SOURCE CODE

The whole project source code can be found on this GitHub repository.

DATA STREAMS EXPERIMENT

All experiments were performed on *Apple Inc.* raw OHLC data (4 features) that was extended using implemented feature engineering into 52 features. Added features depended on OHLC columns directly or were stock indicators such as RSI Indicator or MACD lines. Data concerned stock operating days from 1980-12-12 to 2025-05-09 on a daily interval.

We performed a comprehensive grid search over five model families—Hoeffding Tree, Extremely Fast Decision Tree, Random Forest, MLP, and XGBoost—systematically varying each model's hyperparameters (e.g., `grace_period`, `max_depth`, `delta` for stream trees; `n_estimators`, `max_depth`, `learning_rate`/`max_features` for XGBoost/Random Forest; `hidden_layer_sizes`, `learning_rate_init`, `alpha` for MLP). All other hyperparameters were used with default settings. For each configuration, we evaluated five drift detectors (ADWIN, KSWIN, DummyDriftDetector, PageHinkley, and a Bollinger Band detector) across five learning thresholds (100, 200, 500, 1000, 2000 samples), repeating each combination five times to mitigate randomness. Learning threshold and Bollinger Band detector will be explained in the next paragraph. Overall, 15750 tests were performed. After each run, we recorded overall accuracy, number of detected drifts, and the exact parameter settings. This exhaustive sweep allows us to compare how model choice, hyperparameter settings, drift detection strategy, and buffer size interact to affect streaming prediction performance.

Learning threshold is the number of initial samples that were omitted to let non-incremental models to be trained in batch manner and in case of incremental ones were omitted to keep consistency. When drifts occured models were retrained from scratch. Bollinger Bands are a technical analysis tool traditionally used in financial markets to identify price volatility. The Bollinger Band detector is a custom concept drift detector inspired by this financial indicator. It maintains a sliding window of recent 'close' price values, calculates the mean and standard deviation, and determines whether the latest value lies significantly outside the expected range $\mu \pm k\sigma$, where k is a parameter. Every drift detection either in incremental or non-incremental model triggered model reloading.

We also tested a novel feature selection method – TStatFeatureSelector. The TStatFeatureSelector is a custom feature selection component designed for binary classification tasks in data streams. It selects the most informative features by computing two-sample t-statistics for each feature, based on differences in feature distributions between the two classes (label 0 and label 1). For each incoming instance (x, y) where x is a dictionary of feature values and y is the binary class label (0 or 1), the selector maintains class-conditional streaming statistics using `river.stats.Var`. This includes the mean, variance, and count of each feature, updated incrementally without storing the entire data history. Every `update_interval` steps, the selector calculates a two-sample t-statistic for each feature using:

$$t = \frac{|\mu_0 - \mu_1|}{\sqrt{\frac{\sigma_0^2}{n_0} + \frac{\sigma_1^2}{n_1}}}, \tag{1}$$

where:

- μ_0, μ_1 are the means of the feature for class 0 and class 1,
- σ_0^2, σ_1^2 are the variances,
- n_0, n_1 are the sample counts for each class.

Features are sorted by their computed t-statistics (in descending order). The top k features with the highest t-values are selected as the most discriminative. The selector is integrated as a transformer in a `river` pipeline. It passes through only the selected features to the downstream model, which allows efficient learning in evolving data streams.

288 tests of this feature selection method were conducted in comparison with `SelectKBest`. We performed a comprehensive grid search over two model families – Hoeffding Tree and Extremely Fast Decision Tree. For each configuration, we evaluated three drift detectors (ADWIN, KSWIN, and a Bollinger Band detector), repeating each combination three times. Learning threshold was set on 1000. Similarly, we recorded overall accuracy, number of detected drifts, and the exact parameter settings. In each experiment, we chose 15 best features.

MODELS OVERVIEW

Firstly, we'll analyse how models perform generally and later each of them will be checked one by one.

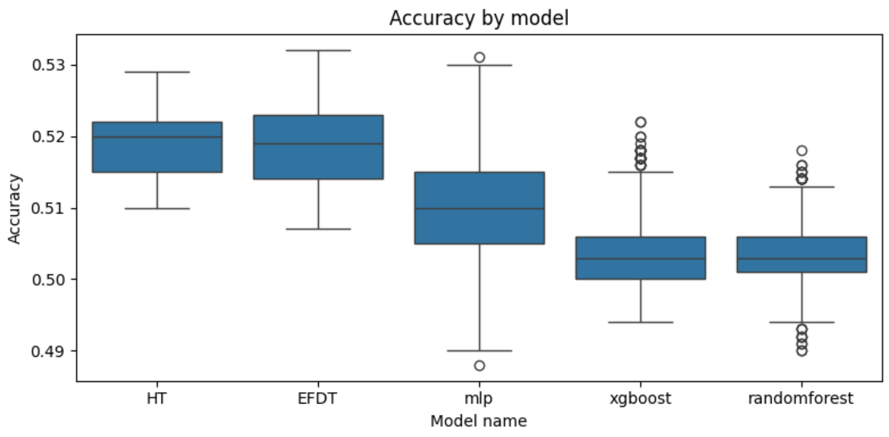

Figure 1: Models comparison by accuracy

From most general comparison we can draw conclusions that even though models behave differently, differences are not extreme as range of accuracy is between 0.53 and 0.49. The best performing models were those that were incremental, namely Hoeffding Tree and Extremely Fast Decision Tree, where the second one has higher spread and better best tests. Also, MLP is able to achieve decent scores, but presumably only in a specific configuration.

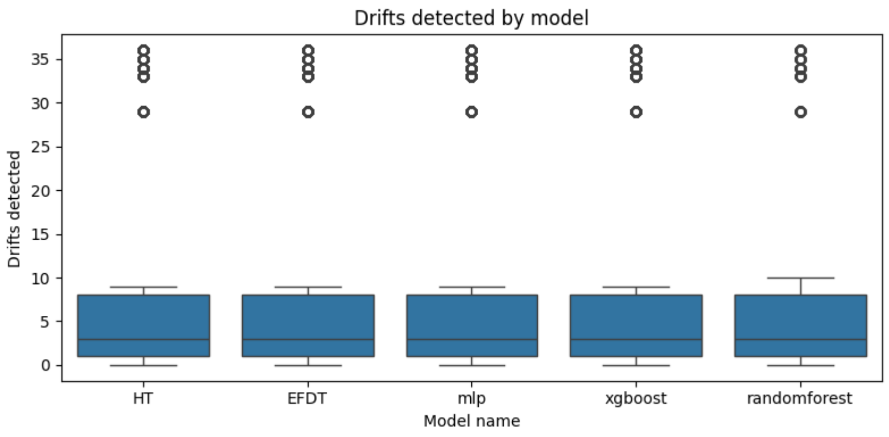

Figure 2: Models comparison by drifts detected

It's easily visible that the number of detected drifts was very similar and did not depend on model type at a glance.

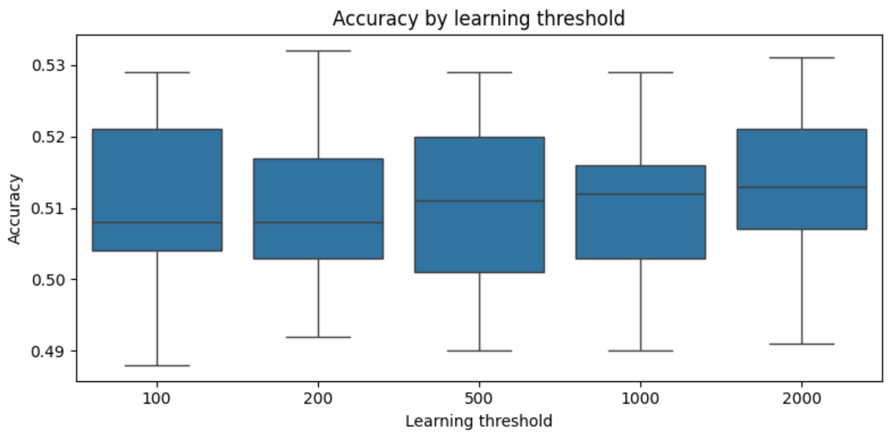

Figure 3: Models comparison by learning threshold

Learning threshold seem not to be having any particular trend.

Hoeffding Tree Classifier

Below is a table showing influence of learning threshold (LT) to accuracy.

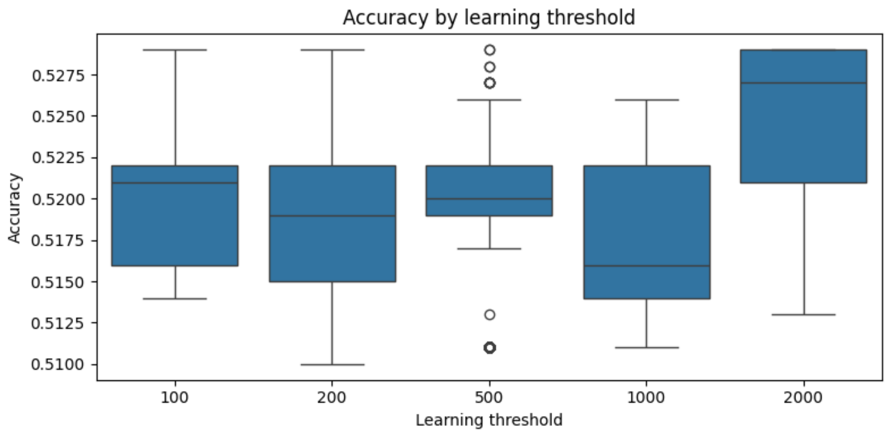

Figure 4: Learning threshold - HT

Omitting first 100-1000 rows not seem to be especially concerning, but when first 2000 rows are omitted model gets better. It indicates that perhaps data then changes and a drift should be detected. Now, let's track drift detection performance.

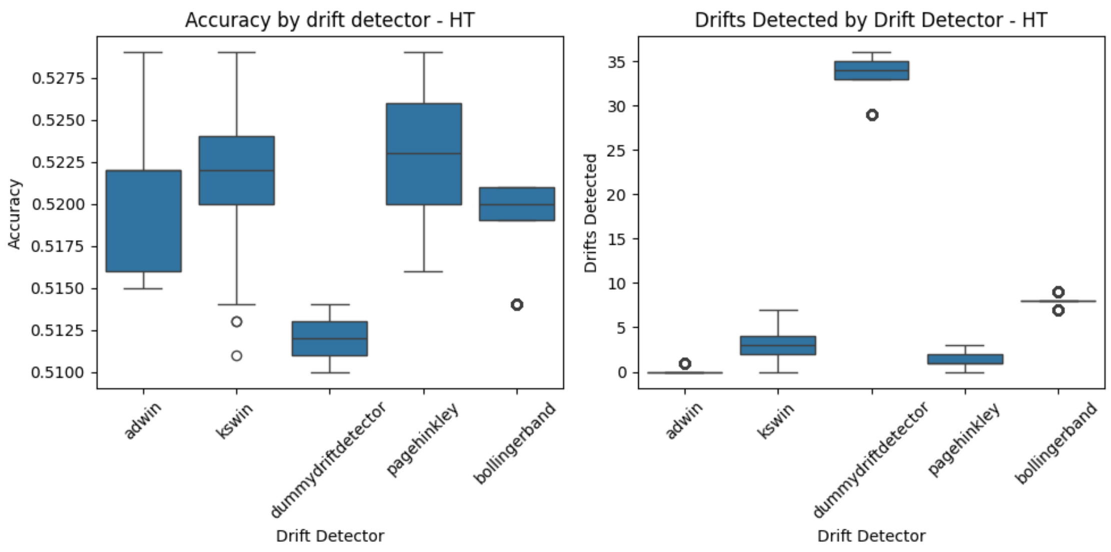

Figure 5: Drift Comparison - HT

From a drift perspective, the Page-Hinkley detector seems most reasonable. It overall gets the best performance. What's interesting it does not catch highest number of drifts, which states it's quality. Dummy Drift Detector predicted the highest number of drifts, but due to it's randomness models' performances were worst then. ADWIN on the other hand hardly predicts any drifts. Bollinger band is the only drift detector based on data rather than predictions, thus it varies very little, only due to learning threshold changes.

Below overview of the model under various parameters, grid search included values of `grace_period` in (100, 200, 300), `max_depth` of a tree in (4, 8, 12), and `delta` in (1e-3, 1e-5, 1e-7). Also learning threshold (LT) is shown, category of drift and average number of drifts, 'acc' stands for 'accuracy'. Top 7 and worst 7 combinations are shown concatenated, excluding Dummy Drift Detector as it always performs the worst.

grace_period	max_depth	delta	LT	drift	avg(drifts)	avg(acc)	max(acc)
100	12	0.001	2000	adwin	0.000	0.529	0.529
100	12	0.001	2000	pagehinkley	0.000	0.529	0.529
100	12	1e-05	2000	adwin	0.000	0.529	0.529
100	12	1e-05	2000	pagehinkley	0.000	0.529	0.529
100	12	1e-07	2000	adwin	0.000	0.529	0.529
100	12	1e-07	2000	pagehinkley	0.000	0.529	0.529
100	4	0.001	2000	adwin	0.000	0.529	0.529
300	12	1e-07	1000	bollingerband	8.000	0.514	0.514
300	4	0.001	1000	bollingerband	8.000	0.514	0.514
300	4	1e-05	1000	bollingerband	8.000	0.514	0.514
300	4	1e-07	1000	bollingerband	8.000	0.514	0.514
300	8	0.001	1000	bollingerband	8.000	0.514	0.514
300	8	1e-05	1000	bollingerband	8.000	0.514	0.514
300	8	1e-07	1000	bollingerband	8.000	0.514	0.514

Table 1: Model Args Comparison Table - HT (7 best and worst)

The best results for the Hoeffding Tree model were achieved with ADWIN and Page-Hinkley detectors with consistent model parameters ($grace\_period = 100$, $max\_depth = 12$), using a high learning threshold ($LT = 2000$), where no drifts were detected and the average accuracy reached its maximum (0.529). In contrast, the Bollinger Band detector, used with a lower threshold ($LT = 1000$), consistently triggered 8 drifts and yielded lower accuracy (0.514). This suggests that while Bollinger Bands are more sensitive to changes, they may not translate to better predictive performance in this context. This suggests that overly sensitive drift detection can negatively impact model performance and higher learning threshold may help the model generalize better and reduce false drift detections.

Extremely Fast Decision Tree Classifier

Below is a table showing influence of learning threshold (LT) to accuracy.

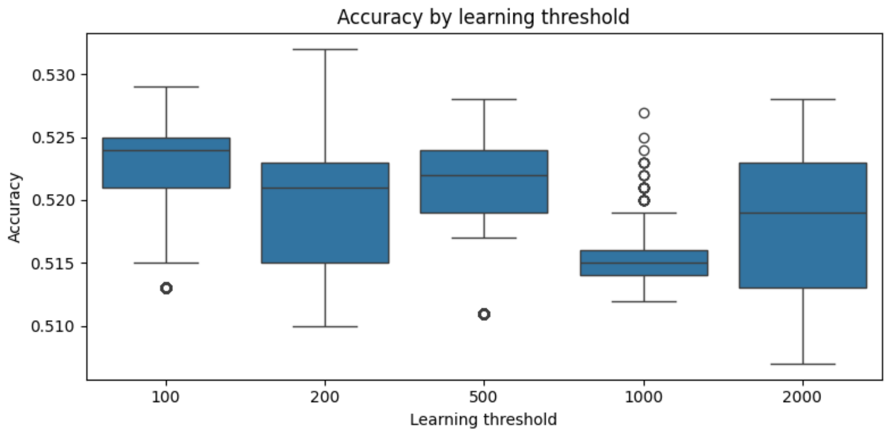

Figure 6: Learning threshold - EFDT

Figure indicates that accuracy varies with a learning threshold change, but there isn't an obvious pattern. There is a minor skew in favor of the smaller learning thresholds. Now, let's track drift detection performance.

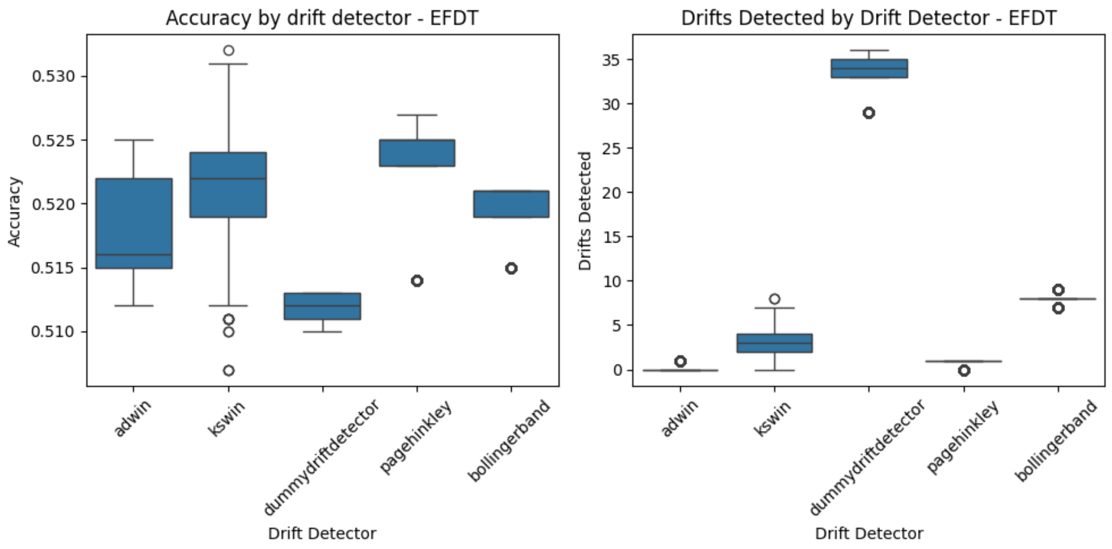

Figure 7: Drift Comparison - EFDT

From a drift perspective, again, the Page-Hinkley detector seems most reasonable as with Hoeffding Tree model. It overall gets the best performance, but there is non-omittable number of tests with KSWIN having higher accuracy scores. PH sensed almost always around 2 drifts and KSWIN between 0 and 10. Dummy Drift Detector detected the most, but it performed poorly.

Below overview of the model under various parameters, grid search included values of `grace_period` in (100, 200, 300), `max_depth` of a tree in (4, 8, 12), and `delta` in (1e-3, 1e-5, 1e-7). Also learning threshold (LT) is shown, category of drift and average number of drifts, 'acc' stands for 'accuracy'. Top 7 and worst 7 combinations are shown concatenated, excluding Dummy Drift Detector as it always performs the worst.

grace_period	max_depth	delta	LT	drift	avg(drifts)	avg(acc)	max(acc)
300	12	1e-05	200	kswin	3.200	0.525	0.532
300	8	1e-05	200	kswin	3.400	0.527	0.531
100	4	1e-07	200	kswin	3.600	0.523	0.531
100	8	0.001	200	kswin	3.400	0.526	0.530
100	4	1e-05	200	kswin	4.800	0.524	0.530
200	8	1e-05	200	kswin	3.000	0.524	0.530
100	12	0.001	200	kswin	2.600	0.524	0.530
300	12	1e-07	2000	adwin	1.000	0.512	0.512
300	4	0.001	2000	adwin	1.000	0.512	0.512
300	4	1e-05	2000	adwin	1.000	0.512	0.512
300	4	1e-07	2000	adwin	1.000	0.512	0.512
300	8	0.001	2000	adwin	1.000	0.512	0.512
300	8	1e-05	2000	adwin	1.000	0.512	0.512
300	8	1e-07	2000	adwin	1.000	0.512	0.512

Table 2: Model Args Comparison Table - EFDT (7 best and worst)

The highest average accuracies for EFDT were obtained using the KSWIN drift detector with a low learning threshold ($LT = 200$), achieving up to 0.527 accuracy. These configurations triggered a moderate number of drifts (∼3–5), suggesting a good balance between sensitivity and performance. In contrast, the worst-performing configurations employed ADWIN with a high threshold ($LT = 2000$), resulting in lower accuracy (0.512) and minimal drift detection (1.0), indicating under-sensitivity to changes in the data stream. Models' parameters not seem to influence performance of the model.

MLP Classifier

Below is a table showing influence of learning threshold (LT) to accuracy.

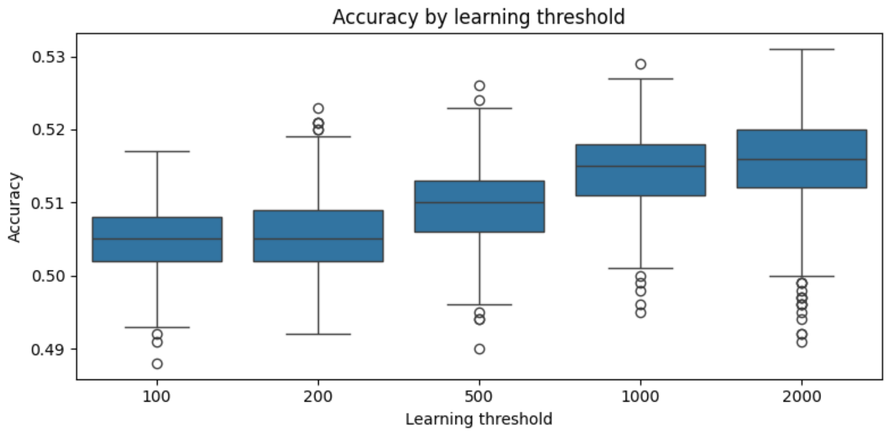

Figure 8: Learning threshold - MLP

As MLP is a non-incremental, batch-based model it's reasonable to trust suggestion indicated from the plot that the bigger the batch is, the better results model gets. Now, let's track drift detection performance.

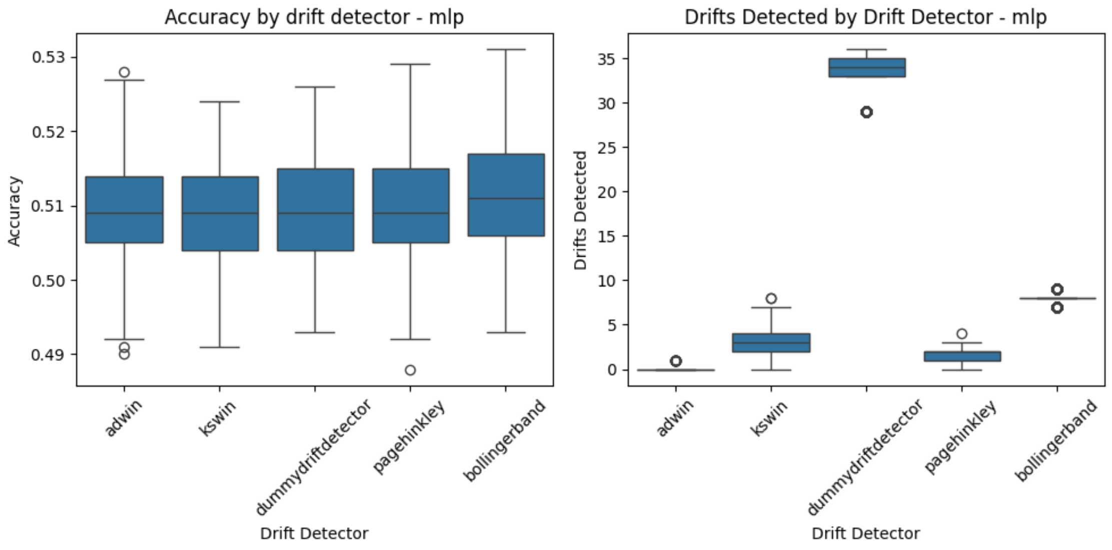

Figure 9: Drift Comparison - MLP

All drift detectors seem to obtain results on a similar level, which is interesting, because number of drifts is significantly different, ranging from 0 to 35 for different detectors.

Below is an overview of the model under various parameters, grid search included values of `hidden_layer_sizes` in (50, 100, 150) neurons and one layer, `learning_rate` (lr) in (0.001, 0.01, 0.1), and `alpha` in (1e-5, 1e-4, 1e-3). Also learning threshold (LT) is shown, category of drift and average number of drifts, 'acc' stands for 'accuracy'. Top 7 and worst 7 combinations are shown concatenated, excluding Dummy Drift Detector as it always performs the worst.

The best-performing MLP configurations achieved average accuracies above 0.52 with relatively low drift counts. Notably, a setup with 100 neurons, learning rate 0.001, and alpha 0.0001 using the Bollinger Band detector at $LT = 2000$ yielded strong performance (avg(acc) = 0.522, avg(drifts) = 7.0). Similarly, a smaller network with 50 neurons and Page-Hinkley (avg(drifts) = 1.2) also performed well. In contrast, configurations using DummyDriftDetector and KSWIN with low learning thresholds ($LT = 100$–200) showed significantly lower accuracy (avg(acc) ≤ 0.502), despite detecting many drifts or none at all. This highlights that both appropriate regularization and

neurons_no	lr	alpha	LT	drift	avg(drifts)	avg(acc)	max(acc)
150	0.01	0.001	2000	bollingerband	7.000	0.516	0.531
100	0.001	0.0001	2000	bollingerband	7.000	0.522	0.530
50	0.001	0.001	1000	pagehinkley	1.200	0.522	0.529
50	0.001	1e-05	2000	bollingerband	7.000	0.521	0.529
100	0.1	1e-05	2000	bollingerband	7.000	0.520	0.529
50	0.1	0.001	2000	pagehinkley	1.800	0.520	0.528
150	0.1	0.0001	2000	adwin	0.000	0.518	0.528
150	0.01	1e-05	200	dummydriftdetector	35.000	0.502	0.505
100	0.01	1e-05	200	adwin	0.000	0.502	0.505
150	0.001	0.0001	200	dummydriftdetector	35.000	0.501	0.505
100	0.01	1e-05	200	dummydriftdetector	35.000	0.501	0.505
150	0.1	0.0001	100	kswin	4.000	0.501	0.505
150	0.01	0.0001	100	kswin	4.000	0.500	0.505
50	0.001	0.001	100	pagehinkley	1.400	0.500	0.504

Table 3: Model Args Comparison Table - MLP (7 best and worst)

correct drift detection are crucial to optimizing MLP performance in streaming scenarios.

XGBoost Classifier

Below is a table showing influence of learning threshold (LT) to accuracy.

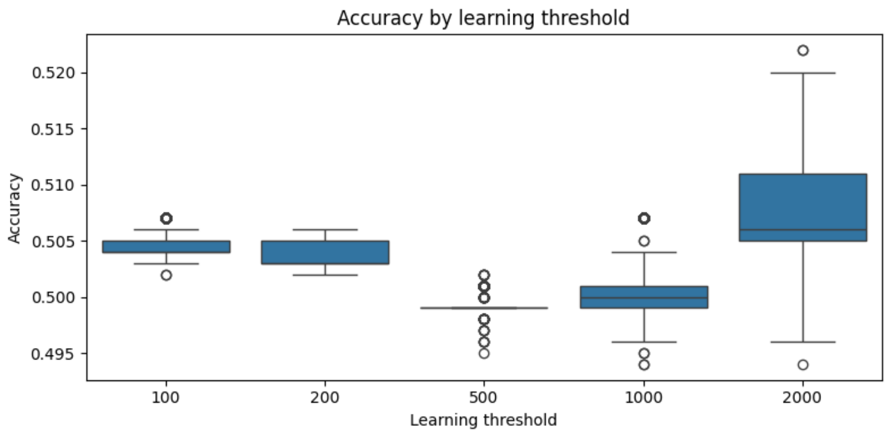

Figure 10: Learning threshold - XGBoost

Learning threshold has significant influence on performance of XGBoost. Firstly, values of 100, 200 do better than 500, and 1000. Also accuracy of tests with LT of 2000 vary, achieving both highest and lowest accuracy scores. Now, let's track drift detection performance.

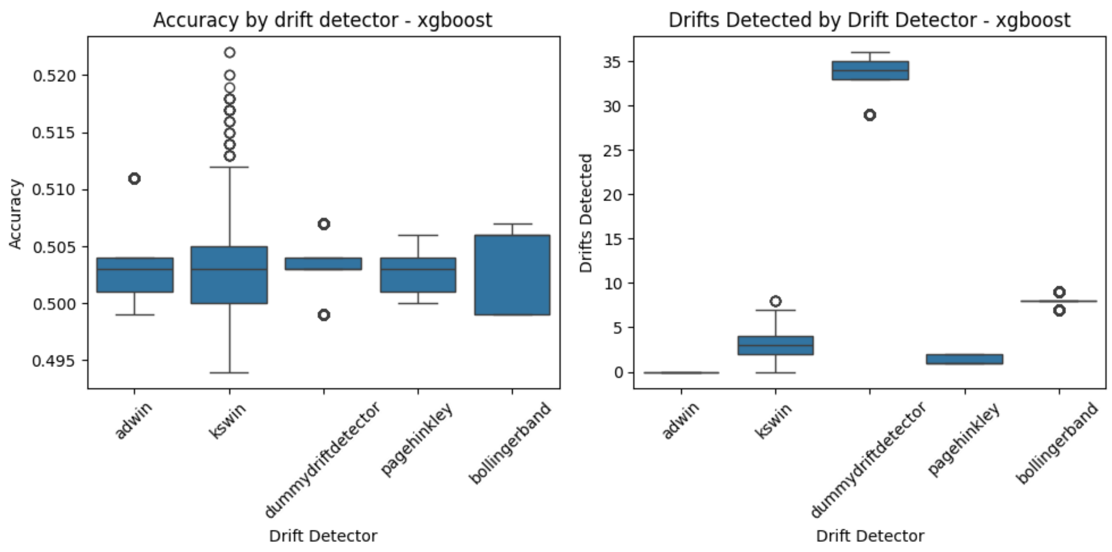

Figure 11: Drift Comparison - XGBoost

All drift detectors have medians at a similar level, Bollinger Band Detector a little bit stronger. Generally they seem to perform alike, except for KSWIN, whose outliers on the plot have best accuracy values.

Below is an overview of the model under various parameters, grid search included values of `n_estimators` in (100, 200, 300), `learning_rate` (lr) in (0.01, 0.1, 0.2), and `max_depth` of the trees in (3, 6, 9). Also learning threshold (LT) is shown, category of drift and average number of drifts, 'acc' stands for 'accuracy'. Top 7 and worst 7 combinations are shown concatenated, excluding Dummy Drift Detector as it always performs the worst.

n_est.	max_depth	lr	LT	drift	avg(drifts)	avg(acc)	max(acc)
200	3	0.1	2000	kswin	4.000	0.509	0.522
300	6	0.1	2000	kswin	3.800	0.509	0.522
100	9	0.1	2000	kswin	2.600	0.516	0.520
100	3	0.1	2000	kswin	2.200	0.514	0.520
100	6	0.2	2000	kswin	2.800	0.511	0.519
200	9	0.2	2000	kswin	2.000	0.514	0.518
200	6	0.2	2000	kswin	3.400	0.512	0.518
300	9	0.2	500	dummydriftdetector	34.000	0.499	0.499
300	9	0.2	1000	bollingerband	8.000	0.499	0.499
300	3	0.1	500	kswin	1.400	0.499	0.499
100	9	0.1	500	kswin	1.600	0.499	0.499
200	3	0.1	500	kswin	1.800	0.498	0.499
300	6	0.2	500	kswin	1.200	0.498	0.499
100	9	0.01	500	kswin	1.200	0.498	0.499

Table 4: Model Args Comparison Table - xgboost (7 best and worst)

The XGBoost model showed solid performance when paired with the KSWIN drift detector and higher learning thresholds (LT = 2000). The top configurations achieved average accuracies above 0.51, with the best at avg(acc) = 0.516 using 100 estimators, a maximum depth of 9, and learning rate 0.1. Notably, drift frequencies remained moderate (2–4 drifts on average), suggesting that KSWIN effectively detected meaningful changes without overreacting. Conversely, models tested with lower LTs (e.g., 500) and different detectors such as DummyDriftDetector or Bollinger Band exhibited significantly lower accuracies (around 0.499) despite triggering up to 34 drifts. These results emphasize that in XGBoost, both learning threshold and the choice of drift detector (particularly KSWIN) are critical for maintaining performance under concept drift scenarios. The XGBoost model demonstrated its strongest performance when using a moderate number of estimators (100–200), rather (6, 9) tree depths, and a learning rate of 0.1 or 0.2. Specifically, the configuration with 100 estimators, depth 9, and learning rate 0.1 achieved the best average accuracy of 0.516 and a relatively low drift frequency of 2.6 on average, suggesting an effective balance between model complexity and stability.

Random Forest Classifier

Below is a table showing influence of learning threshold (LT) to accuracy.

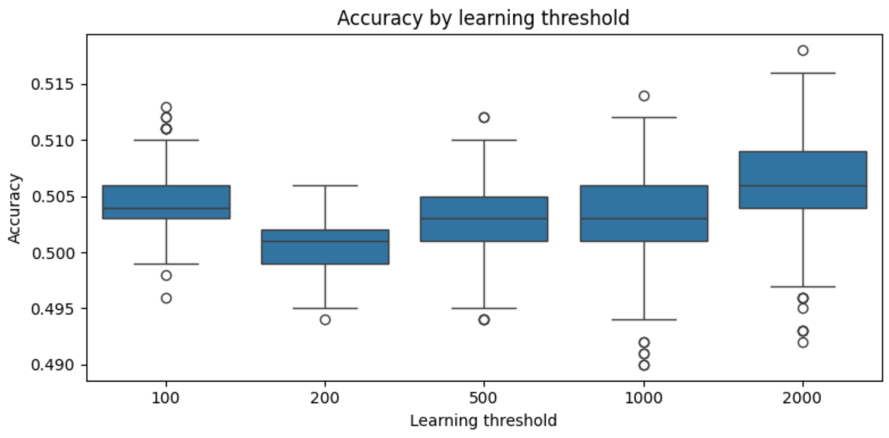

Figure 12: Learning threshold - Random Forest

Learning threshold has significant influence on performance of Random Forest. Values of 100, 500 and 1000 perform similarly, 200 behaves poorly and 2000 the best. Now, let's track drift detection performance.

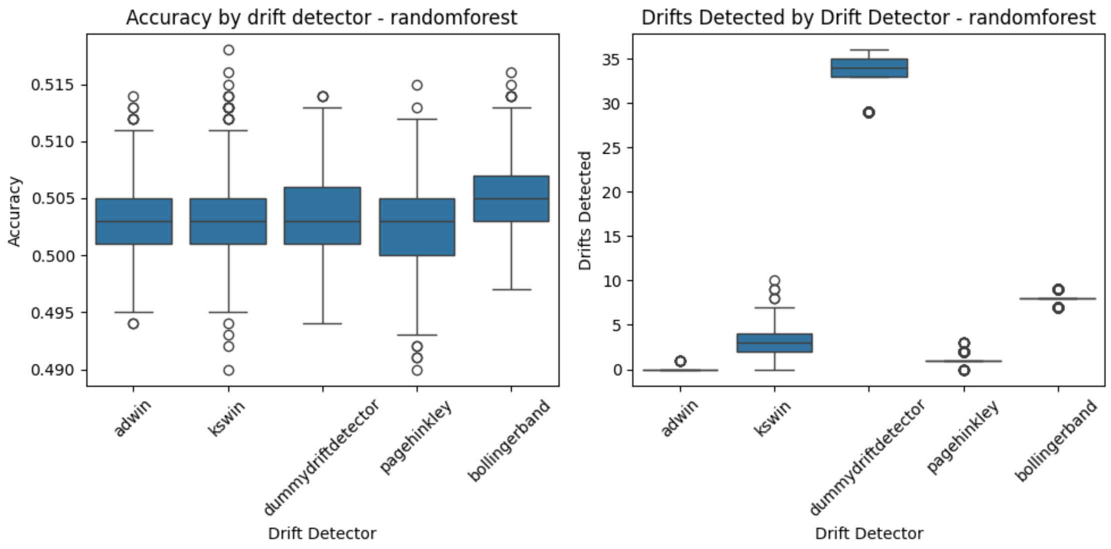

Figure 13: Drift Comparison - Random Forest

All detectors seem to perform similarly, despite different number of detected drifts.

Below is an overview of the model under various parameters, grid search included values of `n_estimators` in (100, 200, 300), `max_depth` of the trees in (None, 10, 20), and `max_features` in ('log2', 'sqrt), and. Also learning threshold (LT) is shown, category of drift and average number of drifts, 'acc' stands for 'accuracy'. Top 7 and worst 7 combinations are shown concatenated, excluding Dummy Drift Detector as it always performs the worst.

n_est.	max_depth	max_feat.	LT	drift	avg(drifts)	avg(acc)	max(acc)
100	20	log2	2000	kswin	3.200	0.507	0.518
100	20	sqrt	2000	bollingerband	7.000	0.512	0.516
300	20	log2	2000	kswin	4.000	0.506	0.516
300	None	sqrt	2000	bollingerband	7.000	0.510	0.515
300	20	sqrt	2000	pagehinkley	1.200	0.508	0.515
200	10	sqrt	2000	kswin	2.600	0.506	0.515
300	10	sqrt	2000	kswin	3.800	0.510	0.514
100	10	sqrt	500	dummydriftdetector	34.000	0.498	0.501
300	20	log2	200	pagehinkley	1.000	0.499	0.500
200	20	sqrt	500	dummydriftdetector	34.000	0.499	0.500
200	20	sqrt	200	adwin	0.000	0.498	0.500
300	None	log2	500	dummydriftdetector	34.000	0.498	0.500
300	None	sqrt	200	pagehinkley	1.200	0.498	0.500
100	20	log2	500	dummydriftdetector	34.000	0.498	0.500

Table 5: Model Args Comparison Table - Random Forest (7 best and worst)

The Random Forest model achieved its best performance using a maximum tree depth of 20 (or unrestricted), n_est. and max_features were irrelevant. The most accurate configuration was with 100 trees, depth 20, and 'sqrt' features, paired with the Bollinger Band drift detector and a high LT of 2000, resulting in an average accuracy of 0.512 and a maximum of 0.516, though at the cost of 7 drift detections. Similar accuracy levels (around 0.510–0.508) were observed with Page-Hinkley and KSWIN, though these detectors typically triggered fewer drifts (1.2–4.0), indicating more stable learning behavior. Increasing the number of estimators beyond 100 did not consistently improve performance, suggesting diminishing returns. Notably, KSWIN paired with 'log2' feature selection and depth 20 also provided solid performance (0.507 avg. acc., 3.2 drifts). On the other hand, configurations with low LT (≤ 500) or using DummyDriftDetector and ADWIN significantly underperformed. These settings triggered excessive or no drifts (e.g., 34 with Dummy or 0 with ADWIN) and consistently produced accuracies below 0.500.

FEATURE SELECTION OVERVIEW

In this chapter, we'll analyse effectiveness of feature selection methods based on performance of models and drift detectors. Let's start with comparing them depending on accuracy by model and feature selector.

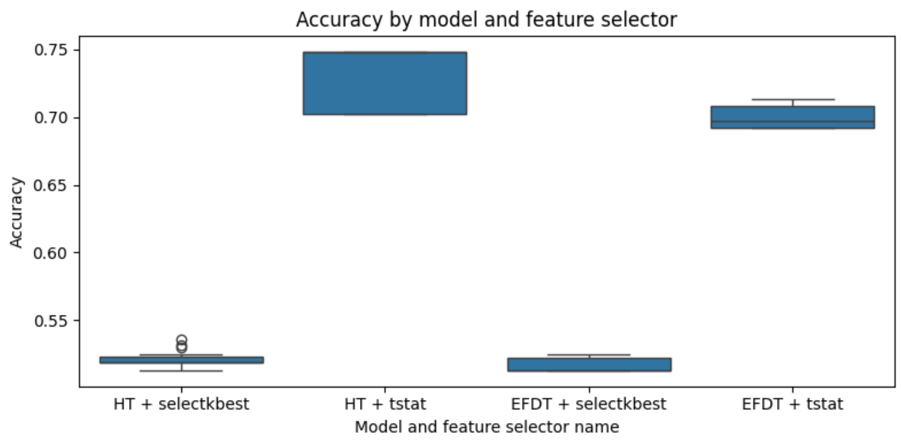

Figure 14: Feature selection comparison by models

The difference of performances between `SelectKBest` and `TStatFeatureSelector` methods is significant. Accuracy of the first one is around 0.51 - 0.53, mean for the second one we have evaluation over 0.7. However, both models perform similarly – we can notice slightly better results for Hoeffding trees.

Now, let's take a look on assessments of feature selectors based on drift detectors.

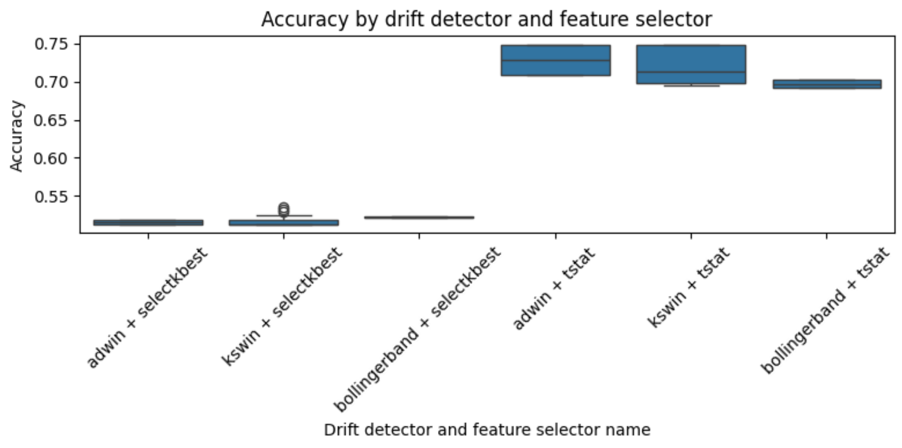

Figure 15: Feature selection comparison by models

Alike on previous graph, we can notice higher accuracies for `TStatFeatureSelector` method. However, this selector seems to perform more unstable, especially looking on boxplots for ADWIN and KSWIN detectors. There's rather no significant different on results among drift detectors when comparing feature selectors.

Ensembles

Ensemble functionality

Two primary approaches to ensembling have been developed for stock prediction tasks, supporting both incremental and batch learning paradigms.

Ensembles for incremental models

These ensembles are designed for online learning scenarios where models update incrementally with each new data instance. The built-in ensembling features of the **river** library were employed, such as:

- BaggingClassifier,
- AdaBoostClassifier,
- ADWINBaggingClassifier,
- SRPClassifier,
- LeveragingBaggingClassifier.

General purpose manual ensembles

This approach is more flexible and can be used with batch learning models (e.g., from scikit-learn) or when a custom combination of incremental models is desired outside of standard **river** ensemble structures. For now, only **majority voting** technique has been implemented in this manner.

Experiment descriptions

A set of experiments has been conducted to check the ensembling efficiency for our classification workflow. The default parameters listed below have been fixed for these experiment to make comparison possible:

- feature_selector – selectkbest
- drift_detector – adwin
- learning_threshold – 1000

Batch ensemble with majority vote

A majority vote ensemble has been evaluated for all supported batch learning models. It turned out that the ensemble accuracy was 0.51 so it stood close to the mean batch model effectiveness.

River ensembles for iterative models

This experiment evaluated various **river** ensemble strategies on all supported iterative classifiers. Two iterations have been run twice for 5 and 10 base models. Increasing the number of base models turned out not to increase the average precision.

There are mean accuracies for all tested ensembles visible on figure 16. Clearly, **Bagging-Classifie** and **ADWINBaggingClassifier** performed better than others with accuracy close to 0.52.

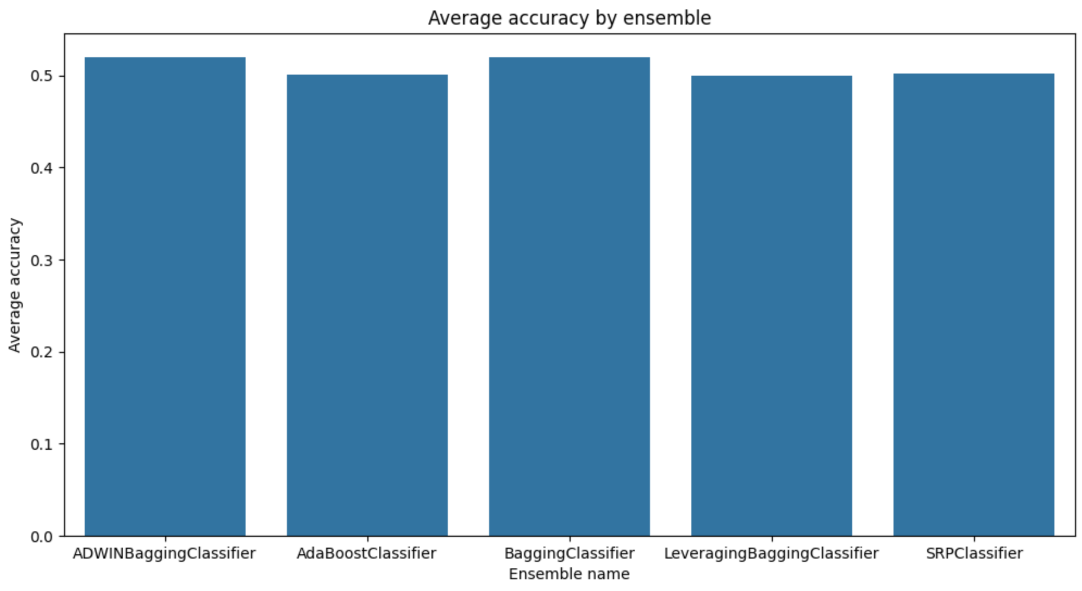

Figure 16: Accuracy by ensemble type

Mean accuracy for each iterative model has been shown on figure 17. There's no significant difference between these two and their accuracy is around 0.52.

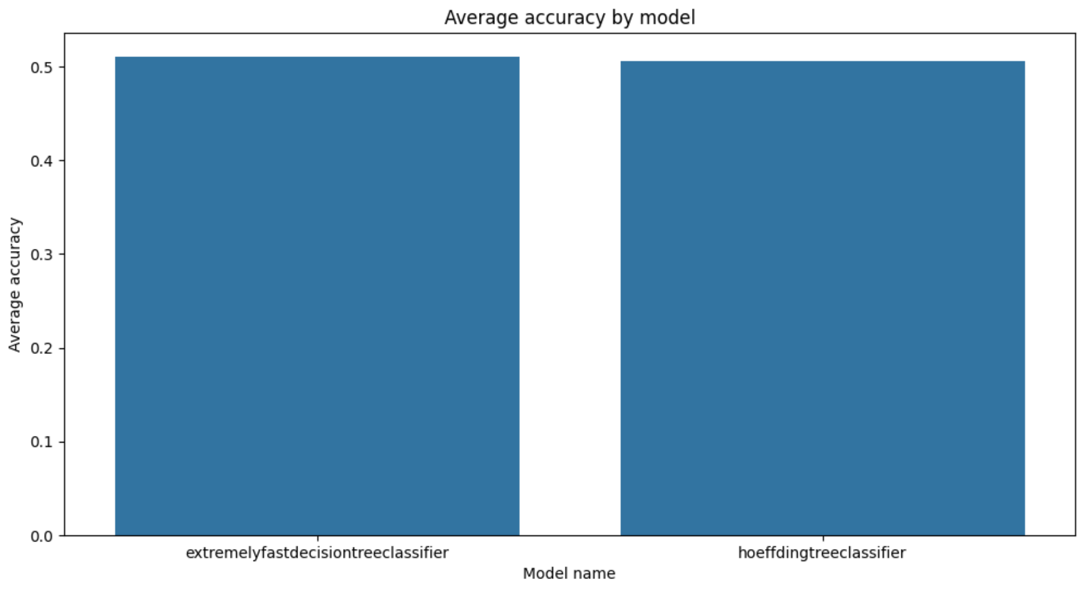

Figure 17: Accuracy by base model type

Although ensembles don't yet seem to perform better than single models, with other parameters adjusted and for different data streams they should exhibit superiority.

CONCLUSIONS FROM THE EXPERIMENT

Let's start with learning threshold. At first glance it might seem controversial to drop from prediction 1000, 2000 samples, which is sequentially 4 and 8 years of stock data. On the other hand both incremental and non-incremental models have the right to be first trained and later on work in online manner. Hoeffding Tree, MLP, XGBoost, and Random Forest worked best with 2000 LT, only EFDT with 200.

Now, speaking about a model, the most stable and with decent models is Hoeffding Tree. However, EFDT reaches similar accuracy scores, but they vary under and below HT ones. Third competitor is MLP, it's capable of predicting nicely, despite average lower performance than previously mentioned. In case of HT grace_period=100 and max_depth=12 should be chosen. For EFDT grace_period=300, max_depth=8 and delta=1e-05. And with MLP parameters (hidden_layer_sizes, learning_rate, alpha, drift_detector) equal to either (100, 0.001, 0.0001, bollingerband) or (50, 0.001, 0.001, pagehinkley). The difference is that second one has significantly less drift detections, meaning it should be more stable and faster.

Let's get now to feature selection. `TStatFeatureSelector` looks promising comparing to SelectKBest method, but it needs to be tested on more combinations of parameters and on other datasets to ensure its effectiveness. However, it behaves more unstably, especially for Hoeffding tree and when comparing with drift detectors.

Last, but not least, drift detection. It should be discussed in accordance with selected models. HT performed best when 0 drifts were detected, which was achievable with ADWIN and PageHinkley (Also no drift detection could be considered). EFDT without discussion worked best with KSWIN. MLP as mentioned in previous paragraph, either Bollinger Bands or PageHinkley.

FURTHER ANALYSIS

Further experiments are supposed to take more datasets into consideration – both real and synthetic. Among these, we want to consider US100 companies, Standardised, OHLC + SMAs, EMAs, stock indicators. Also, we'll consider more tests of feature selectors (especially `TStatFeatureSelector` method) on the current dataset. Moreover, it's necessary to check the effectiveness of these selectors when setting different number of chosen features. We also need to choose final model, drift detector, learning threshold, and finally, best pipeline with all the considered elements.

References

[1] Bartosz Krawczyk, Leandro L. Minku, João Gama, Jerzy Stefanowski, and Michał Woźniak. Ensemble learning for data stream analysis: A survey. *Information Fusion*, 37:132–156, 2017.

[2] G. Raghava Rao I. Bose b P. Ravisankar, V. Ravi. Detection of financial statement fraud and feature selection using data mining techniques. *Information Fusion*, 50:491–500, 2011.