**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 by oneself. To gain a working familiarity with machine learning algorithms, I have coded up and implemented several classification algorithms towards the goal of classifying handwritten numbers.

Generally speaking, algorithm complexity did not translate to improved model complexity. Only modest improvements, if any, were made by increasing the polynomial degree of fit for regression models. Depending on regularization parameters, increasing the polynomial degree of fit could actually result in overfitting. Similarly, neural network accuracy decreased as the number of hidden layers increased due to overfitting.

**2. Datasets and algorithms used**

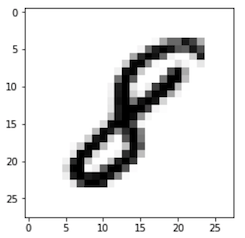
The dataset used in this project is the infamous MNIST dataset[1]. The MNIST dataset consists of 70000 28x28 pixel images of handwritten integer numbers from 0 to 9 (example provided in figure BLAH). Each image has a label that gives the integer number. The goal of working with the MNIST dataset is to 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 1 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.

Fig. 1. Example of MNIST image. Image is a 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.

Various classification algorithms were tested. The algorithms are split into two broad categories: 1) Regression and support-vector machine algorithms (summarized in table BLAH), and 2) Neural network algorithms (summarized in table BLAH). For the regression algorithms, regularization was necessary since number of images, outnumbered the 784 unique features. I used both L1 and L2 regularizations for regressions involving least squares, hinge, and squared hinge loss functions. Additionally, all regression and SVM algorithms examined 0th, 1st, and 2nd – degree polynomial fits for the data. 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 least squares algorithm was solved via matrix inversion while L2 least squares algorithm implemented gradient descent. All support vector machine algorithms implemented the LinearSVC function from python sklearn.svm and LinearSVC modules.

Table 1. Regression and support-vector machines



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

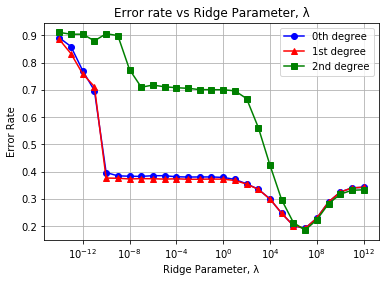
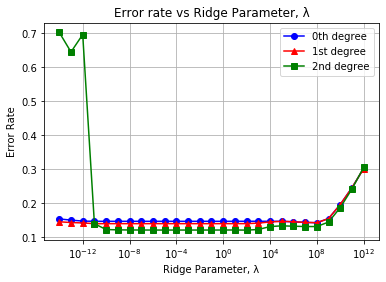
Neural Networks



**3. Results**

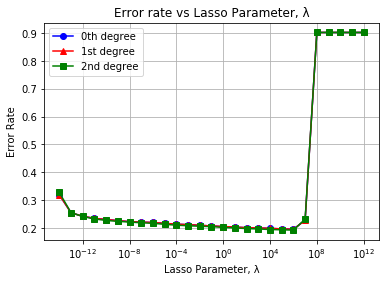
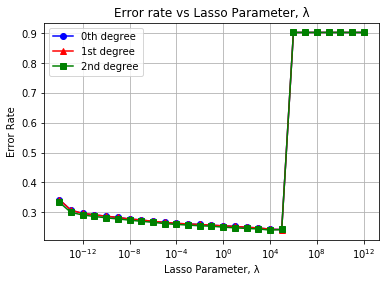
**3.1. Regression and support vector machines**

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.

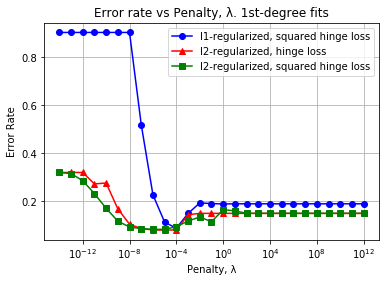
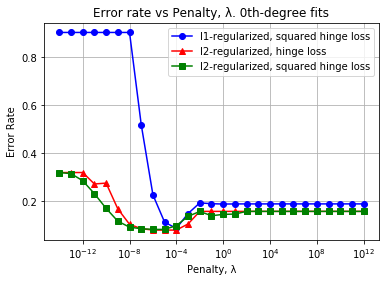


Figure

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.



The support vector machine algorithms showed notable variability across different regularization schemes (L1 and L2). However, polynomial degree had little impact on fitting performance. Similarly, choosing between hinge loss and squared hinge loss resulted in very similar fits and error rates. Squared hinge loss was chosen in addition to hinge loss since it was compatible with both L1 regularization and L2 regularization in the python sklearn.svm and LinearSVC modules.



(b)

(a)

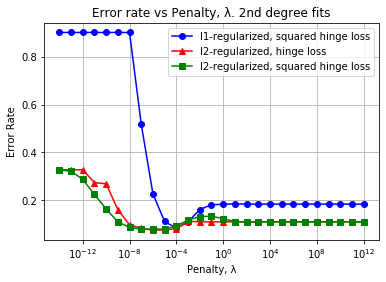
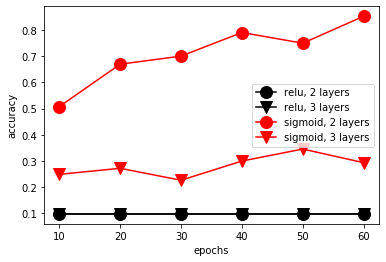
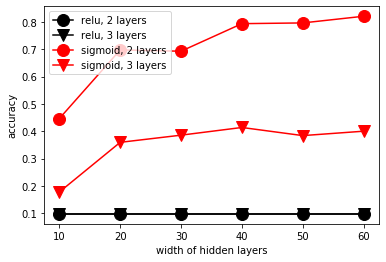
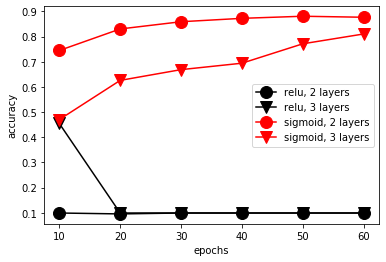
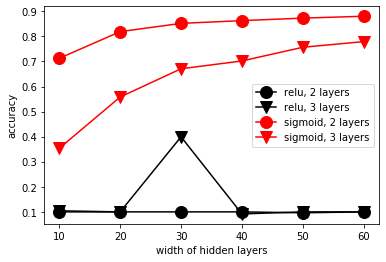


Fig 1. Testing error rate of hinge-loss classifiers for (a) 0th-, (b) 1st-, and (c) 2nd- degree polynomial fitting using various regularization schemes.

(c)

**3.1. Neural Networks**

**4. Strengths and limitations of methods**

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

Preprocessing is an important step to building a classifier. Thankfully, the MNIST dataset is well known for requiring little-to-no preprocessing to test machine learning algorithms https://medium.com/tebs-lab/how-to-classify-mnist-digits-with-different-neural-network-architectures-39c75a0f03e3.

For this reason, preprocessing was avoided in order to focus on algorithm implementation. Similarly, cross-validation can result in a more-robust assessment but is not necessary for achieving tangible results with the MNIST dataset.

**4. Conclusion**