EP2420 Project 2 Online learning with sample selection and change detection

Chenming Du[*]*, Ruihan Zhang*[*]

December 22, 2022

Project Overview

In Machine Learning (ML), typically we do offline learning, meaning that we wait until all the training data we need are done collecting and then start to train our models. Good results in offline learning usually requires long times of data collecting and training. But in real-world scenarios, the characteristics of the data could vary every now and then, and meanwhile, we would want to use our model right away. That's how the idea of online learning emerges, where our training data is always updating, and so it is with our models.

In this project, Reservoir Sampling (RS) is used as our main approach to update the training data pool. Several different algorithms for choosing the time to update the model are investigated and compared. For simplicity, we update all our models by re-training from scratch. We measure the prediction accuracy in the form of Normalized Mean Absolute Error (NMAE). The data sets used in our project are traces from a testbed at KTH which runs a video-on-demand service and a key-value store service under dynamic load.

The project is divided into 4 successive Tasks, each conducted in a week.

Background

In this section, we provide a description of the change detection algorithms (Periodic, STUDD [1], and ADWIN [2]) and the sample selection algorithm (Reservoir Sampling [3]) used in this project. In the first method, the model is trained periodically with fixed time intervals, which will act as our trivial method. However, if the time interval is set to small, it will cause high computation costs. If the time interval is set to large, it may miss the points where the model should be updated. In order to make smarter decisions on these points, STUDD comes into use.

STUDD (Student-Teacher approach for Unsupervised Drift Detection) [1] is an unsupervised approach that is used to detect concept drift. Concept drift refers to the phenomenon where data distribution evolves. In other words, the cause of the concept drift is that the relationship between the input and output data changes over time, meaning the predictions made by a model trained on older historical data are no longer correct. These concept drifts degrade the performance of the model and cause disruptions.

The main idea of STUDD is to build two predictive models (The Student model and the Teacher model). The teacher model acts as the main predictive model, and the Student model is used to mimic the behavior of the teacher as shown in Figure 1. Because the approach is unsupervised, meaning no future labels are provided, the whole approach is carried out by monitoring the student's mimicking loss. The loss is illustrated as a function of the discrepancy between the Student model's prediction and the Teacher model's prediction in the same instance. Then the loss acts as the input for a detector (i.e. Page-Hinkley test [4]) which is responsible for detecting concept drift.

[*]These authors contributed equally to this work.

In summary, the Teacher model is used to make predictions on the data stream, while the concept drift detection is performed on the loss of the Student model.

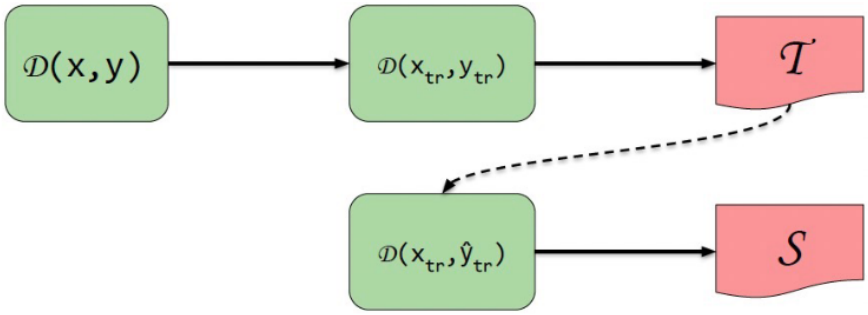


Figure 1: Student and Teacher model in STUDD. x_{tr} and y_{tr} refer to the x and y values from the the training set. $\hat{y}_{tr}$ means the predictions of the teacher model on these samples. T means the teacher model, while S means the student model.

ADWIN (ADaptive WINdowing) [2] is another algorithm used to detect change points. Unlike STUDD, ADWIN maintains a dynamic sliding window whose size varies. In the initial setup of the algorithm, ADWIN will maintain a window of size W. Then when the two subwindows of W exhibit distinct averages, the older portion of the window is dropped. In this case, the size of the window will keep growing when the data is stationary and shrink when the change takes place by dropping outdated data.

Reservoir Sampling (RS) [3] is a sample selection algorithm designed for a large amount of data stream. It uses very little space and computation cost to maintain a uniformly random selection from a relatively large dataset whose length is unknown beforehand.

So it is helpful for online learning when you monitor the incoming data but only want a small sample of it. The algorithm initializes a buffer load or data cache of size n by filling the data until it's full, then it decides at k/n probability to keep the k_{th} coming data in the buffer or not.

The procedure of Reservoir Sampling is summarized as follows:

Assume the cache size is n and the size of the whole data set is N.

1. Copy the first n elements from the whole data set to the cache.
2. Iterate from n to $N-1$. In each iteration j, generate a random number **rand** from 0 to j. And then judge whether the random number **rand** is less than n. If so, replace the element at index **rand** in the cache with the item at index j in the whole data set.

Data Description

The allocation of the data set is shown in Table 1.

Data set	
VoD flashcrowd (35000 samples)	Ruihan Zhang
KV periodic (28000 samples)	Chenming Du

Table 1: Data set allocation

For both data sets, we use the tree-based feature selection algorithm (i.e. DecisionTreeRegressor) to select the top 16 features. We apply the standardization column-wise on both data sets. We also remove

the outliers for both data sets by keeping 99% of the samples. The number of samples after doing outlier removal is shown in Table 2. For all predictive models, we use the random forest regression (20 trees) and the prediction error is illustrated in NMAE.

Data set (after outlier removal)	
VoD	34659 samples
KV	27720 samples

Table 2: Data set after doing outlier removal

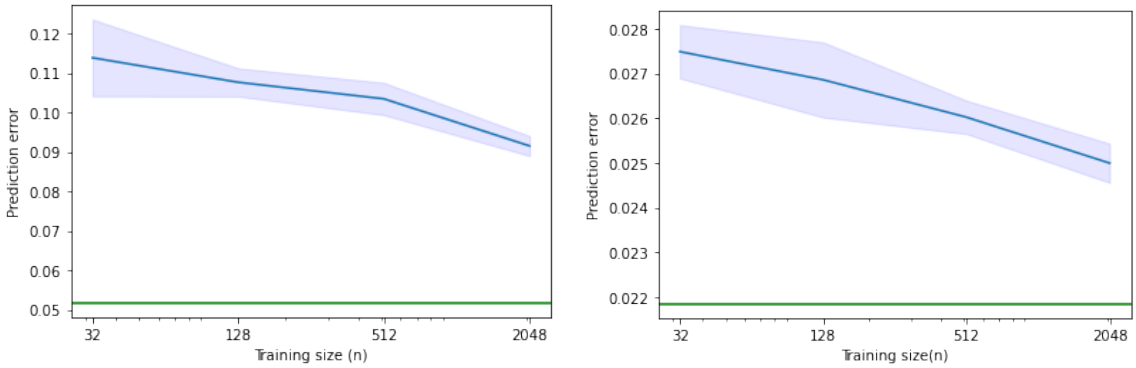
Task 1 - Offline and online learning on a small-size training set

Offline learning

The training set and the test set are built in the following way:

1. The training set is selected uniformly at random with size 32, 128, 512, and 2048.
2. The test set is obtained by selecting uniformly at random with size 1000 without including the samples in the training set.
3. Calculate the prediction error and repeat the above procedure for 10 times.

Figure 2 shows our results for offline learning.



(a) Prediction error vs training set size (n) for VoD

(b) Prediction error vs training set size (n) for KV

Figure 2: The relationship between the prediction error and the size of the training set. The green line shows the prediction error with 70% samples for training and the remaining 30% samples for testing. Figure 2(a) shows the VoD service, while Figure 2(b) shows the KV service. In both figures, the x-axis shows the training set size and the y-axis shows the mean value of the prediction error. The blue shadow shows the 95% confidence interval.

From Figure 2, we could get the following two conclusions:

- For both data sets, when the size of the training set increases from 32 to 2048, the prediction error decreases. The reason is that the model becomes more accurate with more inputs.
- For both data sets, the prediction error obtained using offline learning is higher than the prediction error obtained with a ratio of 7:3 of the training set and test set. The reason is that from Table 2, we could get that for the VoD data set, the number of 70% samples will be 24261, while for the KV data set, the number of 70% samples will be 19404. We could see both numbers are much larger than 2048.

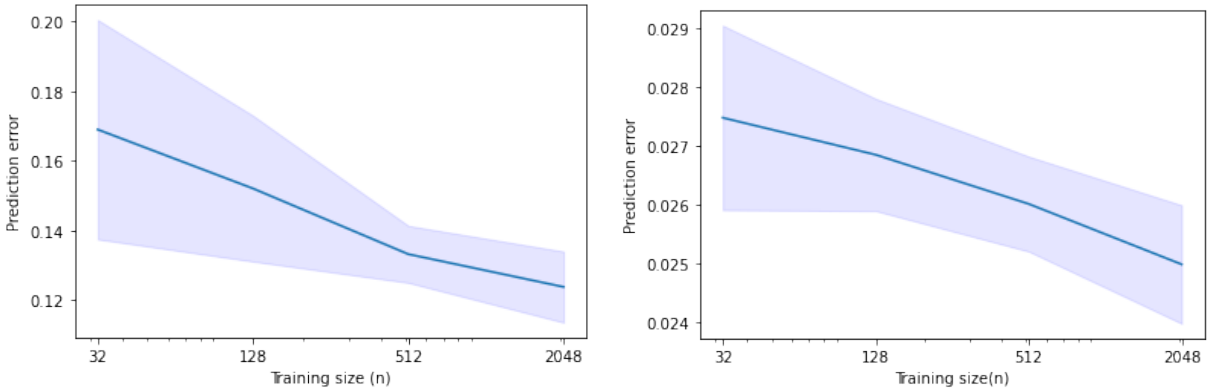
Online learning

Compared with Offline learning, here the training set is constructed using a sample selection algorithm **Resorvoir Sampling (RS)**. The description of algorithm is described in the Background section.

The training set and the test set are obtained in the following way:

1. Choose a random starting point t_0 uniformly in the range $[1, 5000]$.
2. Use the samples of the sequence (x_{t_0}, y_{t_0}), (x_{t_0+1}, y_{t_0+1}), ..., $(x_{t_0+2999}, x_{t_0+2999})$ as input for the RS algorithm and fill the cache of n samples, which acts as the training set.
3. The test set is composed of the samples of the sequence $(x_{t_0+3000}, y_{t_0+3000})$, $(x_{t_0+3001}, x_{t_0+3001})$, ... until the end of the trace.
4. Repeat the above procedure 10 times.

Figure 3 shows our results for online learning.



(a) prediction error vs n for VoD service

(b) prediction error vs n for KV service

Figure 3: The prediction error in function of different cache sizes for online learning. Figure 3(a) shows the figure for VoD service and Figure 3(b) shows the figure for KV service. The x-axis shows the training size and the y-axis shows the mean value of the prediction error. The blue shadow shows the 95% confidence interval.

From Figure 3, we could find that, like offline learning, when the training size increases, the prediction error decreases because more samples to train means better model accuracy. But the overall error is slightly worse than the offline learning. We assume that this is because, for online learning, the cache only consists of samples of a certain time interval, the samples in the cache cannot fully represent the distribution of future samples.

Discussion

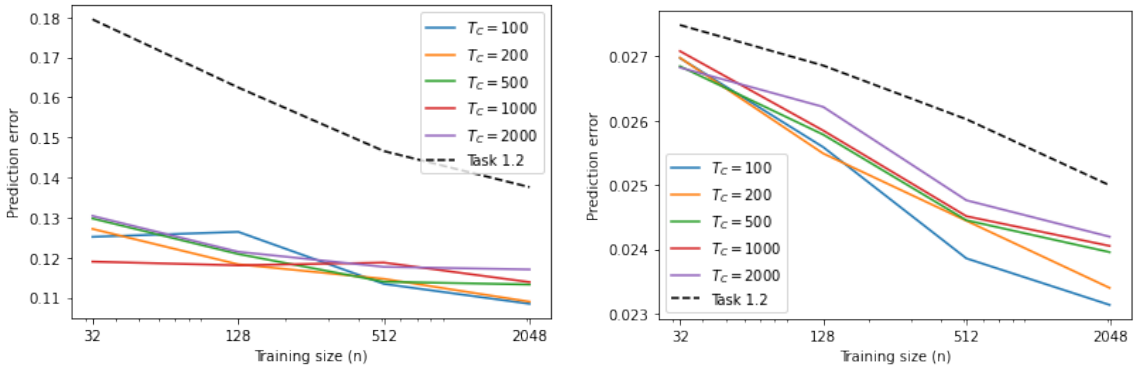
When N is relatively much larger than n The computational complexity of the RS algorithm can be achieved in $O(N)$ time, which is not dependent on n, since the one iteration is at a fairly constant cost. But Li [5] points out that the optimal CPU time can reach $O(n(1+log(N/n)))$ which will decrease if we have a smaller n.

Task 2 - Online learning with periodic model re-computation

Compared with the online learning method in Task 1, here in Task 2, we continuously update the cache and retrain the model. The steps to perform online learning with periodic model re-computation are summarized as follows:

1. Choose a random starting point t_0 from the range $[1, 5000]$.
2. Fill the cache of size n using the samples from $(x_{t_0+3000}, y_{t_0+3000})$ using RS.
3. The initial cache is continuously updated from the timestamp $t_0 + n$ to the end of the trace. At the same time, the model is retrained on different timestamps (In our experiment, the timestamps where the model will be retrained is shown in the form of $[t_0 + 3000 + a * T_C]$ where a is an integer and the retrained model will be evaluated on the following T_C subsequent samples.
4. The prediction error is calculated based on the concatenated prediction value from the whole data set.
5. Choose different T_C values and repeat the above procedure 10 times.

Figure 4 shows our results for the two data sets.



(a) Prediction error vs training set size (n) for VoD

(b) Prediction error vs training set size (n) for KV

Figure 4: The prediction error in function of the cache size (n) under different T_C values. The black dashed line shows the prediction error in function of the cache size without re-computation (Online learning in Task 1), and the other curves show the prediction error in function of the cache size (n) with periodic model re-computation (Different T_C value corresponds to different color curve). Figure 4(a) shows the result for the VoD data set and figure 4(b) shows the result for the KV data set. The y-axis shows the mean value of the prediction error and the x-axis shows the size of the cache.

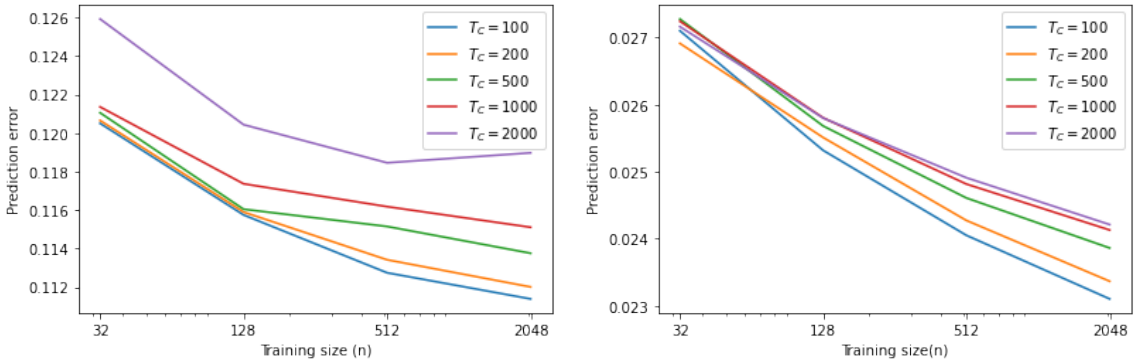
From Figure 4, we can summarize the following conclusions:

- For both data sets, we could find that compared with the black dashed line, with periodic model re-computation, we could get a lower prediction error.
- When the cache size increases, we could find that the prediction error decreases accordingly. This is because when the cache size is larger, the possibility of replacing the elements in the cache will increase. So when the model is retrained, it can get more characteristics of the recent samples, thus getting a smaller NMAE.

Discussion

During the experiment, we expect to get the conclusion that when the re-computation interval T_C is small, we could get a better prediction accuracy, in other words, lower prediction error. Because with a lower re-computation interval, we will retrain the model more frequently, thus the more closely the training data and the test data will be.

However, we only see the phenomenon when the cache size is large (In Figure 4, $n = 2048$). So in order to further study the impact of T_C on the error curve, we want to reduce the randomness of the average prediction error in the experiment. So we run the test 100 times for each n and T_C and the results are shown in Figure 5.



(a) Prediction error vs cache size (n) for VoD over 100 times

(b) Prediction error vs cache size (n) for KV over 100 times

Figure 5: The prediction error in function of the cache size (n) under different T_C values over 100 times.

From Figure 5, we could find that the results are compelling:

- The curves are no longer intersecting with each other, although some still appear when the cache size is small. But a more clear and more stable order of the curves is emerging - The smaller the T_C is, the lower the prediction error will be.
- When n is small, the average prediction error points for different T_C are more close to each other. When n becomes larger, the points become more distinguishable, meaning that the re-computation interval is having more impact on the performance when the cache size is larger. On the other side, the randomness of the cache as its size gets smaller tends to cover the impact of T_C.

To summarize, a higher re-computation frequency in online learning is beneficial, and this impact is more obvious when the cache size is large.

Task 3 - Online learning with change detection using STUDD

In this task, we used the STUDD algorithm to trigger model re-computations. In this task, we have three random forest models with the same parameters. One is our major prediction model, the second is acting as the teacher model, and the last is acting as the student model.

The main way for doing this task is shown as follows:

1. We select a random number in the range [1,5000] named t_0, and we use the samples in the range [t_0, $t_0 + 3000$] to fill the cache using the Reservoir (RS) algorithm.
2. We update the cache continuously as we did in Task 2.

3. We start the change detection at time $t_0 + 3000$. In order to start the detection, we use the samples $[t_0+2000, t_0+3000]$ to first train the teacher model and predict the values using these training samples. Then we replace the y values in these samples with the predictions from the teacher model to train the student model.

4. Then when we have the student model and the teacher model, we predict the predictions from samples $t_0 + 3000$. With these two predictions, we calculate the loss function as $abs(student\_prediction - teacher\_prediction)/teacher\_prediction$. We use the loss function as the input for the Page-Hinkley test [**?**]. Whenever a change is detected, the teacher model and the student model will be retrained on the latest 1000 samples until the end of the trace. After the student and the teacher model are retrained, we will update the loss function. (For example, if a change is detected in t_1, we will use the samples $[t_1 - 1000, t_1]$ to retrain the two models and the loss function will be updated).

5. From the previous three steps, we can get the timestamps where the model used for prediction should be retrained using the samples in the cache. In order to do the prediction, we use the following diagram (Assume we have three different timestamps where a change happens: $[t_1, t_2, t_3]$:

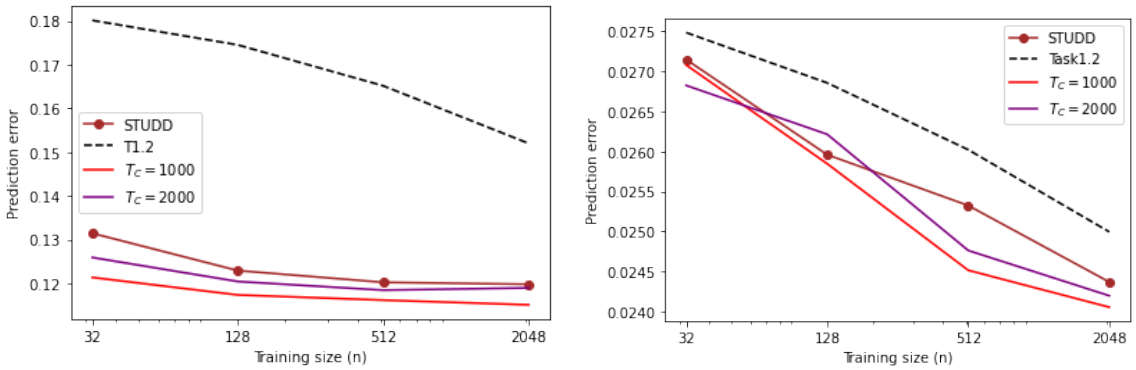
   - The model for the prediction will be retrained on timestamp t_1 using the samples in the cache in t_1 and the prediction will be done on the samples in $[t_1, t_2]$.
   - The model for the prediction will be retrained on timestamp t_2 using the samples in the cache in t_2 and the prediction will be done on the samples in $[t_2, t_3]$.
   - Do similar operations if there are more timestamps.

6. After we get all the predictions on the whole data trace, we will calculate the prediction error. We will repeat the whole procedure 10 times.

By adjusting the parameter threshold in the Page-Hinkley test, we're able to increase or reduce the average number of model re-computations in one run. We have found during our experiment process that, the lower the threshold is, the more detected change points hence more number of model re-computations there will be.

In this Task, in order to reasonably compare with Task 2, we set the parameters so that the average interval between two nearby change points is near 2000, meaning the average number of re-computations in these two tasks will be close, except one is proactive, and the other is static. By setting delta=0.001 and threshold=5 in the VoD dataset, we have 13 average re-computations in each run. By setting delta=0.0005 and threshold=1 in KV dataset, we have 10 average re-computations in each run.

Our expectation is that STUDD is a smarter algorithm than the periodic model re-compute method. Because it's built on Concept Drift Change which is adaptive to the data stream. So, if the total number of model re-computation is equal, the schedule of model updating produced by STUDD will be a better one than simply periodic.



(a) NMAE vs n for VoD service

(b) NMAE vs n for KV service

Figure 6: Online learning using STUDD comparing against periodic re-computation and online learning on a small-size training set. The brown curve shows our measured average NMAE in 10 times.

However, the results are not as we expected. Looking at the STUDD and $T_C = 2000$ curve in Figure 6, these two methods are close in the number of model re-computations, but we actually don't see a better prediction error in STUDD. Despite the result in KV $n = 128$, where STUDD outperforms $T_C = 2000$ and approaches $T_C = 1000$, we could draw the conclusion that the periodic schedule is simpler yet generally better than STUDD. The $T_C = 1000$ curve is shown here to illustrate their relative difference. We think the possible reasons for this not matching our expectations are:

1. The STUDD algorithm may not be suitable for these two data-sets.
2. The choice of the loss function which we use as the input for the Page-Hinkley test could be alternated by other forms.
3. The hyper-parameters in the Page-Hinkley test are not tuned to the best. Maybe at other number of re-computations will STUDD perform correctly.
4. STUDD doesn't work stably. It's affected by the randomness of only 10 times repetitive experiment.

But online learning using STUDD in two data-set both perform much better than Task 1.2(no re-computation). We think this is because no matter what algorithm one uses to trigger the model re-computation, as long as it re-computes, the model will be more updated, so the performance will be better than not updating at all.

Task 4 - Online learning with change detection using ADWIN

Similar to STUDD, ADaptive WINdowing (ADWIN) is another concept drift detection algorithm that can be utilized in unsupervised online learning. But this algorithm is originally designed for a uni-variate data stream, so how to adapt it to the multi-dimensional case becomes our focus in this Task.

A basic approach is to run 16 ADWIN in parallel on every feature in X, and any detected drift from them will trigger our model re-computation. However, this approach is not applicable because it produces too many points, which cannot be mediated by setting the δ parameter in ADWIN to a big number.

So, we want to aggregate all the detected points and select the best possible points from them, hoping we can update our model in time whenever a concept drift happens without updating too much.

ADWIN with k-means clustering

The first method we explored is to do clustering on all ADWIN detection points. So that cluster centroids can be used as our updating points. We use K-means as our clustering method. In this case, the number of initial centroids in K-means will equal the number of re-computations in our aggregation result. However, we found that this is in contradiction to the idea of online learning, because the clustering process is offline learning, which can only be done after we have obtained all the data. But we will show the results anyway.

ADWIN with sliding window election

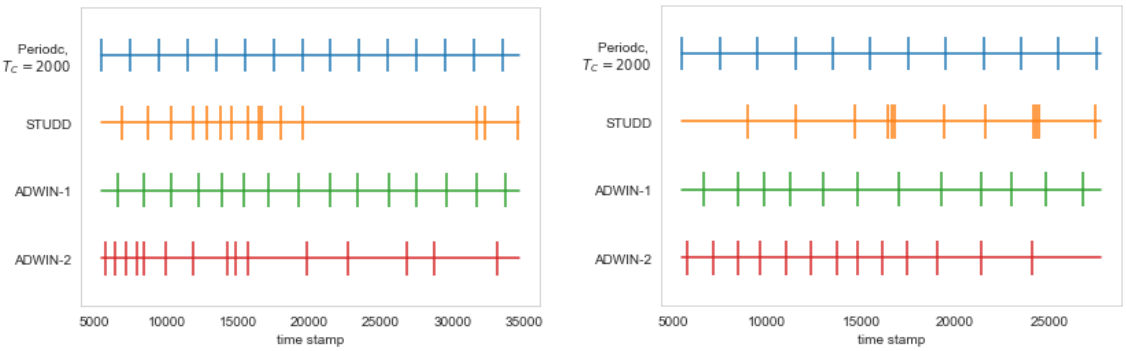
In order to make the detection method also "online", we used this sliding window algorithm to elect those points with the highest density in our candidate points produced by ADWIN: Windows size w, threshold λ and minimum distance d are three adjustable parameters. At time t_n, a re-computation is triggered only if both of these conditions are met:

1. The number of detected candidates in this window $(t_n - w, t_n)$ is above the threshold λ.
2. We have accumulated enough distance from the last time it was triggered. $t_n - t_{n-1} > d$

Discussion

Figure 7 shows an example of detected re-computation points using different algorithms under $t_0 = 2500$. We have adjust the parameters in every algorithm so that they produce the same number of re-computation points for a clearer view of the different distributions in each algorithm. In this chart, in VoD service 7(a), there are 15 points, and in KV 7(b) 11 points. We can see that:

- The periodic method is taking evenly distributed time intervals for re-computation.
- The results from STUDD are relatively the most uneven. In one period of time, it re-computes very rapidly, in another time, it rarely updates.
- ADWIN-1 is very close to the periodic method, because k-means are not likely to converge with closely located centroids.
- ADWIN-2 using our sliding window method seemed like the most probable method that best captures the concept drift points in our view. It's not too even, and not too sparse or dense.



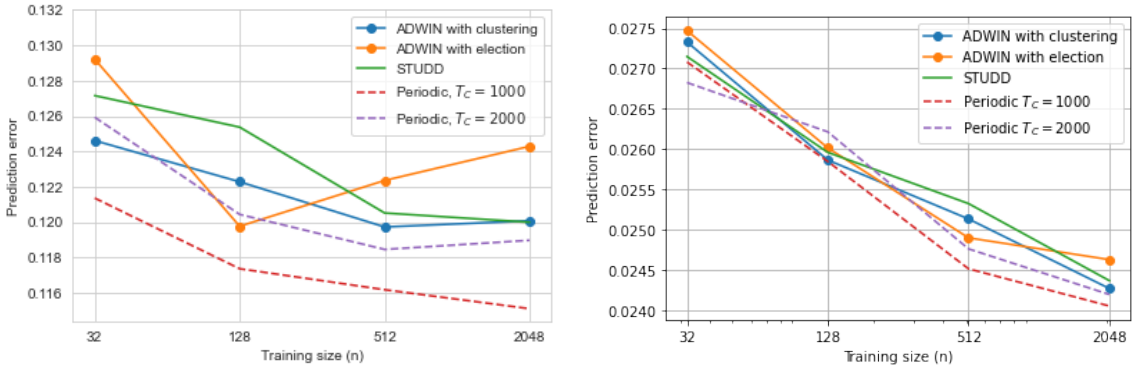
(a) Re-computation points in VoD

(b) Re-computation points in KV

Figure 7: Re-computation points using different algorithms in timeline (from $t = 5500$ to the end of data trace): each vertical line marks a re-computation point. ADWIN-1 is computed using k-means clustering, ADWIN-2 is computed using sliding window election method.

Figure 8 are our results running these different change detection algorithms. STUDD and ADWIN are supposed to compete with the periodic method under $T_C = 2000$, but we also drew the $T_C = 1000$ curve to show the relative distance. Here are our findings:

- Except at $n = 128$, the periodic method is generally the best of all, which is out of our expectation. Those methods we deemed smarter are actually worse in performance.
- ADWIN with k-means clustering is the most close to periodic, better than the other two. Because from Figure 7 we see that the distribution is also very even, similar to periodic.
- STUDD is better than ADWIN with our election method in many cases.(except $n = 128$ in VoD, $n = 512$ in KV)



(a) Prediction error vs training size in VoD

(b) Prediction error vs training size in KV

Figure 8: Online training performance using different change detection methods.

We have tested hundreds of times on these results, but there never was any algorithm that out performs the simplest periodic training. It could be that none of the algorithms we tested fail to find the real concept drift points. But noticing that the more even the updating points are, the better the performance, we doubt that the design of the updating strategy might be misleading.

Suppose that we have found the real change points accurately, and we update our models at those points. But the problem is that the training data that we have obtain at those points will be a reflection of the old data before the change point, which is not what we desired. We actually want the training data that could reflect what's after the change point, not before. In other words, we might want to select training sample after the change point, and then start our training.

References

[1] V. Cerqueira, H. M. Gomes, A. Bifet, and L. Torgo, "STUDD: A Student-Teacher Method for Unsupervised Concept Drift Detection," *CoRR*, vol. abs/2103.00903, 2021. arXiv: 2103.00903.

[2] A. Bifet and R. Gavaldà, "Learning from Time-Changing Data with Adaptive Windowing," in *Proceedings of the 7th SIAM International Conference on Data Mining*, vol. 7, Apr. 2007.

[3] C. C. Aggarwal, *Data Mining: The Textbook*. Cham: Springer, 2015.

[4] A. Bifet, R. Gavaldà, G. Holmes, and B. Pfahringer, *Machine Learning for Data Streams with Practical Examples in MOA*. Cambridge, MA: MIT Press, 2018.

[5] K.-H. Li, "Reservoir-Sampling Algorithms of Time Complexity O(n(1 + Log(N/n))),"
ACM Trans. Math. Softw., vol. 20, pp. 481–493, Dec. 1994. Place: New York, NY, USA Publisher: Association for Computing Machinery.