Developing a Clustering-Based Semi-Supervised Learning Algorithm for Data Streams

Vaishnav Manapetty Ajith

***Abstract*— In today's world, we encounter an enormous volume of continuous data streams. The task of processing and deriving insights from this data can be overwhelming. To address this challenge, we propose the use of a machine learning algorithm, specifically, a clustering-based semi-supervised learning algorithm. This choice is motivated by the recognition that data from these streams is often imperfect, exhibiting issues such as partial labelling, missing values, and noise. Therefore, a clustering-based semi-supervised learning algorithm has been put forward as a solution, as it is capable of handling diverse and inconsistent data, including both labelled and unlabeled data types. After conducting thorough background research, one prominent algorithm has been identified as capable of effectively addressing the project's requirements: the cluster and label classifier. In this project, variations of the cluster and label method were employed to build the proposed algorithm. At the current stage, the proposed model utilizes the cluster and label method to assign a pseudo label to the data points without a label, and then an ensemble OzaBag classifier is built and trained on both labeled and pseudo-labeled data. And then OzaBag classifier is used to classify stream data. This model is evaluated using metrics such as accuracy and execution time. The model that is used has an average accuracy rate of 68% and an execution time of 100 seconds when processing 100,000 data instances.**

I. INTRODUCTION

In today's rapidly evolving world, many devices and sensors are being utilized in everyday activity, and they are generating enormous amounts of data in the form of data streams [1]. Due to the overwhelming amount of data, it is impractical for humans to manually process and derive meaningful insight from the data. That is why a machine learning algorithm that is capable of processing, interpreting, and making high-level predictions from data streams in real-time is essential. Examples of real-time processing and prediction from data streams are evident from various fields. For example, in finance, real-time analysis of stock market data can help investors make critical investment decisions.

However, data from data streams are not perfect. The data that is extracted from the world could be partially labelled, missing values, noisy etc. This is due to external factors such as network delay, expensive labelling processes, corrupted data, etc [2]. That is why we need a semi-supervised learning algorithm which can handle inconsistent types of data (mainly labelled and unlabeled types of data).

This project is about developing a basic clustering-inspired semi-supervised learning method for data streams. The aim is to analyze existing algorithms in the field of semi-supervised learning for data streams and propose a novel approach that is based on clustering. Which can leverage unlabeled and labelled types of data, when creating the model. Concept drift is also an important factor when developing a machine-learning algorithm. It is the change in the relationship between the input and target data as time progresses, and this occurs because data is constantly evolving [3]. Which causes the accuracy of the model to severely decline. This means the clustering-inspired semi-supervised learning method needs to be able to overcome the concept drift problem.

We can evaluate how effective the algorithm is using evaluation metrics such as accuracy. We can also evaluate how efficient the model is by checking the execution time of the model. The execution time refers to how fast the model takes to pre-process data, learn, and predict data, and it is essential that the execution time is at least a reasonable time. So, the aim is to build a model that has a high accuracy rate and has a reasonable execution time.

The proposed model utilizes the cluster and label method to assign a pseudo label to the data points without a label, and then an ensemble OzaBag classifier is built and trained on both labeled and pseudo-labeled data. And then OzaBag classifier is used to classify stream data. This model is evaluated using metrics such as accuracy and execution time. The model that is used has an average accuracy rate of 68% and an execution time of 100 seconds when processing 100,000 data instances. This meets the requirements.

The biggest environmental and sustainability issue this project faces is the amount of energy consumption that is being used to develop this machine learning algorithm. However, the energy consumption that is being used during the development process of the algorithm is going to be minimal because no high-energy consumption tools would be used. So, any environmental or sustainability impacts are going to be very minimal. The machine learning algorithm is most likely going to use a minimal amount of energy as well since this algorithm isn't going to be used on an industry level. It would still be good practice to be using energy-efficient tools and make the algorithm as efficient as possible.

This project was supervised by Heitor Gomes (primary).

The chosen development methodology for this project is the Agile methodology, where each sprint is a cycle of planning, executing, and evaluating. Sprints are time-boxed iterations during which a developer would work on delivering a set of product features or functionality. The beginning of the sprint is the planning phase. In the planning phase, a product backlog will be used. The items in the product backlog contain the requirements and features that the machine learning algorithm needs in the future. They are constantly updated throughout the development cycle. After that, a set of backlog items needs to be chosen to be worked on in the current sprint. Also, important to break the items down into small tasks.

After the planning phase, is the development phase where I work on implementing the tasks identified in the planning phase. This means that this is the phase where we design, code, implement, evaluate/test the machine learning algorithm on MOA.

After the development phase is the evaluation phase. At the end of the sprint, I conduct a review with the supervisor, collect feedback, and adjust the product backlog based on the feedback. After that continue to repeat the sprint until all the items in the product backlog are finished.

The Agile development lifecycle emphasizes iterative and incremental progress, promoting frequent delivery of working software and continual feedback and learning. Cycling through the planning, executing, and evaluating phases in each sprint, allows me to adapt to changes, improve my processes, and deliver value to stakeholders in an iterative manner.

While developing this project, there would be no specialized equipment/hardware needed for this project. The only hardware device required is a computer that can run the following software: IntelliJ, MOA, and Git. IntelliJ IDEA is an IDE (Integrated Development Environment) that acts as a software tool to help code, debug, and test the semi-supervised machine learning algorithm in the project. It also offers advanced refactoring, debugging, and code editing tools which can help streamline the development process for the machine learning algorithm for this project.

The software required for this project is MOA, IntelliJ, and Git. MOA (Massive Online Analysis) is an open-source software framework that is designed for data stream mining in real time. This is a critical piece of software as it allows users to build and run machine learning experiments on evolving data streams. It is also possible to implement the semi-supervised machine learning algorithms that are being developed in this project on MOA. The software also includes evaluation and visualization tools that enable users to monitor the performance of the learning algorithms, assess the quality of the models, and analyze the results which would be essential for this project. Because MOA is built on top of the Java Programming language, it would be necessary to use Java in this project. This means that a Java Development Kit would need to be installed on the computer, to be able to develop the machine learning algorithm.

Git needs to be installed on a computer for this project so it can interact with GitLab. GitLab is a software tool that provides many tools that would be useful in this project. It allows users to manage and host git repositories, version control the code used in this project, review code, and offer project management features such as milestones, issue tracking, etc. This would be very handy since the Agile methodology process uses those key features.

II. • Related Work

After conducting thorough background research, two prominent solutions or algorithms have been identified to be able to address the project's requirements effectively. The three algorithms are Cluster and Label, and SmSCluster.

The Cluster and Label approach [2] uses a clustering algorithm and a voting scheme to select the target label for each data instance. During the training process of the cluster and label algorithm, the clustering algorithm will consistently receive data instances and then group similar data instances together in an unsupervised manner, which results in a K number of clusters. In each cluster, it is expected that most of the data instances in the cluster would belong to the same class. The most common class label will become the representative class for that cluster.

So, when a new data instance joins a cluster, if it's unlabeled, the data will be assigned the representative class of the cluster as its target label. To keep track of the frequency of the labelled data in each cluster, the cluster has a data structure called label features. This allows the algorithm to track how many instances of each class are in the cluster, When the algorithm is trying to predict the label of a new data instance, the algorithm finds the closest cluster to the data point. The representative class of the closest cluster is issued as a prediction of the new data point.

The SmSCluster algorithm [4] is slightly different to the Cluster and Label approach. The SmSCluster algorithm receives the streaming data in chunks. Each chunk contains a group of instances, and the SmSCluster algorithm starts building a new model by applying a clustering algorithm to the group of instances to create a K number of clusters. The clustering algorithm uses an EM algorithm to produce clusters that minimize both intra-cluster dispersion and at the same time the impurity of each cluster regarding its labels. This means it can create compact and tight clusters, where the data instances within the clusters are like each other.

A summary of the statistics of the instances belonging to each cluster is saved as a micro cluster. The micro clusters for a particular chunk of data serve as a model to classify other data instances. When the algorithm wants to classify new data points, it uses the K-nearest neighbor algorithm, which identifies Q-nearest clusters. The most frequent label in these clusters will be the predicted label of the data instances.

To deal with the stream evolution, an ensemble of L-such models is used. So, when a new model is built from a new data chunk, we update the ensemble by choosing the best L models from the L + 1 models (previous L models plus the new model), based on each model's accuracies on the labelled training data inside the new data chunk [5].

The advantage that both algorithms have is that they do not need a fully labelled dataset for the algorithms to work. They are both able to work with a limited amount of labelled data, making them suitable for scenarios where acquiring labelled data is costly or time-consuming.

One of the disadvantages of the SmSCluster is that it can be sensitive to the initial parameter settings, such as the number of clusters or the Q value which is used to find Q-nearest clusters. Careful tuning and validation of these parameters are required to achieve optimal results.

Depending on the clustering algorithm that Cluster and Label use, it might not be able to distinguish between datasets with overlapping clusters. This algorithm works best when most instances within each cluster share the same class label but struggles with more complex labelling scenarios.

The reason these solutions can be used as a benchmark is that they both fulfil the aim of this project, which is to build a cluster-based semi-supervised learning algorithm that combats concept drift. It is possible to compare the prototype model with the following solutions using evaluation metrics such as g-means, kappa statistic, mean absolute error, etc. The solution with the superior metric values means that the solution provides the best performance.

III. Design

Throughout this project, various algorithm designs were considered and have now been narrowed down to two designs. The first design solution is referred to as the Cluster and Label Then Sub-Classifier (CLSC) while the other is known as the Cluster and Label Then Extra Classifier (CLEC). These two solutions represent extended versions of the CL algorithm.

When data/instances are sent to both algorithms. It first arrives as part of prediction dataset so that the algorithm can make a prediction. Then the data/instances are sent to the model again as part of the training dataset to train the CL model. This can be seen in Figures 1 and 2.

The CLSC design was a discounted design which involves employing the first component of the CL method during the training phase of the algorithm. Which is essentially, grouping incoming training data into clusters. Figure 1 shows where the training data is fed into the CL model and algorithm 1 shows how it works.

In the training phase, the algorithm will find the closest cluster C_x for each instance X using the Euclidean distance formula (1). If the instance is labeled, then the clusterer C creates new clusters or updates existing clusters by training on the instance and the label. If X is unlabeled, then the algorithm provides it a pseudo-label by finding the most common class label (the representative class label) within C_x. Then C creates new clusters or updates existing clusters by training on the instance and the pseudo-label [2]. In the CLSC design, all the data points are stored within the clusters, and either have a label or a pseudo-label assigned to them.

$$d(a, b \dots z) = \sqrt{\sum_i^n (a_i - b_i \dots - z_i)^2} \qquad (1)$$

Algorithm 1: Cluster And Label: Training

```
Input: A clusterer C
while stream is active do
    X ←nextInstance()
    C_x ←closestCluster(X)
    If X is labeled then
        C.train(X,y)
    else
        ŷ← C_x.getPseudoLabel(X)
        C.train(X, ŷ)
```

The reason why the unlabeled X is given the representative class label within C_x is because of the cluster assumption [8]. The cluster assumption states that data points belonging to the same cluster belong to the same class. In this case we are assuming that X is a part of C_x.

CLSC and the CL methods are similar; however, CLSC differs in the classification stage of the algorithm. The classification stage is when the data stream sends data points to the CLSC model, so that the model can make a prediction of what class belongs to that data point. This can be seen in Figure 1.

In the classification phase, the clusters are utilized to classify data. The algorithm first determines the closest cluster to the unlabeled instance. Then, using the data points that are within the cluster, a supervised classifier gets trained/updated and subsequently is used to predict the label for the instance.

The classification phase of CLSC leverages both labeled and pseudo-labeled data from the cluster so that the algorithm can train/update and use a supervised classifier to make predictions. The reasoning behind this is that pseudo-labeled data effectively augments the size of the labeled dataset. In many

machines learning tasks, having a larger labeled dataset can lead to more robust and accurate models.

However, the biggest flaw with this approach is that clusters are subsets the stream data, and they may not always represent overall data. If the clusters are not well-balanced, it may introduce bias in classification. Unlabeled data points may be misclassified because they align with a different part of the feature space. Which is why this design was ultimately discounted.

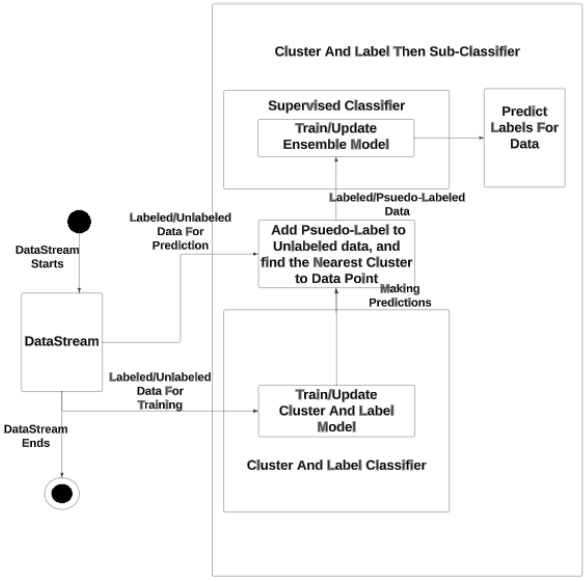

Fig. 1: Overview and behavior of the CLSC Algorithm.

The CLEC design that was used in the project also employs the CL method in the training phase just like the CLSC method. This is because it also uses algorithm 1 in the training phase. But, in the CLEC design, it does not use clusters, instead it uses micro clusters which is a summarized representation of a cluster of data points. The CLEC design also employs a slightly different classification method using algorithm 2.

For the classification phase, the algorithm first determines the C_x to the X. Then, a pseudo label ŷ is then given to the instance if it's unlabeled. Then using that (labelled or pseudo labeled) data point, a supervised classifier gets trained/updated, and then is used to predict what class belongs to that instance. This is visualized in Figure 2.

Algorithm 2: CLEC: Classifying

Input: An instance X, A supervised classifier S, A clusterer C

```
X ←nextInstance()
Cx ←closestCluster(X)
Prediction← null
If X is unlabeled then
    ŷ← Cx.getPseudoLabel(X)
    S.train(X, ŷ)
    Prediction← S.getVotesForInstance(X, ŷ)
else
    S.train(X, y)
    Prediction← S.getVotesForInstance(X, y)
return Prediction
```

Overall, CLEC has the same advantages that CLSC has, which is the ability to use labeled and pseudo-labeled data to train a supervised machine learning algorithm. The ability to create and use pseudo-labeled data also increases the size of the labeled dataset which can lead to more robust and accurate models. However, CLEC uses the entire available dataset to make predictions, whereas CLSC doesn't, which makes CLEC the better algorithm. This is because using the entire available dataset is more likely to represent the overall data.

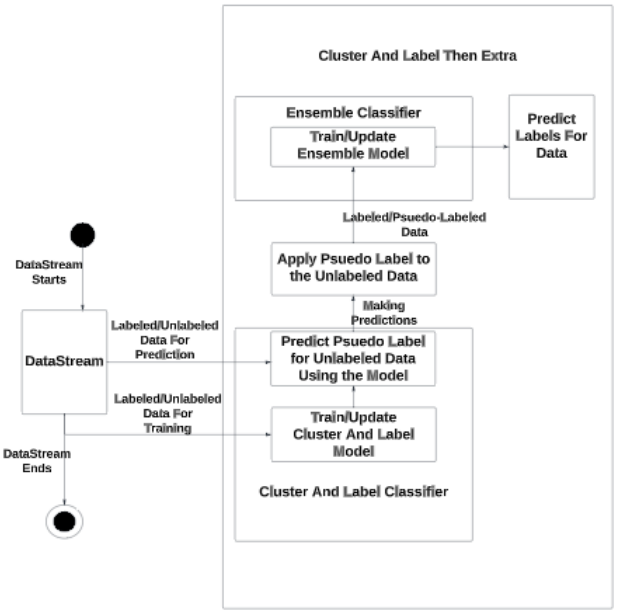

Fig. 2: Overview and behavior of the CLEC Algorithm

II. Implementation

In the implementation phase of the project, the CLEC design was used to build two solutions called Cluster and Label Then Classify Stats (CLTC) [10] and Cluster and Label Then Classify Ensemble (CLTCE) [9]. Both solutions have the same implementation for the training components which can be visualized in Figure 3. However, both use different implementations of the Classification component which can also be visualized in Figure 4.

For the training component, both solutions use algorithm 1 to build, train or update the CL model. They both use the CluStream algorithm to cluster the incoming training data into micro clusters. The reason why CluStream was the best

clustering algorithm is it is designed to be computationally efficient for streaming data [7]. This is partly because it uses micro clusters, which is a summarized representation of a cluster of data points. So, the clusters don't need to store any data points at all. The algorithm also uses a sliding time window to make sure to retain recent micro-clusters, and to gradually discard older micro clusters when they become too old. This allows CluStream to adapt to evolving data distribution and detect concept drift over time because it's constantly adapting to the newest data, and it also gradually removes redundant data.

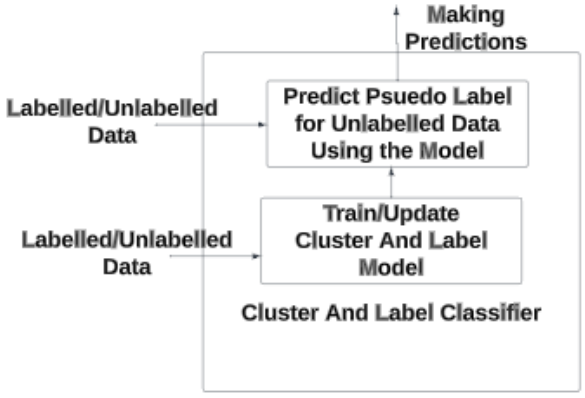

Fig. 3: The Training Component of CLEC Algorithm

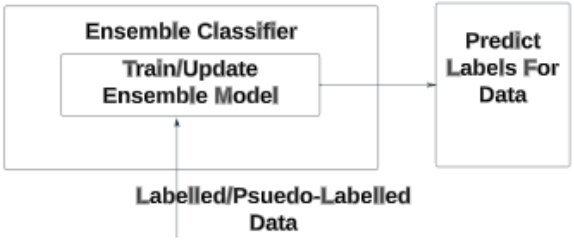

Fig. 4: The Classification Component of CLEC Algorithm

For the classification component, both solutions use algorithm 2 to build, train or update the supervised classifier model. They both use an ensemble algorithm as their supervised classifier to make the final prediction of each instance. This is because ensemble machine learning algorithms combine predictions from multiple models to produce a more robust prediction. This is because if an ensemble has a diverse set of base models, the ensemble can still perform well even if some individual models underperform or are affected by outliers. However, the biggest drawback in using an ensemble is that they are computationally expensive and time-consuming due to the need for training and storing multiple models. The reason why these two solutions are a bit different is that CLTCE uses an ensemble algorithm that was manually built with five different sub learners and CLTC uses an ensemble algorithm from MOA which is called the OzaBag ensemble.

The solution proposed in this project does not need to address environmental sustainability considerations. This is because the amount of energy this solution uses is minimal and will not affect the environment unless it used on an industry scale which is very unlikely. This is because the solution does not require any high energy tools or programs to run.

III. Evaluation

To evaluate the solutions that I have built, I'm going to use two main metrics which are the accuracy of the models, and the how long each model takes to process 100,000 data instances (execution time). Because the ideal model would be something that can predict the target variable at a high accuracy rate with a reasonable execution time to pre-process data, learn, and predict data.

Table I and II show the performance (the accuracy and the execution time) of all the models under different datasets. The models used was the CL algorithm (as a benchmark), the two models that has been developed in this project, which is the CLTCE and CLTC algorithm, and hoeffdfing tree algorithm. The hoeffdfing tree algorithm is a supervised learning algorithm that is used to compare against the semi supervised learning algorithms.

Each of the models except for hoeffdfing tree, had to process 100,000 data instances, with 90% of them being unlabeled data. The hoeffdfing tree algorithm only takes labeled data as it a supervised learning model.

From what we can see in Table I, is that the hoeffdfing tree algorithm does exceptionally well as it has the highest accuracy amongst the models under every single data set. This is to be expected because it does not use unlabeled data. It does appear that CLTCE is the worst performing semi supervised learning algorithm, as it has the lowest accuracy in every single data set. Amongst the semi supervised learning algorithms, the CLTC algorithm does the best in the SEA data set with an accuracy of roughly 60%. Whereas the Cluster and Label algorithm does the best in the SINE data set with an accuracy of roughly 63%. So, to conclude, there is no semi supervised learning algorithm that is inherently better than its peers.

TABLE I: Comparing semi supervised learning algorithms accuracy on different datasets.

Data Sets	Cluster And Label Accuracy (%)	CLTCE Accuracy (%)	CLTC Accuracy (%)	Hoeffdfing Tree Accuracy (%)

SEA	55.407	53.85	60.231	86.42
STAGGER	88.89	88.89	88.89	99.91
SINE	63.306	46.46	58.1	96.51

In Table II, we see that amongst the semi supervised learning algorithms, that the Cluster and label algorithm is generally the quickest algorithm, as it has the fastest time in the SEA and STAGGER dataset and is only 3 seconds slower than the CLTC algorithm in the SINE dataset. This is to be expected, because the CLTC and CLTCE algorithm is an extended version of the Cluster and Label algorithm. Since both these algorithms use the Cluster and Label model as well as an ensemble model.

TABLE II: Comparing semi supervised learning algorithms execution time on different datasets.

Data Sets	**Cluster And Label (s)**	**CLTCE (s)**	**CLTC (s)**	**Hoeffdfing Tree (s)**
SEA	103.23	116.63	110.43	0.32
STAGGER	55.46	92.90	60.45	0.20
SINE	131.42	174.37	128.40	0.17

Figures 5 and 6 show the accuracy of the CLTC and CLTCE and the CL algorithm. However, in Figure 5 and 6 we also see accuracy of the pseudo label within these algorithms as well. In Figure 6, the accuracy of the sub classifiers is shown as well. These models also had to process 100,000 data instances, with 90% of them being unlabeled data.

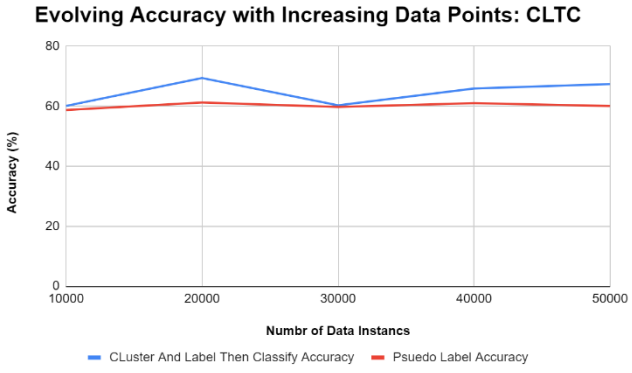

Fig. 5: The evolving accuracy of the CLTC algorithm and the pseudo label accuracy inside the CLTC algorithm using the SINE data set.

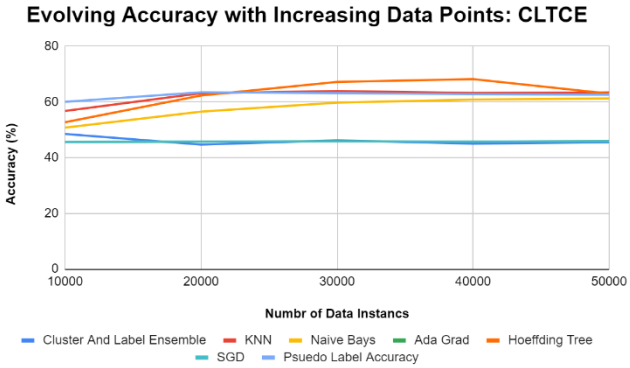

Fig. 6: The evolving accuracy of the CLTCE algorithm and the pseudo label and the sub classifiers accuracy inside the CLTCE algorithm using the SINE data set.

After evaluating Figure 5, the pseudo labeling generally performed better than the overall performance of the CLTC algorithm. However, this is most likely since the models were processing a SINE data set. The CL algorithm outperforms both the CLTCE and CLTC algorithm when it comes to accuracy in the sine data set as seen in Table I. This is important because the pseudo labeling process is based on the CL algorithm. This is further shown in Figure 8, where the pseudo-label accuracy was consistently below the CLTC algorithm when it was running on SEA dataset. This means that the accuracy of the CLTC algorithm is not completely dependent on the CL algorithm.

After evaluating Figure 6, it is safe to say that CLTCE algorithm has underperformed. This is because the accuracy of three of the sub classifiers and the pseudo labelling accuracy has consistently been higher than the CLTCE algorithm. This means that the two underperforming sub classifiers has dragged the CLTCE algorithm to their level. This algorithm isn't also performing poorly due to the SINE dataset, because the algorithm also performed poorly on the SEA dataset as you can see in Figure 7.

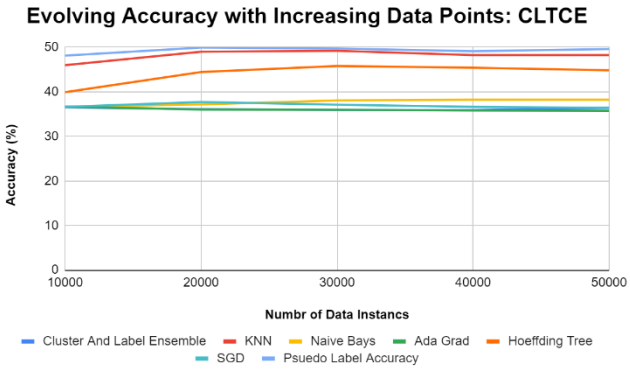

Fig. 7: The evolving accuracy of the CLTCE algorithm and the pseudo label and the sub classifiers accuracy inside the CLTCE algorithm using the SEA data set.

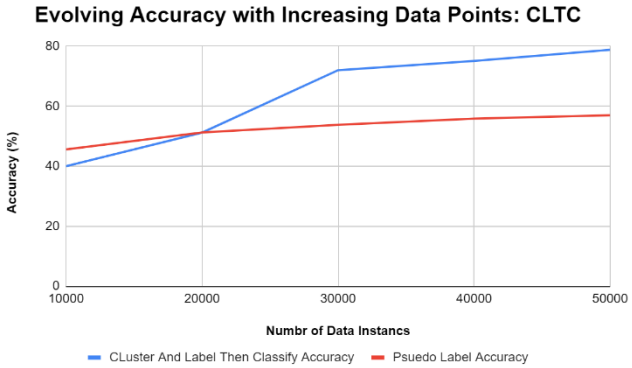

Fig. 8: The evolving accuracy of the CLTC algorithm and the pseudo label accuracy inside the CLTC algorithm using the SEA data set.

Figures 9,10 and 11 show the model's performance when it experiences concept drift. In all three figures, all the models experience a drop in accuracy when all the models process roughly around 50,000 data instances. This means that concept drift is being simulated around that time. However, all three models can slowly recover from the concept drift, after 10,000 instances later. This means that all the models can adapt to concept drift.

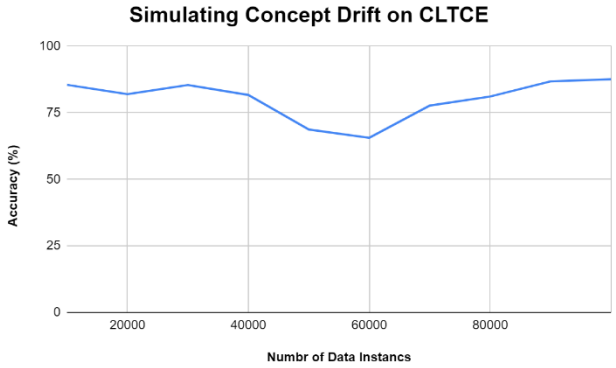

Fig. 9: Simulating Concept Drift on the CLTCE algorithm.

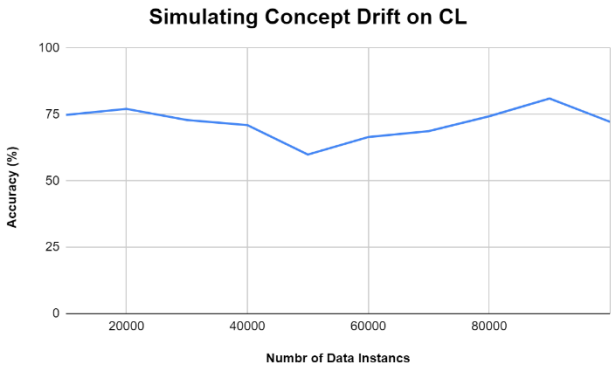

Fig. 10: Simulating Concept Drift on the CL algorithm.

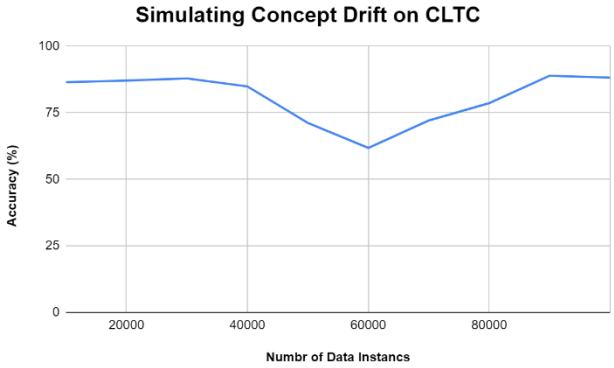

Fig. 11: Simulating Concept Drift on the CLTC algorithm.

To summarize, when evaluating the two algorithms that I've built, the algorithm that performed the best was CLTC, as it was able to consistently get higher accuracy percentages than the CLTCE algorithm. The CLTC model had an average accuracy rate of 68% and an execution time of 100 seconds when processing 100,000 data instances which is better than the 162 second exection time and 62% that CLTCE achieved. Also, the CLTCE algorithm has performed poorly, since the accuracy of the CLTCE algorithm usually imitates the accuracy of the worst performing sub classifier, which is not ideal.

The biggest limitations for the CLTC algorithm were that it sometimes underperforms against the pseudo-label classifier inside the algorithm. The idea is that CLTC algorithm would outperform the pseudo-label classifier inside CLTC. This could be done by adding weights to the pseudo-label classifier. By adding appropriate weightage to the pseudo labeled data. The CLTC algorithm could perform even better.

IV. Conclusion And Future Work

In the future, other types of semi supervised learning algorithms asides from cluster based supervised learning, such as active learning or transfer learning.

V. References

[1] M. Bahri, A. Bifet, J. Gama, H. M. Gomes, and S. Maniu, "Data stream analysis: Foundations, major tasks and tools," WIREs Data Mining and Knowledge Discovery, vol. 11, no. 3, Mar. 2021, doi: https://doi.org/10.1002/widm.1405

[2] M. H. Le Nguyen, H. M. Gomes and A. Bifet, "Semi-supervised Learning over Streaming Data using MOA," 2019 IEEE International Conference on Big Data (Big Data),Los Angeles, CA, USA, 2019, pp. 553-562, doi: 10.1109/BigData47090.2019.9006217

[3] J. Gama, I. Žliobaitė, A. Bifet, M. Pechenizkiy, and A. Bouchachia, "A survey on concept drift adaptation," ACM Computing Surveys, vol. 46, no. 4, pp. 1–37, Apr. 2014, doi: https://doi.org/10.1145/2523813

[4]M. M. Masud, J. Gao, L. Khan, J. Han and B. Thuraisingham, "A Practical Approach to Classify Evolving Data Streams: Training with Limited Amount of Labeled Data," 2008 Eighth IEEE International Conference on Data Mining, Pisa, Italy, 2008, pp. 929-934, doi: 10.1109/ICDM.2008.152

[5] M. J. Hosseini, A. Gholipour, and H. Beigy, "An ensemble of cluster-based classifiers for semi-supervised classification of non-stationary Data

Streams," Knowledge and Information Systems, vol. 46, no. 3, pp. 567–597, 2015. doi:10.1007/s10115-015-0837-4

[6] S. Jaramillo-Valbuena, S. Augusto-Cardona, and J. A. Aldana, "Performance evaluation of different clustering algorithms for data streams," RevistaESPACIOS,https://www.revistaespacios.com/a19v40n38/19403817.html

[7]Sybernix, "CluStream — A Framework for Clustering Evolving Data Streams," Medium, Jul. 25, 2017. https://sybernix.medium.com/clustream-a-framework-for-clustering-evolving-data-streams-b2f8b2d65ae (accessed Oct. 15, 2023).

[8]J. E. van Engelen and H. H. Hoos, "A survey on semi-supervised learning," Machine Learning, Nov. 2019, doi: https://doi.org/10.1007/s10994-019-05855-6.

GitLab Links to the Project:

[9]V.M Ajith, "ENGR489", GitHub, https://gitlab.ecs.vuw.ac.nz/course-work/project489/2023/ajithvais/engr489/-/blob/main/moa/src/main/java/moa/classifiers/semisupervised/ClusterAndLabelThenClassifyEnsemble.java?ref_type=heads

[10] V.M Ajith, "ENGR489", GitHub, https://gitlab.ecs.vuw.ac.nz/course-work/project489/2023/ajithvais/engr489/-/blob/main/moa/src/main/java/moa/classifiers/semisupervised/ClusterAndLabelThenClassifyStats.java?ref_type=heads