2nd Assignment COMPX523-20A [v4]

Heitor Murilo Gomes, Jacob Montiel, Albert Bifet

May 2020

1 Overall Description

In this assignment, you will implement an ensemble classifier for data streams and rolling window features' extractors. In general, this assignment has two parts, each of them graded as follows:

1. Coding (50% of the final score);
2. Experimentation and Analysis (20% of the final score).

Each of these parts are detailed in the following sections. Important information:

- **Deadline: See Moodle**
- **This assignment can be developed in groups of 1 or 2 students.**
- Parts of the assignments are mandatory for groups of 2 students and optional for single student groups. These are marked as **(Groups of 2 Required)**.
- You must code your assignment using either MOA or scikit-multiflow.
- You need to build a classifier that is compatible with either MOA or scikit-multiflow. In MOA the classifier class will extend the **AbstractClassifier** abstract class, and implements the **MultiClassClassifier** and **CapabilitiesHandler** interfaces.
- You need to build a filter that is compatible with either MOA or scikit-multiflow.
- Your submission must contain a zip file with 3 files:
  1. a file containing the classifier implementation (a single .java or .py);
  2. a pdf report with the experiments and analysis; and

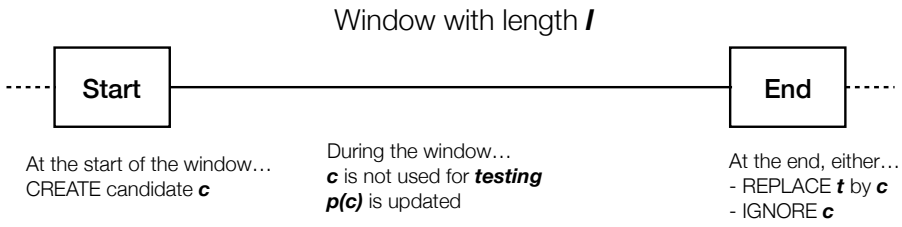

Figure 1: The candidate c updates in a given window

3. a README file with details about how to run your code and reproduce the experiments. Optionally, the README file can be replaced by a jupyter notebook with the execution of the experiments. Notice that a jupyter notebook doesn't replace the PDF report.

Throughout this assignment you will find **(Optional)** questions and requests. These do not give you extra credit nor they influence your final score. They are challenges that may go beyond what was taught in the course, or require further programming and analysis efforts. In summary, you are encouraged to try them, but they are not required.

2 Coding

You must create a new classifier, but you are encouraged to look at the existing ensemble classifiers to observe how ensemble members are usually stored, hyperparameters conventions, and other common code. Notice that your ensemble classifier must be implemented in either MOA or scikit-multiflow in a way that it can be executed as a stand-alone classifier.

The ensemble must contain the implementations to satisfy the requirements described in the following subsections.

2.1 Ensemble Implementation

The ensemble will have a fixed number of learners S. Updates to the ensemble (i.e. additions and removals) will occur every l instances, such that these can be identified as windows of length l instances. At the start of each window, a new base model should be added as a candidate c to join the ensemble. At the end of each window, the ensemble will either: **REPLACE** another learner e currently in the ensemble by c; or **IGNORE** c and discard it. This process is depicted on Figure 1.

The **base learner** for the ensemble will be the **Hoeffding Tree** algorithm. **Optional**: You are encouraged to think about how your ensemble could be implemented using any base learner. The majority of the ensemble algorithms available in MOA/scikit-multiflow do not enforce the use of a specific base learner. However, it would be not easy to accomplish that in this assignment as the diversity induction strategy assumes that we are using a Hoeffding Tree. Another approach is to enforce that the base learner will be a subclass of Hoeffding Tree, such as the Hoeffding Adaptive Tree. **However, if your ensemble is designed to only accept the basic Hoeffding Tree algorithm as the base learner that is fine.**

The **training** and testing **methods** depend on how diversity is induced (see 2.1.4) and the voting method (see 2.1.2). Notice that even though the ensemble is updated in windows of length l, the Hoeffding tree models are continuously updated, i.e. they should not be reset every window and they should be updated during the window as well.

You can name the ensemble algorithm, but if you cannot find a good name, just call it TheEnsemble (we late refer to it using this name).

2.1.1 Predictive performance $p(e)$

The estimated predictive performance $p(e)$ of each member e of the ensemble E must be stored and continuously updated. Concretely, immediately before using an instance x_t for training an ensemble member e, x_t should be used for assessing whether e would correctly predict it, i.e. $h_e(x_t) = y_t$. Notice that if the instance x_t is first used for training e, then the estimated predictive performance would be misleading. By default, $p(e)$ corresponds to the accuracy of e, i.e. number of correct predictions divided by total number of predictions made by e. **(Optional)**: Encapsulate $p(e)$ such that other metrics, such as Kappa T, could be used.

2.1.2 Prediction - Weighted votes

The ensemble prediction should be determined by a weighted majority vote strategy. The weights of each member e are determined according to the predictive performance $p(e)$.

During prediction, the algorithm should sum the weighted votes assigned to each class label to decide the ensemble prediction. For example, given an ensemble with four models, such that $p(g) = 0.90$, $p(m) = 0.85$, $p(n) = 0.81$, and $p(q) = 0.92$, where the predictions for a given x were $h_g(x) = 0$, $h_m(x) = 0$, $h_n(x) = 0$, and $h_q(x) = 1$. The weighted votes for class 0 $w(0) = 0.90 + 0.85 + 0.81 = 2.56$ and for class 1 $w(1) = 0.92$, as a consequence, the ensemble prediction is class 0.

2.1.3 Dynamic updates

After processing l instances, the algorithm should be updated by replacing the worst e^* (min $p(e)$) by the new model c. If $p(c) \leq p(e^*)$, i.e. the predictive performance of c is worst than the minimum predictive performance observed in the ensemble $p(e^*)$, then c should be discarded (i.e. IGNORED).

2.1.4 Training - Inducing diversity

There are multiple ways of inducing diversity into an ensemble model [3]. One can vary in which instances or features the models will be trained, modify the outputs of the training instances, or even combine several strategies. In this assignment, you are asked to implement a strategy that vary the hyperparameters of the base Hoeffding Trees. Concretely, implement a method that create new candidate trees c with random values for the following hyperparameters.

- **Grace period** gp. The number of instances a leaf should observe between split attempts. Range of values (10; 200). Step = 10.
- **Split Confidence** sc. The allowable error in split decision. Range of values (0.0; 1.0). Step = 0.05.
- **Tie Threshold** t. Threshold below which a split will be forced to break ties. Range of values (0.0; 1.0). Step = 0.05.

For example, a given c may be initialized with $gp = 110$, $sc = 0.05$ and $t = 0.55$. There is not much difference between a tree trained using $sc = 0.046$ and $sc = 0.05$, thus we define a step of 0.05 for sc and t, and a step of 10 for gp.

2.1.5 (Groups of 2 Required) Measuring diversity

To verify if a diversity induction strategy was successful, one approach is to estimate how diverse the base models are w.r.t their predictions. Intuitively, if model a always predict the same labels as model b, then they are homogeneous. To estimate 'how' diverse two models on cases other than the extremes (always agree, always disagree), several approaches were defined, such as the Yule's Q statistic or the Kappa statistic.

In this assignment, you are asked to calculate the Kappa statistic $k(a, b)$ for the predictions of each pair of models u and v in the ensemble. Notice that you will need to keep counters for how every pair of models predicted. *Adapted from [3]*: This can be achieved by constructing a contingency table C_{ij}, such that the value at the intersection of a row i and a column j stores the amount of instances $x \in X$ where $h_v(x) = i$ and $h_u(x) = j$. Table 2.1.5 shows an example of contingency table C_{ij} for a k-class problem. The diagonal in matrix C_{ij} contains the concomitant decisions of the pair, thus a naive approach to weight their similarity is to sum its values and divide it by the amount of instances n, as shown in Equation 1.

All possible outputs of a pair of classifiers h_v and h_u for a multiclass classification problem with k possible labels

	$h_u(x) = 0$	$h_u(x) = 1$	...	$h_u(x) = (k-1)$
$h_v(x) = 0$	C_{00}	C_{01}	...	$C_{0(k-1)}$
$h_v(x) = 1$	C_{10}	C_{11}	...	$C_{1(k-1)}$
...	...	...	...	...
$h_v(x) = (k-1)$	$C_{(k-1)0}$	$C_{(k-1)1}$	...	$C_{(k-1)(k-1)}$

$$\Theta_1 = \frac{1}{n}\sum_{i=0}^{k} C_{i,i} \tag{1}$$

The Kappa Cohen's statistic [2] κ statistic measures inter-rater agreement for categorical variables and it was first used in [6] as a pairwise diversity measure for ensemble learners, since it corrects Θ_1 by considering the probability that two classifiers agree by chance according to the observed values in C_{ij}, namely Θ_2 (see Equation 2). The kappa κ statistic is shown in Equation 3.

$$\Theta_2 = \sum_{i=0}^{K}\left(\sum_{j=0}^{K}\frac{C_{i,j}}{n}\cdot\sum_{j=0}^{K}\frac{C_{j,i}}{n}\right) \tag{2}$$

$$k = \frac{\Theta_1 - \Theta_2}{1 - \Theta_2} \tag{3}$$

The interpretation of κ is that if h_u and h_v agree on every instance then $\kappa = 1$, and if their predictions coincide by chance, then $\kappa = 0$. Negative values of κ occurs when agreement is weaker than expected by chance, but this rarely occurs in real problems.

Be cautious when calculating $k(c, *)$, i.e. the Kappa statistic between the candidate c and each of the other learners during the window for which c was introduced. Specifically, take extra care when you REPLACE an existing e by c, remember to overwrite the respective counters.

(Optional) Implement the same method for measuring diversity in Leveraging Bagging [1] and Adaptive Random Forest [4] to estimate their diversity as well. This will allow for a more detailed analysis of the results in the Evaluation part.

2.2 (Groups of 2 Required) Rolling window features

Implement a filter that is independent of any learner, capable of enriching a feature set by adding novel features that represent statistics of the raw data calculated over a rolling window. The time unit is defined in terms of the number of instances, which is easier to define and operate on. The Simple Moving Average (SMA) and the Exponential Moving Average (EMA) should be implemented.

SMA. The Simple Moving Average (SMA) represents the average value calculated over a sliding window of size w as shown in equation 4. For example, assuming one feature f_0, the SMA of the last 10 values for f_0 and are shown in Table 1.

$$SMA(x, w, T) = \begin{cases} x_0, & T = 1 \\ SMA(x, w, w - T), 1 < T < w & \\ \frac{1}{w} \sum_{i=T-w}^{T} x_i, & T > w \end{cases} \tag{4}$$

where T is the current instance index, w the window size and x_i holds the value of feature x. The feature identifier was omitted, otherwise it would look something like x_i^j, i.e. feature j at time i.

Observation: The first values on the stream should be calculated according to the number of instances available. For example, the SMA of $w = 5$ when there are only 3 values available should be calculated using the 3 values, i.e. assuming $w = 3$. This is just for completeness and to avoid doubts during implementation.

EMA. The Exponential Moving Average (EMA) is a moving average that assigns a greater weight according to α on the most recent data as shown in equation 5. Table 1 presents an example of EMA for feature f_0.

$$EMA(x, T, \alpha) = \begin{cases} x_0, & T = 1 \\ \alpha x_{T-1} + (1 - \alpha) EMA(x_{T-1}, \alpha), & T > 1 \end{cases} \tag{5}$$

where the coefficient α is the degree of weighting decrease (constant between 0 and 1). A higher α discounts older values faster. By default, let's use $\alpha = 0.3$. There are several ways of initializing $EMA(x, w, 1, \alpha)$, we are setting it to x_0, even though other approaches are also viable, such as using the SMA of the first values.

In MOA, you can extend **AbstractStreamFilter.java** and base your implementation on the other classes that implement AbstractStreamFilter. The basic idea, is that the filter to create the EMA and SMA versions of the attributes will be applied to a stream, which in turn will them be used for other problems. Include options to define which features should have SMA or EMA created and also to control their hyperparameters w and α.

Important: Notice how on table 1 the EMA and SMA do not include the current values of the feature f_0, the last f_0 used in any calculation is always the value immediately before the current instance.

3 Evaluation and Analysis

The results should be presented using a table format, followed by a short analysis about them guided by the questions associated with each experiment. More details are presented in the subsections for each experiment. At the end of this

f_0	$SMA(f_0, 10)$	$EMA(f_0, \alpha = 0.4)$
26	26	26
20	26	26
34	23	23.6
24	26.67	27.76
17	26	26.26
17	24.2	22.56
35	23	20.33
15	24.71	26.20
13	23.5	21.72
19	22.33	18.23
37	22	18.54
28	23.1	25.92
27	23.9	26.75

Table 1: Calculating SMA and EMA for 13 values. Red cells indicate exceptions to the general equation, e.g. calculating SMA with when $T < w$

section you will find more details about how to prepare, run and report your experiments using either scikit-multiflow or MOA.

Evaluation framework. For all experiments, use a test-then-train evaluation approach and report the final result obtained, i.e. the result obtained after testing using all instances.

Metrics. For all experiments, report **accuracy**, **time**[1].

Datasets. electricity, covertype, SEA_abrupt (drift at 50000), SEA_gradual (drift at 50000, width=10000), and RTG_2abrupt (drift at 30000 and 60000). Refer to attached materials to obtain the ".arff" and ".csv" versions.

3.1 Experiment 1: TheEnsemble variations and sanity check

In the first part of the evaluation, you are asked to perform experiments using TheEnsemble and a single Hoeffding Tree. This experiment will produce 3 Tables of results (algorithms (columns) vs datasets (rows)), where $S = K$ represents the number of learners, $seed$ is the initializer to the random object[2], and l is the length of the window. Important: you don't need to report the time taken for this experiment (it would require yet another take).

- First table ($l = 1000$ for all TheEnsemble variations): HoeffdingTree, TheEnsemble($S = 5$), TheEnsemble($S = 10$), TheEnsemble($S = 20$), and TheEnsemble($S = 30$);
- Second table ($S = 20$ and $l = 1000$ for all): TheEnsemble($seed = 1$), TheEnsemble($seed = 2$), TheEnsemble($seed = 3$), TheEnsemble($seed =$

[1]In MOA, time can be estimated by the **CPU Time**

[2]The actual value of the random seed doesn't matter much, as long as it is not the same for every experiment.

4), TheEnsemble($seed = 5$)

- Third table ($S = 20$ for all): TheEnsemble($l = 500$), TheEnsemble($l = 1000$), TheEnsemble($l = 2000$), TheEnsemble($l = 5000$), TheEnsemble($l = 10000$)

Questions:

1. How does TheEnsemble compares against a single hoeffding tree? Also, does TheEnsemble improve results if more base models are available? Present a table with results varying from $S = \{5, 10, 20, 30\}$, where S is the total number of learners.

2. **(Groups of 2 Required)** What is the impact of the randomization strategy on the diversity of the base learners? 1st repeat each experiment 5 times varying the random seed of the classifier and present a table with the average and standard deviation of each result. 2nd Present a plot, for each dataset, depicting the average Kappa statistic over time to support your claim.

3. Is the ensemble able to recover from concept drifts? Present a plot depicting the accuracy over time to support your claim. There should be a plot for each dataset comparing whichever version of TheEnsemble that produced the best results on the first table against the single hoeffding tree.

4. What is the impact of the l hyperparameter? Discuss the results varying the l hyperparameter.

5. **(Optional)** Which combination of l and S presents the best results for each dataset (on average). This will require another separate table where l and S are combined together.

Important: To plot results, use a windowed evaluation. In MOA, EvaluatePrequential with its default Evaluator should be sufficient. The code from the following repository can be used for the plots, but notice that any other tool or language can be used.
`https://github.com/hmgomes/data-stream-visualization`

3.2 Experiment 2: TheEnsemble vs. others

Compare the best result of TheEnsemble, obtained in the previous experiment, against Adaptive Random Forest (ARF) [4], Leveraging Bagging (LB) [1] and the Dynamic Weighted Majority (DWM) [5] for each dataset. Use $S = 20$ for all the ensembles and subspace $m = 60\%$ for ARF. This will produce two tables with results:

- First table: The accuracy for each ensemble and dataset.

- Second table: The processing time for each ensemble and dataset.

Questions:

1. How does TheEnsemble performs in terms of predictive performance against the others?
2. In terms of time, is TheEnsemble more efficient than the other methods?
3. **(Optional)** Create another table varying the number of learners in the other datasets as well. Then discuss which ensemble method scales better in terms of accuracy, i.e. produces better results as we increase the number of learners for the given datasets.
4. **(Optional)** What design choice made affect the computational resources the most for TheEnsemble? For example, one design choice is how the algorithm update learners, how diversity is induced, etc. Notice that the value of l or S is not a design choice, it is an hyperparameter configuration.

3.3 (Groups of 2 Required) Experiment 3: Feature engineering

For the electricity dataset, present (at the least) 3 transformations of the stream to include EMA and SMA. Produce the results using the transformed datasets using a single Hoeffding Tree, TheEnsemble, ARF, LB, and DWM. Use the same configurations as in Experiment 2, for the ensembles. This will produce one table where columns are the ensembles and the rows are the versions of the dataset.

Observation: Notice that the past values of the class labels are available for transformation and the creation of a feature that uses the previous class labels is possible. This is similar to the **TemporallyAugmentedClassifier** that we saw in class.

Questions:

1. Is it possible to improve the results without changing the ensemble configurations? In other words, just by transforming the data?
2. Which transformations were the most effective? In other words, are there any specific feature that when transformed lead to better results?

References

[1] A. Bifet, G. Holmes, and B. Pfahringer. Leveraging bagging for evolving data streams. In *PKDD*, pages 135–150, 2010.

[2] J. Cohen et al. A coefficient of agreement for nominal scales. *Educational and psychological measurement*, 20(1):37–46, 1960.

[3] H. M. Gomes, J. P. Barddal, F. Enembreck, and A. Bifet. A survey on ensemble learning for data stream classification. *ACM Computing Surveys (CSUR)*, 50(2):1–36, 2017.

[4] H. M. Gomes, A. Bifet, J. Read, J. P. Barddal, F. Enembreck, B. Pfharinger, G. Holmes, and T. Abdessalem. Adaptive random forests for evolving data stream classification. *Machine Learning*, 106(9-10):1469–1495, 2017.

[5] J. Z. Kolter and M. A. Maloof. Dynamic weighted majority: An ensemble method for drifting concepts. *Journal of Machine Learning Research*, 8(Dec):2755–2790, 2007.

[6] D. D. Margineantu and T. G. Dietterich. Pruning adaptive boosting. In *ICML*, volume 97, pages 211–218. Citeseer, 1997.