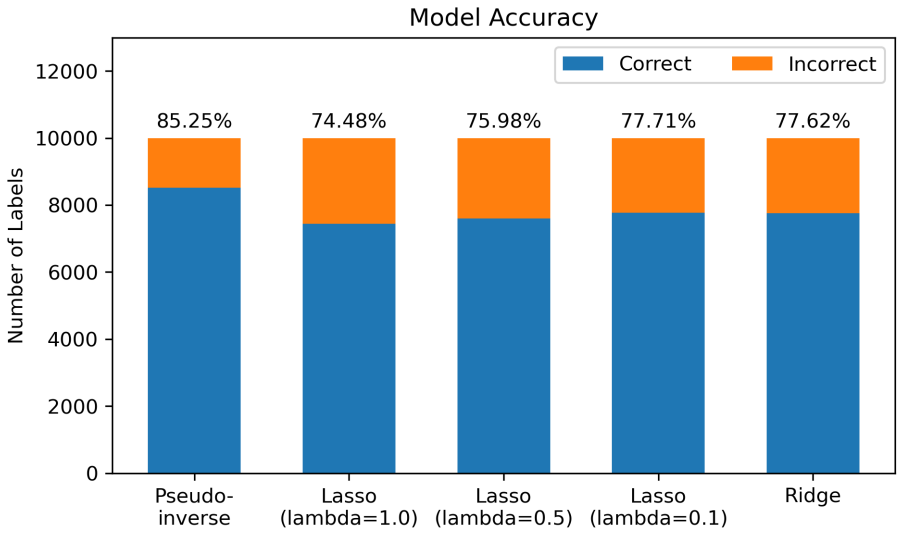
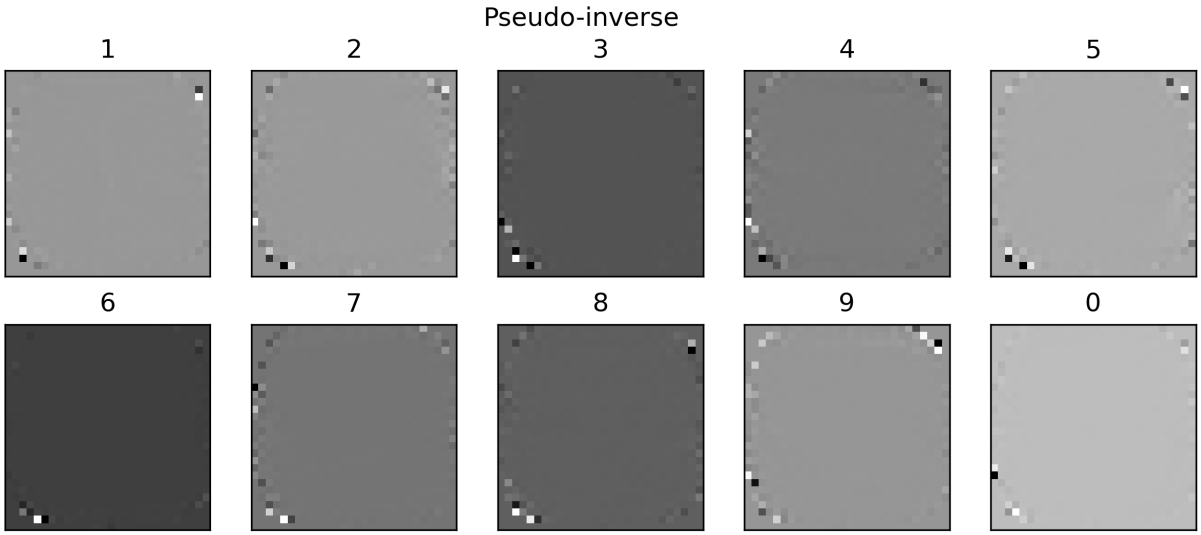


Figure 3: (Problem 1) Accuracy of each model (# correct/# total) for all digits using various $\mathbf{AX} = \mathbf{B}$ solvers.



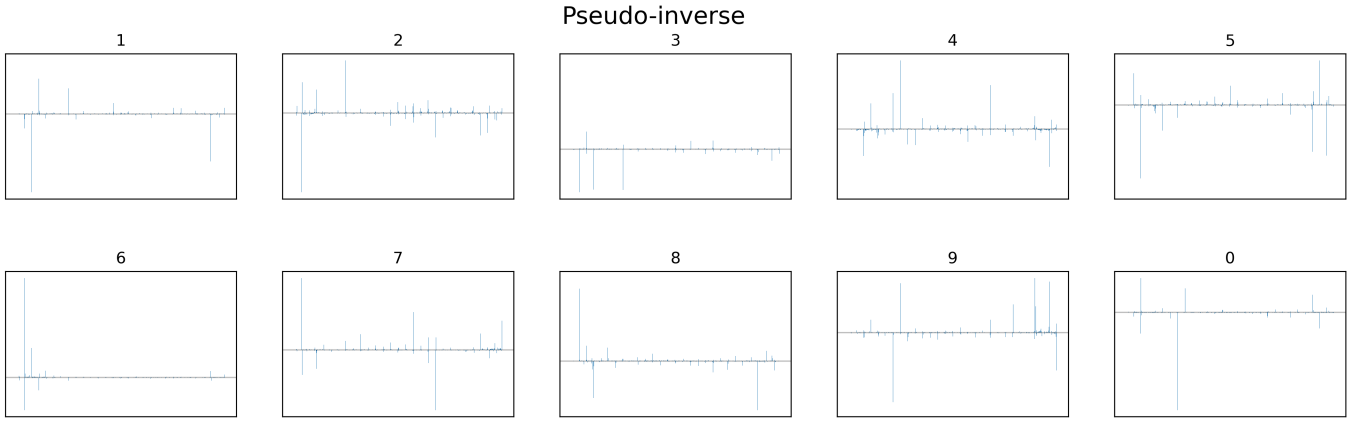


Figure 4: (Problem 2) Top: images of the weights matrix for each digit for the pseudo-inverse method using all pixels. Bottom: Values of the weights of each pixel.

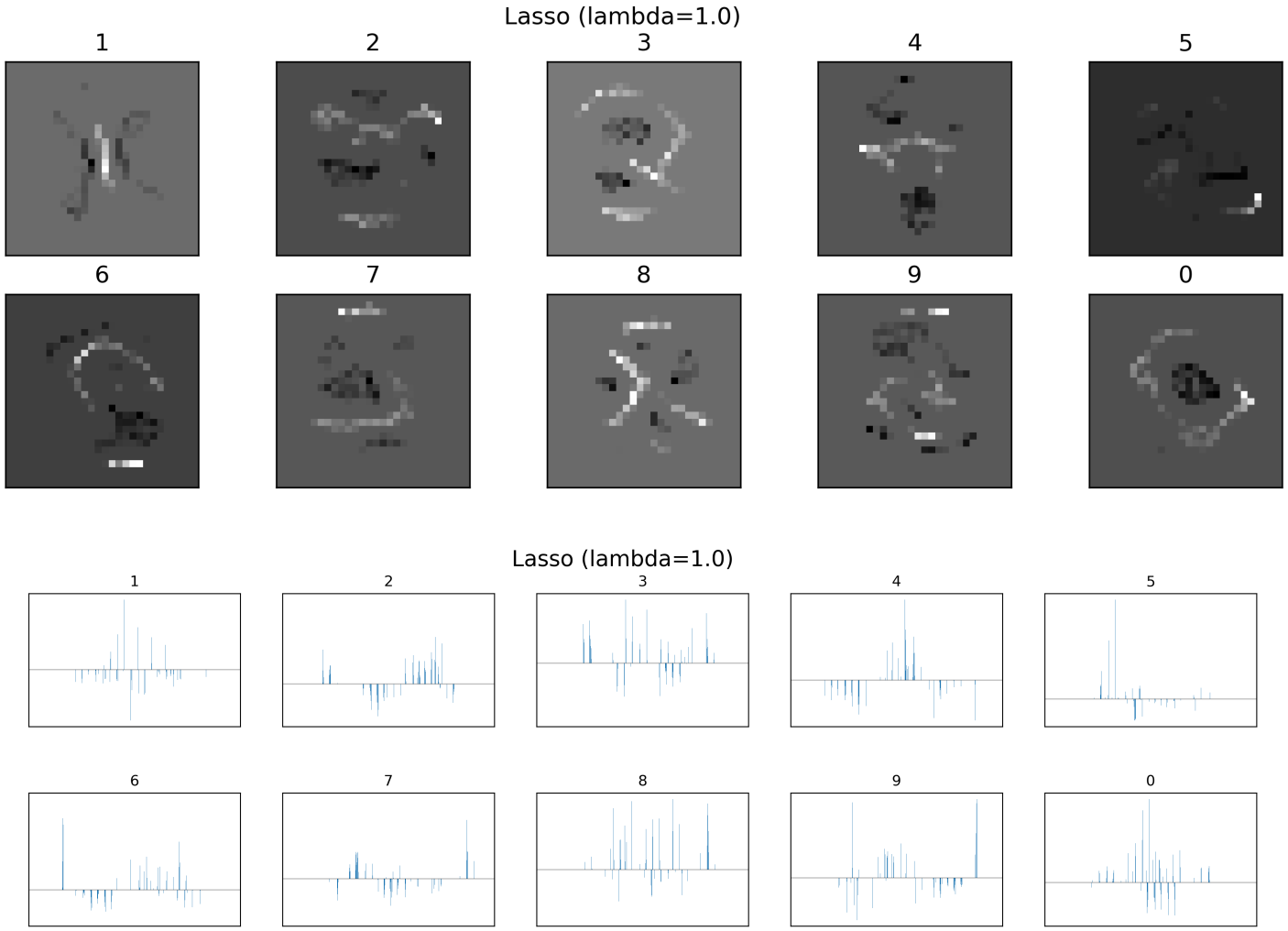


Figure 5: (Problem 2) Top: images of the weights matrix for each digit for the Lasso method (with $\lambda = 1.0$) using all pixels. Bottom: Values of the weights of each pixel.

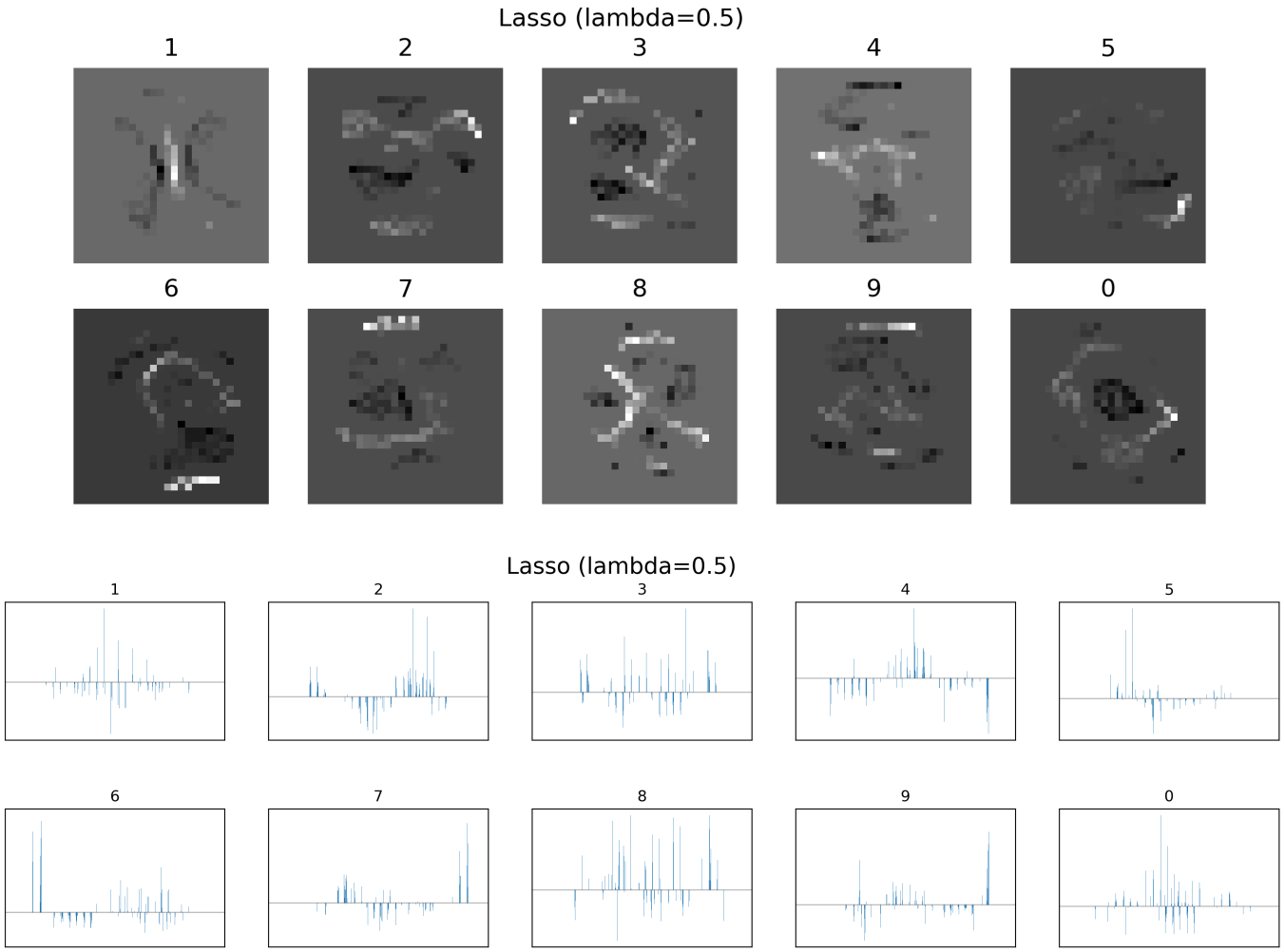
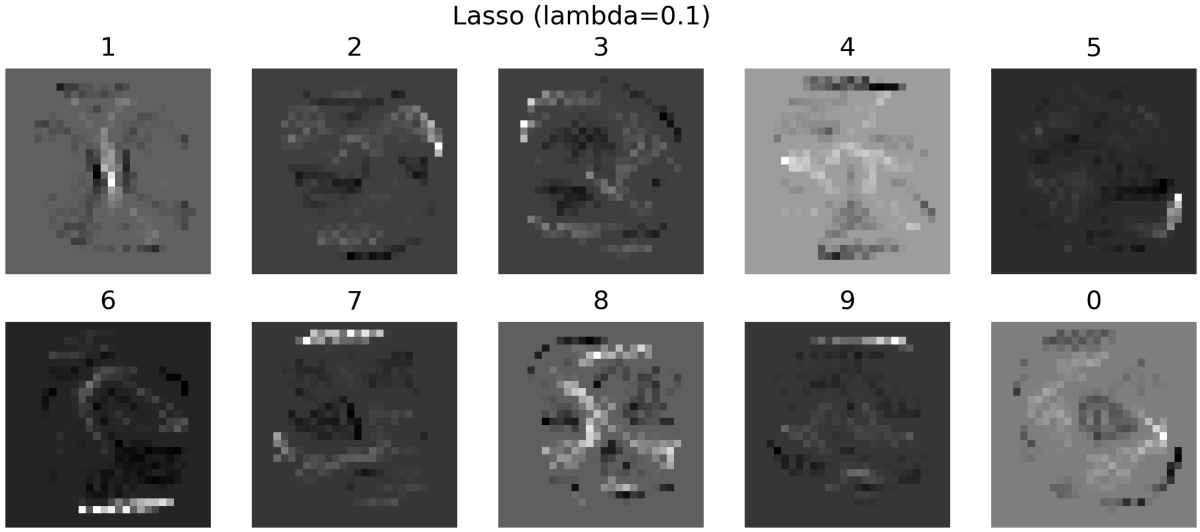


Figure 6: (Problem 2) Top: images of the weights matrix for each digit for the Lasso method (with $\lambda = 0.5$) using all pixels. Bottom: Values of the weights of each pixel.



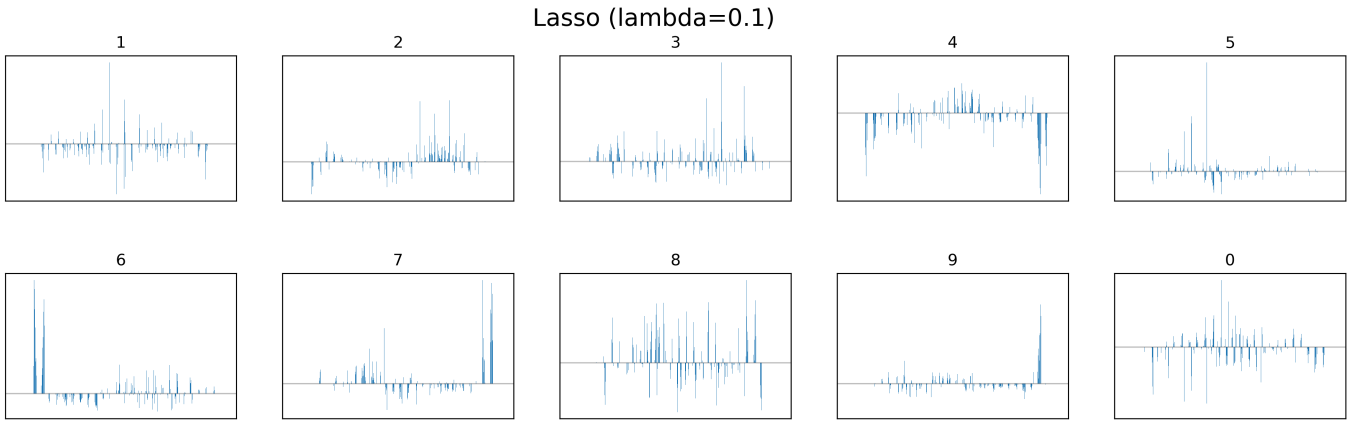


Figure 7: (Problem 2) Top: images of the weights matrix for each digit for the Lasso method (with $\lambda = 0.1$) using all pixels. Bottom: Values of the weights of each pixel.

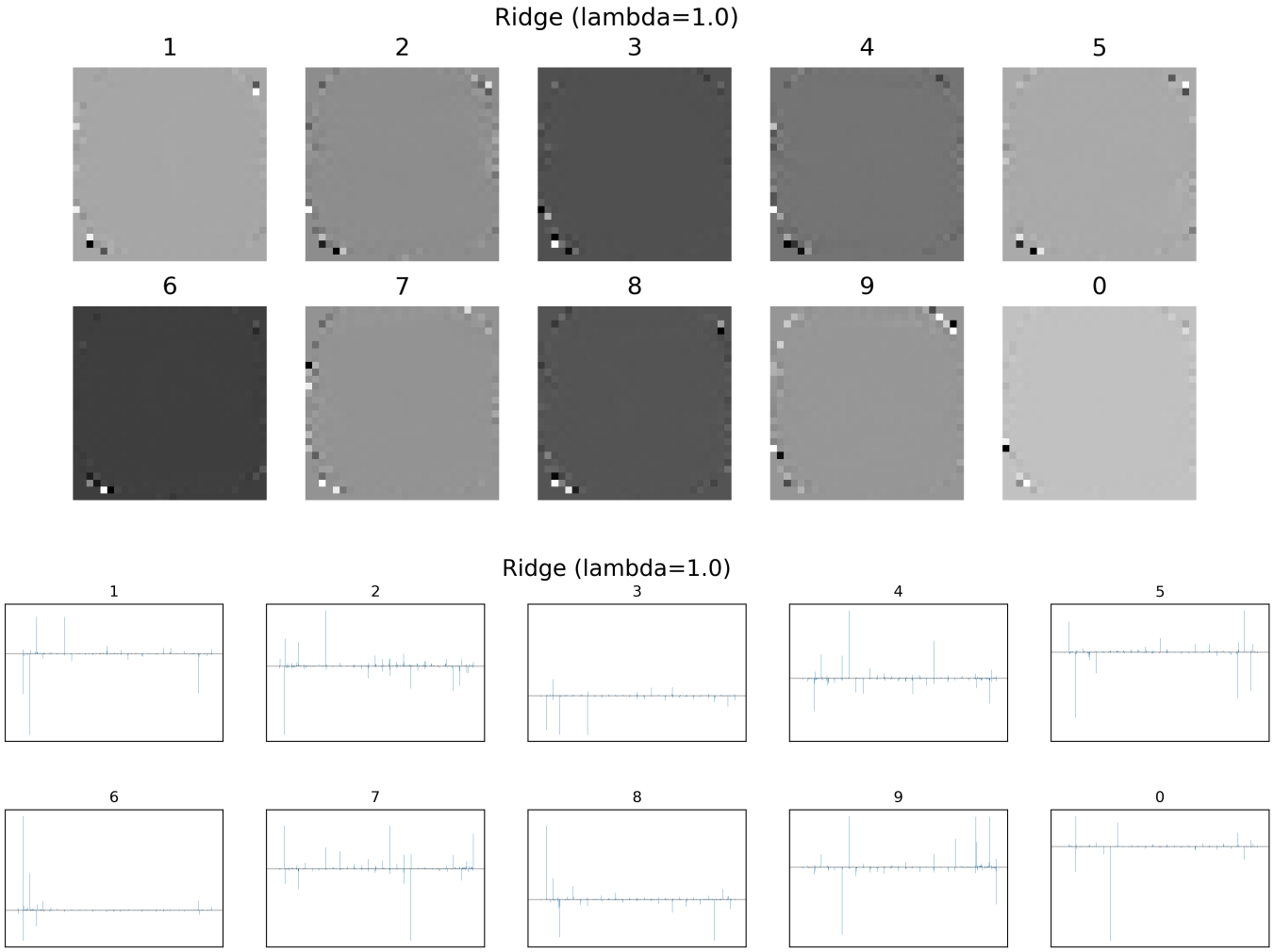


Figure 8: (Problem 2) Top: images of the weights matrix for each digit for the Ridge method using all pixels. Bottom: Values of the weights of each pixel.

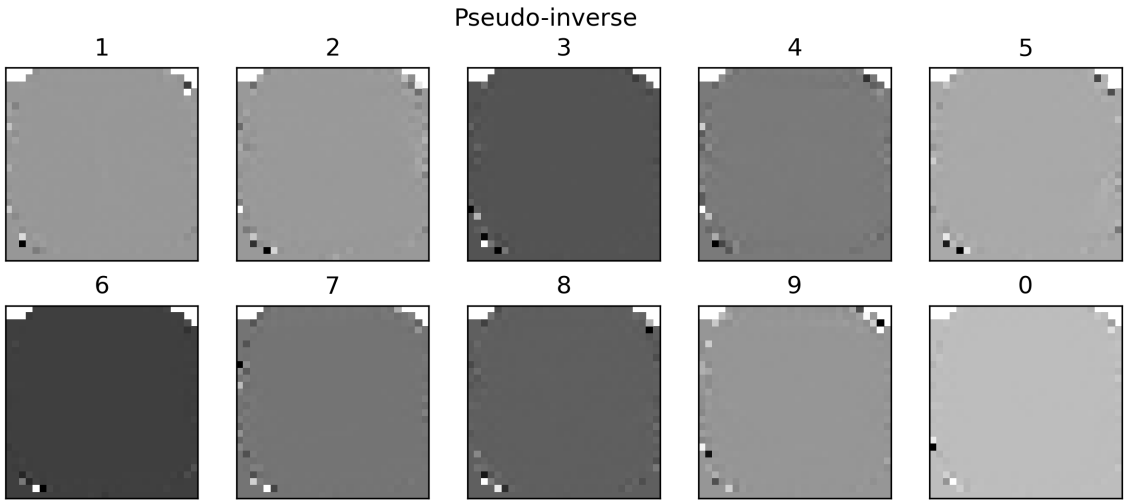


Figure 9: (Problem 2) The weight matrix from Figure 4 with zero elements omitted.

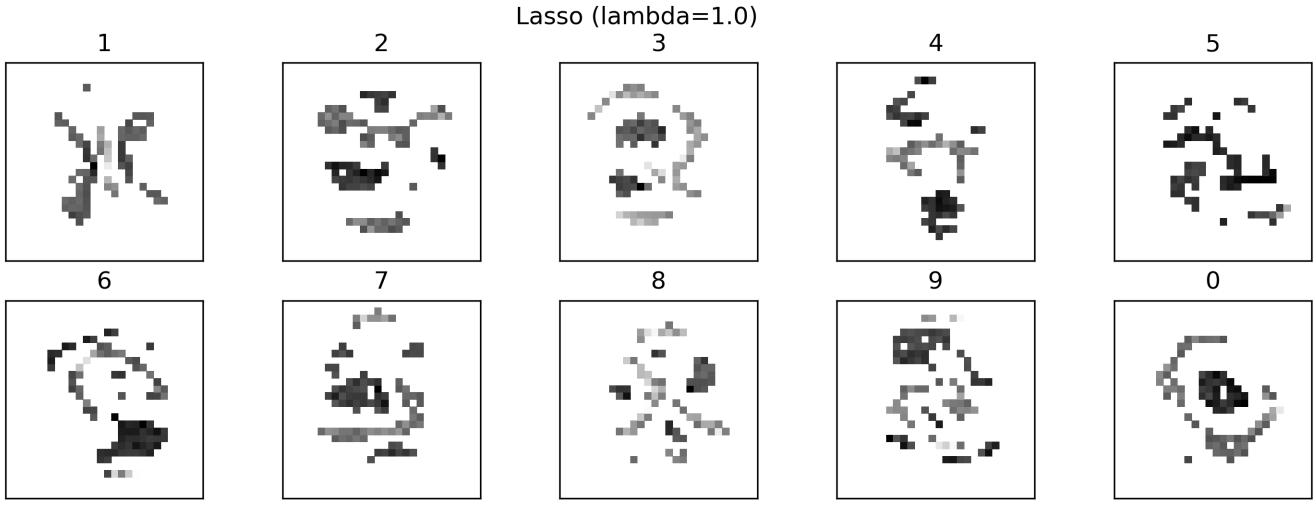


Figure 10: (Problem 2) The weight matrix from Figure 5 with zero elements omitted.

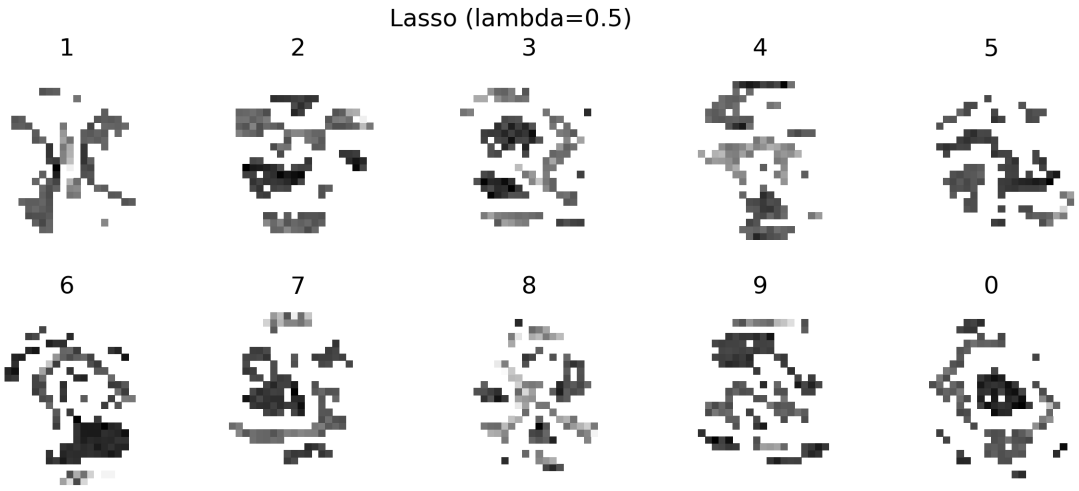


Figure 11: (Problem 2) The weight matrix from Figure 6 with zero elements omitted.

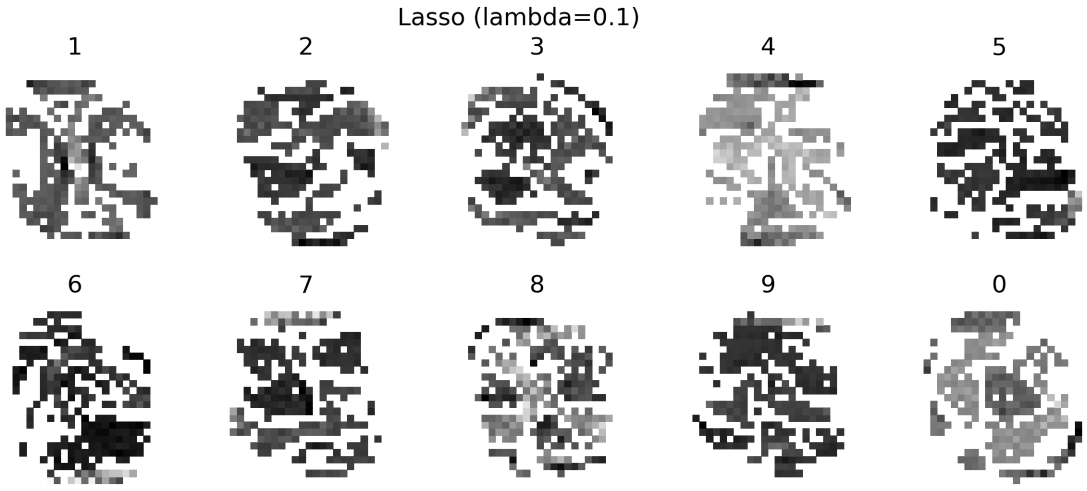


Figure 12: (Problem 2) The weight matrix from Figure 7 with zero elements omitted.

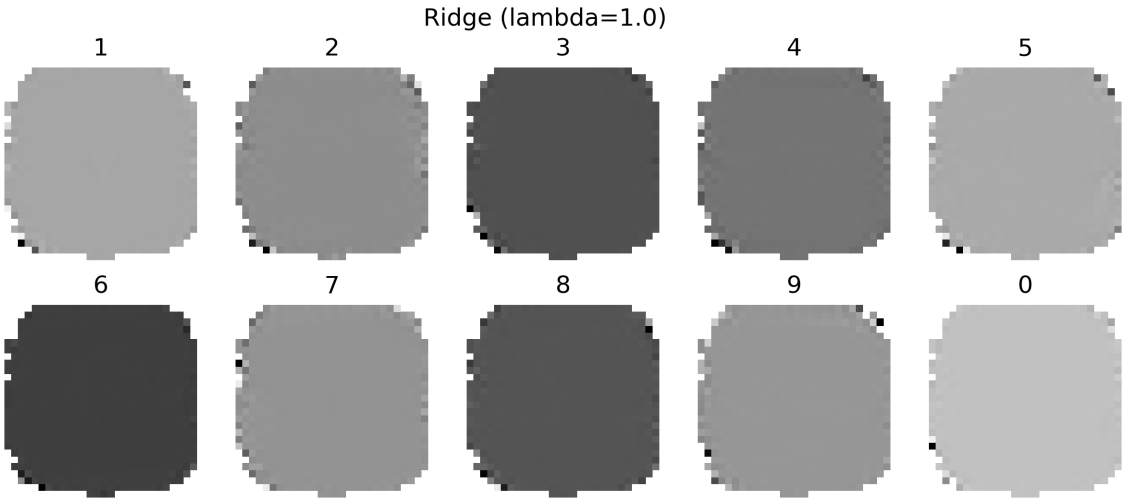


Figure 13: (Problem 2) The weight matrix from Figure 8 with zero elements omitted.

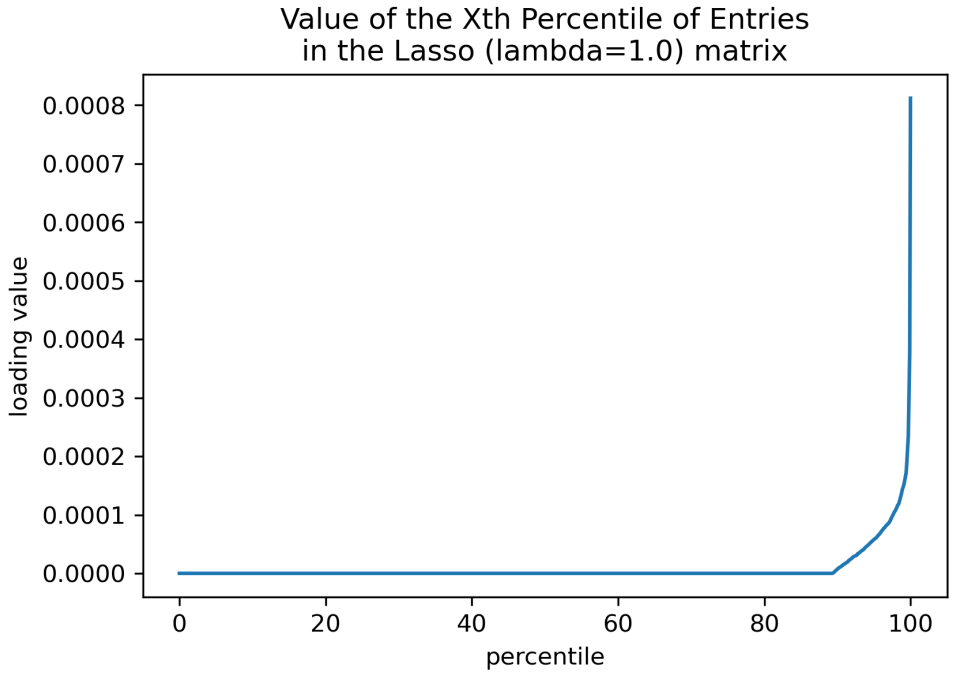


Figure 14: (Problem 2) Percentiles of the absolute values of the weights for the Lasso ($\lambda = 1.0$) matrix for all digits and all pixels.

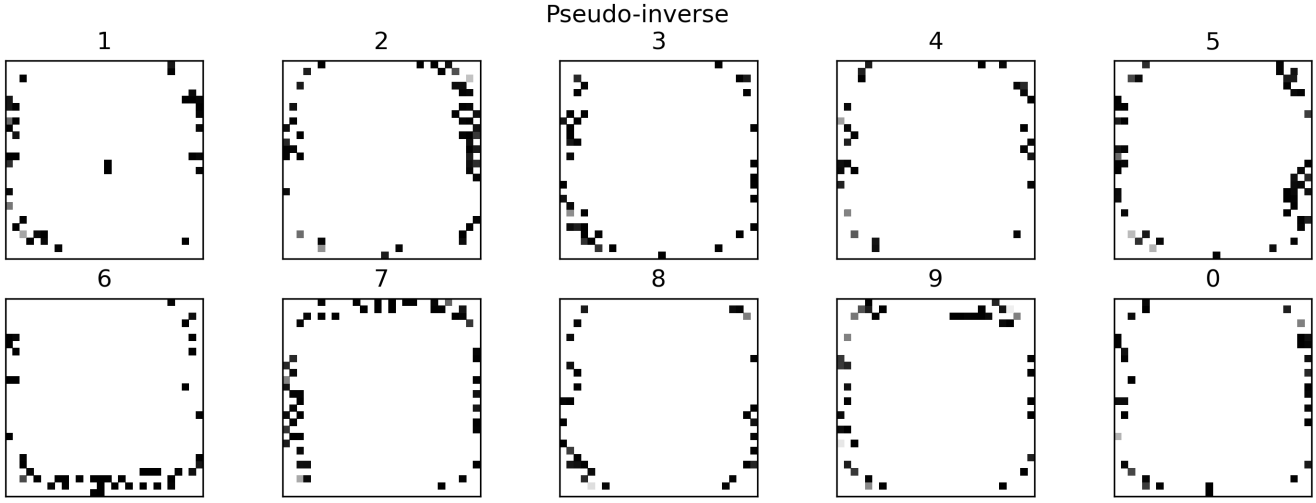


Figure 15: (Problem 3) Image of the weight matrices for each digit of the Pseudo-inverse matrix using only the top 10% of pixel weights.

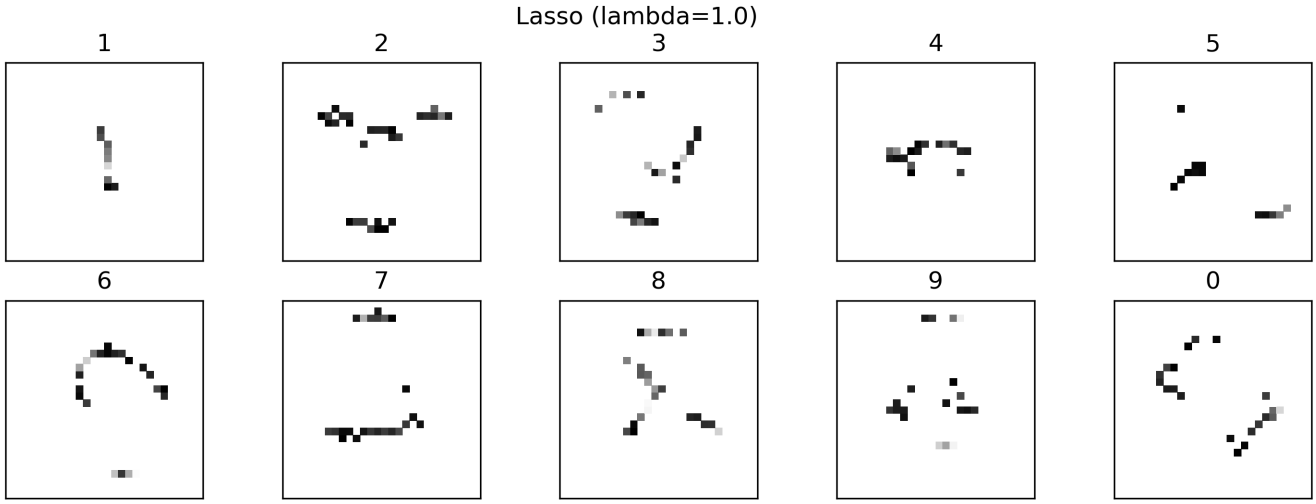


Figure 16: (Problem 3) Image of the weight matrices for each digit of the Lasso ($\lambda = 1.0$) matrix using only the top 10% of pixel weights.

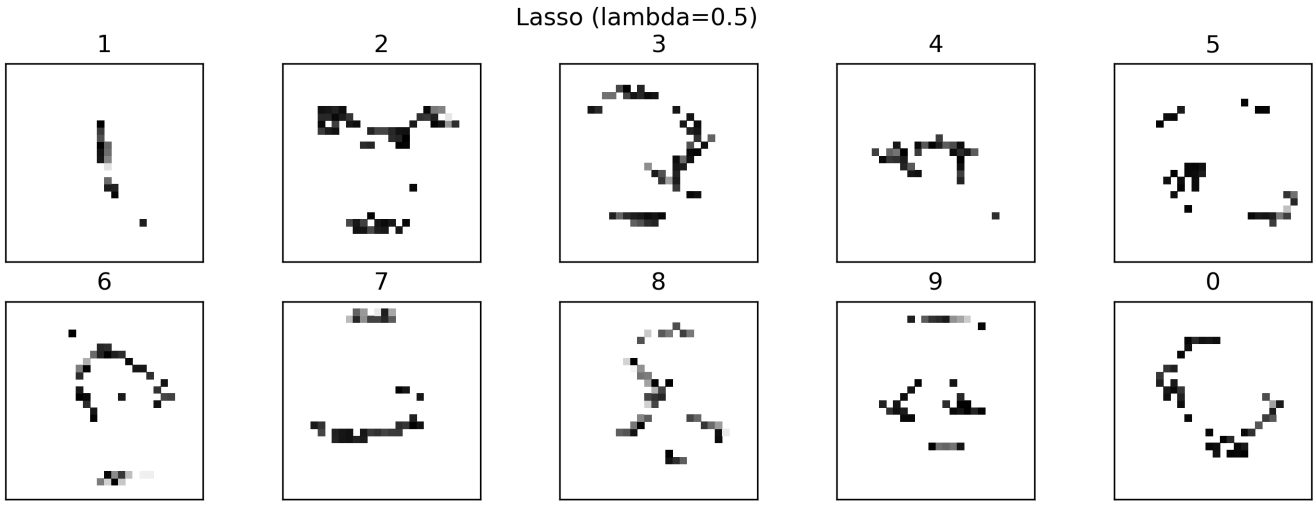


Figure 17: (Problem 3) Image of the weight matrices for each digit of the Lasso ($\lambda = 0.5$) matrix using only the top 10% of pixel weights.

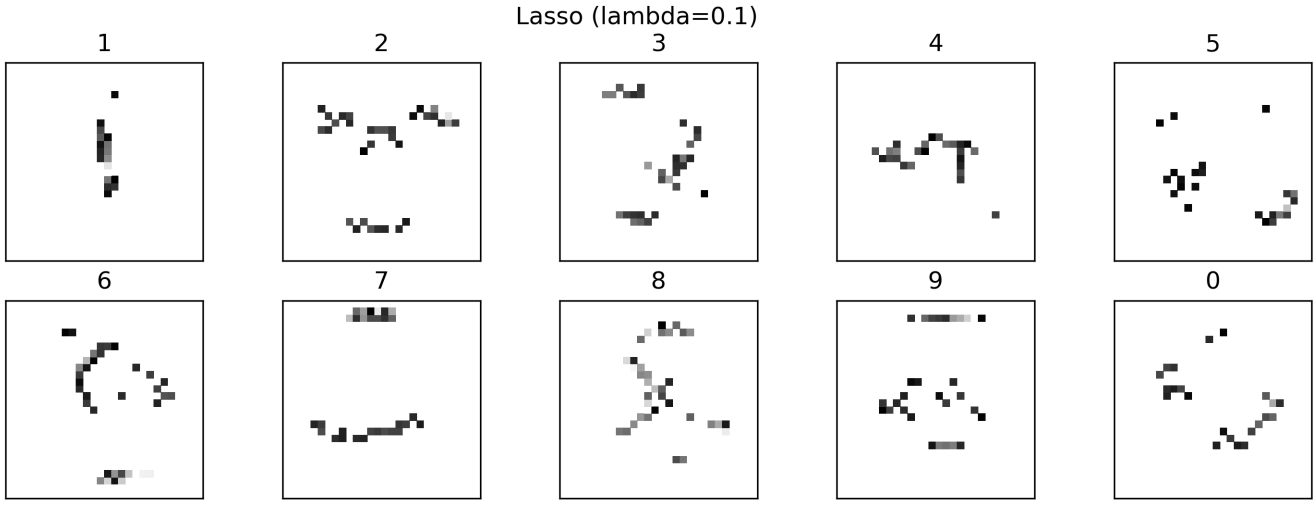


Figure 18: (Problem 3) Image of the weight matrices for each digit of the Lasso ($\lambda = 0.1$) matrix using only the top 10% of pixel weights.

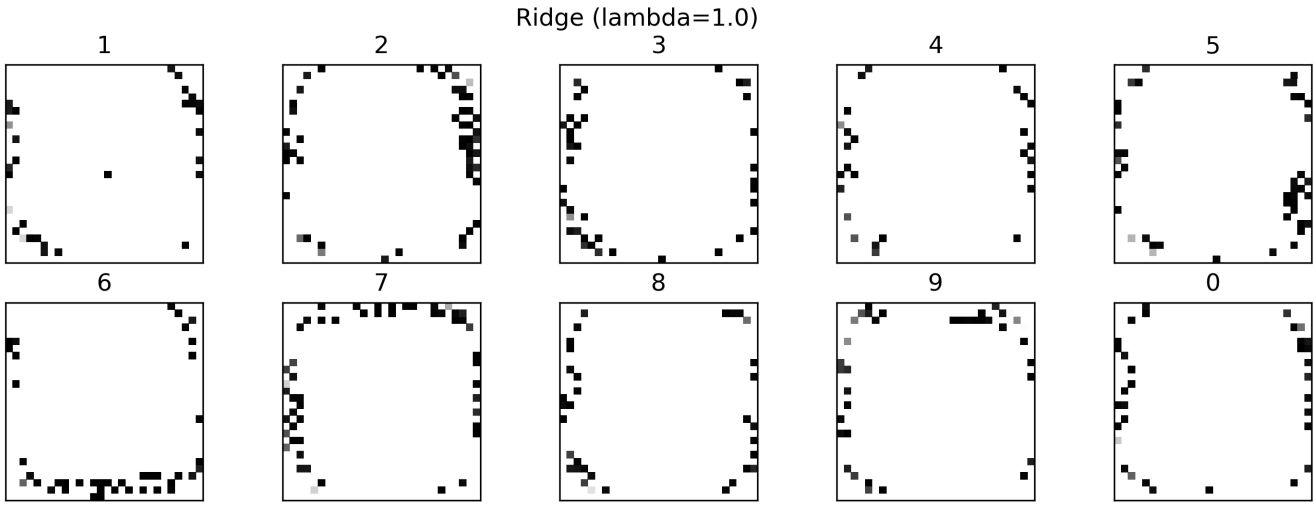


Figure 19: (Problem 3) Image of the weight matrices for each digit of the Ridge matrix using only the top 10% of pixel weights.

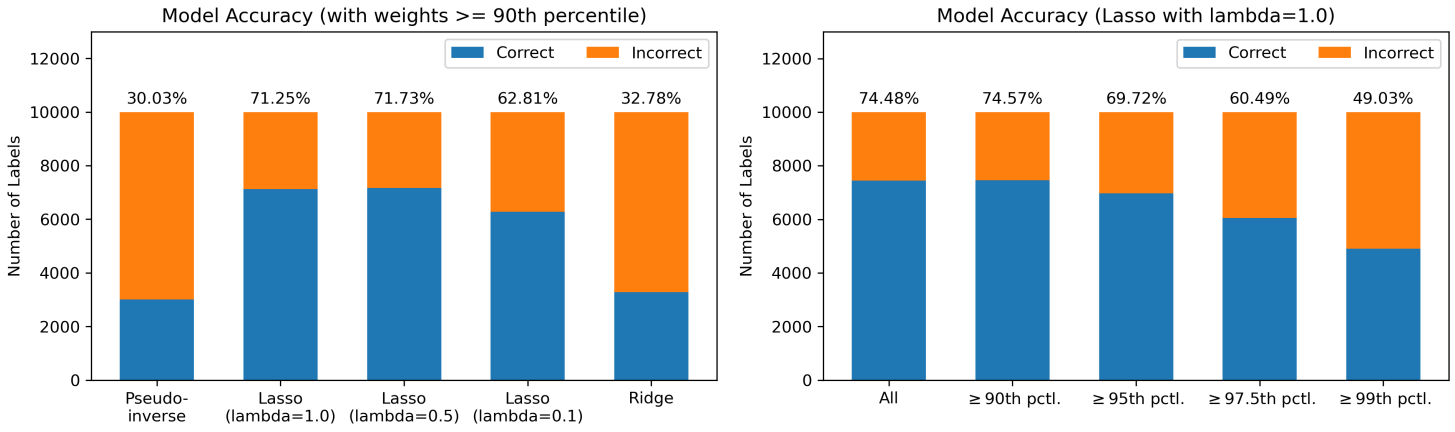


Figure 20: (Problem 3) Left: accuracy for each model using on the top 10% of pixel weights. Right: accuracy for the Lasso model ($\lambda = 1.0$) using various percentiles as cutoffs for the most significant pixel weights.

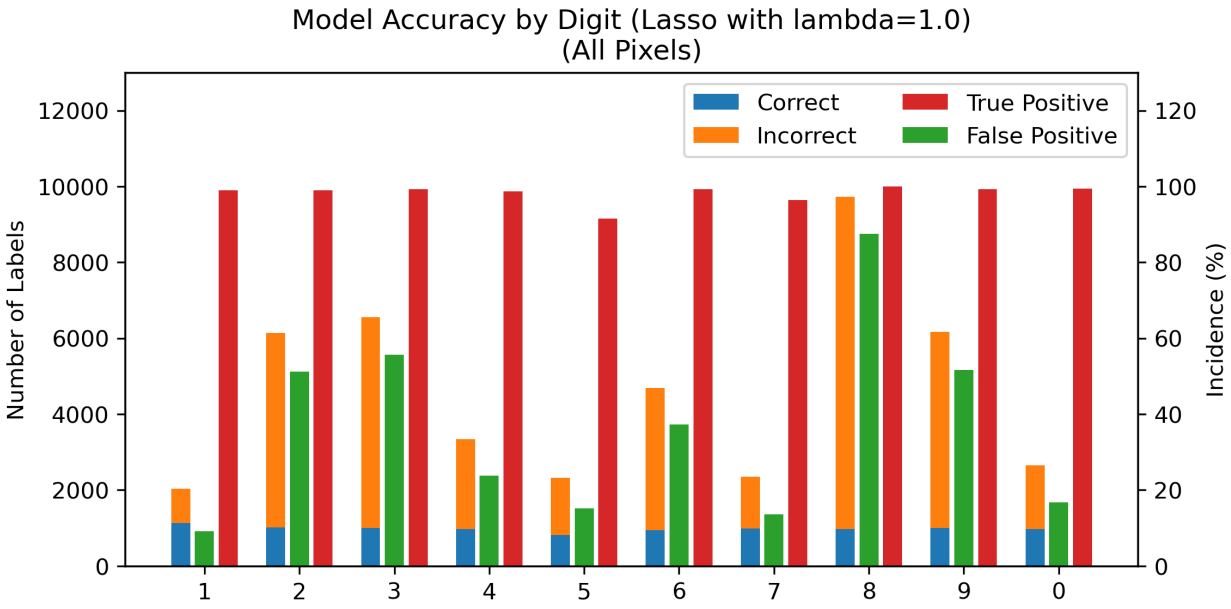


Figure 21: (Problem 4) Accuracy, true positive rate, and false positive rate of the Lasso model ($\lambda = 1.0$) trained on individual digits using all pixels.

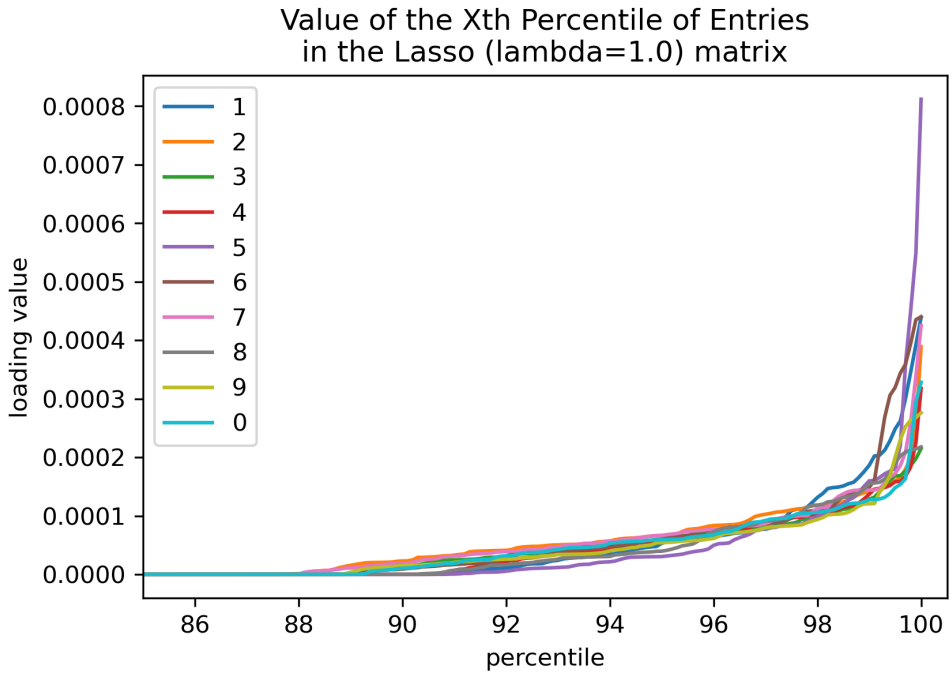


Figure 22: (Problem 4) Percentiles of the absolute values of the weights for the Lasso ($\lambda = 1.0$) matrix trained on each digit.

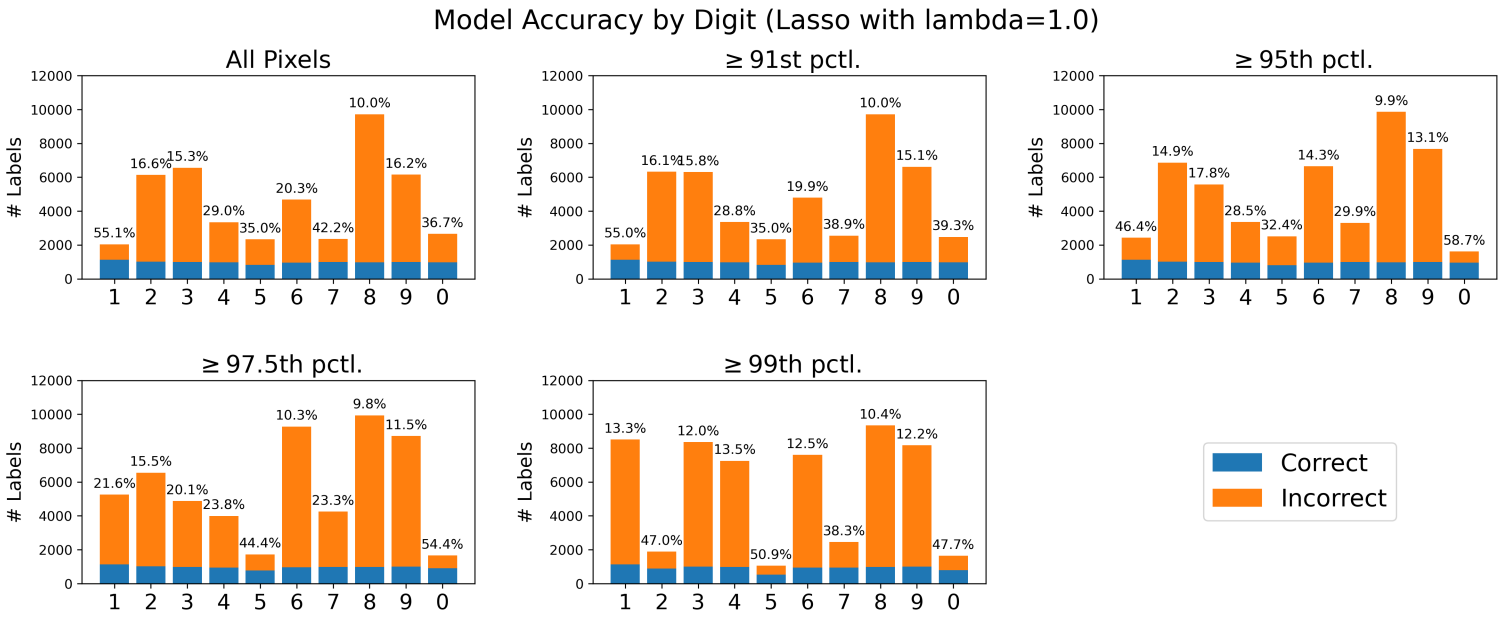


Figure 23: (Problem 4) Accuracy of the Lasso model ($\lambda = 1.0$) trained on individual digits using only the top 9% of pixel weights.

Python Code

(*hw6_midterm2.py* in the GitHub repository linked at the beginning of this document)

```
"""
hw6_midterm2.py

Python code for Homework 6 (Midterm #2),
AMATH 584, Fall 2020

Author: Jacqueline Nugent
Last Modified: December 14, 2020
"""
import random
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.patches as mpatches

from mnist import MNIST
from sklearn import linear_model
from tabulate import tabulate


################################################
#######s######   Set file paths   #############
################################################
# data available from http://yann.lecun.com/exdb/mnist/
file_dir =
    '/Users/jmnugent/Documents/__Year_3_2020-2021/AMATH_584-Numerical_Linear_Algebra/Homework/python/amath584/
data_dir = file_dir + 'data/'
save_dir = file_dir + 'figures/'


################################################
##### Some useful functions - used later ######
################################################

def convert_label(label):
    """Convert an integer label (0-9) into a vector.
    """
    vector = np.zeros(10)
    vector[label-1] = 1

    return vector


def reshape_pic(array, n=28):
    """Reshaped input array into nxn array
    """
    new = np.reshape(array, (n, n))

    return new


def set_label(vec):
    """
    Assigns the 0-9 label to the label vector based on which
    entry has the largest magnitude.
    """
```

```
    ind = np.where(np.abs(vec) == np.max(np.abs(vec)))[0][0]
    if ind == 9:
        label = 0
    else:
        label = ind+1

    return label


def check_accuracy(labels, truth):
    """
    Returns the number of correct and incorrect labels.
        labels = 2D matrix of size (10, #samples) of the modeled labels
        truth = 1D list of the actual labels (0-9)
    Ex: [n_right, n_wrong] = check_accuracy(lasso_labs, test_labs)
    """
    n_right = 0
    n_wrong = 0

    for i in range(labels.shape[1]):
        error = set_label(labels[:, i]) - truth[i]
        if error == 0:
            n_right += 1
        else:
            n_wrong += 1

    return [n_right, n_wrong]


def check_accuracy_all(labels, truth):
    """
    Returns the number of correct and incorrect labels (with 1 loading matrix)
        labels = 1D matrix of size (#samples) of the modeled labels
        truth = 1D list of the actual labels (0-9)
    Ex: [n_right, n_wrong] = check_accuracy(lasso_labs, test_labs)
    """
    n_right = 0
    n_wrong = 0

    labels_rounded = [int(np.round(x)) for x in labels]
    for i in range(len(labels_rounded)):
        if labels_rounded[i] == truth[i]:
            n_right += 1
        else:
            n_wrong += 1

    return [n_right, n_wrong]


def plot_loadings_bar(A, method, name, n=28, save=False,
                      save_dir=save_dir):
    """
    Bar plot of the loadings for each digit

        A = matrix of weights (10xn^2)
        method = method name (e.g. 'Lasso (alpha=0.1)')
        name = name for the file to save (e.g. 'lasso_01')
        n = dimension of picture in each direction (default=28 pixels)
```

```
        save = True if you want to save the image
        save_dir = directory to save the file to
    """
    if A.shape != (10, n**2):
        raise Exception('Weight matrix A has incorrect dimensions: {}'.format(A.shape))

    fig, axes = plt.subplots(2, 5, figsize=(18, 5))
    plt.subplots_adjust(hspace=0.5)

    for r in range(2):
        for c in range(5):
            ax = axes[r, c]

            if r == 0:
                i = c
            else:
                i = c+5

            ax.bar(np.arange(0, n**2), A[i, :].flatten())
            ax.axhline(0, linewidth=1, alpha=0.25, color='k')

            if i == 9:
                ax.set_title('0')
            else:
                ax.set_title(str(i+1))

            ax.get_xaxis().set_ticks([])
            ax.get_yaxis().set_ticks([])

    plt.suptitle(method, fontsize=18)

    if save:
        plt.savefig(save_dir + 'bar_plot_loadings_' + name + '.png', dpi=300,
                    bbox_inches='tight')

    plt.show()


def plot_weights(A, method, name, n=28, save=False, no_zero=False, save_dir=save_dir):
    """
    Plot (and save) the weights for each method.

        A = matrix of weights (10xn^2)
        method = method name (e.g. 'Lasso (alpha=0.1)')
        name = name for the file to save (e.g. 'lasso_01')
        n = dimension of picture in each direction (default=28 pixels)
        save = True if you want to save the image
        no_zero = plot by omitting zeros (to see sparsity)
        save_dir = directory to save the file to
    """
    if A.shape != (10, n**2):
        raise Exception('Weight matrix A has incorrect dimensions: {}'.format(A.shape))

    if no_zero:
        A = np.where(A==0, np.nan, A)
        name += '_no_zeros'

    fig, axes = plt.subplots(2, 5, figsize=(12, 4))
```

```
    for r in range(2):
        for c in range(5):
            ax = axes[r, c]

            if r == 0:
                i = c
            else:
                i = c+5

            ax.set_aspect('equal')
            ax.pcolormesh(reshape_pic(A[i, :], n=n), cmap='gray')

            if i == 9:
                ax.set_title('0')
            else:
                ax.set_title(str(i+1))

            ax.get_xaxis().set_ticks([])
            ax.get_yaxis().set_ticks([])

    plt.suptitle(method)

    if save:
        plt.savefig(save_dir + 'weight_matrix_' + name + '.png', dpi=300,
                    bbox_inches='tight')
    plt.show()

def plot_hist_nonzero(A, method, name, save=False, save_dir=save_dir,
                      xlim=(-.02, .02)):
    """
    Plot (and save) a histogram of the aboslute value of nonzero points of the matrix.

        A = matrix of weights (10xn^2)
        method = method name (e.g. 'Lasso (alpha=0.1)')
        name = name for the file to save (e.g. 'lasso_01')
        save = True if you want to save the image
        save_dir = directory to save the file to
        xlim = limits on x axis (default=(-.02, .02), for pinv)
    """
    fig, axes = plt.subplots(2, 5, figsize=(12, 4))
    plt.subplots_adjust(hspace=0.6, wspace=0.4)

    for r in range(2):
        for c in range(5):
            ax = axes[r, c]

            if r == 0:
                i = c
            else:
                i = c+5

            flat = A[i, :].flatten()
            nonzero = flat[flat != 0.0]
            ax.hist(nonzero, bins=100)

            ax.set_yscale('log')
            ax.set_ylim((1, 1000))
```

```
            ax.set_xlim(xlim)

            if i == 9:
                ax.set_title('0')
            else:
                ax.set_title(str(i+1))

    plt.suptitle('Histogram of nonzero loadings ({})'.format(method))

    if save:
        plt.savefig(save_dir + 'nonzero_hist_' + name + '.png', dpi=300,
                    bbox_inches='tight')

    plt.show()


def check_accuracy_digit(labels, truth):
    """
    Returns the number of correct and incorrect labels. # wrong is the
    number of times the model identifies the digit when it's not, not
    (len(truth) - n_right).
        labels = 2D matrix of size (10, #samples) of the modeled labels
        truth = 1D list of the actual labels (0-9)
    Ex: [n_right, n_wrong] = check_accuracy(lasso_digit_labels, test_labs)
    """
    n_right = 0
    n_wrong = 0

    for i in range(len(labels)):
        if truth[i] == 1 and labels[i] == 1:
                n_right += 1
        elif labels[i] == 1 and truth[i] != 1:
                n_wrong += 1

    return [n_right, n_wrong]


################################################
############ Load in the MNIST data ############
################################################
# this method comes from
    https://stackoverflow.com/questions/40427435/extract-images-from-idx3-ubyte-file-or-gzip-via-python
# and uses this package: https://pypi.org/project/python-mnist/
mndata = MNIST('data')

# there are 60000 items in each training object
# and 10000 in the testing
train_imgs, train_labs = mndata.load_training()
test_imgs, test_labs = mndata.load_testing()

# the imgs are now arrays of arrays of integers for each picture
# the labels are now numpy arrays
train_imgs = np.asarray(train_imgs)
test_imgs = np.asarray(test_imgs)
train_labs = np.asarray(train_labs)
test_labs = np.asarray(test_labs)

# check out the sizes
```

```
for x in [train_imgs, train_labs, test_imgs, test_labs]:
    print(x.shape)

# get the size and dimensions for each picture (n x n)
ntrain, n2 = train_imgs.shape
n = int(np.sqrt(n2))
ntest = test_imgs.shape[0]

# turn the list of labels into vectors
B_train = np.zeros((ntrain, 10))
B_test = np.zeros((ntest, 10))

for i in range(ntrain):
    B_train[i, :] = convert_label(train_labs[i])

for i in range(ntest):
    B_test[i, :] = convert_label(test_labs[i])

# plot the first few training images to see how they look
for i in range(6):
    ax = plt.subplot(2, 3, i+1)
    ax.imshow(reshape_pic(train_imgs[i]), cmap='gray')
    ax.set_title(train_labs[i])
    ax.axis('off')
plt.show()

# plot the first few testing images to see how they look
for i in range(6):
    ax = plt.subplot(2, 3, i+1)
    ax.imshow(reshape_pic(test_imgs[i]), cmap='gray')
    ax.set_title(test_labs[i])
    ax.axis('off')
plt.show()

# look at the shapes again
print(train_imgs.shape)
print(B_train.shape, '\n')
print(test_imgs.shape)
print(B_test.shape)


################################################
####### 1. Using various AX=B Solvers #########
################################################

# -------- (a) pseduo-inversse --------

# using the list of labels so you just get 1 loading matrix
# (tells you the overall structure/features of all digits)
A_pinv_all = train_labs @ np.linalg.pinv(train_imgs.T)
pinv_labs_all = A_pinv_all @ test_imgs.T

# how accurate was it?
[pinv_corr_all, pinv_wrong_all] = check_accuracy_all(pinv_labs_all, test_labs)
ac_lab_pinv_all = str(pinv_corr_all/(pinv_corr_all+pinv_wrong_all)*100.) + '%'
print(ac_lab_pinv_all + ' correct')

# using the B matrix so you get 1 loading matrix for each digit
```

```
A_pinv = B_train.T @ np.linalg.pinv(train_imgs.T)
pinv_labs = A_pinv @ test_imgs.T

# how accurate was it?
[pinv_corr, pinv_wrong] = check_accuracy(pinv_labs, test_labs)
print(str(pinv_corr/(pinv_corr+pinv_wrong)*100.) + '% correct')

# check out what the weights look like - should show you
# the important pixels!
plot_weights(A_pinv, 'Pseudo-inverse', 'pinv', save=True)
plot_weights(A_pinv, 'Pseudo-inverse', 'pinv', no_zero=True, save=True)

# some accuracy plots, etc...
plot_loadings_bar(A_pinv, 'Pseudo-inverse', 'pinv', save=True)
plot_hist_nonzero(A_pinv, 'Pseudo-inverse', 'pinv', save=True)


# -------- (b) lasso (lambda=1.0) --------

# using the list of labels so you just get 1 loading matrix
# (tells you the overall structure/features of all digits)
lasso_all = linear_model.Lasso().fit(train_imgs, train_labs)
A_lasso_all = lasso_all.coef_
lasso_labs_1_all = A_lasso_all @ test_imgs.T

# how accurate was it?
[lasso1_corr_all, lasso1_wrong_all] = check_accuracy_all(lasso_labs_1_all, test_labs)
ac_lab_lasso_all = str(np.round(lasso1_corr_all/(lasso1_corr_all+lasso1_wrong_all)*100., 2))
    + '%'
print(ac_lab_lasso_all + ' correct')

# using the B matrix
lasso = linear_model.Lasso().fit(train_imgs, B_train)
A_lasso = lasso.coef_
lasso_labs_1 = A_lasso @ test_imgs.T

# how accurate was it?
[lasso1_corr, lasso1_wrong] = check_accuracy(lasso_labs_1, test_labs)
print(str(lasso1_corr/(lasso1_corr+lasso1_wrong)*100.) + '% correct')

# check out what the weights look like - should show you
# the important pixels!
plot_weights(A_lasso, 'Lasso (lambda=1.0)', 'lasso_1', save=True)
plot_weights(A_lasso, 'Lasso (lambda=1.0)', 'lasso_1', no_zero=True, save=True)

# some accuracy plots, etc...
plot_loadings_bar(A_lasso, 'Lasso (lambda=1.0)', 'lasso_1', save=True)
plot_hist_nonzero(A_lasso, 'Lasso (lambda=1.0)', 'lasso_1', xlim=(-2e-4, 2e-4),
                save=True)


# -------- (c) lasso (lambda=0.5) --------

# using the list of labels so you just get 1 loading matrix
# (tells you the overall structure/features of all digits)
lasso_05_all = linear_model.Lasso(alpha=0.5).fit(train_imgs, train_labs)
A_lasso_05_all = lasso_05_all.coef_
lasso_labs_05_all = A_lasso_05_all @ test_imgs.T
```

```
# how accurate was it?
[lasso05_corr_all, lasso05_wrong_all] = check_accuracy_all(lasso_labs_05_all, test_labs)
ac_lab_lasso_05_all = str(lasso05_corr_all/(lasso05_corr_all+lasso05_wrong_all)*100.) + '%'
print(ac_lab_lasso_05_all + ' correct')

# using the B matrix
lasso_05 = linear_model.Lasso(alpha=0.5).fit(train_imgs, B_train)
A_lasso_05 = lasso_05.coef_
lasso_labs_05 = A_lasso_05 @ test_imgs.T

# how accurate was it?
[lasso05_corr, lasso05_wrong] = check_accuracy(lasso_labs_05, test_labs)
print(str(lasso05_corr/(lasso05_corr+lasso05_wrong)*100.) + '% correct')

# check out what the weights look like - should show you
# the important pixels!
plot_weights(A_lasso_05, 'Lasso (lambda=0.5)', 'lasso_05', save=True)
plot_weights(A_lasso_05, 'Lasso (lambda=0.5)', 'lasso_05', no_zero=True, save=True)

# some accuracy plots, etc...
plot_loadings_bar(A_lasso_05, 'Lasso (lambda=0.5)', 'lasso_05', save=True)
plot_hist_nonzero(A_lasso_05, 'Lasso (lambda=0.5)', 'lasso_05', xlim=(-2e-4, 2e-4),
                save=True)


# -------- (d) lasso (lambda=0.1) --------

# using the list of labels so you just get 1 loading matrix
# (tells you the overall structure/features of all digits)
lasso_01_all = linear_model.Lasso(alpha=0.1).fit(train_imgs, train_labs)
A_lasso_01_all = lasso_01_all.coef_
lasso_labs_01_all = A_lasso_01_all @ test_imgs.T

# how accurate was it?
[lasso01_corr_all, lasso01_wrong_all] = check_accuracy_all(lasso_labs_01_all, test_labs)
ac_lab_lasso_01_all = str(lasso01_corr_all/(lasso01_corr_all+lasso01_wrong_all)*100.) + '%'
print(ac_lab_lasso_01_all + ' correct')

# using the B matrix
lasso_01 = linear_model.Lasso(alpha=0.1).fit(train_imgs, B_train)
A_lasso_01 = lasso_01.coef_
lasso_labs_01 = A_lasso_01 @ test_imgs.T

# how accurate was it?
[lasso01_corr, lasso01_wrong] = check_accuracy(lasso_labs_01, test_labs)
print(str(lasso01_corr/(lasso01_corr+lasso01_wrong)*100.) + '% correct')

# check out what the weights look like - should show you
# the important pixels!
plot_weights(A_lasso_01, 'Lasso (lambda=0.1)', 'lasso_01', save=True)
plot_weights(A_lasso_01, 'Lasso (lambda=0.1)', 'lasso_01', no_zero=True, save=True)

# some accuracy plots, etc...
plot_loadings_bar(A_lasso_01, 'Lasso (lambda=0.1)', 'lasso_01', save=True)
plot_hist_nonzero(A_lasso_01, 'Lasso (lambda=0.1)', 'lasso_01', xlim=(-2e-4, 2e-4),
                save=True)
```

```
# -------- (d) ridge --------

# using the list of labels so you just get 1 loading matrix
# (tells you the overall structure/features of all digits)
ridge_all = linear_model.Ridge().fit(train_imgs, train_labs)
A_ridge_all = ridge_all.coef_
ridge_labs_all = A_ridge_all @ test_imgs.T

# how accurate was it?
[ridge_corr_all, ridge_wrong_all] = check_accuracy_all(ridge_labs_all, test_labs)
ac_lab_ridge_all = str(ridge_corr_all/(ridge_corr_all+ridge_wrong_all)*100.) + '%'
print(ac_lab_ridge_all + ' correct')

# using the B matrix
ridge = linear_model.Ridge().fit(train_imgs, B_train)
A_ridge = ridge.coef_
ridge_labs = A_ridge @ test_imgs.T

# how accurate was it?
[ridge_corr, ridge_wrong] = check_accuracy(ridge_labs, test_labs)
print(str(ridge_corr/(ridge_corr+ridge_wrong)*100.) + '% correct')

# check out what the weights look like - should show you
# the important pixels!
plot_weights(A_ridge, 'Ridge (lambda=1.0)', 'ridge', save=True)
plot_weights(A_ridge, 'Ridge (lambda=1.0)', 'ridge', no_zero=True, save=True)

# some accuracy plots, etc...
plot_loadings_bar(A_ridge, 'Ridge (lambda=1.0)', 'ridge', save=True)
plot_hist_nonzero(A_ridge, 'Ridge (lambda=1.0)', 'ridge', save=True)


# ----- Plots for when there is one A matrix for each method -----

### look at the A matrices ###

fig, axes = plt.subplots(2, 3, figsize=(8, 6))
plt.subplots_adjust(hspace=0.3)

ax1 = axes[0, 0]
ax1.set_aspect('equal')
ax1.pcolormesh(reshape_pic(A_pinv_all), cmap='gray')
ax1.set_title('Pseudo-inverse\n{}'.format(ac_lab_pinv_all))
ax1.get_xaxis().set_ticks([])
ax1.get_yaxis().set_ticks([])

ax2 = axes[0, 1]
ax2.set_aspect('equal')
ax2.pcolormesh(reshape_pic(A_lasso_all), cmap='gray')
ax2.set_title('Lasso (lambda=1.0)\n{}'.format(ac_lab_lasso_all))
ax2.get_xaxis().set_ticks([])
ax2.get_yaxis().set_ticks([])

ax3 = axes[0, 2]
ax3.set_aspect('equal')
ax3.pcolormesh(reshape_pic(A_lasso_05_all), cmap='gray')
ax3.set_title('Lasso (lambda=0.5)\n{}'.format(ac_lab_lasso_05_all))
```

```
ax3.get_xaxis().set_ticks([])
ax3.get_yaxis().set_ticks([])

ax4 = axes[1, 0]
ax4.set_aspect('equal')
ax4.pcolormesh(reshape_pic(A_lasso_01_all), cmap='gray')
ax4.set_title('Lasso (lambda=0.1)\n{}'.format(ac_lab_lasso_01_all))
ax4.get_xaxis().set_ticks([])
ax4.get_yaxis().set_ticks([])

ax5 = axes[1, 1]
ax5.set_aspect('equal')
ax5.pcolormesh(reshape_pic(A_ridge_all), cmap='gray')
ax5.set_title('Ridge\n{}'.format(ac_lab_ridge_all))
ax5.get_xaxis().set_ticks([])
ax5.get_yaxis().set_ticks([])

axes[1, 2].axis('off')

plt.suptitle('Weights for All Digits by Method', fontsize=14)

plt.savefig(save_dir + 'weights_matrix_accuracy_all_digits_all_methods.png', dpi=300,
    bbox_inches='tight')

plt.show()


### bar plot of loadings ###

fig, axes = plt.subplots(2, 3, figsize=(14, 7))
plt.subplots_adjust(hspace=0.4, wspace=0.4)

ax1 = axes[0, 0]
ax1.bar(np.arange(0, 784), A_pinv_all.flatten(), color='C0')
ax1.axhline(0, linewidth=1, alpha=0.25, color='k')
ax1.set_title('Pseudo-inverse\n{}'.format(ac_lab_pinv_all))

ax2 = axes[0, 1]
ax2.bar(np.arange(0, 784), A_lasso_all.flatten(), color='C0')
ax2.axhline(0, linewidth=1, alpha=0.25, color='k')
ax2.set_title('Lasso (lambda=1.0)\n{}'.format(ac_lab_lasso_all))

ax3 = axes[0, 2]
ax3.bar(np.arange(0, 784), A_lasso_05_all.flatten(), color='C0')
ax3.axhline(0, linewidth=1, alpha=0.25, color='k')
ax3.set_title('Lasso (lambda=0.5)\n{}'.format(ac_lab_lasso_05_all))

ax4 = axes[1, 0]
ax4.bar(np.arange(0, 784), A_lasso_01_all.flatten(), color='C0')
ax4.axhline(0, linewidth=1, alpha=0.25, color='k')
ax4.set_title('Lasso (lambda=0.1)\n{}'.format(ac_lab_lasso_01_all))

ax5 = axes[1, 1]
ax5.bar(np.arange(0, 784), A_ridge_all.flatten(), color='C0')
ax5.axhline(0, linewidth=1, alpha=0.25, color='k')
ax5.set_title('Ridge\n{}'.format(ac_lab_ridge_all))
```

```
ylim_1 = (-0.05, 0.45)
ylim_2 = (-0.0025, 0.0055)
ylim_3 = (-0.15, 0.45)

ax1.set_ylim(ylim_1)
ax2.set_ylim(ylim_2)
ax3.set_ylim(ylim_2)
ax4.set_ylim(ylim_2)
ax5.set_ylim(ylim_3)

axes[1, 2].axis('off')

plt.suptitle('Weights for All Digits by Method', fontsize=14)

plt.savefig(save_dir + 'bar_plot_loadings_all_digits_all_methods.png', dpi=300,
           bbox_inches='tight')

plt.show()


################################################
#### 2. Determine most informative pixels #####
################################################
# NOTE: the code for #2 and #3 may overlap between sections

### plot the overall accuracy ###

fig, ax = plt.subplots(figsize=(7, 4))

corr_counts = [pinv_corr, lasso1_corr, lasso05_corr, lasso01_corr, ridge_corr]
wrong_counts = [pinv_wrong, lasso1_wrong, lasso05_wrong, lasso01_wrong, ridge_wrong]

inds = np.arange(len(corr_counts))
width = 0.6

rects1 = ax.bar(inds, corr_counts, width, color='C0')
rects2 = ax.bar(inds, wrong_counts, width, bottom=corr_counts, color='C1')

ax.set_xticks(np.arange(0, len(corr_counts)))
ax.set_xticklabels(['Pseudo-\ninverse', 'Lasso\n(lambda=1.0)', 'Lasso\n(lambda=0.5)',
                   'Lasso\n(lambda=0.1)', 'Ridge'])

ax.set_ylabel('Number of Labels')
plt.legend([rects1[0], rects2[0]], ['Correct', 'Incorrect'], loc='upper right', ncol=2)
ax.set_title('Model Accuracy')
ax.set_ylim(0, 13000)


# label for accuracy
#
    https://matplotlib.org/3.1.1/gallery/lines_bars_and_markers/barchart.html#sphx-glr-gallery-lines-bars-and-
accuracy = [(corr_counts[i]/(corr_counts[i]+wrong_counts[i]))*100. for i in
    range(len(corr_counts))]
accuracy_labs = ['{:.2f}%'.format(x) for x in accuracy]
for i in range(len(rects2)):
    height = rects1[i].get_height() + rects2[i].get_height()
    ax.annotate(accuracy_labs[i],
               xy=(rects2[i].get_x() + rects2[i].get_width() / 2, height),
```

```
                xytext=(0, 3), textcoords='offset points',
                ha='center', va='bottom')

plt.savefig(save_dir + 'all_data_accuracy_comparison.png', dpi=300, bbox_inches='tight')

plt.show()


### look at the sparsity ###

def num_zeros(A):
    """ Print number of zero weights for each digit
    """
    num_zeros = np.zeros(10)
    for i in range(10):
        num_zeros[i] = len(A[i, :][A[i, :] == 0])

    print(num_zeros)

# from most to least sparse:
num_zeros(A_lasso)
num_zeros(A_lasso_01)
num_zeros(A_lasso_05)
num_zeros(A_ridge)
num_zeros(A_pinv)

############################################################
## 3. Apply your most important pixels to the test data ###
############################################################
# NOTE: the code for #2 and #3 may overlap between sections

### pick most important pixels by using some percentile as a threshold ###

pct = 90

pinv_pct = np.percentile(np.abs(A_pinv.flatten()), pct)
lasso_pct_1 = np.percentile(np.abs(A_lasso.flatten()), pct)
lasso_pct_05 = np.percentile(np.abs(A_lasso_05.flatten()), pct)
lasso_pct_01 = np.percentile(np.abs(A_lasso_01.flatten()), pct)
ridge_pct = np.percentile(np.abs(A_ridge.flatten()), pct)

new_A_pinv = np.where(A_pinv > pinv_pct, A_pinv, 0)
new_A_lasso_1 = np.where(A_lasso > lasso_pct_1, A_lasso, 0)
new_A_lasso_05 = np.where(A_lasso_05 > lasso_pct_05, A_lasso_05, 0)
new_A_lasso_01 = np.where(A_lasso_01 > lasso_pct_01, A_lasso_05, 0)
new_A_ridge = np.where(A_ridge > ridge_pct, A_ridge, 0)

new_pinv_labs = new_A_pinv @ test_imgs.T
new_lasso_labs = new_A_lasso_1 @ test_imgs.T
new_lasso_05_labs = new_A_lasso_05 @ test_imgs.T
new_lasso_01_labs = new_A_lasso_01 @ test_imgs.T
new_ridge_labs = new_A_ridge @ test_imgs.T

### accuracy with that subset ###

lab_list = [new_pinv_labs, new_lasso_labs, new_lasso_05_labs,
           new_lasso_01_labs, new_ridge_labs]
```

```
new_corr_counts = [[]]*len(lab_list)
new_wrong_counts = [[]]*len(lab_list)

for i in range(len(lab_list)):
    new_corr_counts[i], new_wrong_counts[i] = check_accuracy(lab_list[i], test_labs)

### plot it ###

pct=90
fig, ax = plt.subplots(figsize=(7, 4))

inds = np.arange(len(corr_counts))
width = 0.6

rects1 = ax.bar(inds, new_corr_counts, width, color='C0')
rects2 = ax.bar(inds, new_wrong_counts, width, bottom=new_corr_counts, color='C1')

ax.set_xticks(np.arange(0, len(new_corr_counts)))
ax.set_xticklabels(['Pseudo-\ninverse', 'Lasso\n(lambda=1.0)', 'Lasso\n(lambda=0.5)',
                    'Lasso\n(lambda=0.1)', 'Ridge'])

ax.set_ylabel('Number of Labels')
plt.legend([rects1[0], rects2[0]], ['Correct', 'Incorrect'], loc='upper right', ncol=2)
ax.set_title('Model Accuracy (with weights >= {}th percentile)'.format(pct))
ax.set_ylim(0, 13000)


# label for accuracy
#
    https://matplotlib.org/3.1.1/gallery/lines_bars_and_markers/barchart.html#sphx-glr-gallery-lines-bars-and-
accuracy = [(new_corr_counts[i]/(new_corr_counts[i]+new_wrong_counts[i]))*100. for i in
    range(len(new_corr_counts))]
accuracy_labs = ['{:.2f}%'.format(x) for x in accuracy]
for i in range(len(rects2)):
    height = rects1[i].get_height() + rects2[i].get_height()
    ax.annotate(accuracy_labs[i],
                xy=(rects2[i].get_x() + rects2[i].get_width() / 2, height),
                xytext=(0, 3), textcoords='offset points',
                ha='center', va='bottom')

plt.savefig(save_dir + 'geq_{}th_pct_accuracy_comparison.png'.format(pct), dpi=300,
    bbox_inches='tight')

plt.show()

### from most to least sparse with the cuttoff: ###

print('{}th percentile'.format(pct))
num_zeros(new_A_lasso_1)
num_zeros(new_A_lasso_01)
num_zeros(new_A_lasso_05)
num_zeros(new_A_ridge)
num_zeros(new_A_pinv)

### plot these "most important" pixels ###

plot_weights(new_A_pinv, 'Pseudo-inverse', 'pinv_geq_90th', no_zero=True, save=True)
plot_weights(new_A_lasso, 'Lasso (lambda=1.0)', 'lasso_1_geq_90th', no_zero=True, save=True)
```

```
plot_weights(new_A_lasso_05, 'Lasso (lambda=0.5)', 'lasso_05_geq_90th', no_zero=True,
    save=True)
plot_weights(new_A_lasso_01, 'Lasso (lambda=0.1)', 'lasso_01_geq_90th', no_zero=True,
    save=True)
plot_weights(new_A_ridge, 'Ridge (lambda=1.0)', 'ridge_geq_90th', no_zero=True, save=True)


### test the cutoff with Lasso (lambda=1.0) ###

xspace = np.arange(0, 100.1, .1)
plt.plot(xspace, np.percentile(np.abs(A_lasso).flatten(), xspace))
plt.title('Value of the Xth Percentile of Entries\nin the Lasso (lambda=1.0) matrix')
plt.xlabel('percentile')
plt.ylabel('loading value')

plt.savefig(save_dir + 'lasso_loading_percentiles-see_uptick_at_90.png',
           dpi=300, bbox_inches='tight')
plt.show()

### pick most important pixels by using some percentile as a threshold ###

lasso90 = np.percentile(np.abs(A_lasso.flatten()), 90)
lasso95 = np.percentile(np.abs(A_lasso.flatten()), 95)
lasso975 = np.percentile(np.abs(A_lasso.flatten()), 97.5)
lasso99 = np.percentile(np.abs(A_lasso.flatten()), 99)

A_lasso_90 = np.where(np.abs(A_lasso) >= lasso90, A_lasso, 0)
A_lasso_95 = np.where(np.abs(A_lasso) >= lasso95, A_lasso, 0)
A_lasso_975 = np.where(np.abs(A_lasso) >= lasso975, A_lasso, 0)
A_lasso_99 = np.where(np.abs(A_lasso) >= lasso99, A_lasso, 0)

lasso_labs_90 = A_lasso_90 @ test_imgs.T
lasso_labs_95 = A_lasso_95 @ test_imgs.T
lasso_labs_975 = A_lasso_975 @ test_imgs.T
lasso_labs_99 = A_lasso_99 @ test_imgs.T


lasso_lab_list = [lasso_labs_1, lasso_labs_90, lasso_labs_95, lasso_labs_975,
                  lasso_labs_99]

lasso_corr_counts = [[]]*len(lasso_lab_list)
lasso_wrong_counts = [[]]*len(lasso_lab_list)

for i in range(len(lasso_lab_list)):
    lasso_corr_counts[i], lasso_wrong_counts[i] = check_accuracy(lasso_lab_list[i], test_labs)

### plot accuracy by pctl ###

fig, ax = plt.subplots(figsize=(7, 4))

inds = np.arange(len(lasso_corr_counts))
width = 0.6

rects1 = ax.bar(inds, lasso_corr_counts, width, color='C0')
rects2 = ax.bar(inds, lasso_wrong_counts, width, bottom=lasso_corr_counts, color='C1')

ax.set_xticks(np.arange(0, len(lasso_corr_counts)))
ax.set_xticklabels(['All', '$\geq$90th pctl.', '$\geq$95th pctl.',
```

```
                '$\geq$97.5th pctl.', '$\geq$99th pctl.'])

ax.set_ylabel('Number of Labels')
plt.legend([rects1[0], rects2[0]], ['Correct', 'Incorrect'], loc='upper right', ncol=2)
ax.set_title('Model Accuracy (Lasso with lambda=1.0)')
ax.set_ylim(0, 13000)

# label for accuracy
#
    https://matplotlib.org/3.1.1/gallery/lines_bars_and_markers/barchart.html#sphx-glr-gallery-lines-bars-and-
accuracy = [(lasso_corr_counts[i]/(lasso_corr_counts[i]+lasso_wrong_counts[i]))*100. for i in
    range(len(lasso_corr_counts))]
accuracy_labs = ['{:.2f}%'.format(x) for x in accuracy]
for i in range(len(rects2)):
    height = rects1[i].get_height() + rects2[i].get_height()
    ax.annotate(accuracy_labs[i],
                xy=(rects2[i].get_x() + rects2[i].get_width() / 2, height),
                xytext=(0, 3), textcoords='offset points',
                ha='center', va='bottom')

plt.savefig(save_dir + 'lasso_pctl_accuracy_comparison.png', dpi=300, bbox_inches='tight')

plt.show()


############################################################
#### 4. Redo the analysis with each digit individually ####
############################################################

# -------- (a) all pixels, lasso (lambda=1.0) --------

### change the B vectors to be specific for each digit ###

B_0 = np.where(train_labs==0, 1, 0)
B_1 = np.where(train_labs==1, 1, 0)
B_2 = np.where(train_labs==2, 1, 0)
B_3 = np.where(train_labs==3, 1, 0)
B_4 = np.where(train_labs==4, 1, 0)
B_5 = np.where(train_labs==5, 1, 0)
B_6 = np.where(train_labs==6, 1, 0)
B_7 = np.where(train_labs==7, 1, 0)
B_8 = np.where(train_labs==8, 1, 0)
B_9 = np.where(train_labs==9, 1, 0)

B_list = [B_1, B_2, B_3, B_4, B_5, B_6, B_7, B_8,
          B_9, B_0]

### generate the loadings matrix for all digits (using all pixels) ###

A_lasso_digits = [[]]*10
lasso_digit_labels = [[]]*10

for i in range(10):
    lasso_digit = linear_model.Lasso().fit(train_imgs, B_list[i])
    A_lasso_digits[i] = lasso_digit.coef_
    lasso_digit_labels[i] = np.sign(A_lasso_digits[i] @ test_imgs.T)

# quick look at the A matrices (but they're exactly the same as before!)
```

```
for x in A_lasso_digits:
    plt.imshow(reshape_pic(x), cmap='gray')
    plt.show()

### plot accuracy for each digit using all pixels ###

test_labs_0 = np.where(test_labs == 0, 1, 0)
test_labs_1 = np.where(test_labs == 1, 1, 0)
test_labs_2 = np.where(test_labs == 2, 1, 0)
test_labs_3 = np.where(test_labs == 3, 1, 0)
test_labs_4 = np.where(test_labs == 4, 1, 0)
test_labs_5 = np.where(test_labs == 5, 1, 0)
test_labs_6 = np.where(test_labs == 6, 1, 0)
test_labs_7 = np.where(test_labs == 7, 1, 0)
test_labs_8 = np.where(test_labs == 8, 1, 0)
test_labs_9 = np.where(test_labs == 9, 1, 0)

test_lab_list = [test_labs_1, test_labs_2, test_labs_3, test_labs_4,
                 test_labs_5, test_labs_6, test_labs_7, test_labs_8,
                 test_labs_9, test_labs_0]

### plot the first few to see how they look ###

for j in range(10):
    if j == 9:
        num = 0
    else:
        num = j+1

    # indices where the picture is of digit #num
    digit_inds = np.where(test_lab_list[j] == 1)[0]
    not_digit_inds = np.where(test_lab_list[j] != 1)[0]

    # indices where there is a false positive of digit #num
    not_digit_inds = np.where(test_lab_list[j] != 1)[0]
    false_pos_inds = []
    for n in not_digit_inds:
        if lasso_digit_labels[j][n] == 1:
            false_pos_inds.append(n)

    for i in range(3):
        # randomly select a digit out of the list
        ind = random.choice(digit_inds)

        ax = plt.subplot(2, 3, i+1)
        ax.imshow(reshape_pic(test_imgs[ind]), cmap='gray')
        if lasso_digit_labels[j][ind] == 1:
            ax.set_title('Identified', fontsize=11)
        else:
            ax.set_title('Missed', fontsize=11)
        ax.axis('off')

    for k in range(3, 6):
        # randomly select a digit out of the list
        ind = random.choice(false_pos_inds)

        ax = plt.subplot(2, 3, k+1)
        ax.imshow(reshape_pic(test_imgs[ind]), cmap='gray')
```

```
        ax.set_title('Identified as {}'.format(num), fontsize=11)
        ax.axis('off')

    plt.suptitle('Random Sample of Lasso (lambda=1.0) Results for {}'.format(num),
                 fontsize=14)
    name = 'ALL_PIXELS_ex_of_correct_and_false_pos_for_{}'.format(num)
    plt.savefig(save_dir + name, dpi=300, bbox_inches='tight')

    plt.show()


### calculate accuracies ###

corr_list_digit = [[]]*10
wrong_list_digit = [[]]*10
accuracy_digit_list = [[]]*10
false_pos_list = [[]]*10
true_pos_list = [[]]*10

for i in range(10):
    if i == 9:
        n = 0
    else:
        n = i+1
    corr_list_digit[i], wrong_list_digit[i] = check_accuracy_digit(lasso_digit_labels[i],
                                                                   test_lab_list[i])

    # how often the digit was identified correctly and the not-digits were
    # identified correctly
    accuracy_digit_list[i] = corr_list_digit[i]/(corr_list_digit[i] + wrong_list_digit[i])*100.

    fp = 0
    tp = 0
    for j in range(ntest):
        if lasso_digit_labels[i][j] == 1 and test_lab_list[i][j] != 1:
            fp += 1
        elif lasso_digit_labels[i][j] == 1 and test_lab_list[i][j] == 1:
            tp += 1

    # how often the digit was identified when it wasn't the true label
    # and how often the digit was identified correctly
    false_pos_list[i] = fp/ntest*100.
    true_pos_list[i] = tp/np.sum(test_lab_list[i])*100.

    print('{n}: {a}% correct'.format(n=n, a=accuracy_digit_list[i]))
    print('{n}: {f}% false positives'.format(n=n, f=false_pos_list[i]))
    print('{n}: {t}% true positives'.format(n=n, t=true_pos_list[i]))


### bar chart for accuracy by digit ###

fig, ax = plt.subplots(figsize=(8, 4))

inds = np.arange(10)
width = 0.2

rects_corr = ax.bar(inds, corr_list_digit, width, color='C0')
rects_wrong = ax.bar(inds, wrong_list_digit, width, bottom=corr_list_digit, color='C1')
```

```
ax.set_xticks(np.arange(0.25, 10.25))
ax.set_xticklabels([*np.arange(1, 10), 0])
ax.set_ylabel('Number of Labels')
ax.set_ylim(0, 13000)

ax2 = ax.twinx()
ax2.set_ylabel('Incidence (%)')
ax2.set_ylim(0, 130)
rects_tp = ax2.bar(inds+0.5, true_pos_list, width, color='C3')
rects_fp = ax2.bar(inds+0.25, false_pos_list, width, color='C2')

plt.legend([rects_corr[0], rects_wrong[0], rects_tp[0], rects_fp[0]],
          ['Correct', 'Incorrect', 'True Positive', 'False Positive'],
          loc='upper right', ncol=2)
ax.set_title('Model Accuracy by Digit (Lasso with lambda=1.0)\n(All Pixels)')

plt.savefig(save_dir + 'DIGIT_ALL_PIXELS_lasso_accuracy_comparison.png', dpi=300,
    bbox_inches='tight')

plt.show()

### make a table of the percentages ###
dig_list = [*np.arange(1, 10), 0]
acc_labs_dig = ['{:.1f}%'.format(x) for x in accuracy_digit_list]
fp_labs = ['{:.1f}%'.format(x) for x in false_pos_list]
tp_labs = ['{:.1f}%'.format(x) for x in true_pos_list]

cols = [[]]*10
for i in range(10):
    cols[i] = [dig_list[i], acc_labs_dig[i], tp_labs[i], fp_labs[i]]

print('All Pixels:')
print(tabulate(cols, ['Digit', 'Accuracy', 'True Positive Rate', 'False Positive Rate'],
             tablefmt='fancy_grid'))


# -------- (b) pick the most important pixels for each digit --------

### test the cutoff ###

xspace = np.arange(0, 100.1, .1)

for i in range(10):
    if i == 9:
        num = 0
    else:
        num = i + 1
    plt.plot(xspace, np.percentile(np.abs(A_lasso_digits[i]).flatten(), xspace),
            label=num)

plt.legend()
plt.xlim((85, 100.5))
plt.title('Value of the Xth Percentile of Entries\nin the Lasso (lambda=1.0) matrix')
plt.xlabel('percentile')
plt.ylabel('loading value')

plt.savefig(save_dir + 'DIGIT_lasso_loading_percentiles-see_uptick_at_91.png',
```

```
          dpi=300, bbox_inches='tight')
plt.show()

### pick the most important pixels by using some percentile as a threshold ###

# pick most important pixels by using some percentile as a treshold

pct_list = [0, 91, 95, 97.5, 99]
digit_pcts = [[]]*len(pct_list)
new_A_digits = [[]]*len(pct_list)
new_digit_labs = [[]]*len(pct_list)

# 0
digit_pcts[0] = [np.percentile(np.abs(x.flatten()), pct_list[0])
                 for x in A_lasso_digits]
new_A_digits[0] = [np.where(np.abs(A_lasso_digits[i]) >= digit_pcts[0][i],
                            A_lasso_digits[i], 0)
                   for i in range(10)]
new_digit_labs[0] = [np.sign(new_A_digits[0][i] @ test_imgs.T)
                     for i in range(10)]

# 91
digit_pcts[1] = [np.percentile(np.abs(x.flatten()), pct_list[1])
                 for x in A_lasso_digits]
new_A_digits[1] = [np.where(np.abs(A_lasso_digits[i]) >= digit_pcts[1][i],
                            A_lasso_digits[i], 0)
                   for i in range(10)]
new_digit_labs[1] = [np.sign(new_A_digits[1][i] @ test_imgs.T)
                     for i in range(10)]

# 95
digit_pcts[2] = [np.percentile(np.abs(x.flatten()), pct_list[2])
                 for x in A_lasso_digits]
new_A_digits[2] = [np.where(np.abs(A_lasso_digits[i]) >= digit_pcts[2][i],
                            A_lasso_digits[i], 0)
                   for i in range(10)]
new_digit_labs[2] = [np.sign(new_A_digits[2][i] @ test_imgs.T)
                     for i in range(10)]

# 97.5
digit_pcts[3] = [np.percentile(np.abs(x.flatten()), pct_list[3])
                 for x in A_lasso_digits]
new_A_digits[3] = [np.where(np.abs(A_lasso_digits[i]) >= digit_pcts[3][i],
                            A_lasso_digits[i], 0)
                   for i in range(10)]
new_digit_labs[3] = [np.sign(new_A_digits[3][i] @ test_imgs.T)
                     for i in range(10)]

# 99
digit_pcts[4] = [np.percentile(np.abs(x.flatten()), pct_list[4])
                 for x in A_lasso_digits]
new_A_digits[4] = [np.where(np.abs(A_lasso_digits[i]) >= digit_pcts[4][i],
                            A_lasso_digits[i], 0)
                   for i in range(10)]
new_digit_labs[4] = [np.sign(new_A_digits[4][i] @ test_imgs.T)
                     for i in range(10)]
```

```
### get the % accuracy by choice of important pixels ###

corr_list_digit_pcts = [[]]*len(pct_list)
wrong_list_digit_pcts = [[]]*len(pct_list)
accuracy_digit_list_pcts = [[]]*len(pct_list)
false_pos_list_pcts = [[]]*len(pct_list)
true_pos_list_pcts = [[]]*len(pct_list)

for j in range(len(pct_list)):
    corr_list_dig = [[]]*10
    wrong_list_dig = [[]]*10
    accuracy_digit_list = [[]]*10
    false_pos_dig = [[]]*10
    true_pos_dig = [[]]*10

    for i in range(10):
        corr_list_dig[i], wrong_list_dig[i] = check_accuracy_digit(new_digit_labs[j][i],
                                                                   test_lab_list[i])

        # how often the digit was identified correctly and the not-digits were
        # identified correctly
        accuracy_digit_list[i] = corr_list_dig[i]/(corr_list_dig[i] + wrong_list_dig[i])*100.

        fp = 0
        tp = 0
        for k in range(ntest):
            if new_digit_labs[j][i][k] == 1 and test_lab_list[i][k] != 1:
                fp += 1
            elif new_digit_labs[j][i][k] == 1 and test_lab_list[i][k] == 1:
                tp += 1

        # how often the digit was identified when it wasn't the true label
        # and how often the digit was identified correctly
        false_pos_dig[i] = fp/ntest*100.
        true_pos_dig[i] = tp/np.sum(test_lab_list[i])*100.

    corr_list_digit_pcts[j] = corr_list_dig
    wrong_list_digit_pcts[j] = wrong_list_dig
    accuracy_digit_list_pcts[j] = accuracy_digit_list
    false_pos_list_pcts[j] = false_pos_dig
    true_pos_list_pcts[j] = true_pos_dig

### plot the % accuracy ###

fig, axes = plt.subplots(2, 3, figsize=(19, 7))
plt.subplots_adjust(hspace=0.5, wspace=0.3)

inds = np.arange(10)
width = 0.8

for r in range(2):
    for c in range(3):
        ax = axes[r, c]

        if r == 0:
            i = c
        else:
            i = c+3
```

```
        if i != 5:
            rects_corr = ax.bar(inds, corr_list_digit_pcts[i], width, color='C0')
            rects_wrong = ax.bar(inds, wrong_list_digit_pcts[i], width,
                bottom=corr_list_digit_pcts[i], color='C1')

            ax.set_xticks(np.arange(0, 10))
            ax.set_xticklabels([*np.arange(1, 10), 0], fontsize=16)
            ax.set_ylabel('# Labels', fontsize=14)
            ax.set_ylim(0, 12000)
            if i == 0:
                ax.set_title('All Pixels', fontsize=18)
            elif i == 1:
                ax.set_title('$\geq${}st pctl.'.format(pct_list[i]), fontsize=18)
            else:
                ax.set_title('$\geq${}th pctl.'.format(pct_list[i]), fontsize=18)

            # label for accuracy
            #
                https://matplotlib.org/3.1.1/gallery/lines_bars_and_markers/barchart.html#sphx-glr-gallery-line
            acc_labs_pct = ['{:.1f}%'.format(x) for x in accuracy_digit_list_pcts[i]]
            for j in range(len(rects_corr)):
                height = rects_corr[j].get_height() + rects_wrong[j].get_height()
                ax.annotate(acc_labs_pct[j],
                            xy=(rects_corr[j].get_x() + rects_wrong[j].get_width() / 2, height),
                            xytext=(0, 3), textcoords='offset points',
                            ha='center', va='bottom')

        else:
        # proxy legend
            ax.axis('off')
            corr_patch = mpatches.Patch(color='C0', label='Correct')
            wrong_patch = mpatches.Patch(color='C1', label='Incorrect')
            ax.legend(handles=[corr_patch, wrong_patch], fontsize=18,
                      loc='center')

plt.suptitle('Model Accuracy by Digit (Lasso with lambda=1.0)', fontsize=20)

plt.savefig(save_dir + 'DIGIT_PCTS_lasso_accuracy_comparison.png', dpi=300,
    bbox_inches='tight')

plt.show()

### make tables of percentages ###

# 91st
dig_list = [*np.arange(1, 10), 0]
acc_labs_dig = ['{:.1f}%'.format(x) for x in accuracy_digit_list_pcts[1]]
fp_labs = ['{:.1f}%'.format(x) for x in false_pos_list_pcts[1]]
tp_labs = ['{:.1f}%'.format(x) for x in true_pos_list_pcts[1]]

cols = [[]]*10
for i in range(10):
    cols[i] = [dig_list[i], acc_labs_dig[i], tp_labs[i], fp_labs[i]]

print('>= 91st percentile')
print(tabulate(cols, ['Digit', 'Accuracy', 'True Positive Rate', 'False Positive Rate'],
             tablefmt='fancy_grid'))
```

```
# 95th
dig_list = [*np.arange(1, 10), 0]
acc_labs_dig = ['{:.1f}%'.format(x) for x in accuracy_digit_list_pcts[2]]
fp_labs = ['{:.1f}%'.format(x) for x in false_pos_list_pcts[2]]
tp_labs = ['{:.1f}%'.format(x) for x in true_pos_list_pcts[2]]

cols = [[]]*10
for i in range(10):
    cols[i] = [dig_list[i], acc_labs_dig[i], tp_labs[i], fp_labs[i]]

print('>= 95th percentile')
print(tabulate(cols, ['Digit', 'Accuracy', 'True Positive Rate', 'False Positive Rate'],
              tablefmt='fancy_grid'))

# 97.5th
dig_list = [*np.arange(1, 10), 0]
acc_labs_dig = ['{:.1f}%'.format(x) for x in accuracy_digit_list_pcts[3]]
fp_labs = ['{:.1f}%'.format(x) for x in false_pos_list_pcts[3]]
tp_labs = ['{:.1f}%'.format(x) for x in true_pos_list_pcts[3]]

cols = [[]]*10
for i in range(10):
    cols[i] = [dig_list[i], acc_labs_dig[i], tp_labs[i], fp_labs[i]]

print('>= 97.5th percentile')
print(tabulate(cols, ['Digit', 'Accuracy', 'True Positive Rate', 'False Positive Rate'],
              tablefmt='fancy_grid'))

# 99th
dig_list = [*np.arange(1, 10), 0]
acc_labs_dig = ['{:.1f}%'.format(x) for x in accuracy_digit_list_pcts[4]]
fp_labs = ['{:.1f}%'.format(x) for x in false_pos_list_pcts[4]]
tp_labs = ['{:.1f}%'.format(x) for x in true_pos_list_pcts[4]]

cols = [[]]*10
for i in range(10):
    cols[i] = [dig_list[i], acc_labs_dig[i], tp_labs[i], fp_labs[i]]

print('>= 99th percentile')
print(tabulate(cols, ['Digit', 'Accuracy', 'True Positive Rate', 'False Positive Rate'],
              tablefmt='fancy_grid'))
```