Final Report
Linear Programming

Yang-Shan Chen, Yu-Hsiang Liu, Yu-Chi Wang, Chih-En Lin

Part I

1 Algorithm Description

The goal is to find a linear classifier $f(g(y))$ and $g(y) = \theta^T y$. In our method, the decision function $f(x)$ is trivial as following.

$$f(x) = \begin{cases} 1, \; if \; x > 0 \\ 0, \; if \; x \leq 0 \end{cases}$$

Therefore, we focus on how to find the best coefficient θ to approach this problem.

In the first step, we selected a small portion (e.g. 120 points) of the training data to build up an initial classifier. With this classifier, we can identify which data points are useful and can be implemented into the rest of the training data. If a data point is close enough to the classifier, we will select it to be trained. After we have selected a certain amount (e.g. 12 points) of data points, we will update our classifier to be more accurate. In the end, we were able to obtain the final classifier after going through the whole training set.

The method we used to build the classifier is gradient descent with logistic regression, and the following equation is our cost function.

$$\text{Cost}(\theta) = -\frac{1}{m}\sum_{i=1}^{m}[y_i \log(h_\theta(x_i)) + (1 - y_i)\log(1 - h_\theta(x_i))], h_\theta(x) = \frac{1}{1+e^{-\theta^T x}}$$

Where y is the label and x is the feature. We can see that if $\theta^T x$ is much greater than 0, then $h_\theta(x)$ will be close to 1. In this case, we won't get punished if $y = 1$. However, we will get penalized heavily if $y = 0$. And it's the same concept when $\theta^T x$ is much less than 0. So we know it's a reasonable cost function which we expect to minimize.

In the training process, we iterated through each data point and calculated the gradient by getting the partial derivative of the function. Then, we updated the weight and bias of the classifier according to gradient and learning rate.

$$\theta_i = \theta_i - \alpha\frac{\partial}{\partial\theta_i}\text{Cost}(\theta), \; \alpha \text{ is the learning rate}$$

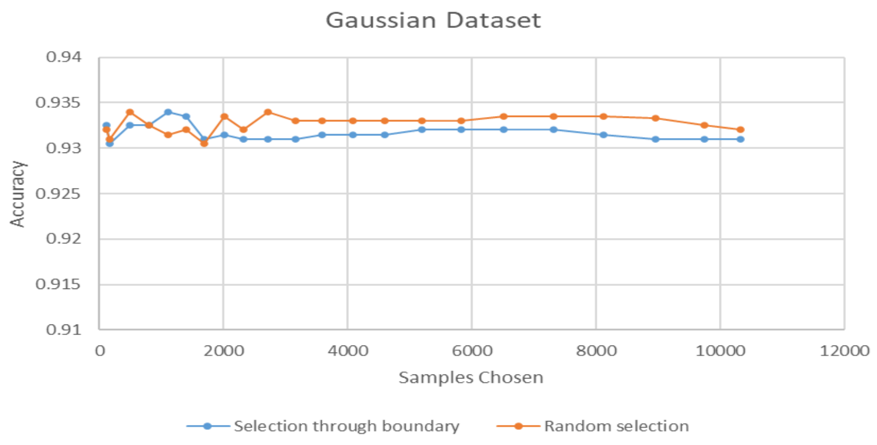
2 Parameter Elaboration

m: number of total training set
n: number of selected samples
x: feature of selected samples

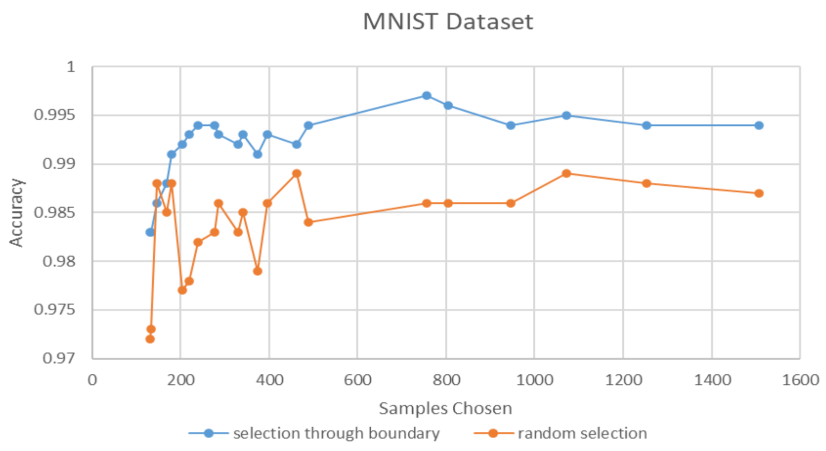
y: label of selected samples
dim: the dimension of data points plus 1(constant feature)
θ: coefficients of classifier, including weight and bias
mean: mean value of selected samples
std: standard deviation of selected samples
pretrain: selected $\frac{m}{\text{pretrain}}$ samples to build the initial classifier
batch: updated our classifier whenever our algorithm found $\frac{m}{\text{batch}}$ more useful samples
boundary: If the distance from the data point to the hyper-plane of the classifier was smaller than the boundary, we would select it. We manipulated this variable to change the number of chosen training sample points.
lr: learning rate of training model

3 Performance

Our Algorithm versus Randomly Selected
Gaussian Dataset



In the Gaussian dataset, the accuracy difference of the two methods is less than 0.3%. Due to the fact that the accuracy difference is so small, we were not able to identify that our algorithm is better than the random selection method. Moreover, the accuracy of the two methods are already high even when using a small sample size, so we couldn't observe the trend of increasing accuracy.
MNIST Dataset



In the MNIST dataset, the accuracy of our algorithm slightly outperforms the random selection by 1.0% on average. As shown in the graph, we can see that accuracy is increasing along with the increment of chosen

samples. As the number of chosen samples reaches 300, the trend of accuracy becomes stable.
Our Algorithm versus Full Training Dataset
Gaussian Dataset:
Accuracy: 0.9325 v.s. 0.931, Selected samples: 120 v.s. 12000
As you can see from the results of the Gaussian dataset, we only used 120 data points to achieve almost the same accuracy.
MNIST Dataset
Accuracy: 0.9910 v.s. 0.9908, Selected samples: 180 v.s. 13007
As you can see from the results of the MNIST dataset, we only used 180 data points and achieved almost the same accuracy.
Number of Data Samples to Achieve the Values of Accuracy(50%, 65%, 80%, 90%)
Our algorithm is good enough so that even with just a few samples, we were able to obtain a high accuracy. With only 120 chosen data points in the Gaussian dataset, we can get over 93% accuracy. With only 130 chosen data points in the MNIST dataset, we can get over 98% accuracy.

Part II

1 Algorithm Description

We came up with two methods for part II. First, we found two center points of two kinds of labels c_1, c_2, that is, the mean value of the data.
Method1: We found the perpendicular bisector of the line segment of these two center points, the line p_0, p_1. If a data point is close enough to this perpendicular bisector, it will be selected and included in the process of training our classifier.
Method2: We divided the two distances of the data point x and center points.That is to say, $\frac{\overline{xp_0}}{\overline{xp_1}}$. If this ratio is close enough to 1, it will be selected and included in the process of training.

2 Parameter Elaboration

In addition to using the same parameters from part I: m, n, x, y, dim
dis: Used in **Method1**. If the ratio of the distance from the data point to the perpendicular bisector and the distance $\overline{p_0p_1}$was smaller than this dis and in the right half-space, we would select it. By manipulating this variable, we were able to change the number of chosen training sample points for **Method1**.
divide: Used in **Method2**. We would select the data point if the ratio between the two distances of the data point and the two center points was smaller than this divide, larger than 1/divide, and also happened to be in the right half-space. We also manipulated this variable to change the number of chosen training sample points for **Method2**.

3 Integer Linear Programming

Method1
$\min \mathbf{1}^T n$, subject to: $n_i \in \{0,1\}, b_i = (p_1 - p_0)^T(x_i - \frac{p_1+p_0}{2})$

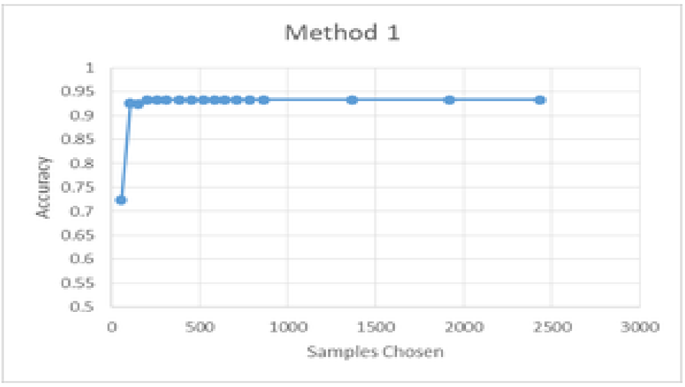
$$n_i = \begin{cases} 1, \; if \; [(b_i > 0 \; and \; y_i = 1) \; or \; (b_i <= 0 \; and \; y_i = 0)] \; and \; \frac{b_i}{\overline{p_0p_1}^2} \leq \text{ dis} \\ 0, \; otherwise \end{cases}$$

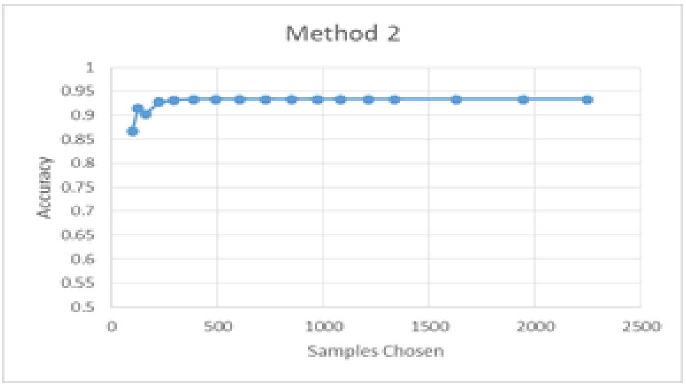
where x is the feature and y is the label, $n_i = 1$ means that we choose this sample. At the same time, we still want to maintain the accuracy.
Method2:
Method2 is basically the same as Method1.

We only modify the condition $\frac{b_i}{p_0 p_1{}^2} \leq$ dis to $\frac{1}{\text{divide}} \leq \frac{||x_i - p_0||}{||x_i - p_1||} \leq$ divide
However, it's not a linear constraint since we have 2-norm terms of x_i.

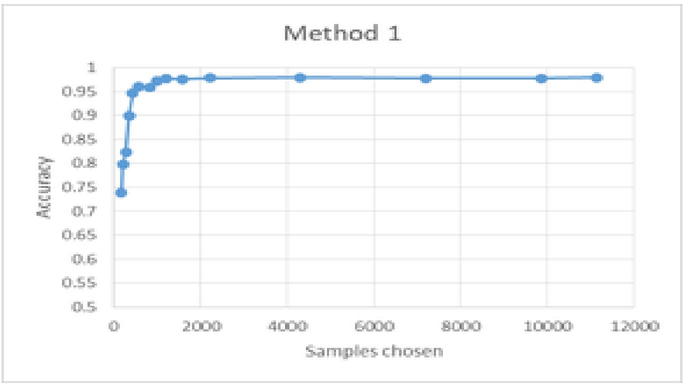
4 Performance

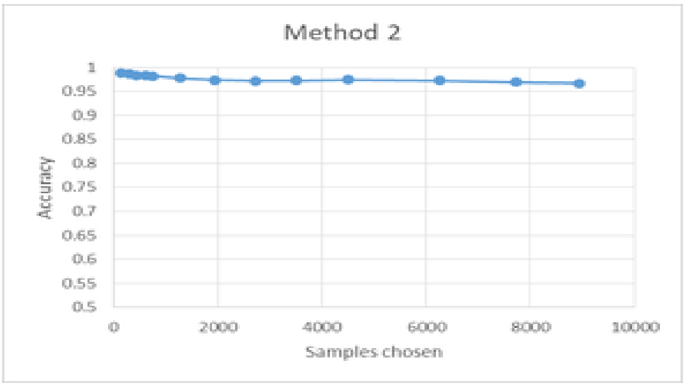
Gaussian Dataset





Both **Method1** and **Method2** have the same trend in the Gaussian dataset. After reaching 250 samples chosen, the accuracy is higher than 93%.
MNIST Dataset





The result of **Method1** and **Method2** differ when the chosen samples are less than 400. But after reaching 400, both the accuracies become very high, at about 98%.
Number of Data Samples to Achieve the Values of Accuracy(50%, 65%, 80%, 90%)
Gaussian:
Method1(accuracy/number of samples):52.95%/17, 66.15%/50, 80.7%/62, 91.3%/68
Method2(accuracy/number of samples):50.4%/13, 65.1%/36, 79.8%/90, 90.7%/102
MNIST:
Method1(accuracy/number of samples):53%/48, 66.48%/146, 79.8%/217, 90.01%/352
Method2(accuracy/number of samples):69.39%/36, 88.07%/44, 96.25%/63
PartI v.s. PartII
For the Gaussian dataset, the difference between Part I and Part II (regardless of Method1 or Method2) are observable when the chosen samples are less than 250. After reaching 250, the accuracies remain at the highest level, around 93%. For the MNIST dataset, the differences in Part I and Part II are similar to the Gaussian dataset. In other words, when the chosen samples happen to be less than 1000, the accuracies from Part I and Part II (Method 1) can be differentiated. But after reaching 1000, both the accuracies remain at the highest level, 98%. The reason is that the algorithm we used in Part I is training data to build up the classifier. While in Part II, we calculate the two center points, their distance, and the perpendicular bisector. Therefore, when the chosen samples are scarce, the methods we used in Part II will not be accurate, and we will get a worse accuracy than Part I.

Method1 v.s. Method2

We observe that Method2 is better than Method1 while choosing fewer samples(MNIST Dataset). We think that since we have a higher probability to choose samples a farther from the bisector in Method2, we can get a more accurate support vector. The bisector is only an estimate of the classifier, thus if we only get samples extremely close to the bisector, we may only get samples that are actually error data in the perfect classifier.

Appendix

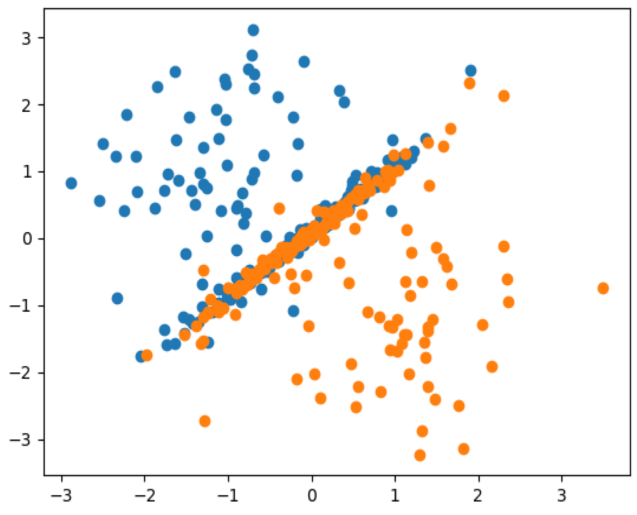
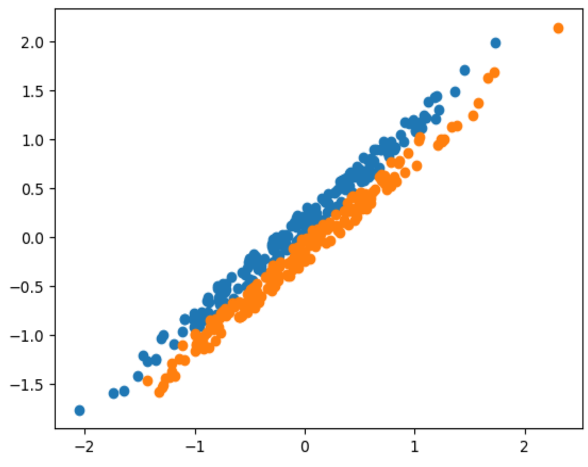


Figure 1: samples chosen by PartI algorithm(391 samples, accuracy: 93.05%)



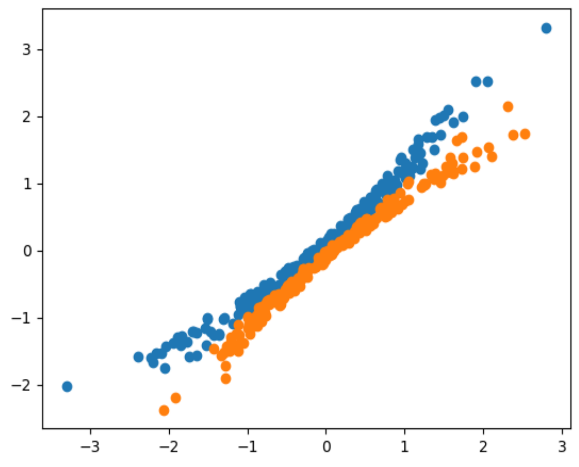


Figure 2: samples chosen by Method1 in PartII algorithm(387 samples, accuracy: 93.3%)

Figure 3: samples chosen by Method2 in PartII algorithm(389 samples, accuracy: 93.3%)