Distributing Active Learning Algorithms

Syed Mostofa Monsur
mostofamonsur9396@gmail.com
Bangladesh University of Engineering and Technology
Dhaka, Bangladesh

Muhammad Abdullah Adnan
abdullah.adnan@gmail.com
Bangladesh University of Engineering and Technology
Dhaka, Bangladesh

ABSTRACT

Active Learning is a machine learning strategy that aims at finding out an optimal labeling sequence for a huge pool of unlabeled data. We observe that sometimes there are not enough labeled data in contrast to unlabeled samples. Moreover, in some scenarios labeling requires a lot of time and expert supervision. In those cases, we need to chalk out an optimal labeling order so that with a relatively small amount of labeled samples the model gives fairly good accuracy levels. This problem becomes more serious when dealing with distributed data that needs a distributed processing framework. In this work, we propose distributed implementations of state of the art active learning algorithms and perform various analyses on them. The algorithms are tested with real datasets on multinode spark clusters with data distributed on a distributed file system (HDFS). We show that our algorithms perform better than randomly labeling data i.e. non-active learning scenarios and show their mutual performance comparisons. The code is publicly available at *https://github.com/dv66/Distributed-Active-Learning*

CCS CONCEPTS

• **Computing methodologies** → **Machine learning**; Distributed algorithms; • **Theory of computation** → *Active learning*.

KEYWORDS

machine learning, active learning, distributed systems, big data

ACM Reference Format:
Syed Mostofa Monsur and Muhammad Abdullah Adnan. 2020. Distributing Active Learning Algorithms. In *7th International Conference on Networking, Systems and Security (7th NSysS 2020), December 22–24, 2020, Dhaka, Bangladesh.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3428363.3428368

1 INTRODUCTION

Active Learning is a machine learning strategy where the learning algorithm can query on the data instances to find out which of the unlabeled data points should be labeled by an annotator[9]. Given a set of labeled and unlabeled data instances, active learning aims at finding an optimal labeling order of the data so that good accuracy can be achieved with a relatively small set of labeled samples.

7th NSysS 2020, December 22–24, 2020, Dhaka, Bangladesh

ACM ISBN 978-1-4503-8905-1/20/12…$15.00
https://doi.org/10.1145/3428363.3428368

Sometimes labels of data instances are easy to obtain – practically they come at no costs. For instance, in detecting spams, rating songs or movies, the labels corresponding to the samples are obtained very easily. Thus we do not have to think much about the annotation process. But there may appear some situations when the task of data annotation may not be trivial. In many supervised machine learning tasks, labeling instances can be costly, time consuming and may require expert supervision. For example, annotating medical imaging data must need supervision of an expert of the domain. In speech recognition, accurate labeling of speech utterances is time consuming and requires expert linguists – thus costly. So in these tasks, minimizing the number of labeled instances is beneficial[14]. In normal supervised learning scenarios, the training set is selected for the learning procedure by a random sampling of the labeled instances. In active learning, the learner can actively choose the training data so that it reduces learner's need for large quantities of labeled data.

Currently a huge amount of data are spread across the machines and data centers all over the world. Distributed systems are required to process all these data. Distributed data storage and processing tools like Apache-Hadoop[2], Apache-Spark[17] are very frequently used to handle this loads of data. Also distributed machine learning is on the rise. So in this work we proposed novel active learning strategies to be adaptable to data spread across machines i.e. in a distributed environment. We introduce two standard distributed active learning procedure that takes distributed data (mostly unlabeled) from a distributed storage and process it iteratively to obtain an optimal labeling order such that high accuracy model can be achieved with a relatively small amount of labeled samples.

- We propose distributed active learning as a distributed optimization problem. In this definition we can seamlessly adopt distributed learning strategies.
- We extend the non-distributed versions of state-of-the-art active learning algorithms : uncertainty sampling, density weighting which we modified for a distributed environment.
- Additionally we introduce an efficient proximity matrix construction algorithm which outperforms spark's distributed matrix API *columnSimilarities*.
- We analyze the distributed active learning procedures in different metrics (accuracy gain per iteration, shuffleBytesRead, shuffleBytesWritten, running time analysis)
- We measure the efficiency of construct-proximity-matrix algorithm with its counterpart columnsimilarities and show that while columnsimilarities work well only with sparse data, our method works very efficiently across various sizes of datasets.

We provide pyspark implementation of two distributed algorithms **dist-US**(distributed uncertainty sampling) and **dist-DW**

(distributed density weighting) which is tested with both real and synthetic datasets and shows good performance.On average, **dist-DW** algorithm takes 200s less than the **dist-US** algorithm when running on our datasets. **dist-US** reads 25% less data than **dist-DW** on average. Again, in case of shuffle write operation, **dist-DW** writes 40% less data than **dist-US** on average. Our **construct-proximity-matrix** algorithm performs better than Spark API **columnSimilarity** for our datasets.

2 RELATED WORKS

Active learning has been extensively studied by researchers from the last decade. These researches has led us to various active learning strategies. Some state of the art active learning strategies include: uncertainty sampling[6], query-by-committee[11], density weighting[10], error[8], variance maximization[3] etc.

Uncertainty sampling is undoubtedly one of the most elegant and popular active strategies. Given a trained hypothesis, what uncertainty sampling does is, it tries to select those unlabeled samples which we are most uncertain about. Variants of uncertainty sampling has been applied extensivly in solving interesting tasks like multiclass active learning, image classification etc. Query by committee is an ensemble like algorithm which tries to reduce variances by taking votes from the classfiers in the ensembles. Error maximization techniques try to select those samples adding which in the labeled set will incur the most reduction of generalized error overall. However these techniques fail to capture the structure of the data thus may perform suboptimally in various context. Density weighting based approaches take structure of the data into consideration. Unlike uncertainty sampling which only looks at samples with most information content, density-weighting consider the representativeness of data samples which is generated from a distribution[10]. Thus it successfully sorts out the outliers which contains more information but is not a representative of a data set.

In recent times, meta learning heuristics and applications of active learning are studied most. Generative Adversarial Active Learning is a technique that adopts the game theoretic approach while applying active learning[18]. Learning active learning[5] is a error maximization based active learning technique in which the query procedure of active learning is treated as a regression problem. Given a trained regressor, the query procedure can predict the maximum error reduction that will be caused by adding an unlabeled sample to the labeled set. Their work performs well on various scenarios involving real and synthetic datasets.

However, none of these work has focused on the context of distributed environment in which the data is distributed and the learning process follows a distributed computing framework. Where as our work considers the potential of extending active learning to a distributed learning platform.

3 MOTIVATION

In this era of big data and cloud based systems, data is flooding through the internet. Big organizations have been moving to big data infrastructure since the beginning of this decade. These organizations has data spread all over the world across data centers. Video streaming services, social networks deal with huge amount of data. Most of these data are unlabeled. Active learning is highly useful in these contexts:

- *Speech Recognition.* Speech recognition needs expert to be labeled accurately thus time consuming.
- *Video Annotation.* Millions of terabytes of video data are spread across the data centers which is an impossible task to annotate[19]
- *Computational Biology.* In predicting protein structures, genome sequencing, labeling is a real difficult task because it induces billions of different possibilites.[13]
- *Medical Imaging.* Biomedical imaging data is mostly labled by experienced experts because they are sensitive.[5]
- *High Energy Physics.* Active learning has shown great performances with high energy physics data.[5]

The problem with these data is that they are mostly distributed thus they need a distributed processing mechanism to handle them. An the same time, it would be a tough task to label all the data. So we need to meet the both ends here, dealing with large amount of distributed data and apply active learning.

This motivates us to harness the power of distributed processing to solve active learning problem.

4 PROBLEM FORMULATION

Before describing Distributed Active Learning, let us revisit the fundamentals active learning framework first.

Say, we have a huge pool of unlabeled data $\mathcal{U}$. The goal of active learning would be to select samples in a order so that the model can be learned quickly with a small amount of data. The main issue here is that the labeling task is hard to do. As we cannot label all the data. It is logical to selectively and iteratively sample datapoints from the huge unlabeled pool that just randomly sampling them.

Let, we have a dataset $\mathcal{D} = \{(x_1, y_1), \ldots, (x_N, y_N)\}$ with N samples. x_i are k-dimensional feature vectors and y_i will be their corresponding labels where $y_i \in \{1, 0\}$ for $i = 1, \ldots, N$. Note that some labels are not known, that is they are unlabeled samples. The set of labled samples and unlabled samples are denoted by $\mathcal{L}$ and $\mathcal{U}$ respectively. We have a classifier model f which predicts the labels given the sample feature as $f(x_i) = \hat{y}_i$. In this scenario, a general active learning framework goes like:

(1) **Initialize:** Start with $\mathcal{L} \subset \mathcal{D}$ and unannotated data $\mathcal{U} = \mathcal{D} \backslash \mathcal{L}$ with an initial model performance metric
(2) Train f with current $\mathcal{L}$
(3) Query procedure chooses $x^* \in \mathcal{U}$ to be labeled and added to the labeled set $\mathcal{L}$.
(4) x^* is given a label y^* by an expert annotator.
(5) $\mathcal{L} = \mathcal{L} \cup \{(x^*, y^*)\}$
(6) 2 to 5 steps are repeated until the model achieves a predefined threshold.

Now lets move on to the formulation of distributed active learning problem. Given a distributed system $C = \{c_1, \ldots, c_k\}$ where d_is are the nodes of the cluster, the data $\mathcal{D} = \mathcal{U} \cup \mathcal{L}$, is distributed across the cluster nodes c_i. Let each of the nodes c_i has a data chunk $\mathcal{D}_i$ in it. So $\mathcal{D}_1 \cup \mathcal{D}_2 \cup \ldots \cup \mathcal{D}_k = \mathcal{D}$ A distributed classifier $\mathcal{F}$ is trained using the $\mathcal{D}_i$s.

The query selection procedure picks a block of instances $\mathcal{B} = \{x_m^*, \ldots, x_n^*\} \subset \mathcal{U}$ for annotation. Then $\{x_m^*, \ldots, x_n^*\}$ are given labels $\{y_m^*, \ldots, y_n^*\}$.

Labeled and unlabeled sets are updated as $\mathcal{U} = \mathcal{U} - \mathcal{B}$ and $\mathcal{L} = \mathcal{L} \cup \mathcal{B}$

The process is iterated until a predefined accuracy value is achieved.

We propose two Distributed Active Learning scenarios:

4.1 Distributed Uncertainty Sampling

Uncertainty sampling suggests to label those samples which we are most uncertain about.

$$x^* = \arg\max_{x \in \mathcal{U}} \mathcal{H}(\hat{y} = y|x) \tag{1}$$

where $\mathcal{H}$ is the uncertainty function.

In a distributed setting, this equation can be reformulated as:

$$\mathcal{X}^* = \arg \max_{\{x_m^*, \ldots x_n^*\} \subset \mathcal{U}_t} \sum_{x \in \{x_m^*, \ldots x_n^*\}} \mathcal{H}(\hat{y} = y \mid x) \tag{2}$$

That is we are no longer selecting one sample but a block of samples $\mathcal{X}^* = \{x_m^*, \ldots x_n^*\}$ so that among all possible permutations this block incurs the greatest total entropy.

4.2 Distributed Density Weighting

The classical density weighting based active learning is defined using the equation:

$$x^* = \arg\max_x \phi_A(x) \times \left(\frac{1}{U} \sum_{x' \in \mathcal{U}} sim(x, x') \right)^\beta \tag{3}$$

Where $\phi_A(.)$ represents the heuristic value of x w.r.t a "base" strategy A like uncertainty sampling, The second term denotes the representativeness of x similarity value to all other instances of the same data source. β is a hyperparameter. Now, in a distributed active learning environment this equation will be changed to this:

$$\mathcal{X}_{ID}^* = \arg \max_{\{x_m^*, \ldots x_n^*\} \subset \mathcal{U}_t} \sum_{x \in \{x_m^*, \ldots x_n^*\}} \phi_A(x) \times \left(\frac{1}{U} \sum_{x' \in \mathcal{U}} S(x, x') \right)^\beta \tag{4}$$

That is we want to find a block of instances who incur a maximum total information density heuristic to a distributed active learning strategy. Here S is a distributed matrix storing the proximity values of the data points.

In both cases, our distributed active learning algorithms output a block of instances with a predefined window size to the oracle/labeling agent to label those samples instead of labeling just one sample per iteration.

5 HDFS AND RDD

HDFS[12] (Hadoop Distributed File System) and RDD[16] (Resilient Distributed Datasets) are key components in the distributed learning scenario that our work introduces. Here is a brief overview of these two:

- *HDFS.* The Hadoop Distributed File System (HDFS) is the primary data storage system used by Hadoop applications. It employs a NameNode and DataNode architecture to implement a distributed file system that provides high-performance access to data across highly scalable Hadoop clusters. HDFS takes in data, breaks the information down into separate blocks and distributes them to different nodes in a cluster, thus enabling highly efficient parallel processing.
- *RDD.* A fault tolerant, distributed data structure that is spread across machines in a Spark in-memory processing environment. RDDs are immutable, partitioned and lazily evaluated. RDDs only allow two kinds of operations: *transformations* and *actions*. A transformation when applied on an RDD, transforms it to another modified RDD. Whereas and action will reduce the RDD to some kind of value.

6 OUR CONTRIBUTION

In this section we present the main contributions of our work. We present three distributed algorithms for distributed active learning.

6.1 Distributed Uncertainty Sampling Algorithm

As mentioned earlier, the distributed uncertainty sampling algorithms uses a distributed file storage (HDFS). We need to specify the indices of data points because in spark-RDD there is no concept of indexing. So our data vectors are distributed across machines. In order to identify each of the sample vectors, we need to zip indices to them. Then at each iteration we use a distributed machine learning model (e.g. Random forest) to predict the class probabilities of each sample. From the class inferences, it is possible to compute the uncertainty in the data. The indices of the samples with the highest amount of uncertainty are chosen for labeling at the end of the iteration.

6.2 Proximity Matrix Construction

In density weighting, we have to consider the similarities among data points. So we have to calculate a proximity matrix here. If dataset becomes larger, the proximity matrix will be huge. We can use the default **columnsimilarities** algorithm[15] provided in spark distributed matrix API. But that algorithm has really poor performance for constructing the proximity matrix in our specific scenario – where we need a proximity matrix in the form of a spark-coordinate-matrix for better filtering and querying proximity values. So we modified the algorithm to minimize data transfer and we propose our create-proximity-matrix algorithm. For example, lets consider a data matrix D. We take each entry vector of D and normalize them. The normalized matrix is U. We compute U^T, the transpose of U. We multiply U and U^T to get the similarity matrix S.

$$\mathcal{U}_i = \frac{\mathcal{D}_i}{|\mathcal{D}_i|} \tag{5}$$

$$\mathcal{S} = U * U^T \tag{6}$$

$$S_{ij} = \frac{\hat{u}_i \hat{u}_j}{|\hat{u}_i||\hat{u}_i|} \tag{7}$$

6.3 Distributed Density Weighting Algorithm

We extend the density weighting scheme of active learning to a distributed context. First, we create the proximity matrix using

Algorithm 1 dist-US

input: distributed file of vectors(F), window size (w), number of estimators in the model (n)
output: an optimal labeling order of unlabeled samples using uncertainty sampling scheme in a distributing setting
F_{RDD} := `makeRDD(`F`)`
D_{train}, D_{test} := `split(`F_{RDD}`)`
$D := D_{train}$`.zipWithIndex()`
$L := D$`.takeSamples(`w`)`
$U := D \setminus L$
while $U \neq \emptyset$ **do**
  M := `createDistributedModel(`L,n`)`
  E := `emptyRDD()`
  for m in M **do**
    $P := m$`.predict(`U`)`
    $E := E \cup P$
  end for
  H := `computeUncertainty(`E`)`
  $H_U := U$`.leftOuterJoin(`H, on=index`)`
  S := `sortDescending(`H_U`)`
  R := `makeRDD(`S`.takeSamples(`w`))`
  $U := U \setminus R$
  send R (for labeling)
  $L := L \cup R$
end while

Algorithm 2 construct-proximity-matrix

input: distributed file of vectors(F)
output: proximity matrix (SC) as spark coordinate matrix
F_{RDD} := `makeRDD(`F`)`
$X := F_{RDD}$`.map`$\left(\hat{x} \rightarrow \frac{\hat{x}}{|\hat{x}|}\right)$
$D := X$`.zipWithIndex().map(`$\hat{x} \rightarrow$ `indexedRow(`$\hat{x}$`))`
I := `makeIndexedRowMatrix(`D`)`
$C := I$`. makeCoordinateMatrix()`
$C^T := C$`. transpose()`
$U := I$`. makeBlockMatrix()`
$U^T := C^T$`. makeBlockMatrix()`
$S := U \times U^T$
$SC := S$`. makeCoordinateMatrix()`
out SC

our **construct-proximity-matrix** algorithm. We zip indices to the data vectors. Then in each iteration we find out the most uncertain samples (we are considering uncertainty sampling as our base strategy). We then compute the information density heuristic for all the data with the information from our previously constructed proximity matrix. We multiply the entropy values and heuristic values. The samples that maximizes the joint heuristic value are the ones to be chosen. In this way, the density scores help to reduce the probability of choosing noisy data thus increase efficiency in labeling efforts. We take a block of maximal heuristic samples and add them to the 'to be labeled' set for labeling.

Algorithm 3 dist-DW

input: distributed file of vectors(F), window size (w), number of estimators in the model (n)
output: an optimal labeling order of unlabeled samples using density weighting (DW) in a distributing setting
F_{RDD} := `makeRDD(`F`)`
D_{train}, D_{test} := `split(`F_{RDD}`)`
$D := D_{train}$`.zipWithIndex()`
$L := D$`.takeSamples(`w`)`
$U := D \setminus L$
SC := **construct-proximity-matrix**(F)`. makeRDD()`
T := `emptyRDD()`
for l in L **do**
  l_{index} := l`. getIndex()`
  $SC_{labeled} := SC$`.filter(`i=l_{index} **or** j=l_{index}`)`
  $T := T \cup SC_{labeled}$
end for
$SC := SC \setminus T$
while $U \neq \emptyset$ **do**
  M := `createDistributedModel(`L,n`)`
  E := `emptyRDD()`
  for m in M **do**
    $P := m$`.predict(`U`)`
    $E := E \cup P$
  end for
  H := `uncertaintyFromBaseStrategy(`E`)`
  $H_{SC} := SC$`.leftOuterJoin(`H, on=index`)`
  $H_U := H_{SC}$`.computeDensityScore()`
  S := `sortDescending(`H_U`)`
  R := `makeRDD(`S`.takeSamples(`w`))`
  $U := U \setminus R$
  send R (for labeling)
  $L := L \cup R$
end while

7 EXPERIMENTAL RESULTS

In this section we will present the experiments run on our algorithms which are tested on both real and synthetic datasets.

7.1 Experimental Setup

For our distributed active learning experiments, we have used a six node Apache-Spark cluster. Each computing node has the following configurations:

- 16G RAM
- 160G Disk Space
- 8 core CPU
- Python 3.4, Apache-Hadoop, Apache-Spark, Lynx installed

Besides we used Apache-Hadoop for the distributed file storage HDFS. All of the data were distributed in HDFS and used by the spark kernel. We used the Random forest[1] implementation by spark MLLib[7] as our distributed machine learning model. We took **number of trees in random forest = 20** as the hyperparameter in our experiments.

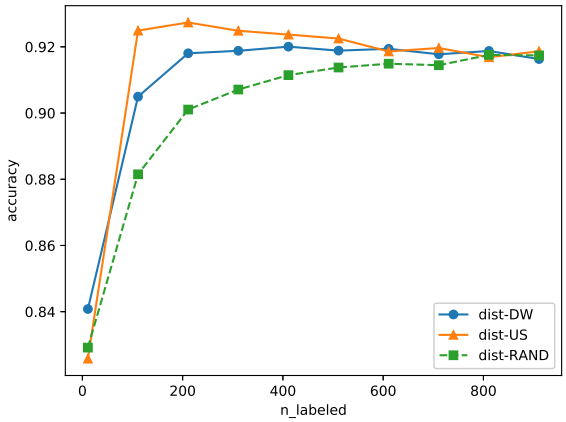

Figure 1: Accuracy gain per iteration for Striatum dataset

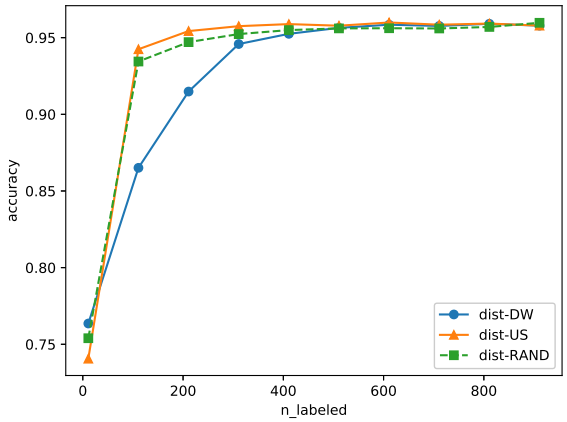

Figure 2: Accuracy gain per iteration for Quick draw dataset

7.2 Datasets

We used both real and synthetic datasets for the performance evaluation of our distributed algorithms. The datasets are :

- *Google QuickDraw Dataset.* The Quick Draw Dataset[4] is a collection of 50 million drawings across 345 categories, contributed by players of the game Quick, Draw!. The drawings were captured as timestamped vectors, tagged with metadata including what the player was asked to draw and in which country the player was located.
- *Striatum Dataset.* This dataset consists of 3D Electron Microscopy stack of rat neural tissue, the task is to detect and segment mitochondria[5].
- *XOR Dataset.* This is a synthetic dataset. The data points are scattered in a checkerboard pattern (xor data).

7.3 Accuracy Gain Per Iteration

Among the three algorithms (**dist-Rand** stands of random sampling i.e. non active learning) **dist-US** has a overall better performance than that of the remaining two. For Striatum dataset, the **dist-DW** algorithm performs better than the others (Fig 1). In the quick draw dataset, **dist-DW** shows poor performance which is even worse than random sampling (Fig 2). Whereas, we observe a higher accuracy gain for **dist-DW** in case of xor dataset (Fig 3). The reason for this variation in case of Quick Draw and XOR is that, we manipulated the data in XOR dataset and introduced skewness artificially. The Density Weighting algorithm finds outliers in data better than the uncertainty sampling based strategy. The density information along with the uncertainty metric provides insight about the data i.e. if the data is actually representative of the whole set. So the modified heuristic for **dist-DW** works better in cases when noisy data or outliers are present in the dataset. We see a performance improvement for **dist-DW** in Fig 3.

7.4 Running time

Algorithm **dist-DW** has a better running time performance than the **dist-US** algorithm. In all three datasets **dist-DW** is faster than

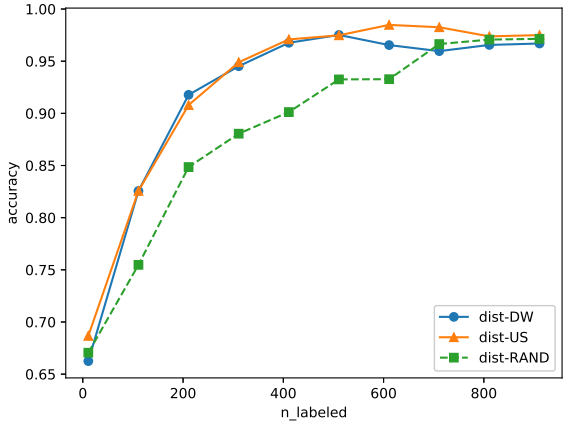

Figure 3: Accuracy gain per iteration for XOR dataset

dist-US. On average, **dist-DW** algorithm takes 200s less than the uncertainty based algorithm **dist-US** when running on the three previously mentioned datasets (Fig 4).

7.5 Accuracy vs Running time

Although **dist-US** runs slower than the **dist-DW** algorithm in the given setting for our three test datasets, it is observed that **dist-US** achieves good accuracy levels with less number of iteration than **dist-DW**. That is with relatively small amount of data **dist-US** shows better results than **dist-DW** but takes a little more time. As we saw in the previous section **dist-DW** is preforming faster than **dist-US** algorithm. Here in Accuracy vs Running time analysis, we see that **dist-US** achieves good accuracy levels faster than **dist-DW** in *Striatum* (Fig 5) and *QuickDraw* (Fig 6) datasets. But it gets slower in the synthetic **XOR** (Fig 7)dataset.

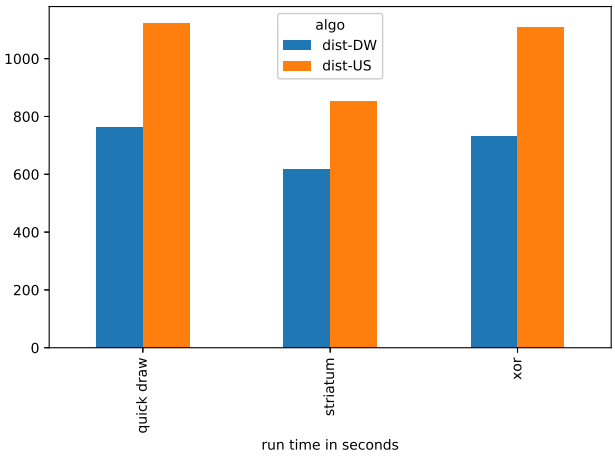

Figure 4: Running time for various datasets

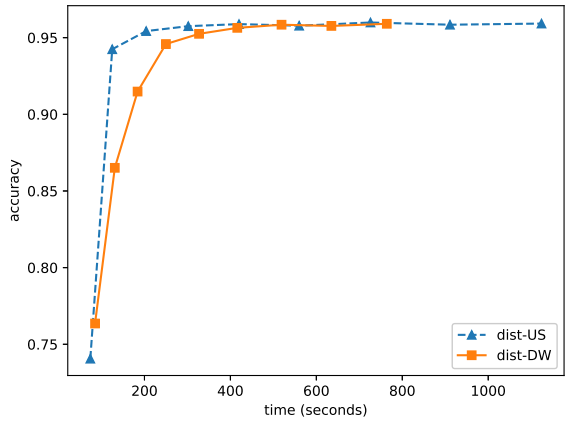

Figure 6: Accuracy vs running time for Quick Draw Data

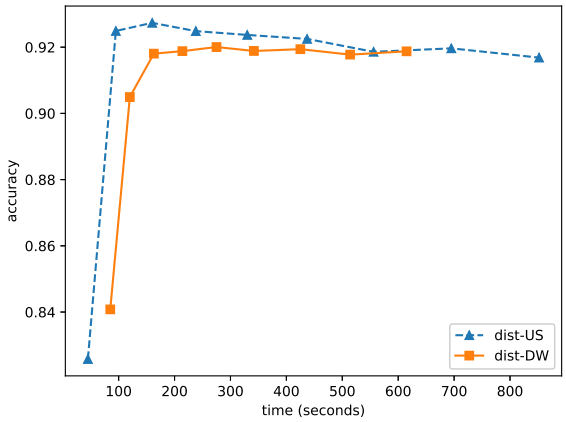

Figure 5: Accuracy vs running time for Striatum Data

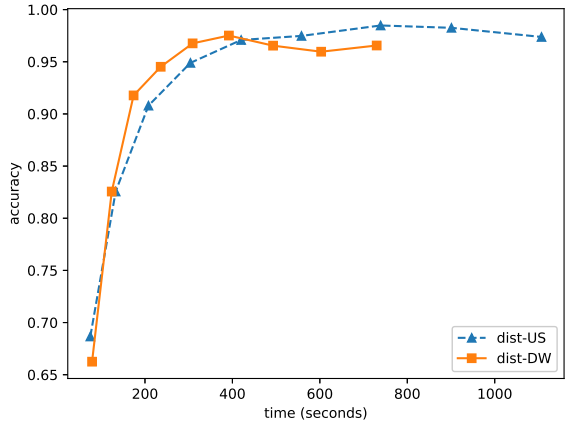

Figure 7: Accuracy vs running time for XOR Data

7.6 Shuffle I/O

We observe **dist-DW** has a higher shuffle I/O than the other two algorithms. In **dist-DW**, the **construct-proximity-matrix** algorithm is used to compute mutual proximity metrics among the data vectors. For this, a large proximity matrix is constructed. A huge number of shuffle operations is needed when data is transferred among the clusters. That is why **dist-DW** has higher shuffle I/O rate than **dist-US**. In shuffle read operation, **dist-US** reads 25% less data than **dist-DW** on average (Fig 8). Again, in case of shuffle write operation, **dist-DW** writes 40% less data than **dist-US** on average (Fig 9)).

7.7 Columnsimilarity Vs Construct-Proximity-Matrix Algorithm

The *Apache-Spark* API **columnSimilarity** is sub optimal than Construct-Proximity-Matrix Algorithm. ColumnSimilarity algorithm is a probabilistic one that works well with sparse data matrices and mostly used with user rating matrices which are reasonably sparse enough for this algorithm. But this does not guarantee high performance with all sorts of data. The **construct-proximity-matrix** algorithm is free from these limitations. In **dist-DW** algorithm we need a proximity matrix which is constructed using **construct-proximity-matrix** algorithm. In table 1 and 2 the running times of **column-similarity** and **construct-proximity-matrix** for Striatum and Quick Draw datasets are shown. From the statistical data we clearly see that **construct-proximity-matrix** outperforms **column-similarity** for our datasets.

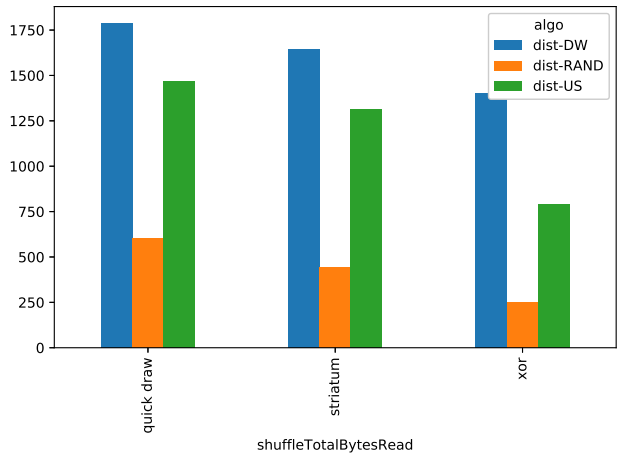

Figure 8: Amount of bytes read (MB) in shuffle for algorithms

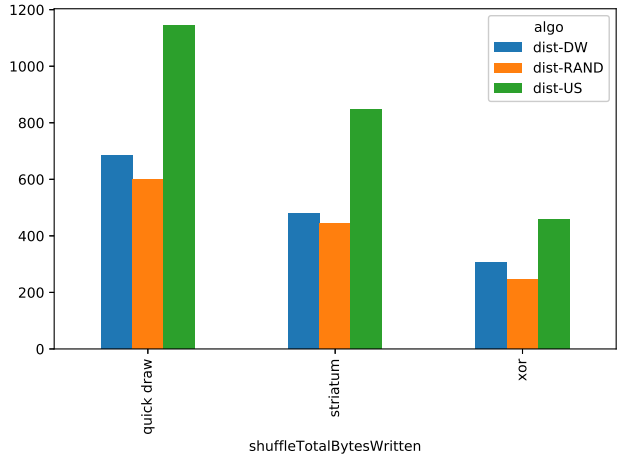

Figure 9: Amount of bytes written (MB) in shuffle for algorithms

Table 1: Running times for Striatum Dataset

Size	Columnsimilarity	Create-Proximity
1000	22s	12s
2000	62s	13s
3000	148s	14s
4000	297s	15s
5000	out-of-memory	17s

8 CONCLUSION AND FUTURE WORKS

In this work, we have dealt with the distributed active learning problem. We formulated the distributed active learning as a distributed extension of pre existing state-of-the-art active learning algorithms. We presented two standard distributed active learning algorithms **dist-US** and **dist-DW** which are extended from their non-distributed counterparts. We also presented an efficient proximity matrix construction procedure **Construct-Proximity-Matrix** Algorithm which performs better than columnsimilarities algorithm. In future we have plans to work on active reinforcement learning in distributed contexts.

Table 2: Running times for Quick Draw Dataset

Size	Columnsimilarity	Create-Proximity
1000	15s	16s
2000	40s	17s
3000	96s	18s
4000	188s	22s
5000	out-of-memory	19s

REFERENCES

[1] Leo Breiman. 2001. Random Forests. *Mach. Learn.* 45, 1 (october 2001), 5–32. https://doi.org/10.1023/A:1010933404324

[2] Jeffrey Dean and Sanjay Ghemawat. 2008. MapReduce: Simplified Data Processing on Large Clusters. *Commun. ACM* 51, 1 (january 2008), 107–113. https://doi.org/10.1145/1327452.1327492

[3] Stuart Geman, Elie Bienenstock, and René Doursat. 1992. Neural Networks and the Bias/Variance Dilemma. *Neural Comput.* 4, 1 (january 1992), 1–58. https://doi.org/10.1162/neco.1992.4.1.1

[4] https://github.com/googlecreativelab/quickdraw dataset. [n. d.]. .

[5] Ksenia Konyushkova, Raphael Sznitman, and Pascal Fua. 2017. Learning Active Learning from Data. In *Advances in Neural Information Processing Systems 30*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (Eds.). Curran Associates, Inc., 4225–4235. http://papers.nips.cc/paper/7010-learning-active-learning-from-data.pdf

[6] David D. Lewis and Jason Catlett. 1994. Heterogeneous Uncertainty Sampling for Supervised Learning. In *In Proceedings of the Eleventh International Conference on Machine Learning*. Morgan Kaufmann, 148–156.

[7] Xiangrui Meng, Joseph K. Bradley, Burak Yavuz, Evan R. Sparks, Shivaram Venkataraman, Davies Liu, Jeremy Freeman, D. B. Tsai, Manish Amde, Sean Owen, Doris Xin, Reynold Xin, Michael J. Franklin, Reza Zadeh, Matei Zaharia, and Ameet Talwalkar. 2015. MLlib: Machine Learning in Apache Spark. *CoRR* abs/1505.06807 (2015). arXiv:1505.06807 http://arxiv.org/abs/1505.06807

[8] Nicholas Roy and Andrew McCallum. 2001. Toward Optimal Active Learning Through Sampling Estimation of Error Reduction. In *Proceedings of the Eighteenth International Conference on Machine Learning (ICML '01)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 441–448. http://dl.acm.org/citation.cfm?id=645530.655646

[9] Burr Settles. 2008. *Curious Machines: Active Learning with Structured Instances*. Ph.D. Dissertation. University of Wisconsin-Madison. Advisor(s) Mark Craven.

[10] Burr Settles. 2012. *Active Learning*. Morgan & Claypool Publishers.

[11] H. S. Seung, M. Opper, and H. Sompolinsky. 1992. Query by Committee. In *Proceedings of the Fifth Annual Workshop on Computational Learning Theory (COLT '92)*. ACM, New York, NY, USA, 287–294. https://doi.org/10.1145/130385.130417

[12] Konstantin Shvachko, Hairong Kuang, Sanjay Radia, and Robert Chansler. 2010. The Hadoop Distributed File System. In *Proceedings of the 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST) (MSST '10)*. IEEE Computer Society, Washington, DC, USA, 1–10. https://doi.org/10.1109/MSST.2010.5496972

[13] Brian C. Smith, Burr Settles, William C. Hallows, Mark W. Craven, and John M. Denu. 2011. SIRT3 Substrate Specificity Determined by Peptide Arrays and Machine Learning. *ACS Chemical Biology* 6, 2 (18 Feb 2011), 146–157. https://doi.org/10.1021/cb100218d

[14] Simon Tong and Daphne Koller. 2002. Support Vector Machine Active Learning with Applications to Text Classification. *J. Mach. Learn. Res.* 2 (march 2002), 45–66. https://doi.org/10.1162/153244302760185243

[15] Reza Bosagh Zadeh and Gunnar Carlsson. 2013. Dimension Independent Matrix Square using MapReduce. *CoRR* abs/1304.1467 (2013). arXiv:1304.1467 http://arxiv.org/abs/1304.1467

[16] Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauley, Michael J. Franklin, Scott Shenker, and Ion Stoica. 2012. Resilient Distributed Datasets: A Fault-tolerant Abstraction for In-memory Cluster

Computing. In *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation (NSDI'12)*. USENIX Association, Berkeley, CA, USA, 2–2. http://dl.acm.org/citation.cfm?id=2228298.2228301

[17] Matei Zaharia, Reynold S. Xin, Patrick Wendell, Tathagata Das, Michael Armbrust, Ankur Dave, Xiangrui Meng, Josh Rosen, Shivaram Venkataraman, Michael J. Franklin, Ali Ghodsi, Joseph Gonzalez, Scott Shenker, and Ion Stoica. 2016. Apache Spark: A Unified Engine for Big Data Processing. *Commun. ACM* 59, 11 (october 2016), 56–65. https://doi.org/10.1145/2934664

[18] Jia-Jie Zhu and José Bento. 2017. Generative Adversarial Active Learning. *CoRR* abs/1702.07956 (2017). http://arxiv.org/abs/1702.07956

[19] Xiaojin Zhu, Andrew B. Goldberg, Ronald Brachman, and Thomas Dietterich. 2009. *Introduction to Semi-Supervised Learning*. Morgan and Claypool Publishers.