ISYE 6740 Homework 6

Nick DiNapoli, ndinapoli6@gatech.edu

Due: November 24, 2021

1 Conceptual questions

1.1 Explain how do we control the data-fit complexity in regression trees.

The data-fit complexity in a regression tree can be controlled by limiting the size of the tree because too large of a tree can overfit the data and too small of a tree can underfit the data. This can be achieved through a number of methods and possibly a combination of said methods. Some lower-level data-fit complexity controlling methods include: controlling the depth of a tree, controlling the number of leaf nodes of a tree, and controlling the number of training data points that a node or leaf node can contain. On a higher level, one can use tree pruning and examine the data-fit complexity tradeoff by growing a large tree, stopping when a minimum node size has reached and then performing pruning by minimizing the cost function. The goal here is to measure the data fitting error (using a specific measure) of each node and keeps nodes with high predictive power. A typical approach is to test if the deletion of a node causes no or little change in error and if so, delete that node. A similar approach can be taken but this time by examining the explained variance after deleting the node.

1.2 What's the main difference between boosting and bagging?

Although there are many similarities between boosting and bagging, the main difference between the two lies in the fact that boosting uses weak learners on weighted data sets, i.e. data points have different weights such that newer models do well where the previous ones did not. Additionally the weak learners are combined linearly via a weighted average with more wight on the models that performed best on training data. Bagging on the other hand runs weak learners using bootstrapping and non-weighted data and get combined in the end by simply averaging the weak learners (same weight for each).

1.3 Explain how OOB errors are constructed and how to use it to understand a good choice for the number of trees in random forest. Is OOB error test or training error and why?

OOB errors are constructed by examining the classification of each point in a set using only the trees that were trained using bootstrap samples that did not include said point. Then, OOB error is obtained by comparing all predictions in the set to ground truth labels as usual. By further examining the OOB error when using a different number of trees, the OOB error can be used to determine a good choice for the number of trees by seeing and using the number of trees when the OOB error saturates or stabilizes (which often differs compared to using test error). OOB error is a test error because the classification of OOB sample points is completed by using trees which have not seen the points and therefore the OOB samples are all considered unseen data.

1.4 Explain what is "kernel trick" and why it is used?

The kernel trick involves considering the inner product of features maps instead of expanding features explicitly. This inner product of feature maps is represented through valid kernel functions (many choices for kernel function so long as the Gram matrix is positive semi-definite). Therefore the feature map is not directly constructed. The trick is particularly useful when the data has many features and transformations of this data can include polynomial combinations of the features which leads to an extremely large number of terms and large computational costs. So the kernel trick is used such that the objective function uses inner products of the transformed feature vectors i.e. the kernel function. This trick now dramatically decreases the computational complexity (O(n)) and cost which is ultimately why it is used. Additionally, the kernel trick allows for a linear decision boundary in the transformed non-linear feature space i.e. a non-linear decision boundary in the original feature space. Often this means higher dimensional transformations can assist with prediction.

2 Random forest and on-class SVM for email spam classifier

In this problem I use the Spambase dataset and evaluate the effectiveness of a classification tree, a random forest, and a one-class SVM (novelty detector) classifier in the classic problem of email spam filtering. This dataset consists of 4601 emails labeled as spam (1) or not (0) and the collection of non-spam emails came from filed work and personal emails and hence the occurrence of the word, "George", and the area code, "650", as indicators in the features of the dataset. This is common for constructing a personalized email spam filter. Per the UCI Machine Learning Repository, there are 57 attributes in this dataset, 48 of which are concerned with the frequency of a specific word, 6 of which are concerned with the frequency of a character, and 3 that measure the length of sequences of consecutive capital letters in different ways.

2.1 CART model

I first construct and visualize a classification tree using the data by taking advantage of the sklearn library in Python. For simplicity here, I use all of data for constructing the tree. Using entirely default parameters, the tree grows quite large reaching a depth of 34 with 299 leaves. Figure 1 shows the visualization of this default-type tree. In this tree, the minimum number of samples required to be at a leaf node is only one. When using a dataset of this size, this can greatly increase the chance of overfitting to the training data, so I use a general rule of thumb to better construct a new classification tree. The rule of thumb is that no leaf node contains less than 5% of the training data. After applying this simplification, the constructed tree becomes much smaller having a depth of 10 and 16 leaves. This "improved" or more general tree is shown in Figure 2.

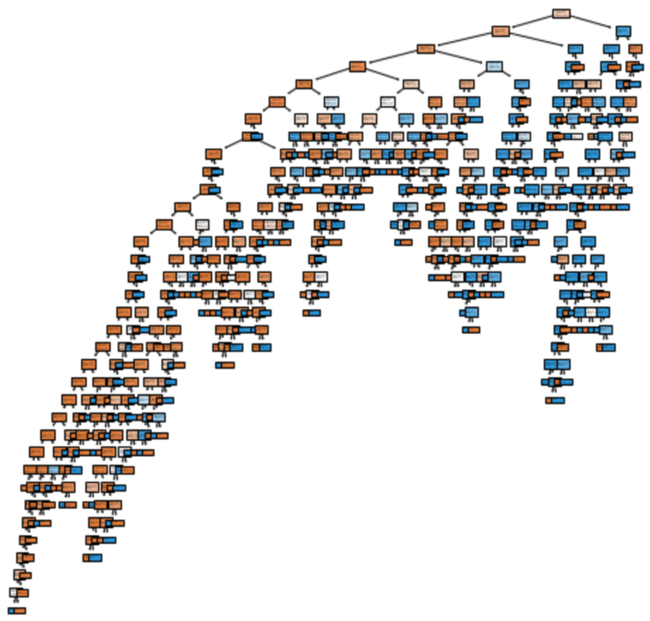

Figure 1: A default classification tree trained using all Spambase data has a depth of 34 with 299 leaves.

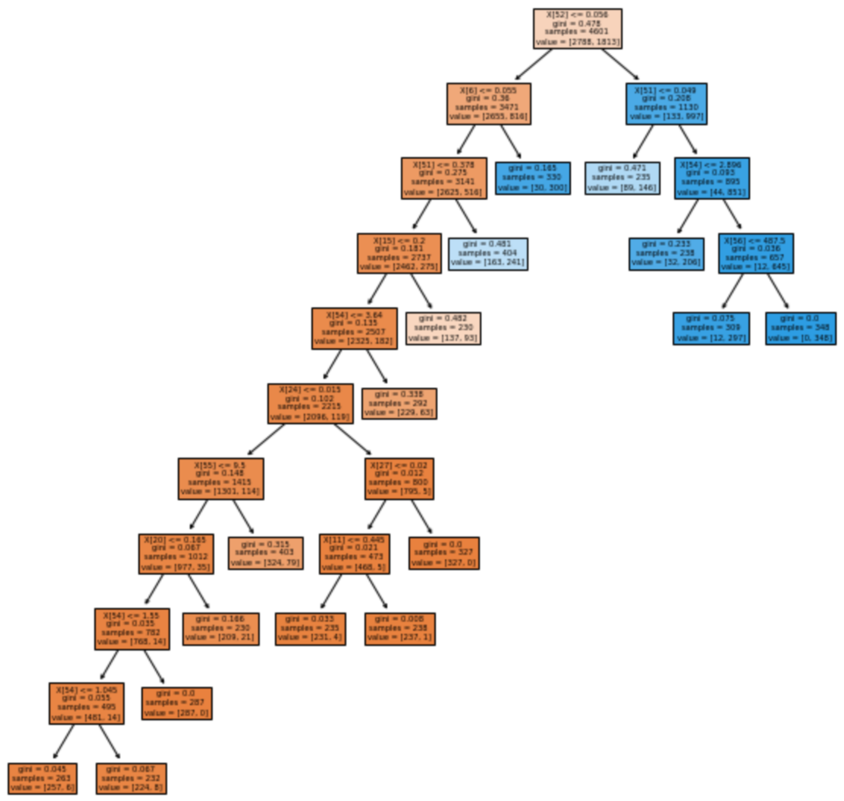

Figure 2: An improved classification tree can simply reduce overfitting and has a depth of 10 with 16 leaves.

2.2 Random forest model versus classification tree

Now I build and train a random forest model to compare the effectiveness of a random forest compared to a classification tree for this task. I shuffle and split the dataset for training (using 80% of the data) and testing (using 20% of the data). Because the ultimate goal is to make predictions on unseen data and to compare models, I retrain the classification tree using just the training set and do the same for the random forest model. As I have learned, the number of trees used in the forest can greatly impact the test error and model effectiveness so I tune the number of trees used in the final model by first analyzing the test error using a different number of trees. The goal is to see when the test error saturates and use the number of trees where it does so, this keeps the final model simple and general, yet effective. Figure 3 shows this analysis; specifically it shows the test error (total misclassification error rate) versus the number of trees for both the random forest and the re-trained classification tree. As expected, the test error for the classification tree is constant when varying the number of trees. From this analysis, I see that the test error for the random forest model is minimized when the number of trees used is 500. Table 1 summarizes the final results for comparing these two models when using the test set. It is clear from Figure 3 and Table 1 that the random forest model with a tuned number of trees achieves a lower test error and better results. Generally, it can be seen that the random forest model achieves better results when there is a sufficient number of trees.

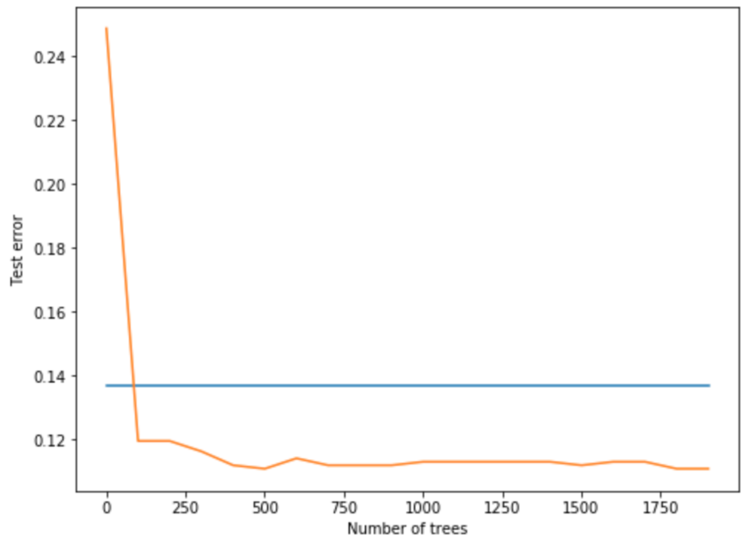

Figure 3: The test error of a classification tree model (shown in blue) and a random forest model (shown in orange) when varying the number of trees.

Model	Test error
Classification tree	13.68%
Random forest (500 trees)	11.07%

Table 1: Comparison of test error between a classification tree and a random forest.

2.3 One-class SVM for spam filtering

Lastly, I use the one-class SVM model and approach to this task which is a novelty detection situation. This is because after shuffling and train/test set splitting (80%/20%), all non-spam emails in the training set are extracted and solely used to train the model. The one-class kernel SVM model I build here uses an RBF kernel. I further enhance this model by tuning the kernel bandwidth to achieve a reasonable performance. Figure 4 shows the test error using the one-class kernel SVM model when varying the kernel bandwidth. I determined that the optimal bandwidth was 0.001601 which allowed the model to achieve a 34.42% test error. It is very clear here that the performance of this model for email spam filtering does not come close to that of the classification tree or random forest (as seen in previous section).

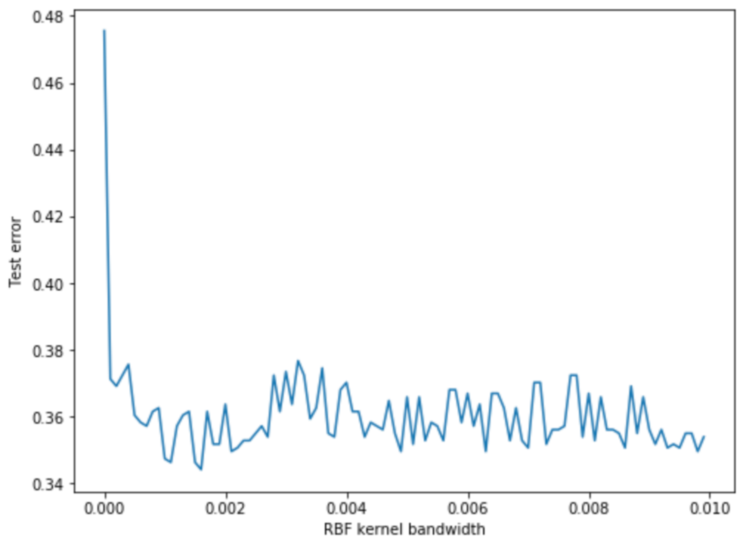

Figure 4: The test error of a one-class kernel SVM model using an RBF kernel as the kernel bandwidth is varied.

3 Locally weighted linear regression and bias-variance tradeoff

In this problem, I consider a dataset with n data points (x_i, y_i) following the linear model in Equation 1,

$$y_i = \beta^{*T} x_i + \epsilon_i, \quad i = 1, \ldots, n, \tag{1}$$

where $\epsilon_i \sim \mathcal{N}(0, \sigma_i^2)$ are independent but not identically distributed.

3.1 Simplification

I start from the form of ridge regression in the context of this problem in Equation 2 to derived the matrix form of the expression,

$$\hat{\beta}(\lambda) = \arg\min_{\beta} \hat{L}(\beta) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \beta^T x_i)^2 + \lambda\|\beta\|_2^2. \tag{2}$$

Although one could entirely rederive the form above, the most important thing to note in this problem is the form of Equation 1. Because $\epsilon_i \sim \mathcal{N}(0, \sigma_i^2)$, and the first term in Equation 1 is just a constant, y_i, is a mean-shifted Gaussian distribution (mean 0 shifted by $\beta^T x_i$). Now when I take the log likelihood of the PDF of the mean-shifted Gaussian I get,

$$\sum_{i=1}^{n} -\frac{1}{2}log(2\sigma_i^2\pi) - \frac{(\beta^T x_i - y_i)^2}{2\sigma_i^2} \tag{3}$$

and I am only concerned with terms with β dependence so I get,

$$\sum_{i=1}^{n} \frac{(\beta^T x_i - y_i)^2}{\sigma_i^2}. \tag{4}$$

Another important note here is that the σ_i's are not identically distributed so they cannot be considered a constant and hence cannot be omitted whereas the -1/2 term can be (negative sign to because now I and concerned with arg min). Now I express this in matrix form,

$$(X\beta - y)^T \Sigma^{-1} (X\beta - y). \tag{5}$$

Remembering to tack on the regularizing term I get a new expression for the maximum likelihood estimate,

$$\hat{\beta}(\lambda) = \arg\min_{\beta} \left\{ (X\beta - y)^T \Sigma^{-1} (X\beta - y) + \lambda \|\beta\|_2^2 \right\} \tag{6}$$

which fits

$$\hat{\beta}(\lambda) = \arg\min_{\beta} \left\{ (X\beta - y)^T W (X\beta - y) + \lambda \|\beta\|_2^2 \right\} \tag{7}$$

so long as W is the square, symmetric, and positive definite diagonal covariance matrix with diagonal entries $1/\sigma_i^2$.

3.2 Closed-form solution

Next, I start with the MLE of β in Equation 7 to find the closed-form solution for $\hat{\beta}(\lambda)$. First, I expand the terms and set the gradient equal to zero and solve from there.

$$\begin{aligned}
\hat{\beta}(\lambda) &= \arg\min_{\beta} \left\{ (X\beta - y)^T W (X\beta - y) + \lambda \|\beta\|_2^2 \right\} \\
\hat{\beta}(\lambda) &= (X\beta)^T W X\beta - y^T W X\beta - Wy(X\beta)^T + y^T W y + \lambda \beta^T I \beta \\
\nabla_\lambda \hat{\beta}(\lambda) &= 0 \\
0 &= 2X^T W (X\beta - y) + 2\lambda I \beta
\end{aligned} \tag{8}$$

Now I divide by 2, expand and rearrange to find β.

$$\begin{aligned}
0 &= X^T W X\beta - X^T W y + \lambda I \beta \\
X^T W y &= (X^T W X + \lambda I)\beta \\
\beta &= (X^T W X + \lambda I)^{-1} X^T W y
\end{aligned} \tag{9}$$

I can also take out of matrix form,

$$\beta_i = \frac{(x_i/\sigma_i^2) y_i}{(x_i^2/\sigma_i^2) + \lambda} \tag{10}$$

3.3 Bias

The bias is the expected difference between the expected prediction and the true solution. For this locally weighted ridge regression problem it can be derived in the following way.

$$\mathbb{E}_x[\mathbb{E}_D[\hat{f}(x)] - h(x)] \tag{11}$$

Considering a fixed point x and using the previous result,

$$\begin{aligned}
&\mathbb{E}_{\hat{\beta}}[(\hat{\beta} - \beta^*)^T x] \\
&\mathbb{E}_{\hat{\beta}}[\hat{\beta}^T x - \beta^{*T} x] \\
&\mathbb{E}_{\hat{\beta}}[\hat{\beta}^T] x - \beta^{*T} x \\
&(x^T (1/\sigma) x + \lambda)^{-1} x^T (1/\sigma) y - \beta^{*T} x \\
&[(x^2/\sigma^2 + \lambda)^{-1} (xy)/\sigma^2] - \beta^{*T} x.
\end{aligned} \tag{12}$$

3.4 Variance

The variance is said to be the difference between what can be expected to be learned and what is actually learned from the dataset. This can also be derived similar to the bias in the following way.

$$\mathbb{E}_x\mathbb{E}_D[(\hat{f}(x) - \mathbb{E}_D[\hat{f}(x)])^2] \tag{13}$$

For a particular x and using a simplification can be written as,

$$\mathbb{E}_{\hat{\beta}}[(x^T\hat{\beta})^2] \tag{14}$$

and for this problem using the previous results,

$$x^2[(x^2/\sigma^2 + \lambda)^{-1}(xy)/\sigma^2 - \beta^{*T}x]^2 \tag{15}$$

3.5 MSE

Lastly, I assume the data are one-dimensional, and that the training dataset consists of two samples $x_1 = 1.5$ and $x_2 = 1$. With test samples, $x = 0.5$ and $x = 2$, the true parameter, $\beta_0^* = 1$, $\beta_1^* = 1$, and the noise variance, $\sigma_1^2 = 2$, $\sigma_2^2 = 1$, I examine the MSE as a function of λ. Figures 5 and 6 show these results. Unfortunately, based on these results, I have serious reason to believe that there is an error in my derivation. I expect the MSE to be a smooth curve with a global minimum at a λ value that is close to where the bias and variance curves intersection (hence tradeodd). Figure 5 seems to be on the right path for what is expected but Figure 6 does not appear to be correct because I expect the two curves to be similar because the same training data is used.

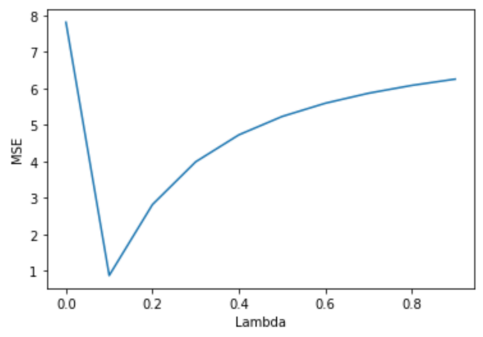

Figure 5: MSE as λ is varied with test point x = 0.5.

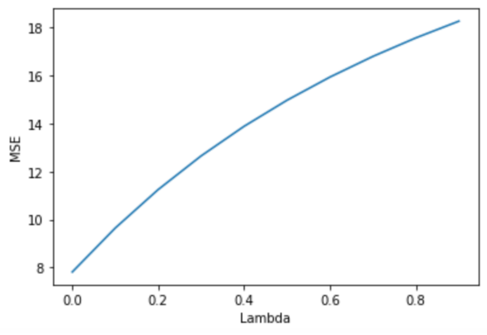

Figure 6: MSE as λ is varied with test point x = 2.

4 Medical imaging reconstruction

For this problem, I aim to mimick a problem in medical imaging reconstruction, specifically in MRI imaging. MRI machines that collect images take a long time to measure each pixel value individually but it is much faster to measure a linear combination of pixel values. For this reason, I measure 1300 linear combinations with weights being random and identically distributed as $\mathcal{N}(0, 1)$. I actually measure a noisy version of this because of the nature of the machine. These measurements are given by the equation,

$$y = Ax + n$$

where n $\sim \mathcal{N}(0, 25 \times I_{1300})$, I_n being the identity matrix of size n. Here, I generate y using this model and a true image of size 50x50 which is a sparse, toy version of an MRI image I aim to collect. Figure 7 shows this toy image for this problem.

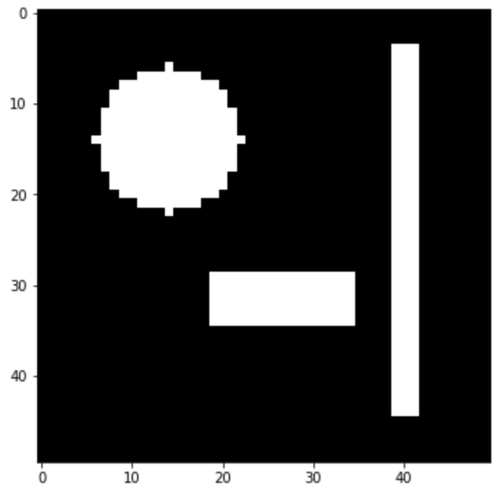

Figure 7: Example MRI image I aim to recover.

I aim to answer the question if I can model y as a linear combination of the column vectors of x to recover a coefficient vector that is close to the original true image. I complete this image recovery task using both lasso, $\min_x \|y - Ax\|_2^2 + \lambda\|x\|_1$, and ridge, $\min_x \|y - Ax\|_2^2 + \lambda\|x\|_2^2$ regression and selecting λ using 10-fold cross-validation.

4.1 Lasso for image recovery

First I use Lasso to recover the coefficient vector. As mentioned, I use 10-fold cross-validation to select λ by varying alpha (alpha is used as the parameter name for λ) and qualitatively and quantitatively analyzing the negative mean squared error. I choose the λ or model that minimizes the MSE and visualize coefficient vector as a means of representing and recovering the original image. After completing this, I find that the optimal choice of λ is 0.011 as seen in Figure 8. This choice of λ results in the final recovered image having 856 non-zero values so the lasso does its job quite well. The recovered image is shown in Figure 9.

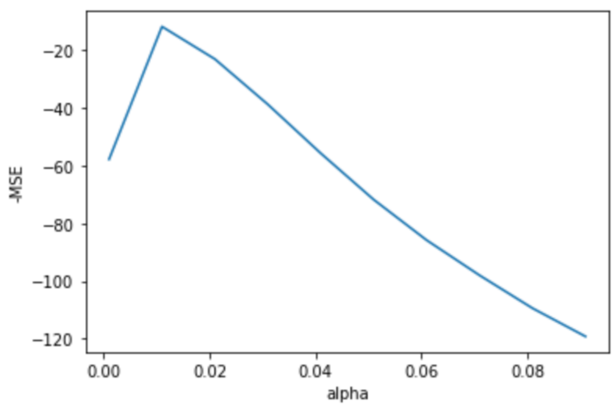

Figure 8: MSE of lasso regression when varying λ

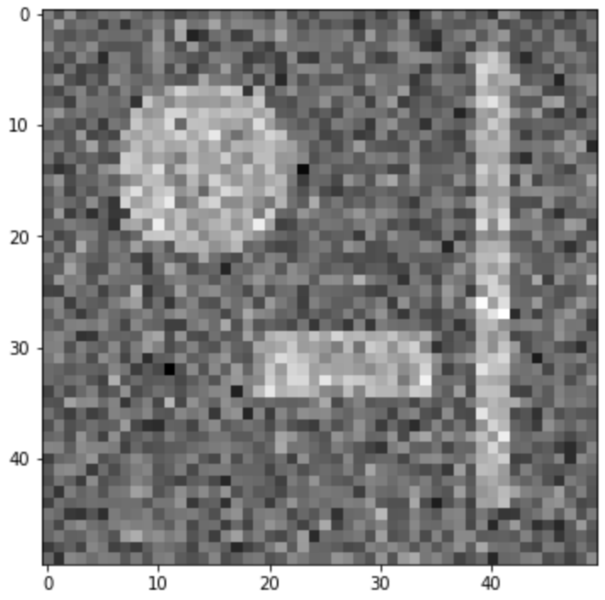

Figure 9: Recovered image using optimal lasso model (856 non-zero values).

4.2 Ridge for image recovery

As a means of comparison, I now use ridge regression with 10-fold cross-validation to complete the same tasks as the previous section. The value of λ that minimizes the MSE is 76.1 as can be seen in Figure 10. With this choice of λ, all 2500 pixels remain as non-zero as expected and the recovered image can be visualized. Figure 11 shows this recovered image. Although the lasso recovered image has many zero values, generally, it seems noisier and the recovered ridge image is darker because many coefficients and hence pixels are driven towards zero (even if they are not zero). For this problem, this arguably leads to an increased signal to noise ratio, which is vastly important in medical imaging, and gives a better recovered image.

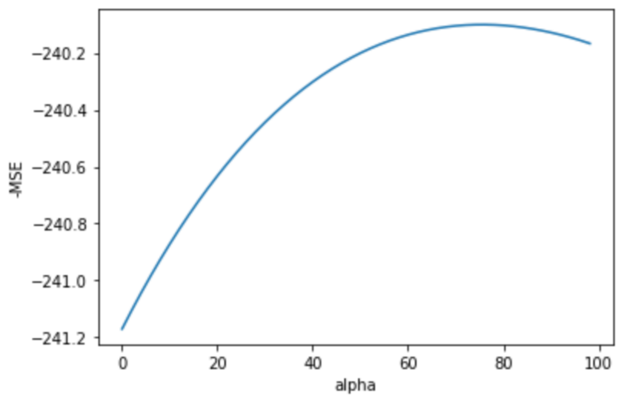

Figure 10: MSE of ridge regression when varying λ

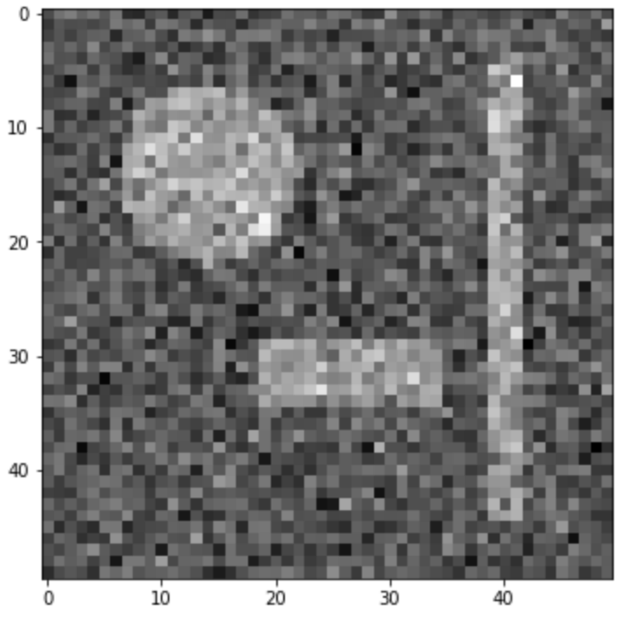

Figure 11: Recovered image using optimal ridge model (2500 non-zero values).

References

[1] http://archive.ics.uci.edu/ml/datasets/Spambase

[2] https://www.cs.princeton.edu/courses/archive/fall18/cos324/files/mle-regression.pdf