Application of classification algorithms to MNIST handwritten letters

Jonathan King, ECE 532

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. Each image has a label giving 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 ten 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. First, the 28x28 matrix is 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 the 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. 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).

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 validation images to total validation images when the trained classifier is used to predict the classes of the validation set. Generally speaking, usually a third "testing" dataset would be used to evaluate the final performance of models that have been tuned via the feedback from the validation step. However, the emphasis of this project is on algorithm implementation and so this step is neglected.

Various classification algorithms were tested. The algorithms were 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 used stochastic gradient descent (SGD) [2]. All support vector machine algorithms used the LinearSVC function from python sklearn.svm and LinearSVC modules [3]. 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.

The keras and tensorflow python modules [4] were used to implement neural networks (summarized in table 2). Originally, neural networks were written in python from scratch but the pre-developed code from keras and tensorflow yielded 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 1. Regression and support-vector machines

Loss Function	Regular-ization	Loss Eqn. w/ Regularization term	Fitting Degrees
LS	L1	$\|\boldsymbol{Aw}-\boldsymbol{d}\|_2^2+\lambda\|\boldsymbol{w}\|_1$	0, 1, 2
	L2	$\|\boldsymbol{Aw}-\boldsymbol{d}\|_2^2+\lambda\|\boldsymbol{w}\|_2^2$	0, 1, 2
Hinge	L2	$\sum_{i=1}^{N}\left(1-d_i\boldsymbol{x}_i^T w\right)_+ +\lambda\|\boldsymbol{w}\|_2^2$	0, 1, 2
Squared hinge	L1	$\sum_{i=1}^{N}(1-d_i\boldsymbol{x}_i^T w)_+^2 +\lambda\|\boldsymbol{w}\|_2^2$	0, 1, 2
	L2	$\sum_{i=1}^{N}(1-d_i\boldsymbol{x}_i^T w)_+^2 +\lambda\|\boldsymbol{w}\|_1$	0, 1, 2

Table 2. Neural Networks

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

3. Results

3.1. Regression and support vector machines

Among the L2-LS regressions (Fig. 1), 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.

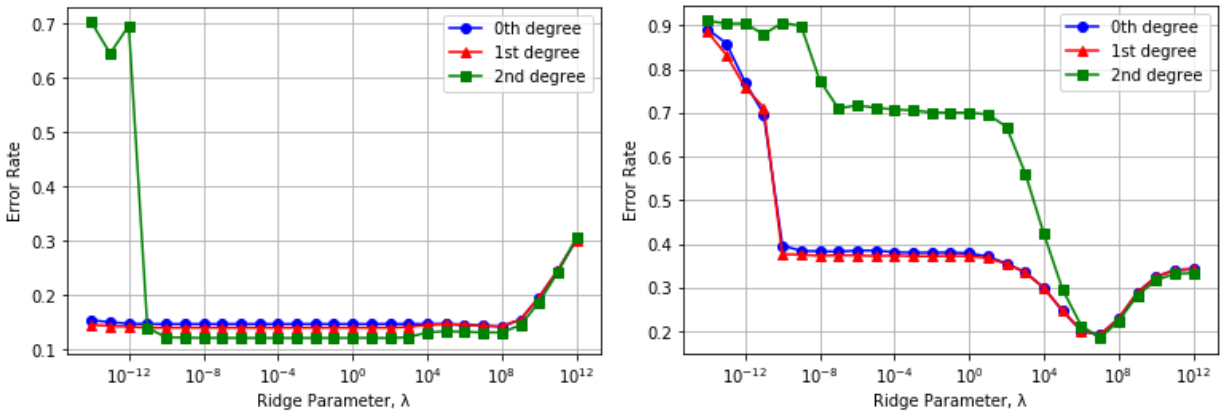

Figure 1. Validation classification error rate for L2-regularized least squares regressions using (a) 60,000 training images and (b) using the first 1,000 training images

L1-LS regression (a.k.a. LASSO regression) trained much faster than L2-LS regression due to the absence of matrix inversion in the L1-LS algorithm. Error rates are plotted in Fig. 2. L1-LS was much more effective when working with a small dataset compared to L2-LS. This makes sense since L1-LS emphasizes the influence of higher-weight features 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. L1-LS should minimize the influence of extraneous lower-weight features. Adding additional features will not improve performance unless the added features are relatively important.

The error rate of support vector machine algorithms showed notable variability across different regularization schemes (L1 and L2). Like L1-LS, hinge loss and squared hinge loss appeared effective for small training sets since even after only training with the first 1000 images, classifiers could achieve validation error rates as low as .15, as shown in Fig. 3. However, hinge loss and squared hinge loss saw large improvements in minimal error rate when training sample size expanded compared to L1-LS. Squared

hinge loss models and hinge loss models maintained vastly different error rates below λ=1E-4 but drew closer when λ>1E-4.

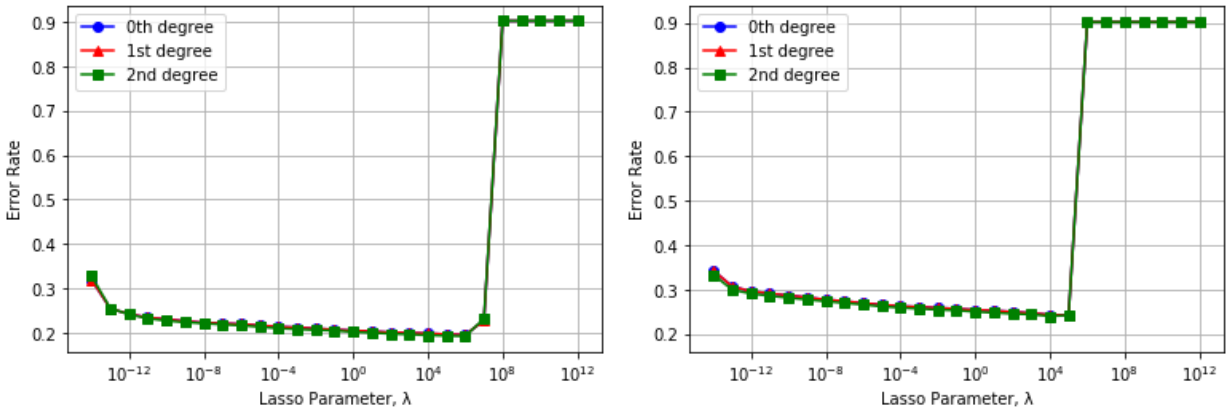

Figure 2. Validation error rate for L1-regularized least squares regressions using (a) 60,000 images for training and (b) using the first 1,000 images for training

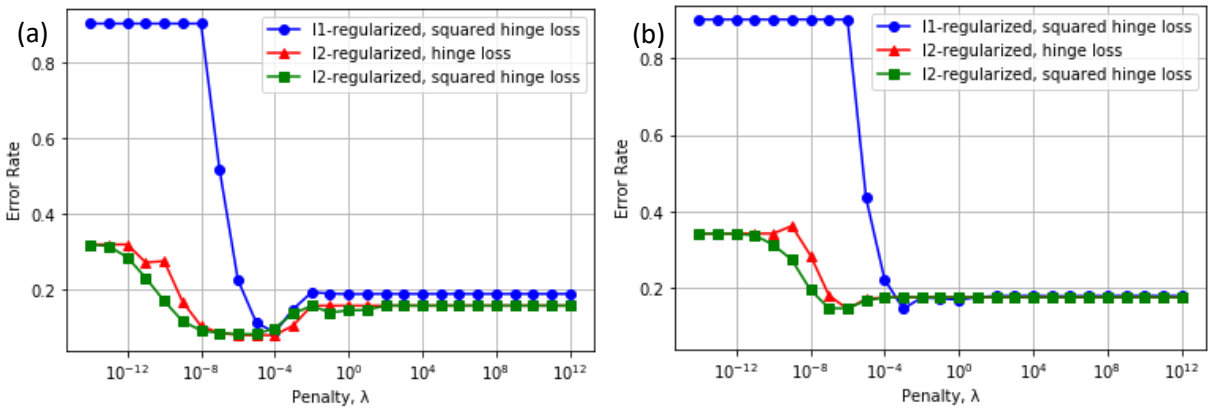

Fig 3. Validation error rate of hinge-loss classifiers with (a) 0th-degree model trained using 60000 images, (b) 0th-degree model trained using only 1000 images. The 0-D classifiers trained with only 1000 images achieve nearly the same accuracy as the 0-D classifiers with all 60000 training images.

Similar to L1-LS, polynomial degree had little impact on fitting performance of hinge loss and squared hinge loss. 1D and 2D fits are plotted in Fig. 4. Performance improvements are visible for λ>1E-4 when degree is increased for L2-regularized models, but this is after the "optimal" λ whereat polynomial degree appears to have no effect.

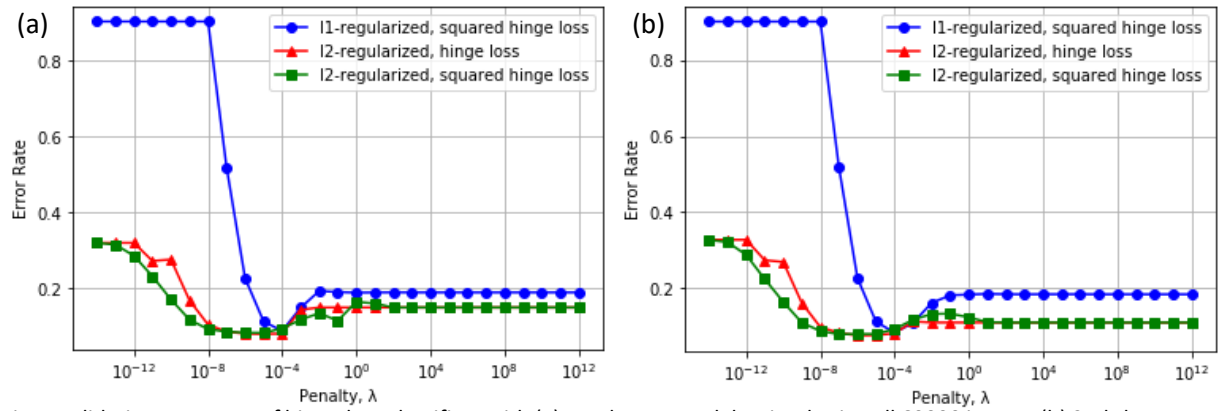

Fig 4. Validation error rate of hinge-loss classifiers with (a) 1st-degree model trained using all 60000 images (b) 2nd-degree model trained using all 60000 images.

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. Each model used the same activation function for all hidden layers.

Fig. 5 shows the results of the neural networks using the LS 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. Valuable information is likely lost as more layers are added.

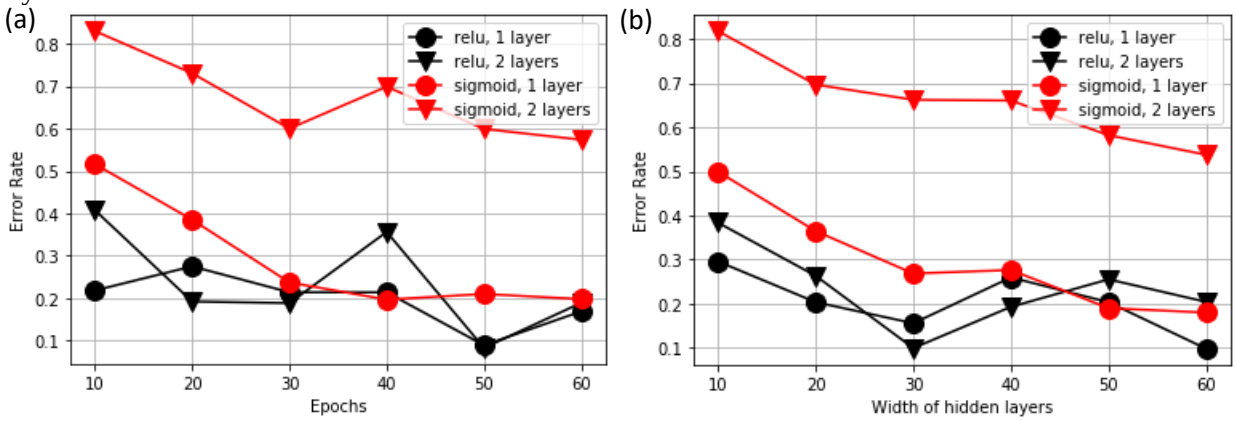

Figure 5. Validation error rate for neural networks using a LS loss function across (a) number of training epochs (b) number of nodes in each hidden layer. Neural networks could have had 1 or 2 hidden layers which were either exclusively relu nodes or sigmoid nodes.

Using the categorical cross-entropy loss function in tandem with the sigmoid activation function resulted in significant improvements in accuracy (Fig. 6). However, relu is incompatible with CCE when applied directly which resulted in predictions that were approximately equal to random guesses.

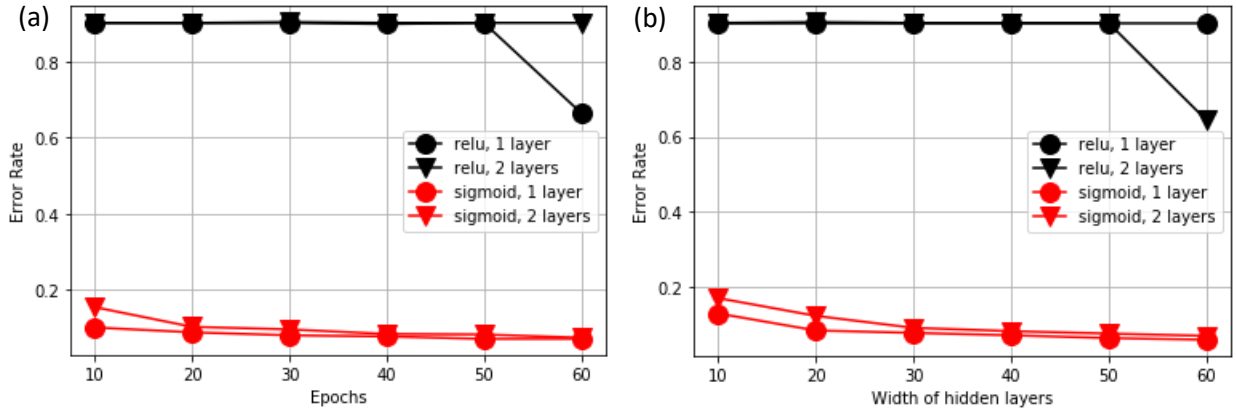

Figure 6. Validation error rate for neural networks using a CCE loss function across (a) number of training epochs (b) number of nodes in each hidden layer. Neural networks could have had 1 or 2 hidden layers which were either exclusively relu nodes or sigmoid nodes.

4. Strengths and limitations of methods

The L2-LS code took significantly longer to run than the L1-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 and limited resources, 0^{th} or 1^{st} degree L1-LS seems to be a good choice for a first attempt. L1-LS was simple, quick, and fairly accurate even with a small training set. Furthermore, unlike L2-LS, the error rate remained relatively invariant across a wide range of λ for the small data set. In other words, L1-LS seemed less prone to misuse. L2-hinge and L2-squared hinge also achieved good accuracy at low-degree, but performance was not as invariant across a wide range of λ as was the case with L1-LS. Hinge and squared hinge methods were able to achieve lower error rates compared to the other regression methods when the training set was expanded.

Neural networks performed poorly when dataset was small (not shown), but achieved high accuracy when using the whole training set. 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 and squared hinge 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, CCE, could only be used for a binary label matrix like the one that I used ("one-hot" encoded). Additionally, CCE is not directly compatible with relu. Flippantly applying activation functions can easily result in poor performance like relu's accuracy in Fig. 6. The addition of more hidden layers hindered model accuracy in the case of sigmoid, illustrating that performance cannot simply be improved by naively adding complexity. Neural network development is much subtler.

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. "7̸") 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

Given the full training data, the various methods implemented achieved optimal validation error rates between .05 and .20, with the exception of the mis-applied relu in Fig. 6. This exercise emphasizes a few key insights. First, there are many tweaks and tricks that can be attempted to achieve better performance in a classifier, but these can have wildly different effects depending on type of classifier being tweaked. These tweaks can range from adding features to extend fitting degree, changing regularization, changing SGD convergence criteria, and others.

However, in this exercise, changes that added to model complexity often resulted in worse performance or resulted in only modest improvements. L2-LS regression was the only method that saw minimum error improve when additional degrees were added, and it did so only for large training sets. Meanwhile, 0^{th}-degree hinge and square hinge methods were more accurate and faster than 2^{nd}-degree L2-LS in all circumstances evaluated. Similarly, increasing epochs and hidden layer width improved the accuracy of the LS neural networks, but the CCE sigmoid neural networks were able to achieve excellent accuracy with a small number of epochs, and a small hidden layer width. For optimal performance, starting with a broad and shallow survey of classifiers would appear to be more effective than fixating on a single method and trying to tweak it at length. Finding the correct classifier and *then* making judicious adjustments to appears to be an effective approach to designing classifiers.

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] L. Bottou, "Large-Scale Machine Learning with Stochastic Gradient Descent," in *Proceedings of COMPSTAT'2010*, 2010.

[3] F. Pedregosa FABIANPEDREGOSA *et al.*, "Scikit-learn: Machine Learning in Python," 2011.

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