Harrison Lam

Digit Recognition

In this project, I attempt to compare different machine learning methodologies where it is used to classify written digits from 0-9, trained on the MNIST dataset found on

https://www.kaggle.com/competitions/digit-recognizer/data

This dataset contains 42000 data points, with the input being a length 784 (28x28) array of numbers from 0 to 255, representing a 28x28 greyscale image.

First, we try to use logistic regression on 40000 training points and 2000 validation points on a simplified problem – modelling the probability of the output being equal to '0',

$$h_\theta(x) = \frac{1}{1 + e^{-\theta^T X}} = P(y_i = 0)$$

The log-likelihood of the training data is then

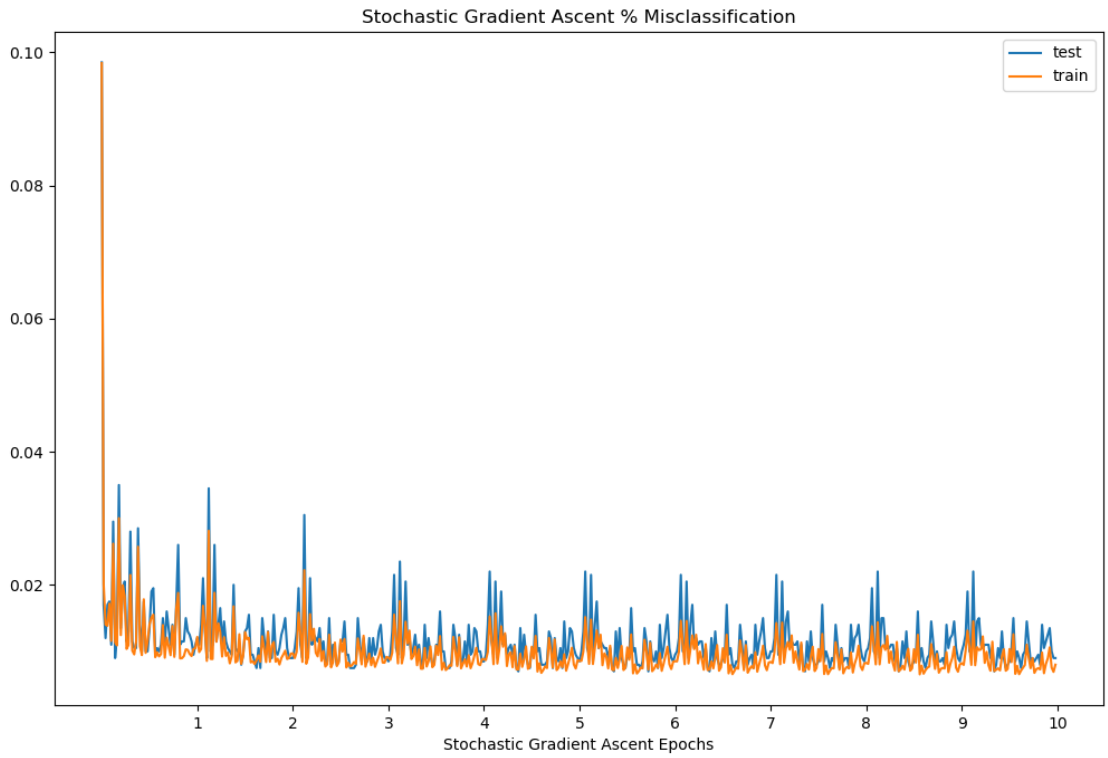
$$l(\theta) = \sum\nolimits_{i=1}^{n} \left(y_i \log\big(h(x_i)\big) + (1 - y_i)\log\big(1 - h(x_i)\big)\right)$$

and the gradient ascent rule for maximising this quantity:

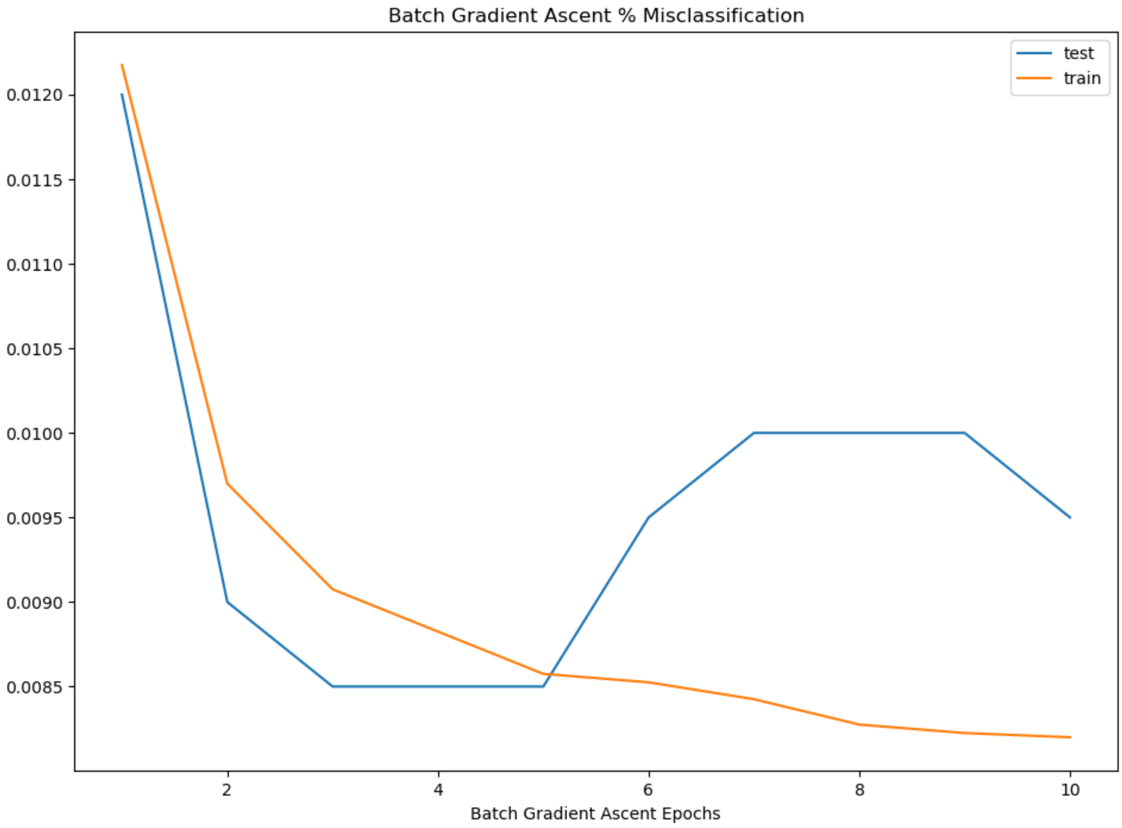
$$\theta_j = \theta_j + \alpha\big(y_i - h_\theta(x_i)\big)x_j$$

Where $y_i = 1\{output = 0\}$

We first compare stochastic gradient ascent and batch gradient ascent. With α set to 0.01, after ten iterations over the training set, stochastic gradient ascent misclassifies approximately 0.9% of the training set and 1% of the validation set.



This is very close to the misclassification rate of batch gradient descent, which is approximately 0.8% for the training set and 1% of the validation set.



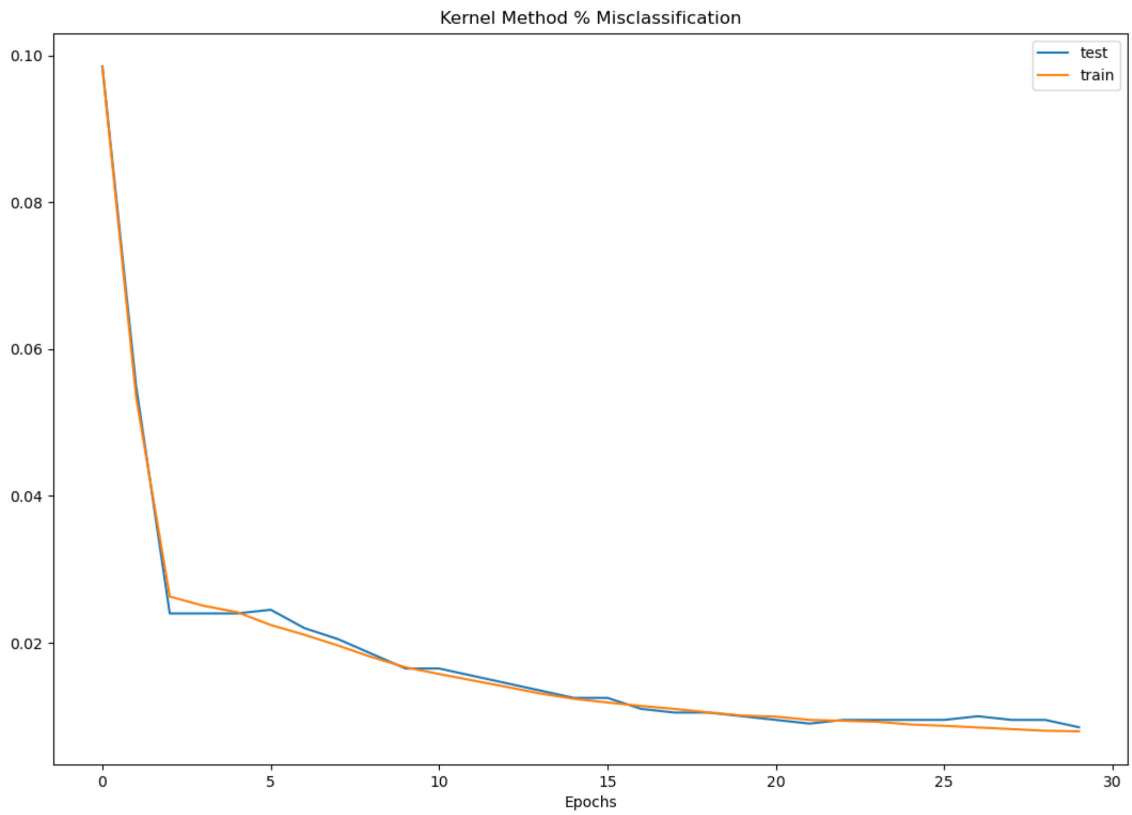
To further decrease the misclassification error, we implement the kernel method with a polynomial kernel, i.e.

$$K(i,j) = <\phi(x_i), \phi(x_j)> = 1 + <x_i, x_j> + <x_i, x_j>^2 + <x_i, x_j>^3$$

We note that θ is a linear combination of the $\phi(x_i)$'s at every step, therefore by substituting $\theta = \sum_{i=1}^{n} \gamma_i \phi(x_i)$ into the gradient ascent equation, we obtain the update rules for the γ_i's instead:

$$\gamma = \gamma + \alpha(Y - expit(K\gamma))$$

where expit represents element-wise application of the function $\frac{e^x}{e^x+1}$.



This method achieves a similar misclassification rate as previous methods (training: 0.8%, validation: 0.9%) only after 30 epochs, and does not decrease any further with more passes.

We now return to the original classification problem. There is a very natural extension to logistic regression – softmax regression – for multi-class classification. In softmax regression,

$$P(y = j \mid x, \theta) = p_x^j = \frac{e^{\theta_j^T x}}{\sum_m e^{\theta_m^T x}}$$

and the objective is to maximise the log-likelihood:

$$l(\theta) = \sum_{i=1}^{n} \log \frac{e^{\theta_{y_i}^T x_i}}{\sum_m e^{\theta_m^T x_i}}$$

We can maximise this using gradient ascent, with the gradient being

$$\nabla_{\theta_k} l(\theta) = \sum_{i=1}^{n} (1\{y_i = k\} - p_{x_i}^k)\, x_i$$

With each of the pixels (values in [0, 255]) scaled to [-1, 1], this method achieves approximately 8% error on both the training and validation set even after hours of iterating through the training set (~12000 iterations), and does not improve much with further iterations. Although line search gradient descent results in a bigger improvement per step, each step is much more computationally expensive and does not improve much past 8% error either. By scaling the values to [0, 1] instead of [-1, 1], we observe that the method achieves much better convergence (negligible effect on previous methods). After approximately 100 iterations, the misclassification is at 9% for both the training and validation sets. To further improve the performance on unseen data, we implement Monte Carlo 10-fold cross-validation, selecting 40000 random data points from the 42000 points as the training set with the other points being in the validation set. We then take the element-wise average of the 10 sets of parameters obtained as the final parameter θ. To test the final performance, we use this θ to predict the 28000 data points in the test set, achieving a 90.175% classification accuracy.