ECE236A Project

Group 18: Catherine Holly Frank, Rushi Bhatt, Ronak Kaoshik

December 9, 2021

1 Part 1

In this project, we designed a binary linear classifier object called MyClassifier. This classifier object can train itself, classify test points, and choose "ideal" training points from a stream of input data. Specifically, the classifier decides whether or not to query each point without knowing any other points in the data set in order to minimize the upload cost.

1.1 train() and test()

The train() function uses a linear program to find a hyperplane $W^T x + b = 0$ that separates the two classes as well as possible. The problem is as follows:

$$\min_{W,b} \sum_{i}^{N_{train}} \max\{0, 1 - S_i(W^T x_i + b)\} \qquad (1)$$

where S_i is the mapped label for data point x_i and $W \in \mathbb{R}^M$, $b \in \mathbb{R}$. The labels are are mapped so that $S_i \in \{-1, 1\}$ regardless of the input labels. This mapping is saved so that we can work in terms of S_i but return the correct labels in test(). We solved equation (1) for W and b using the library cvxpy and the train() input data - an $N_{train} \times M$ matrix containing N_{train} data points with M features each and the associated $N \times 1$ vector of labels. The MyClassifier object is then updated to store the optimal weights, W, and bias, b.

The test function uses the trained weights and bias to classify each input test point. $g(y_i) = W^T y_i + b$ is computed for each y_i in the test set. Then, if $g(y_i) >= 0$, the maximum label is assigned, since y_i lies above the hyperplane. If $g(y_i) < 0$, y_i is below the hyperplane and the minimum label is assigned.

1.2 Sample Selection

In designing our sample selection method, we assumed that each data point is located on a sensor that must independently decide whether or not to transmit data to the training node. Each sensor can download information from the central node (training node) for free - that is, it can access W and b - but there is a cost for uploading any data to the node. Thus, the sample selection method must decide whether to query the label S_i of each point x_i without knowledge of any other points in the set.

Our method iteratively trains the classifier starting from two data points in the data set - one from each class. To ensure that we choose a random two points, we shuffle each data set and choose the first data point with each label. Then, we initialize the classifier's weights and biases using these two points. Next, the algorithm loops through each remaining point, x_i, in the data set and evaluates $|g_k(x_i)| = |W_k^T x_i + b_k|$, where W_k and b_k are the weights and bias trained on k data points. If $|g_k(x_i)| \leq \delta$, then x_i is close to the hyperplane. Thus, it is uncertainly classified by the current model, and it should be included in the training set. Therefore, if $|g_k(x_i)| \leq \delta$, then our algorithm selects x_i, queries S_i, and adds them to MyClassifier's training set and labels objects, respectively. Then, we retrain the classifier on the new training set of size $k+1$ and repeat for all x_i. The code for this is shown in Algorithm (1).

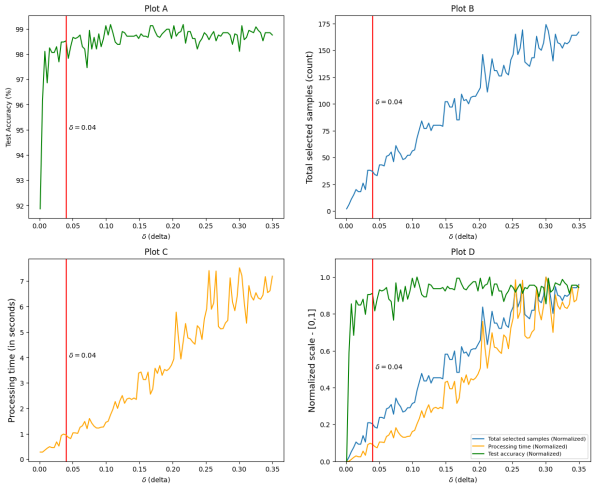

Figure 1: Selecting δ value

Algorithm 1 Our proposed sample selection algorithm

Require: $N_{train} \geq 2$

1. Select point x_{c0} and x_{c1} each from class c_0 and c_1
2. $train()$ the classifier with x_{c0} and x_{c1}

for $i = 2$ to N_{train} **do**
  Calculate $g_k(x_i) = W_k^T x_i + b_k$
  if $g_k(x_i) \leq \delta$ **then**
    Add x_i to $trainSetData$
    Add s_i to $trainSetLabels$
    $train()$ classifier with $k + 1$ points
  end if

Our method leverages iterative training for the classifier to constantly improve the selection criteria for the stream of data and successfully reduces the necessary sample size for training.

1.3 Results

We chose $\delta = 0.04$ by comparing accuracy and classification time over a large range of δ values, as shown in fig-

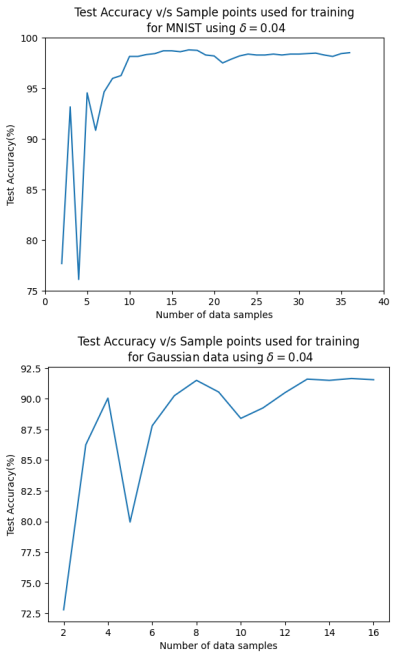

Figure 2: Sample Selection algorithm's classification Accuracy over number of data points selected for MNIST and Gaussian

MNIST Dataset		Gaussian Dataset	
Acc.	**Avg Points**	**Acc.**	**Avg Points**
80 %	3	50 %	3
90 %	6	65 %	5
95 %	10	75 %	9
97 %	15	85 %	12
98.52 %	25	91.5 %	17
100 %	13007 (full set)	**91.75 %**	10000 (full set)

Table 1: Our proposed Sample Selection method's performance on MNIST and Gaussian data sets.

		Sample Sel.	**Rnd. Samp.**
	Acc.	**Avg. Points**	**Avg. Points**
MNIST Dataset	80 %	3	10
	90 %	6	17
	95 %	10	35
	98 %	23	130
Gaussian Dataset	50 %	3	3
	65 %	5	6
	85 %	12	14
	91 %	16	25

Table 2: Sample Selection and Random Sampling performance on MNIST and Gaussian data sets.

ure (1). Specifically, this value was selected for data sets with features mapped to the range $[0, 1]$. $\delta = 0.04$ yielded good accuracy results across all classes in the MNIST data set and Gaussian distribution and ran quickly.

Since the main objective of this problem was to reduce the number of selected training samples N_{train}, we studied the relationship between classifier accuracy and N_{train} for the Gaussian and MNIST data sets, as shown in figure (2). While the accuracy is high at very low N_{train} ($> 70\%$), the true power of this algorithm is seen once $10 - 15$ samples are selected. Then, the accuracy increases to almost ideal (100% for MNIST and 91.75% for the Gaussian) while still only selecting fractions of a percent of the training data. This would cut upload costs by close to 99.9% without sacrificing much accuracy.

Table (1) shows a numerical comparison between the number of points needed to achieve specified classifier accuracy using our algorithm for classes 1 and 7 in the MNIST data set and for classes 1 and -1 for the Gaussian data set. On average, our method achieved remarkably high accuracy for low N_{train}, achieving 98.52% of the accuracy for the classifier trained on the full MNIST data set with only 0.19% of the training points. For the Gaussian, we were able to achieve 99% of the full classifier accuracy with 0.16% of the training points.

Table (2) compares the difference between our sample selection method and randomly choosing training points from the data set. For the MNIST data set, we need about $3-6$ times more random points to achieve the same accuracy as the sample selection. For the Gaussian, we found that we needed about $1 - 1.5$ times the amount of data points. Thus, our method outperforms random sampling. The moderate increase for the Gaussian is likely due to the normal distributions of the classes - randomly selected points are likely to give a good estimation of the true distribution of the data.

We can also consider the accuracy graphically. Figure (3) shows the full Gaussian data set colored by class. The blue line represents the ideal hyperplane trained on all 10000 points, and the orange line represents the hyperplane trained on 21 sample selected points. The lines are close together, and clearly both do a good job of classifying the majority of the points in the sets. We could further decrease the difference between the two lines by including more training points near the boundary.

2 Part 2

In the second part of the project, we assume a genie is given access to the whole labeled data set and can decide which points should be selected. Specifically, we have an ILP-loving genie who wants to use an integer linear

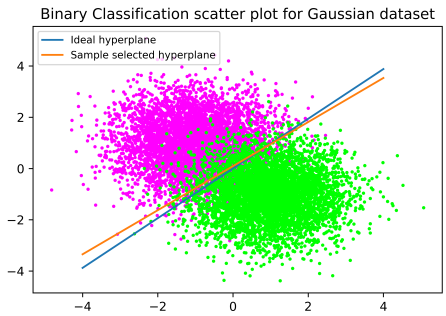

Figure 3: Scatter Plot for Class -1 v/s Class 1

programming approach to choose the points.

2.1 Candidate ILPs

In our tests, we explored two main methods for constructing this ILP. The first looks for the points in the middle of the two classes. This creates a set of points that are close to the boundary between the classes, as shown in the black circle in figure (4). While this is a good method for achieving high accuracy when selecting a large number of points for classification, the performance for smaller data sets was poor. The second method compares each point to the average of each class and selects the points closer their own class's average than the others, as shown in the red circles in figure (4). For this method, the classification is good for small data sets, but does not improve much when we increase the number of training points. We then combined these two methods to find an ILP that gives good results for very small and large N_{train}. We formulated an ILP to select some points close to the class averages to capture the classification between the majority of the data and some points in the boundary region between the two classes to refine the hyperplane.

2.2 ILP Formulation

Our genie finds the N_{train} "ideal" points from a set of N possible points for classification by solving the linear relaxation of the following integer linear program:

$$\min_{\lambda_1,\lambda_2} \quad d_1^T\lambda_1 + d_2^T\lambda_2 \tag{2}$$

$$\text{subject to} \quad 1^T\lambda_2 >= c * N_{train} \tag{3}$$

$$\lambda_1 + \lambda_2 \leq 1 \tag{4}$$

$$1^T(\lambda_1 + \lambda_2) \leq N_{train} \tag{5}$$

$$\sum_i^N S_j(\lambda_{1i} + \lambda_{2i}) = 0 \tag{6}$$

$$0 \leq \lambda_1, \lambda_2 \leq 1, \quad \lambda_1, \lambda_2 \in \mathbb{Z}^N \tag{7}$$

where

$$d_{1i} = S_i||x_i - x_{c1}||_2 - S_i||x_i - x_{c2}||_2 \tag{8}$$

$$d_{2i} = |W_{avg}^T x_i + b_{avg}| \tag{9}$$

and x_{c1} is defined as the average of all points with mapped labels $S_i = 1$ and x_{c2} is the average of all points with mapped labels $S_i = -1$. Further, W_{avg} and b_{avg} define the ideal hyperplane separating these two points. We leverage our train() code to find these values, but the genie could also find these values by simply connecting the two averages with a line and finding the orthogonal hyperplane equidistant from both points. d_{1i} represents the difference between the distance from point x_i to its own class average and the distance between point x_i and the opposite class average. That is, if $d_{1i} < 0$ and $S_i = 1$, x_i is closer to x_{c1} than x_{c2}. d_{2i} represents the distance between x_i and the hyperplane separating the class averages. Small d_{2i} means the point is close to the hyperplane boundary region, so it will be important for refining the classification. We want to choose points that minimize d_{1i} or d_{2i} subject to some constraints, so we introduced two indicator variables λ_1 and λ_2, where $\lambda_{1i} = 1$ means x_i is being selected to minimize d_{1i} and $\lambda_{2i} = 1$ means x_i is being selected to minimize d_{2i}, as is reflected in the objective function, equation (2). Constraint (7) ensures that $\lambda_1, \lambda_2 \in \{0, 1\}$ and, in combination with cons. (4), ensures that each point x_i can only be selected either for the proximity to the true class mean or to the boundary region, but not both. Constraint (3) requires that a fraction c of the selected points be close to the hyperplane. This allowed us to adjust how many points the genie selects from the boundary region, as discussed in section 2.4. Constraint (5) limits the total number of selected points to less than the requested number of samples. Finally, cons. (6) requires that the genie choose an equal number of points from each class. This is included to ensure robust performance at low N_{train}.

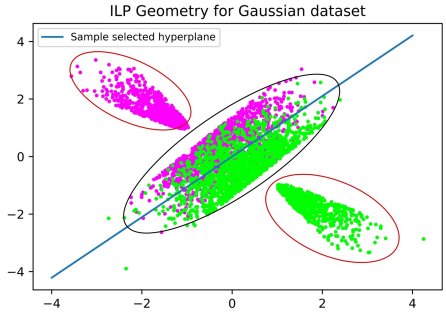

Figure 4: ILP Geometry for Gaussian data set

2.3 LP Relaxation

Rather than solve this computationally intensive ILP, we relaxed constraint (7) by letting $\lambda_1, \lambda_2 \in \mathbb{R}^N$. Then, we solved this LP relaxation using cvxpy to get the ideal λ_1 and λ_2. We constructed the integral solution by rounding λ_1 and λ_2 to get integral vectors and then including the point x_i in the training set if $\max(\lambda_1, \lambda_2) == 1$. That is, we include x_i if it is included in either of the integral solutions for λ_1 or λ_2.

2.4 Results

Figure (4) shows 4000 points selected by our ILP implementation. We can see that this method successfully selects points in the overlap region between the two classes, circled in black, and points closer to the their own class average than the other, circled in red. The first set corresponds to the x_i where $\lambda_{1i} = 1$ and the second to where $\lambda_{2i} = 1$. Our unique, two indicator variable approach described in section 2.3 allowed us to choose these disjoint sets to create a robust ILP across different N_{train}. The ratio c in equation (3) controls the relative size of each set. We selected $c = 7/10$ for our algorithm because we found this achieved the highest accuracy for a fixed N_{train}, as shown in figure (5).

Figure (6) shows the classifier accuracy vs. N_{train} for

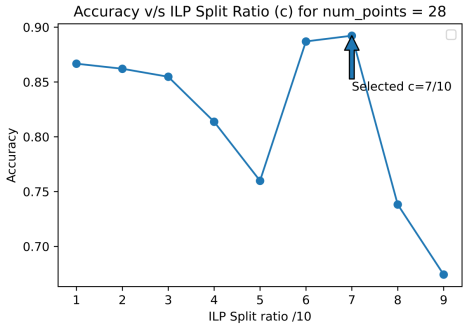

Figure 5: c, the fraction of points selected in the boundary region, vs. accuracy for $N_{train} = 28$.

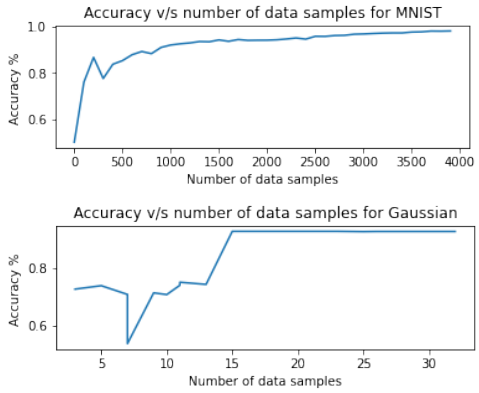

Figure 6: ILP's classification Accuracy over number of data points selected for MNIST and Gaussian

the MNIST and Gaussian sets. We can see that the classifier performance rapidly increases at low N_{train} and then levels off as we increase N_{train} further and approach the maximum accuracy. This is also seen in table (3), which shows that we can achieve 95% of the ideal classification accuracy with roughly 17% of the data for the MNIST set, and almost 100% of the ideal classification accuracy with 0.15% of the data set for the Gaussian. This large performance difference is due to how well each set is characterized by the class means.

If we only selected the black circled points in figure (4), then we would have poor $N_{train} \lesssim 100$ accuracy. On the other hand, if we only selected the red circled points, the accuracy would approach the ideal accuracy much slower. Thus, we chose to combine these selection methods with a two-indicator variable LP because we found it performed best for $N_{train} \lesssim 100$ while still approaching ideal accuracy reasonably fast.

MNIST Dataset		Gaussian Dataset	
Acc.	**Avg. Points**	**Acc.**	**Avg. Points**
50 %	3	53 %	7
80 %	23	70 %	10
90 %	28	75 %	13
95 %	2300	92.7 %	15

Table 3: Our ILP method's performance for MNIST and Gaussian data sets.

		ILP	**Sample Sel.**
	Acc.	**Avg. Points**	**Avg. Points**
MNIST Dataset	50 %	3	<3
	80 %	23	3
	90 %	28	6
	95 %	2300	10
Best Acc.		99.8 % for 9000 pts.	98.52 % for 25 pts.
Gauss. Dataset	50 %	7	3
	70 %	10	6
	75 %	13	7
	85 %	14	11
Best Acc.		92.7 % for 15 pts.	91.5 % for 15 pts.

Table 4: Our proposed ILP's performance v/s Random Sampling's performance on MNIST and Gaussian dataset

3 Stream-Based Selection vs. ILP Genie

Table 4 compares our stream-based sample selection method to the ILP sample selection method. As we can see, the iterative sample selection method out performs the ILP genie for small N_{train}. This is because we assume that the genie must select all points *before* training the classifier, whereas the sample selection classifier retrains itself each time a new data point is selected. This means that the selection criteria is constantly getting better for the stream-based approach, whereas the ILP makes it's decisions without training the classifier - a useful approach when training is computationally expensive. However, we can also see that the best accuracy is higher for the ILP - this is because the sample selection method can only select a point if it is close to one of the iteratively trained hyperplanes, which limits the maximum N_{train} selected, whereas the ILP can continue adding points to increase the classification accuracy.

Our main takeaway from this project is that we must choose the sample selection method to fit the task. If training the classifier is free, then our method from part 1 works very well - by iteratively retraining the hyperplane, the selected points will be closer and closer to the ideal hyperplane. If training the classifier is expensive, then our ILP selection genie would be a good choice. This method selects the most important points for a classification task and only trains the classifier twice - though the first train() call could be replaced by a linear expression, so really only a single train() call is required. By exploring different sample selection techniques, we realized that it is vital to understand the classification task - such as the system architecture, data transfer costs, and training costs - in order to design an ideal sampling algorithm.