**Application of classification algorithms to MNIST handwritten letters**

**1. Introduction**

Classification algorithms are useful tools that can be used for image processing, email span filters, and tumor detection. To gain a working understanding of classification algorithms and other machine learning concepts, it is necessary to experiment with the material oneself. To gain familiarity with machine learning algorithms, I have coded up and experimented with several classification algorithms towards the goal of classifying handwritten numbers. For regression models, five different combinations of loss function and regularization were explored. For neural network models, four different combinations of loss function and activation were explored.

Broadly speaking, algorithm complexity did not translate to improved model validation error rate. More often than not, performance worsened with increase in polynomial degree of fit for regression models. Similarly, neural network accuracy decreased as the number of hidden layers increased in the cases where sigmoid activation was used. L1-regularized least-squares regression and L2-regularized hinge regression appeared effective with small sample sizes while L2-regularized hinge regression and neural networks using categorical cross-entropy loss with sigmoid activation functions appeared effective with large sample sizes.

**2. Datasets and algorithms used**

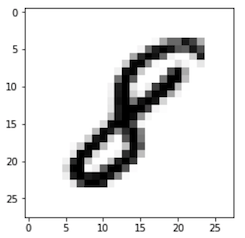
The dataset used in this project is the infamous MNIST dataset [1]. The MNIST dataset consists of 70000 28x28 pixel images of handwritten integer numbers from 0 to 9 (example provided in figure 1). Each image has a label that gives the integer number. The goal of working with the MNIST dataset is to create a machine learning algorithm that can accurately classify images into one of the 10 classes of integer numbers from 0 to 9. Each image can be represented as a 28x28 matrix with each element containing a value between 0 and 255 which corresponds to the darkness of the pixel. Alternatively, the 28x28 matrix can be transformed into a 784x1 vector array for easy processing. For n training images, the 784x1 vectors for each image are stacked into a 784xn matrix which serves as my feature matrix. All implemented classification methods essentially consist of an ensemble of binary classifiers for each possible class. An image is classified according to the class of the binary classifier amongst the ensemble that signals the best fit. To facilitate this process, the set of labels is converted to a nx10 binary label matrix rather than a simple nx1 vector of integers. Within the nx10 binary label matrix, an element label of “1” or “-1” is assigned depending on whether image matches the class of the column or not (“1” and “0” for neural networks).

Fig. 1. MNIST 28x28 pixelated image of a handwritten integer.

MNIST designates 60,0000 of the 70,000 images as “training” images and 10,000 images as “testing” images. For this project, the 60,000 “training” images are used as my training dataset and the 10,000 “testing” images are used as my validation dataset. The error rate reported is the ratio of misclassified testing images to total testing images when the trained classifier is used to predict the classes of the validation training set. Generally speaking, usually a third “testing” dataset would be used to evaluate the final performance of final models that have been tuned via the feedback from the validation dataset. However, the emphasis of this project is on algorithm implementation and so this step is neglected.

Various classification algorithms were tested. The algorithms are split into two broad categories: 1) Regression and support-vector machine algorithms, and 2) Neural network algorithms. For the regression algorithms (summarized in table 1), regularization was necessary since the number of images outnumbered the 784 unique features. I used both L1 and L2 regularizations for regressions involving least squares (LS), hinge, and squared hinge loss functions. The 1st- and 2nd- degree polynomial fits were achieved by expanding the feature matrix and weight matrix to include additional intercept and square terms. L1-LS algorithm was solved via matrix inversion while L2-LS algorithm implemented stochastic gradient descent (SGD). All support vector machine algorithms implemented the LinearSVC function from python sklearn.svm and LinearSVC modules[2]. Squared hinge loss was included since it is compatible with both L1 and L2 regularizations in the python sklearn.svm and LinearSVC modules whereas hinge is not.

Table 1. Regression and support-vector machines

|  |  |  |
| --- | --- | --- |
| **Loss**  **Function** | **Activation** | **No. Hidden**  **Layers** |
| LS | relu | 1, 2 |
| sigmoid | 1, 2 |
| CCE | relu | 1, 2 |
| sigmoid | 1, 2 |

The keras and tensorflow python modules [3] were used to implement neural networks (algorithms summarized in table 2). Originally, neural networks were written in python from scratch but the pre-developed code from keras and tensorflow showed an order-of-magnitude improvement in run time and script length. I experimented with different loss functions, activation functions, number of hidden layers, number of epochs, and hidden layer width and observed changes in error rate. The two loss functions investigated were LS and categorical cross entropy (CCE), the two activation functions were relu and sigmoid, I investigated 1 and 2 hidden layers for each case.

Table 2. Neural Networks

|  |  |  |  |
| --- | --- | --- | --- |
| **Loss  Function** | **Regular-**  **ization** | **Loss Eqn. w/**  **Regularization term** | **Fitting**  **Degrees** |
| LS | L1 |  | 0, 1, 2 |
| L2 |  | 0, 1, 2 |
| Hinge | L2 |  | 0, 1, 2 |
| Squared  hinge | L1 |  | 0, 1, 2 |
| L2 |  | 0, 1, 2 |

**3. Results**

**3.1. Regression and support vector machines**

Among the L2-regularized least squares regressions, the 2nd-degree achieved the best accuracy when the entire 60,000-image training set was used. However, when the training set was much smaller (first 1,000 images), lower-order regression was better. The polynomial fit error rates converged near λ=1E7 across all degrees before increasing drastically at higher λ. Beyond this point, the regularization component of the loss function will dominate and the data itself will have less influence on the fit, leading to increasing prediction error.

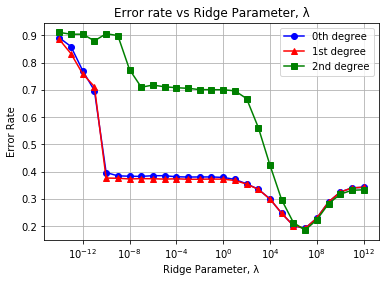
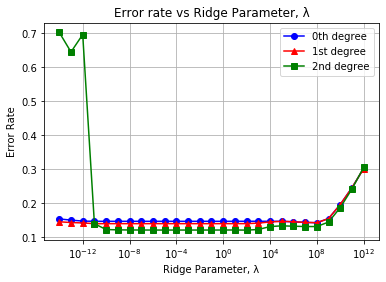


Figure 2. Validation classification error rate for L2-regularized least squares regressions using (a) 60,000 training images and (b) using the first 1,000 training images

L1-regularized least-squares regression (LASSO regression) trained much faster than L2-regularized least-squares regression due to the absence of matrix inversion in the L1 algorithm. Additionally, L1-regularization was much more effective when working with a limited dataset (fig BLAH b) compared to L2. This makes sense since L1-regularization emphasizes the influence of higher-weight features, captures prominent relationships, and results in a sparse weight matrix. Even for smaller datasets, prominent relationships may be ascertainable. Polynomial degree made little difference in L1-regularized fitting accuracy. This result makes sense since L1 regularization should minimize the influence of extraneous lower-weight features. Adding additional features will not improve performance unless the added features are relatively important. INCLUDE TOLERANCE

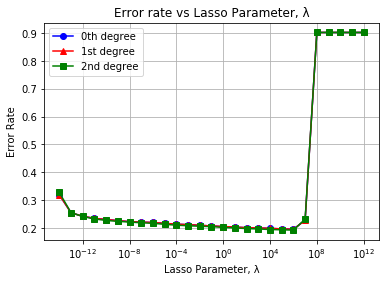
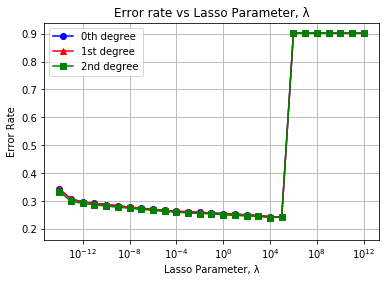
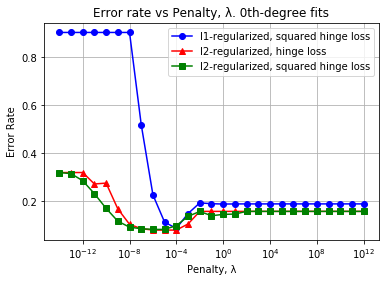
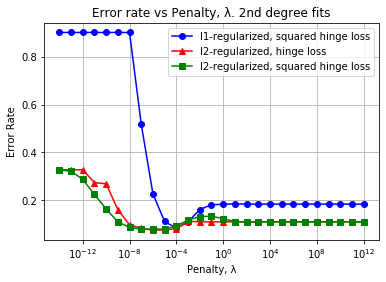
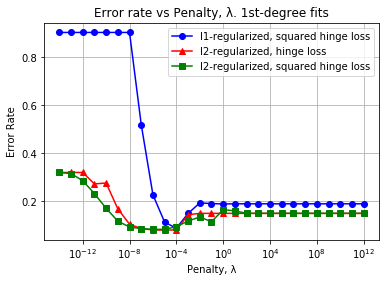
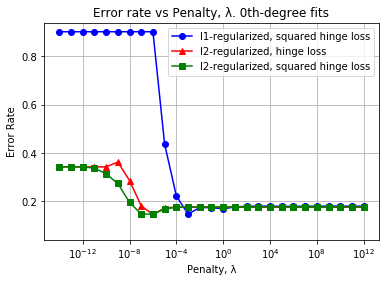
 

Figure 2. Validation error rate for L2-regularized least squares regressions using (a) 60,000 images for training and (b) using the first 1,000 images for training

The error rate of support vector machine algorithms (Fig. 3) showed notable variability across different regularization schemes (L1 and L2). Like L1 LS, hinge loss and squared hinge loss seemed effective for small training sets since even after only training with the first 1000 images, classifiers could achieve error rates as low as .15. Unlike L1 LS, hinge loss and squared hinge loss saw large improvements in minimal error rate when training sample size expanded. Polynomial degree had little impact on fitting performance. Squared hinge loss models and hinge loss models maintained vastly different error rates below λ=1E-4 but drew closer when λ>1E-4.

(c)

(d)

Fig 3. Validation error rate of hinge-loss classifiers with (a) 0th-degree model trained using 60000 images, (b) 0th-degree model trained using only 1000 images. The 0-D classifiers trained with only 1000 images achieve nearly the same accuracy as the 0-D classifiers with all 60000 training images. (c) 1st-degree model trained using all 60000 images (d) 2nd-degree model trained using all 60000 images. Additional features did not improve error rate significantly.

(a)

(b)

**3.1. Neural Networks**

For each model in table 2, epoch number was swept from 10 to 60 epochs, and layer width was swept from 10 to 60 nodes. During the sweeps across epoch number, hidden layer width was fixed to 30 nodes. Similarly, epoch number was fixed to 30 during sweeps across layer width. When using relu activation, all pixel data was scaled from integer values of 0 to 255 to decimals of 0 to 1 to ensure compatibility.

Fig. 4 shows the results of the neural networks using the first loss function, LS. All LS neural nets see modest gains in accuracy as epochs and/or hidden layer widths are increased. However, the addition of a second hidden layer does not visibly improve accuracy for the relu models, and in the case of the sigmoid models, accuracy is decreased. While this may be suggestive of overfitting, I also found increases in error rate in the training set when moving from 1 hidden layer to 2.

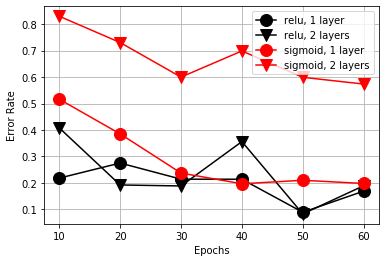
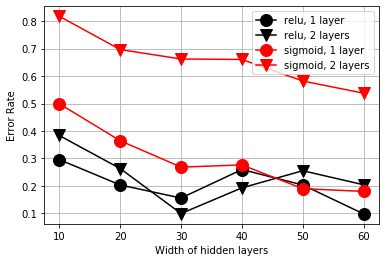
 

Figure 4. Validation error rate for neural networks using a LS loss function across (a) number of training epochs (b) number of nodes in each hidden layer. Neural networks could have had 1 or 2 hidden layers which were either exclusively relu nodes or sigmoid nodes.

Using the categorical cross-entropy loss function in tandem with the sigmoid activation function resulted in significant improvements in accuracy. However, relu is incompatible with CCE when applied directly which resulted in predictions that were approximately equal to random guesses.

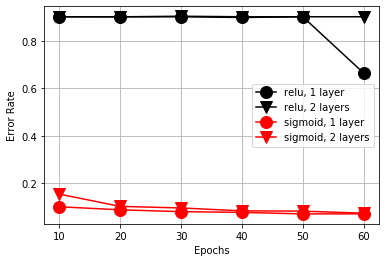
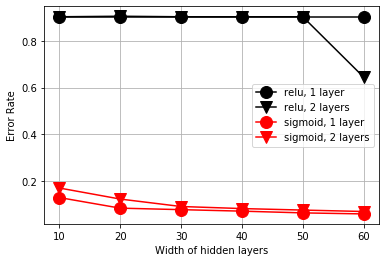
 

Figure 4. Validation error rate for neural networks using a CCE loss function across (a) number of training epochs (b) number of nodes in each hidden layer. Neural networks could have had 1 or 2 hidden layers which were either exclusively relu nodes or sigmoid nodes.

**4. Strengths and limitations of methods**

The L2-regularized LS code took significantly longer to run than the L1-regularized LS code. This is likely due to matrix inversion being a relatively complex operation. While matrix inversion results in an exact solution, an iterative method like SGD may be needed for larger sample sizes where inversion isn’t feasible.

Given a small dataset, 0th or 1st degree L1-regularized LS was an excellent choice. It proved to be fairly accurate even with a small data set.

Neural networks performed poorly when dataset was small

In terms of accuracy, run time, elegance, ease of use, etc., the model implementing a single sigmoid hidden layer with CCE loss was the best model among all models evaluated in this project. Of the non-CCE models, only hinge regression reached similar accuracy, but did so at a very narrow range of λ

Neural network implementation required a bit more finesse and contained more potential pitfalls compared to the regression methods. For instance, the relu activation function and the sigmoid activation function required different data scaling to perform effectively. The second loss function that I used, categorical cross-entropy, could only be used for a binary label matrix like the one that I used (“one-hot” encoded). Additionally, categorical cross entropy is not directly compatible with relu, and would require additional operations, such as performing a softmax operation beforehand. Flippantly applying

Furthermore, the addition of more hidden layers actually hindered model accuracy rather than improved it in the case of sigmoid. While this may be suggestive of overfitting, I also found increases in error rate in the training set when moving from 1 hidden layer to 2.

checked the error rate of the training data

Additionally, adding more hidden layers hindered performance

Broadly speaking, Performance could likely be improved by centering and scaling images or by splitting the “7” class into two separate classes, one class of 7’s with a central cross (e.g. “7”) and one class of 7’s without the central cross (e.g. “7”). Additionally, pixel proximity was not considered in any model. An approach that conserves the relationships between nearby pixels would also likely improve performance.

**5. Conclusion**

Surprisingly, the various methods achieved roughly similar error rates in the validation overall.

**6. References**

[1] “MNIST handwritten digit database, Yann LeCun, Corinna Cortes and Chris Burges.” [Online]. Available: http://yann.lecun.com/exdb/mnist/. [Accessed: 22-Oct-2020].

[2] F. Pedregosa FABIANPEDREGOSA *et al.*, “Scikit-learn: Machine Learning in Python,” 2011.

[3] “Keras: the Python deep learning API.” [Online]. Available: https://keras.io/. [Accessed: 12-Dec-2020].