Distributed System for ML Classifiers

Shivangi Yadav and Achsah Ledala

(1) Problem Statement:

Implement a Distributed System for machine learning classifiers for improved performance by distributing parameters over different servers. We would further analyze the effect of feature size and number of parameters, used for fine-tuning the algorithm, on the performance of classifier.

(2) Motivation:

- Machine Learning based systems are becoming the backbone of modern technologies
- Examples are: Image classification, Recommendation systems, Biometric-based authentication systems, NLP, etc.
- Data is available in abundance i.e., input data size is increasing
- Complex algorithms need multiple parameters to be trained on i.e., can have hundreds and thousands of configurations
- Computational power of single machine can be a bottleneck for classifier's performance, time, and accuracy

(3) Introduction and Background:

In order to build Machine Learning models with high accuracy, a large range of parameters should be explored to train the data. Training the model on these parameters and evaluating the accuracy each time, to choose the best parameters for the given dataset is a time consuming process. Due to this time constraint, we usually explore only a few parameters and build a model having moderately good accuracy within the explored parameters. Using Distributed Systems concept, we can distribute the parameters to a number of systems and thus potentially reduce the time taken to train the models on a large number of parameters to build a model with the best accuracy. In this project we proposed to use this concept in order to choose the best parameters for a model for a given dataset. To achieve this, we have built a distributed system for machine learning classifiers such as,

- **Support Vector Machine (SVM)**
  SVM is a discriminative classifier formally defined by a separating hyperplane. In other words, given labeled training data (supervised learning), the algorithm outputs an optimal hyperplane which categorizes new data samples. In two

dimensional space this hyperplane is a line dividing a plane in two parts where in each class lay in either side as shown in Figure 1.

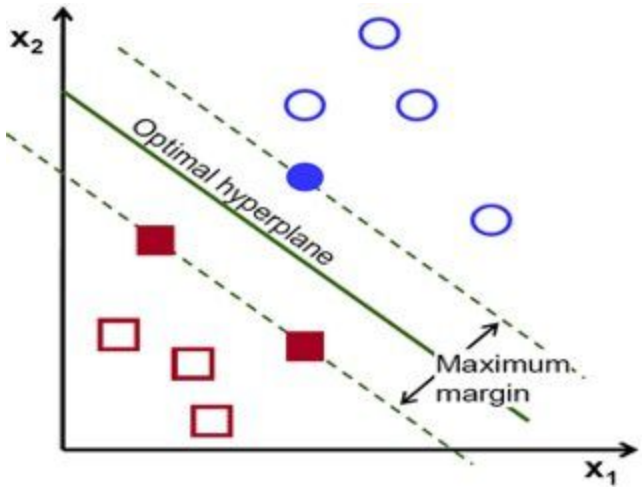

Figure 1: This illustrates a simple example of how SVM classify data points on the basis of their likeness with each other. As given in the figure above, square and circles represents two classes that are separated by a decision boundary (optimal hyperplane with maximum margin). The data points that lies on margins are known as support vectors that helps decide a soft decision boundary over given dataset.
Reference:https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47

The important parameters used in training a SVM classifier involve gamma, a regularization parameter C, a kernel function used to define decision boundary and degree of freedom for kernel function. SVM should have a good margin where the decision boundary is equidistant from both the classes (as shown in figure 1) and allows the points to be in their respective classes without crossing to the other side. Regularization parameter is used to optimize or regulate the number of misclassified points. The gamma parameter is used define how much a training sample influences a decision boundary. Gamma and C are tuning parameters for a nonlinear SVM Classifier with a radial basis function as kernel. If the kernel is linear, then a straight line or a hyperplane is learned as the decision boundary. Otherwise,the decision boundary is non-linear and depends on the degree of kernel function. The range of these parameters used for our project is given below:

- Kernel: Linear, Polynomial, RBF
- Degree: 1 - 4
- Gamma: -log(9) to log(9)
- C: -log(2) to log(14)

- **Artificial Neural Network (ANNs):**
  ANNs are statistical models that are capable of modeling and processing linear and nonlinear relationships between inputs and outputs in parallel. They are characterized by containing adaptive weights along paths between neurons that can be tuned by a learning algorithm that learns from observed data in order to improve the model. In a simple network, the first layer is the input layer, followed by one hidden layer, and lastly by an output layer (see Figure 2). In the literature, many researchers have used ANN with multiple layers, each of which can contain one or more neurons.

  Similar to SVM, performance of ANNs depends on parameters like number of neurons, number of layers, learning rate, activation and solver function. Learning rate is a hyper-parameter that controls how much we are adjusting the weights of our network with respect the loss gradient**.** Activation functions are used to convert an input signal of a node to an output signal. They introduce non linear relationships in the network. The output of a Hidden layer is sent as input to the next layer and so it is not visible as the output of the network. Solver functions are used to optimize the loss function to learn the correct parameters of the model. The range of these parameters used for our project is given below:

  - Number of hidden layers (h) = 1 - 3
  - Number of neurons each layer = 100 - 200
  - Learning rate = 0.001 - 0.05
  - Activation Function = identity, logistic, tanh and relu
  - Solver Function = lbfgs, sgd and adam

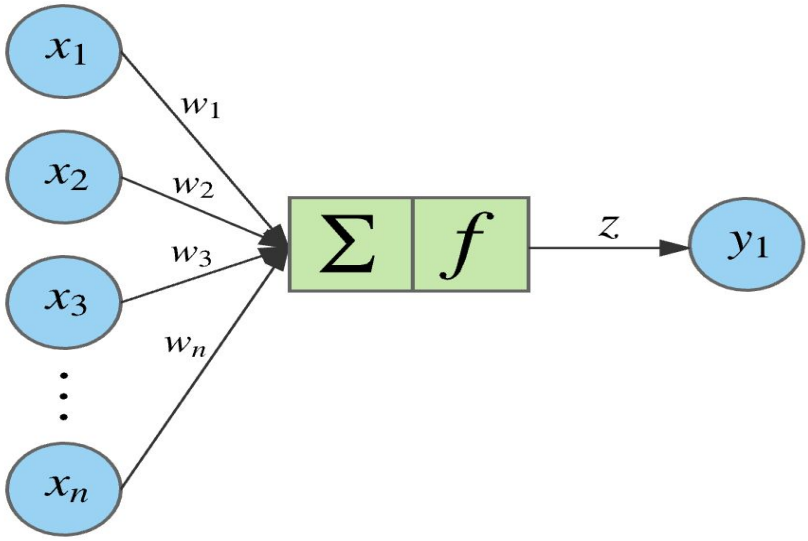

Figure 2: Illustration of a simple Neural Network (NN) with input layer followed by a single perceptron and a output layer. The input layer depends on the feature size of the dataset being used and output layer is usually as big as distinct number of classes in the dataset. The perceptron can be expanded to multiple number of hidden nodes and layers, making the network more dense and complex.
Reference:https://towardsdatascience.com/applied-deep-learning-part-1-artificial-neural-networks-d7834f67a4f6

- **Clustering**

  Clustering is a unsupervised machine learning algorithm.It is the process of grouping similar data points together into clusters such that all points in one cluster are more similar to each other and dissimilar to the points in another cluster as shown in figure 3. Some of the clustering methods are hierarchical clustering, density based clustering, fuzzy clustering, model based clustering and partitioning methods.

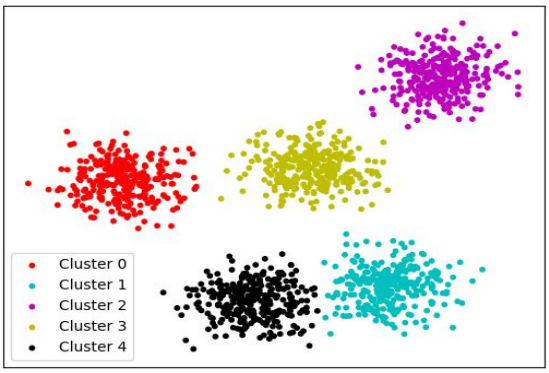

Figure 3: Illustration of how a Clustering Algorithm groups data points that similar to each other into the one cluster. All the points belonging to the same cluster are marked using the same color.
Reference:https://www.imperva.com/blog/clustering-and-dimensionality-reduction-understanding-the-magic-behind-machine-learning/

(4) Datasets Used:

The first dataset we used is the Digits dataset present in sklearn.datasets. This dataset contains images of the digits belonging to 10 classes (that is, digits from 0-9). Each class has approximately 180 samples or patterns. So, the total number of training points is 1,797.
The size of each digit image is 8 x 8 and so the dimension of each training point is 64.

The other dataset we used is the Iris Dataset which contains images of the Iris that belong to two classes - Real and Presentation Attack (PA). Each image is of size 256 x 256 and so the dimension of each training point is 4096. There are 3200 training points in total for this dataset.

(5) Image Pre-Processing and Feature Extraction

The proposed method is based on two important parts: Machine Learning and Distributed Systems. In Machine Learning, we first pre-processed the data and extract features (if required), and then feed the processed data into a classifier for training and testing (see Figure 4). No preprocessing is done in Dataset-I and the image of size 8x8 are used for classifier training. On the other hand, all the images in the iris dataset are center aligned using pupil center coordinates and center-cropped to 256x256 for uniformity in the dataset. After preprocessing, Binary Statistical Image Feature (BSIF) is used to extract textural information for iris images that helps classifier better understand the difference between the classes.

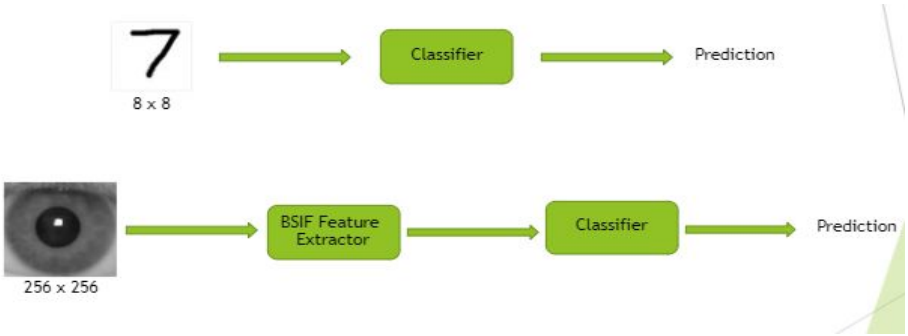

Figure 4: Illustration of data preprocessing and feature extraction for classifier training and testing

(6) Proposed Method

In this project, we implemented a distributed system for ML classifiers over numerous combinations of parameters to obtain the best accuracy. To achieve this, we implemented a reliable communication between multiple servers using Python socket

libraries. Our aim is to divide the task (i.e., different parameter combination) for each classifier on different servers to expand the resources available to train the classifiers more efficiently. User, running the script, has the option of selecting the dataset, classifier of his/her choice and the range for each parameters. If user doesn't give these values then the classifier assigns default parameter range to train and test on the dataset. The pseudo-algorithm for this distributed learning is given below:

Algorithm: Distributed ML Classifier

Input: Classifier(C), Parameters Range, Number of systems (N) and Dataset (D)
Output: Best Parameter Setting and Accuracy

---

(1) Get list of all the combinations of parameters' values (L)
(2) Divide parameter list to send to servers:
    Number of parameters' combination per server (len) = length(L)/N
    Let P be the list containing parameter list for each system
    Here P[0] refers to parameter list for system 1
    And P[N-1] refers to the parameter list for current system
(3) Connect to N-1 Systems using Python Socket libraries
(4) Train and Test C on current system with P[N-1] and dataset D
(5) For n in range(0,N-1):
    send(P[n]) to System n+1
    Wait for acknowledgement from each servers
    Each server Train and Test C on with P[n] and dataset D
(6) Receive results from other servers
    (a) Wait for assigned time to receive results from each system
    (b) If results are not received from a system within the time constraint, then throw an exception to the user and ask if he/she wants to run code again for that system only
    (c) If yes, then repeat steps (5)-(6) for that system. Else, go to step (7)
(7) Close all the connections
(8) Compare the results of each system and return best accuracy and parameter list
(9) Return Total time taken to complete this process (for analysis)

---

(7) System Configurations

In this project, we used seven different systems for three different distributed settings (as described in Results section). Configuration for these systems are given below:

- Two MAC systems with 2.7 GHz Intel Core i7 processor and 16GB RAM
- Three Windows systems with 2.7 GHz Intel Core i5 processor and 8GB RAM
- Two Windows systems with 2.7 GHz Intel Core i5 processor and 16GB RAM

For Single Systems, all the codes were run on MAC system with 2.7 GHz Intel Core i7 processor and 16GB RAM.

(8) Results and Analysis

In this section, we compare the performance of proposed algorithm when classifiers are trained using three, five and seven distributed systems. This is them compared with the performance of single system. The metric used for analyzing the efficacy of proposed method is speed-up from single system, which is calculated using as follows,

$$\text{Speed-Up (S)} = \frac{Time\ Taken\ by\ One\ System\ (T[1])}{Time\ Taken\ by\ N\ Systems\ (T[N])}$$

- **Performance on SVMs**

	Single System		Distributed System (Three Systems)		Distributed System (Five Systems)		Distributed System (Seven Systems)	
	Time Taken (in secs)	Accuracy (in %)	Time Taken (in secs)	Accuracy (in %)	Time Taken (in secs)	Accuracy (in %)	Time Taken (in secs)	Accuracy (in %)
Dataset-I	367.36	99.16	133.42	99.16	109.67	99.16	87.34	99.16
Dataset-II	13870.40	71.67	9358.93	71.67	7798.11	71.67	4679.46	71.67

Speed-Up

Speed-up	Single System	Three Systems	Five Systems	Seven Systems
Dataset-I	1	2.75	3.35	4.21
Dataset-II	1	1.48	1.77	2.96

- **Performance on ANNs**

	Single System		Distributed System (Three Systems)		Distributed System (Five Systems)		Distributed System (Seven Systems)	
	Time Taken (in secs)	Accuracy (in %)	Time Taken (in secs)	Accuracy (in %)	Time Taken (in secs)	Accuracy (in %)	Time Taken (in secs)	Accuracy (in %)
Dataset-I	1536.91	98.61	983.35	98.61	801.97	98.61	592.99	98.61
Dataset-II	7038.94	79	4658.23	79	3120.74	79	1985.02	79

Speed-Up

Speed-up	Single System	Three Systems	Five Systems	Seven Systems
Dataset-I	1	1.56	1.92	2.59
Dataset-II	1	1.51	2.25	3.55

- **Performance on Clustering**

	Single System		Distributed System (Three Systems)		Distributed System (Five Systems)		Distributed System (Seven Systems)	
	Time Taken (in secs)	Accuracy (in %)	Time Taken (in secs)	Accuracy (in %)	Time Taken (in secs)	Accuracy (in %)	Time Taken (in secs)	Accuracy (in %)
Dataset-I	2.11	98.89	1.33	98.89	1.47	98.89	1.34	98.89
Dataset-II	151.84	68.83	95.47	68.83	81.59	68.83	59.45	68.83

Speed-Up

Speed-up	Single System	Three Systems	Five Systems	Seven Systems
Dataset-I	1	1.59	1.44	1.57
Dataset-II	1	1.59	1.86	2.21

- **Analysis**
  As seen from the tables given above, distributed systems give the machine learning algorithms more resources to train and test the data. Hence, the time taken to complete the training process decreases as the number of distributed systems increases. Although, when a classifier like K-clustering is used the speed-up is not as significant as desired for small datasets. This could be because clustering is already fast-enough for a small dataset and adding distributed systems just adds more overhead with the connections and acknowledgement check. Although, significant speed-up can still be seen on big datasets and dense classifiers.
  We also analyzed that as the number of features increase from Dataset-I to Dataset-II, the time taken by the classifier to learn a decision boundary increases. This is because as the number of features increase, classifiers have more information to learn from.This helps to learn better decision boundaries, but it also increases learning times.

  Hence, from the above results we can conclude that distributing the training of a classifier over multiple systems helps speed-up the process, especially if the parameters to be learned and feature size is big.