Regression, Model Selection and DMD

Diego Alba, AMATh 563, 4/2020

Abstract:

The goal of this assignment is to determine a mapping from the image space to the label space of the MNIST dataset with different solvers/algorithms and regularizations and evaluate how each method performs. Moreover, we will investigate which pixels are most characteristic for each digit.

Background

The MNIST dataset contains 28x28 pixel images of handwritten digits, 0-9. Each image is associated with its corresponding digit label, i.e., an integer 0-9. The dataset has already been split into train and test sets, with 60k and 10k images and labels, respectively. A visualization of the images and labels is provided in the Implementation and Results section.

To determine the mapping between images and labels for each digit, we will use different solvers. Considering each image as vector of pixels such that the image space is described with a 60000x784 matrix, and the label space with a 60000x10 (where each label is now an orthogonal unit vector), we can propose a 784x10 mapping from images to labels. We will frame the problem of finding the best mapping as an optimization problem. The different methods we will use are the following:

- Least Squares: minimizing the squared Euclidean 2-norm, x st. min (b - A x)^2, where x is the mapping, A the images, and b the labels. Implemented through np.linalg.lstsq.
- Ridge: least squares with L2, or Ridge, regularization, x st. min (b - A x)^2 + x^2. Implemented through klearn.linear_model.Ridge.
- Lasso: least squares with L1, or Lasso, regularization, x st. min (b - A x)^2 + |x|. Implemented through klearn.linear_model.Lasso.
- Elastic Net: least squares with a mixture of L1 and L2 regularization, x st. min (b - A x)^2 + L|x| + (1-L)x^2, where L si the ratio of L1 regularization. Implemented through klearn.linear_model.ElasticNet.

The purpose of adding the regularization are twofold: to solve an ill-posed problem and to prevent overfitting. Where L1 makes the solution more sparse (most of the variables or weights will be 0), L2 makes all variables take small, non-zero values. The Elastic Net method tries to encompass the benefit of both regularizations. By promoting sparisty, the mapping will ideally be more interpretable.

In order to verify the validity of the model, we will employ k-fold cross validation. With this method, the test dataset will be split into k-folds and train solely in k-1 folds at a time, for a total of k times, changing which folds are used for training. During each iteration, the rest of the data is used for evaluating the model. By measuring the spread of the error, one is able to identify any data partition dependence bias, as well as to get a sense for how well the model will perform on new data.

```
HTML Settings, don't know how to remove these two blocks
```

```
Out[2]:
```

```
Out[3]:
```

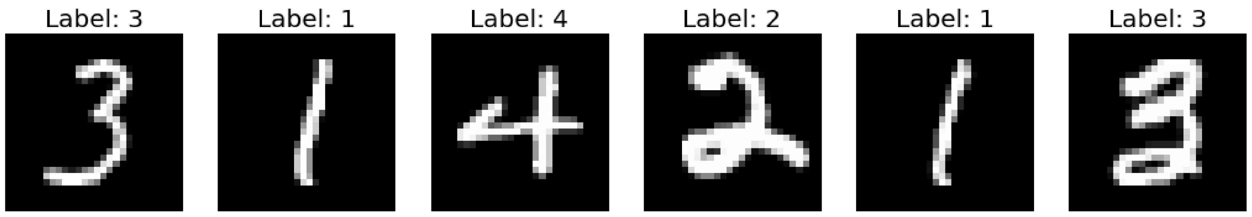
Click here to toggle on/off the raw code.

Implementation and Results

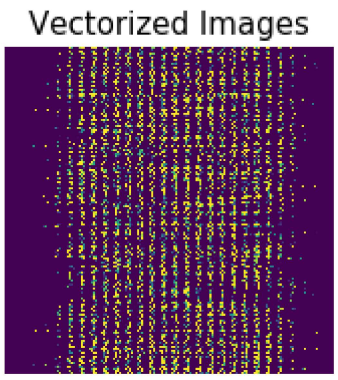
The fist thing we'll set up some settings for the Jupyter notebook and import all of the packages required to carry out the analysis and plotting.

Next, we will download the data. The MNIST can be downloaded from http://yann.lecun.com/exdb/mnist/ (http://yann.lecun.com/exdb/mnist/) or directly imported from packages such as TensorFLow.

To understand what the data contains, we will display several images and their corresponding label.



Next, we will vectorize the dataset. This will allow us to use linear algebra to find the mapping between the pixel values and the digit label. For the images, we will reshape them so every row is an image with 28x28=784 pixel values. For the labels, we will perform a one-hot encoding, meaning every label will be converted from an integer to a binary vector. It could be intersting to compare the perfomance with tensor algebra, without having to reshaping the images.



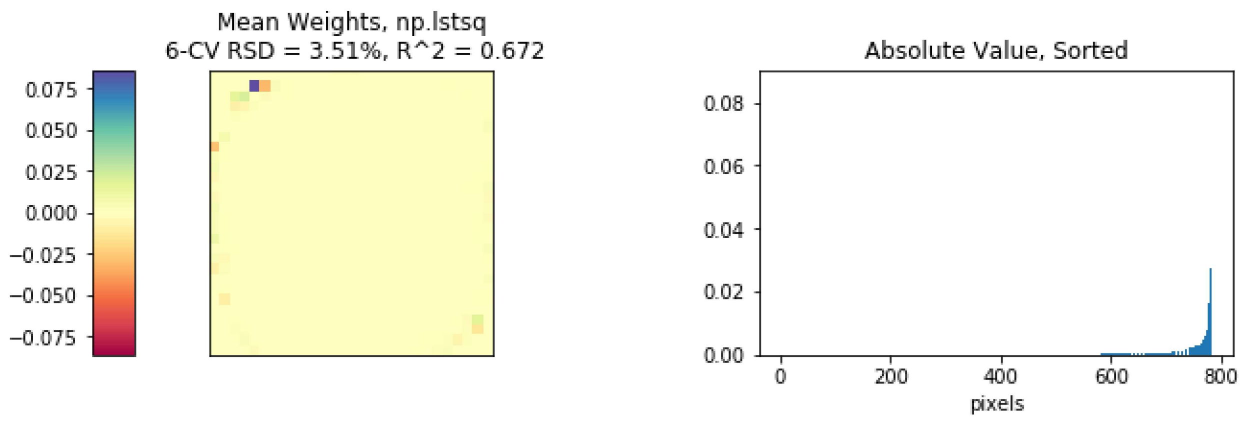
From the image above, it is clear that most of the image information is present at the center. Therefore, we expect the most important pixels to be in this location instead of the edges of the image.

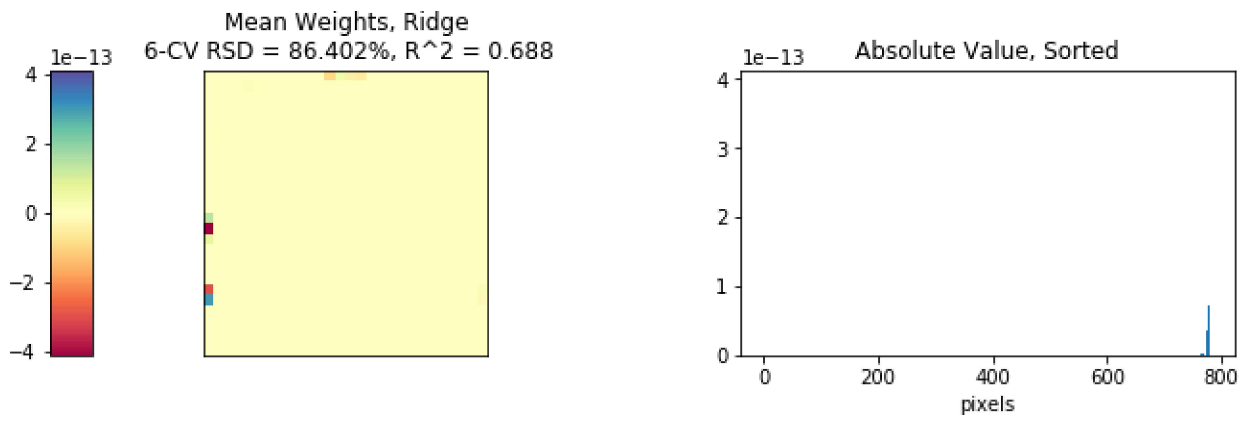
Next, we will use the different algorithms previously described to determine the mapping between images and labels. Moreover, we will use 6-fold cross validation to verify our results are idependet of the data used for training. We report this as relative standard deviation.

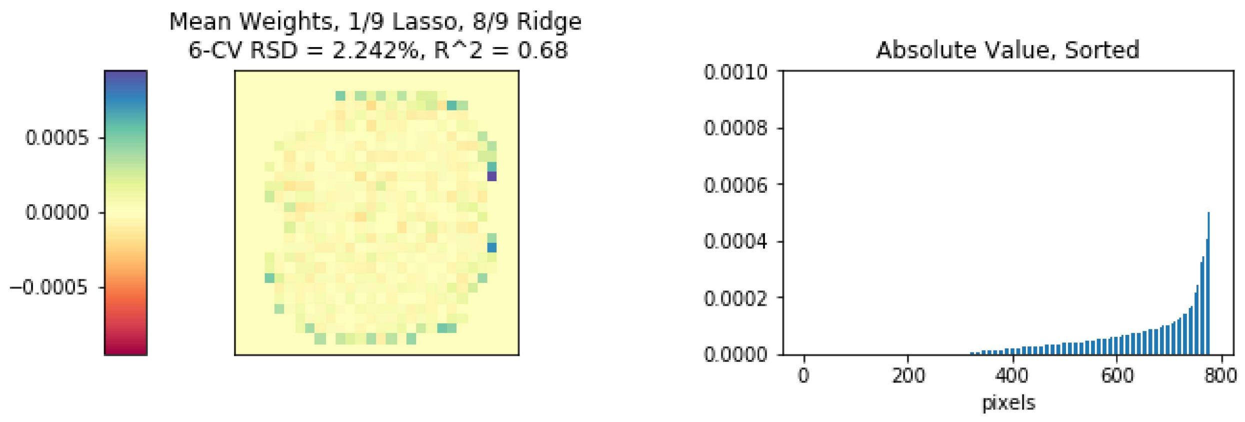
Since we observe a very low RSD, most likely due to the simplicity and the class balance of the data, we go ahead and use the entire initial train split in the model. However, ridge regression does seem to depend on train split.

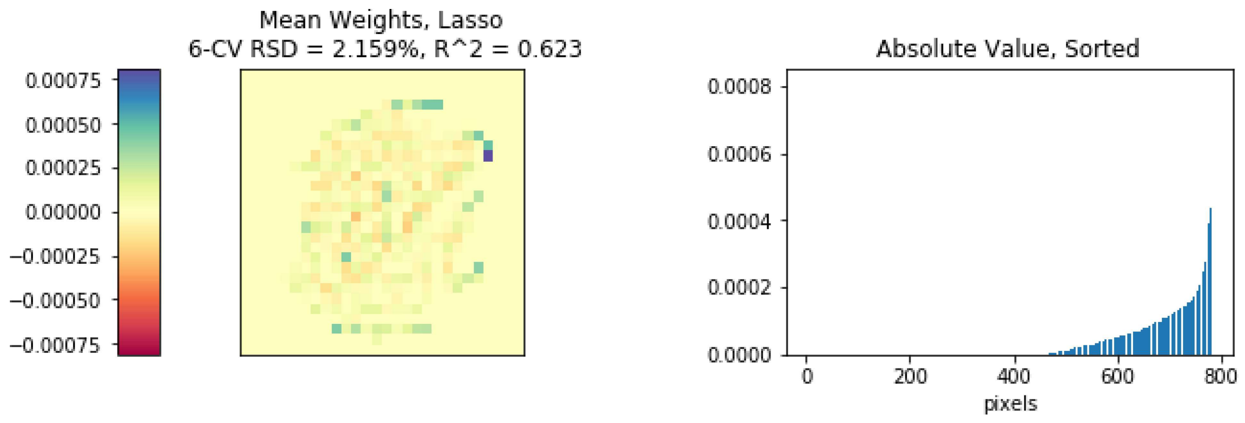
We test the different models in the original 10k test split. To calculate the R^2 value, we take the index of the maxium value in the output vector as the correspoding digit label. Note we made the artibirary choice to report R^2, accuracy or recall can also be calculated.

In the following plots, we will display the mapping (weights for each method, averaged across digits) on the left, and the histogram of absolute vaules of the weights on the right. This will give us a sense of the importance of each pixel, as well as the spread of that importance across all pixels.

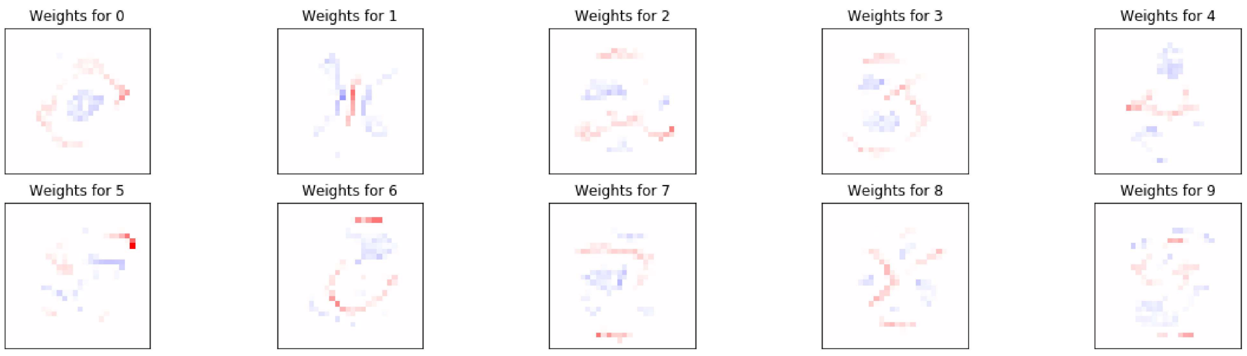






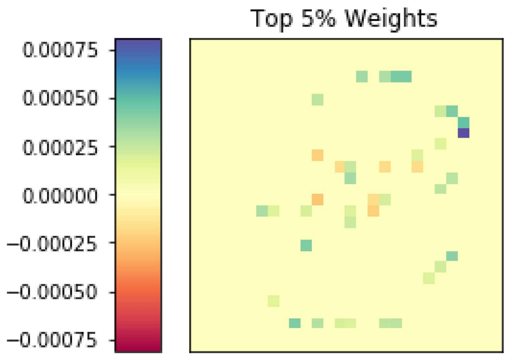


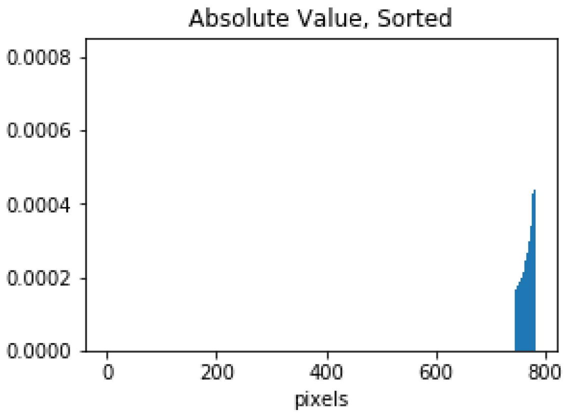
From these results, it is clear Lasso regularization increases sparsity in the mapping, highlighting the most importnat pixels. Note the contrast in magnitude between Ridge and Lasso weights. We can go one step further and break down the mapping for each digit.

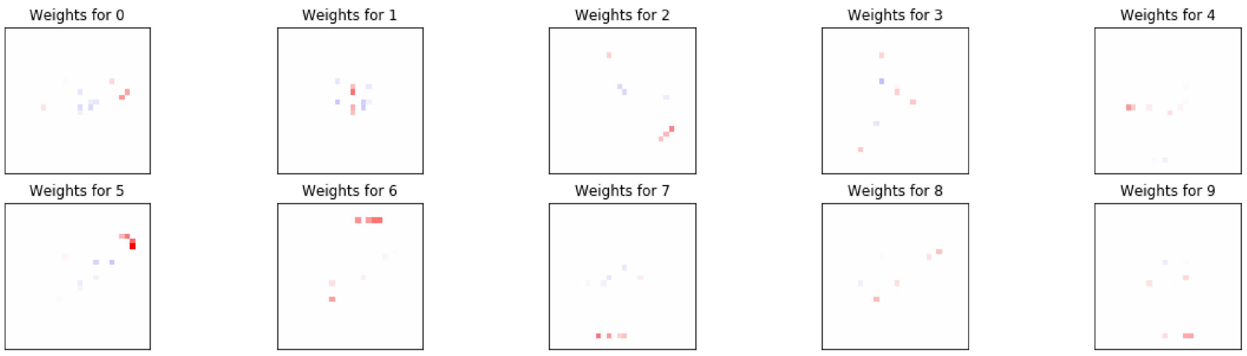


Breaking down the mapping for each digit really lets us explore their inherit definition, i.e., what makes a 3 a 3 and not an 8. In some cases this is really clear, such as 0, 3, or 7, and in other not so much (5, 9). This could be both because these digits are intrinsically harder to distinguish, or because there is an imbalance, either in terms of quantity or quality, in the dataset.

Next, we will explore what happens when we only use a small percentage of the most important pixel weights.

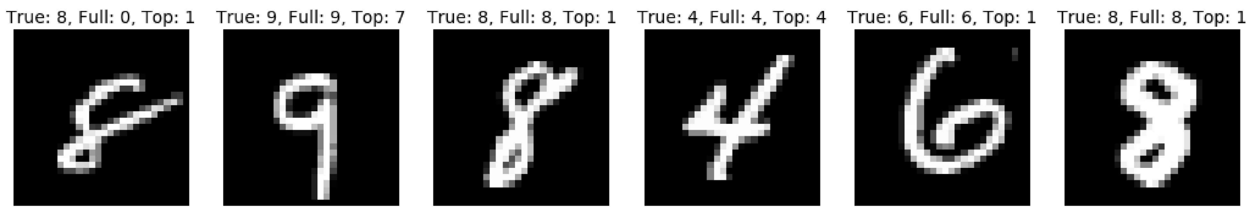


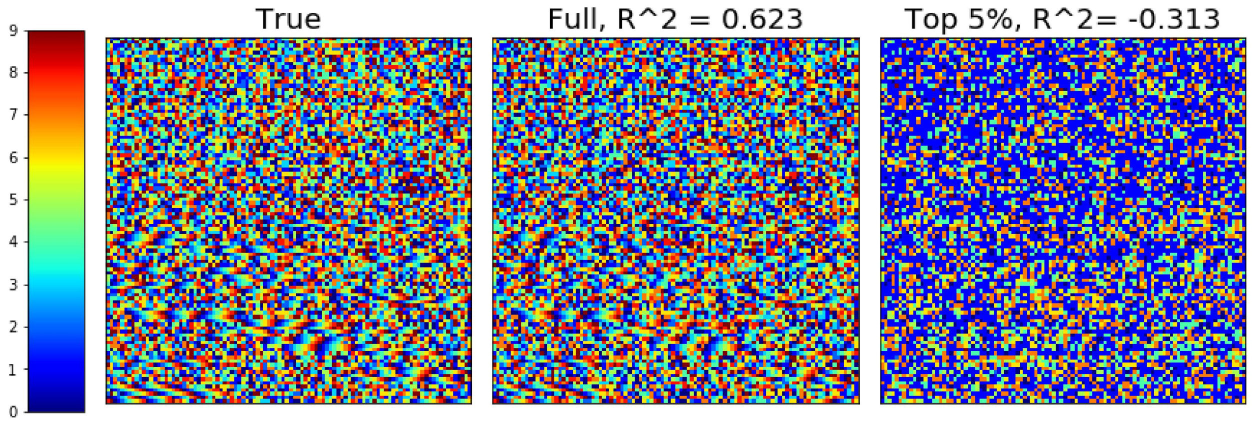




The above graphs show which pixels have been seleceted based on our criteria, and we can already tell that distinguishing between digits with such few weights will be extremely hard.

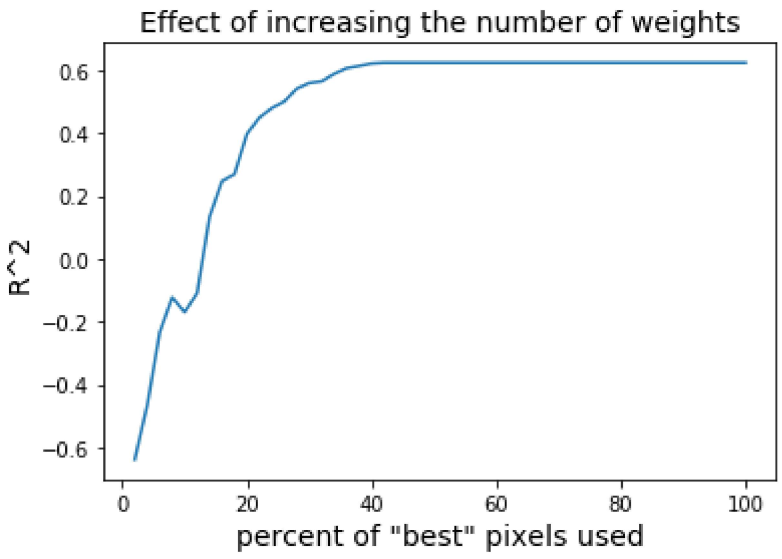
In this next block of code we will calculate the test error and try to visualize the impact of reducing the number of weights on our predictions.





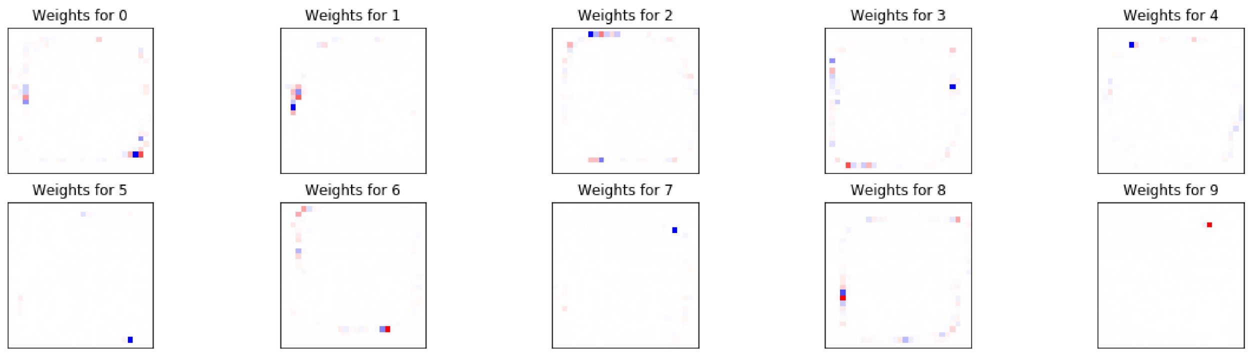
Here, we can see how different the predictions, and hence the error (measured in R^2), are. In the top row we have a sample of predictions, and what each method, Full and Top 5% Weights, predicted. On the bottom row, we show all of the predictions for the test data, and the error for each method. We can see using only 5% of the weight heavily shifts predictions to the lower digits.

Next, we can ask the question of how accurate we can be with as few pixels as possible. For this, we will simply increase the percent of weights used and calculate the error on the test data at each iteration.



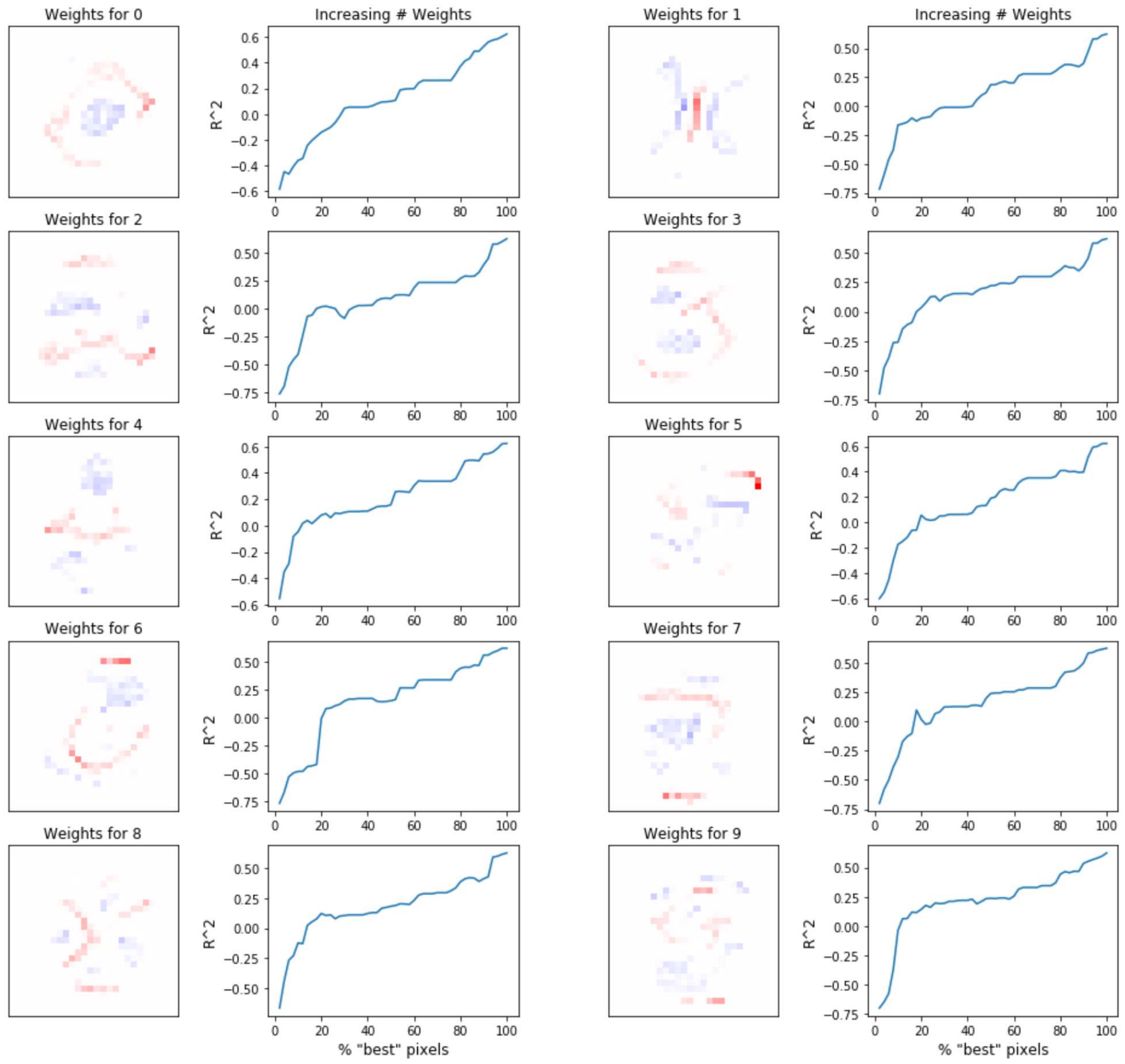
From this graph we can see that our accuracy increases rapidly as we increase the number of weights, but only during the first 40 percent points, after which the error stays constant. We suspect this is highly correlated with the number of none zero weights determined through the Lasso regularization.

Lastly, we will investigate we will investigate which pixels are most characteristic for each digit. We can approach this from several angles. First we'll try doing least squares regression solely on one digit at a time. The most important pixels for each digit are show below.



We can see this method is not very informative, mainly because we did not promote sparsity.

Another option is to analyze the weights from the Lasso regression done earlier,since this weights determine the best mapping such that each digit can be identified.



We can observe every digit, or the mapping for each digit, is different, and not all pixels are weighted the same. For instance, the most important digits to identify a 3 carry more information than those to identify a 0.

Conlcusion

We have shown how to use different methods for solving optimization problems. Particularly, we have seen that the algorithm and regularization used heavily determine the solution. For instances in which we are trying to gain insight into the model and the underlaying variables, promoting sparsity through the L1-norm turns out to be very valuable.

Appendix

The full Jupyter Notebook containing the report and the code used can be found at https://github.com/DIEGOA363/ISCS (https://github.com/DIEGOA363/ISCS)