CPF: Concept Profiling Framework for recurring drifts in data streams

Robert Anderson, Yun Sing Koh, and Gillian Dobbie

Department of Computer Science, University of Auckland, New Zealand
rand079@aucklanduni.ac.nz, {ykoh,gill}@cs.auckland.ac.nz

Abstract. We propose the Concept Profiling Framework (CPF), a meta-learner that uses a concept drift detector and a collection of classification models to perform effective classification on data streams with recurrent concept drifts, through relating models by similarity of their classifying behaviour. We introduce a memory-efficient version of our framework and show that it can operate faster and with less memory than a naïve implementation while achieving similar accuracy. We compare this memory-efficient version of CPF to a state-of-the-art meta-learner made to handle recurrent drift and show that we can regularly achieve improved classification accuracy along with runtime and memory use. We provide results from testing on synthetic and real-world datasets to prove CPF's value in classifying data streams with recurrent concepts.

Key words: Concept drift, recurrent concepts, data streams

1 Introduction

Classifying potentially infinite data streams requires a trade-off between accuracy of our predictions and speed and memory use of our approach. This trade-off is made more difficult by streams potentially changing over time *i.e.* the underlying distribution of data changing (*concept drift*). Classification of streams with drifting concepts benefits from drift handling techniques *e.g.* as described in [1]. For example, if we detect a significant change in the underlying distribution, we can build a new model to learn anew from it, avoiding bias learnt from previous distributions. However, we may lose all we have learnt up to that point.

Over time an underlying data-generating distribution may revert to a previously seen state. Through recognising and understanding these recurring concepts we can often perform better classification of incoming data in a stream by reverting to previously trained models [2]. Repeated patterns are regularly observed in the real world: seasons; boom-bust periods in financial markets; rush-hour in traffic flows; and battery states of sensors, for example. Where these exist, classification can be faster and more accurate by recognising and accounting for them. Identifying repeated patterns improves our understanding of our modelled problem. However, many state-of-the-art techniques use demanding statistical tests or ensemble approaches to identify and handle recurring concepts. These escalate memory use and increase runtime. In situations that rely on speed or

low memory use, this overhead means these techniques cannot always be applied, and so recurring concepts will not be accounted for in the classification. Through this work, we address this problem by proposing a classification framework that utilises recurrent concepts while mininimising runtime and memory usage overhead compared to current state-of-the-art techniques.

We present the Concept Profiling Framework (CPF). This is a meta-learning approach that maintains a collection of classifiers and uses a drift detector. When our drift detector indicates a drift state *i.e.* that our current classifier is no longer suitable, we check our collection of classifiers for one better suited to the current stream. If one meets a set level of accuracy, we will select it as the current classifier; otherwise a new classifier is produced and trained on recent data. If this new classifier behaves similarly to a classifier in our collection (using a measure derived from *conceptual equivalence* [3]), we will choose that existing classifier as our current model (*i.e.* model *reuse*); otherwise we will add the new classifier to the collection and use that as our current classifier.

We introduce two techniques to allow efficient handling of recurrent concepts. First, we regularly compare behaviour of our classifiers, and over time, our certainty about their similarity will improve. If they behave similarly, we can use the older model to *represent* the newer one. Second, we implement a *fading* mechanism to constrain the number of models, a points-based system that retains models that are recent or frequently used. Through observing reuse patterns, we can understand how patterns recur in our stream.

Our contribution is a meta-learning framework that can: utilise observed model behaviour over time to accurately recognise recurring concepts, without relying on additional information about underlying concepts such as in [4]; and regularly outperform a state-of-the-art learning framework, the Recurring Concept Drifts framework (RCD [5]), in terms of accuracy, memory and runtime.

In the next section, we discuss related work that informed the creation of CPF. We then detail our proposed framework. We provide experimental results to: show that using our fading mechanism provides similar quality to a naïve implementation of our algorithm whilst improving memory and runtime; show our approach provides generally better accuracy than RCD, requiring less memory and runtime on eight synthetic datasets; and show CPF keeps this time and memory efficiency while matching RCD in accuracy on five commonly used real-world datasets. Next, we discuss the results and consider our technique in greater detail. Finally, we conclude by summarising our findings and suggesting sensible next steps for developing CPF.

2   Related work

In this section, we discuss the previous work that has informed our development of CPF. Gama et al. [6] provide a fantastic overview of the problems faced while learning in drifting streams and solutions that have been proposed. Since data streams arrive incrementally, models such as Very Fast Decision Trees (VFDTs) [7] have been created to be built incrementally. They achieve a constant time

and memory relationship with the number of instances seen, and are guaranteed to achieve performance similar to a conventional learner. This is through limiting the number of examples required to be seen at any node through utilising the Hoeffding bound [8], which describes the number of instances needed to be representative of an overall dataset within a set probability.

Drift-detection mechanisms try to detect underlying concept drift, so a learning framework can take a corrective action. DDM (Drift Detection Mechanism) [9] monitors the error rate of a classifier. When the mean error rate rises above thresholds based on the minimum error rate seen, it signals a warning or drift. EDDM (Early DDM) [10] introduces two changes to DDM. First, the mean distance between errors is measured instead of error rate. Second, it changes thresholds used for detecting warnings and drifts. Two user-set parameters, α and β decide its sensitivity. If the mean distance drops $\beta \times 2s.d.$ below the minimum mean distance seen, the detector signals a warning, and below $\alpha \times 2s.d.$, it signals a drift, where $1 > \beta > \alpha$. EDDM is made to detect drift more quickly than DDM, with less evidence required.

The approach in [11] handles recurring concepts by building a referee' model alongside every instance a classifier sees, and keeps a collection of these pairs. The referee model judges whether its classifier is likely to correctly classify an instance by learning whether it was correct for previous similar instances. When existing models aren't applicable, a new model is created. Two models are trained at once at any time, and no suggestions are made for constraining the total number of models built over time.

Gomes et al. [4] propose using an additional user-specified context stream alongside a data stream. Their approach relates the current context to the current classifier when it is performing well. Particular contexts become associated to classifiers that are useful in those contexts. After drift, a model is reused if the context learner feels it fits the current context. Otherwise, a new classifier is created. They use *conceptual equivalence* [3], which relates similar models through their classifying behaviour. This technique requires an additional context stream which is difficult to select for a problem that is not well understood.

Gonçalves and De Barros [5] propose RCD, a framework that maintains a classifier collection. For each classifier, a set of recent training instances are stored to represent its concept. When a drift is detected, a statistical test (k-NN) compares the instances in the warning buffer to the training instances for classifiers. If the instances are found to be significantly similar, it uses the corresponding model; otherwise, a new one is created. The statistical testing and buffer stores add significant runtime and memory requirements to their approach.

Ensemble approaches to recurrent learning use multiple classifiers examining a stream, allowing greater chance to find one that functions well at a given time. However, this gain in accuracy often comes with greater memory and runtime overhead compared to single classifier techniques as shown in [12].

3   Concept Profiling Framework (CPF)

In this section, we describe how our proposed technique functions. Our main goal is a learning framework that can recognise recurring concepts through model behaviour and use this to fit existing, pre-trained models rather than new models.
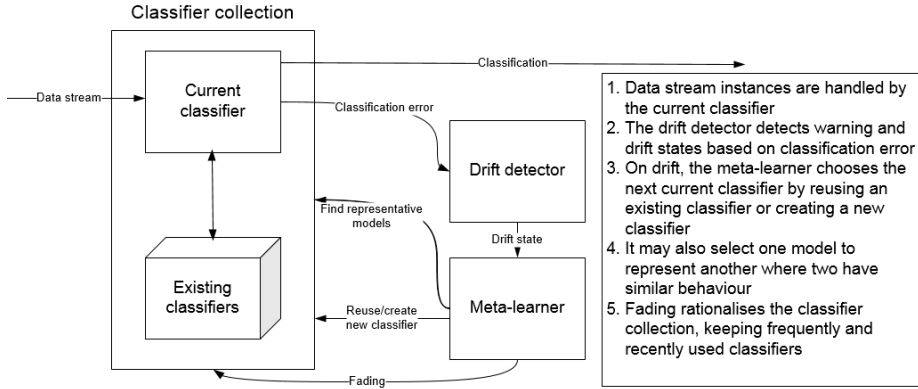

Fig. 1. The Concept Profiling Framework

We use a meta-learning framework (Figure 1) with a collection of one or more incremental classifiers. One is designated as our current classifier. A drift detector signals warning and drift states. On a warning state, the meta-learner will stop training the current classifier and store instances from the data stream in a buffer. If a drift state follows, the meta-learner looks for an existing model in the collection that classifies the warning buffer accurately to use as the current classifier. If it cannot find one, it will create a new model trained on even buffer instances. When an existing model behaves similarly to this new model (when tested on odd buffer instances) that model will be reused; otherwise the new model is trained on odd buffer instances and used. Every model in the collection is tested on the buffer, and the results will be compared and stored. Where it is found that models classify similarly to one another, the older model will *represent* the newer one.

Through regular reuse and representation of particular models (as described below), we hope for particular classifiers to model particular concepts very well over time. In addition, frequency of reuse and representation can show patterns of recurrence in the underlying data. CPF can pair with any classifier that is incremental and can perform in a streaming environment. We use Hoeffding Trees using Naïve Bayes - a version of a CVFDT [13] which creates a Naïve Bayes model at the leaves if it provides better accuracy than using the majority class. This is consistent with the implementation of RCD that the authors suggest in [5]. Our experiments use EDDM to be consistent with RCD. Technically, our

technique can work using a drift detector that has no warning zone, but the buffer will be of a set minimum size rather than informed by the drift detector.

3.1   Model similarity

Our approach uses a similarity measure based upon *conceptual equivalence* used in [4] for comparing models. We adapt their approach and do pairwise comparisons of models' respective errors when classifying given instances. When comparing classifier c_a and classifier c_b, we calculate a score per instance, where $c_a(x)$ and $c_b(x)$ is the classification error for c_a and c_b on a given instance x:

$$Score(x, c_a, c_b) = \begin{cases} 1 & \text{if } c_a(x) = c_b(x) \\ 0 & \text{if } c_a(x) \neq c_b(x) \end{cases}$$

We then calculate similarity for two models over a range of instances, using instances X seen during warning drift periods:

$$Sim(X_1...X_n, c_a, c_b) = \frac{\Sigma(Score(X, c_a, c_b))}{n}$$

If $Sim(X, c_a, c_b) \geq m$ ($m \leq 1$ and is a set similarity margin threshold), we describe the models as similar and likely to represent the same concept. We require a minimum of thirty common instances seen by two classifiers before we measure their similarity. Since new classifiers only train on even instances, we collect at least sixty instances. Our score function provides a Binomial distribution, and the Central Limit Theorem indicates that as we see thirty examples and beyond, this will approximate a Normal distribution. This gives assurance of finding a representative mean $Score$ between two classifiers.

Classifier (c_n)	Errors on warning buffer instances (X) $x_1\ x_2\ x_3\ x_4\ x_5\ x_6\ x_7\ x_8\ x_9\ x_{10}$	Accuracy on buffer	Sim(X, c_1, c_n)	Sim(X, c_{new}, c_n)
Classifier 1 (c_1)	0 1 0 0 0 1 0 0 1 1	6 / 10	10 / 10	4 / 5
Classifier 2 (c_2)	1 1 0 0 1 1 0 1 1 1	3 / 10	7 / 10	2 / 5
Classifier 3 (c_3)	0 0 0 0 0 1 0 0 0 0	9 / 10	7 / 10	5 / 5
New classifier (c_{new}) *T = training instances	0 T 0 T 0 T 0 T 0 T	5 / 5	4 / 5	5 / 5

Fig. 2. Example of calculating model similarity for classifiers $c_1...c_{new}$ on warning buffer X

Every time a drift occurs, all existing classifiers will classify instances in the warning buffer. The results of each classifier will be compared as a bitwise

comparison to see if they both had the same classification error. The similarity matrix stores pairwise comparisons between classifiers, through recording instances seen (or n) and total score (or $\Sigma(Score(X, c_a, c_b))$). We then check for similar models. Figure 2 shows an example of four classifiers being compared using a given warning buffer, with 0 representing a correct classification and 1 an incorrect classification by the classifiers. Of ten instances (X) in our buffer, we can see which our classifiers correctly or incorrectly classify, which is the behaviour we measure to find their Sim. If $m = 0.95$, we can see that $Sim(X, c_{new}, c_3) = 5/5 \geq m$), so could treat our models as similar. In practice, we require instances in the buffer for that decision.

3.2   Reuse and representation

When a drift occurs, CPF will check if any existing classifiers are suitable to reuse for the following data. This is by a two-step check. First, by checking from the oldest classifier, if any classifier achieves $\geq m$ accuracy (our similarity margin threshold), the model will be selected as the classifier to use. Otherwise, a new classifier will be trained on even instances in the buffer, then prequential test-and-trained on odd instances. If an old model has similarity $\geq m$ when tested on odd instances in the buffer, we will *reuse* the old model, as it is similar to the new model we would introduce. Our experiments suggest setting m as 0.95 for consistent performance across a range of datasets. This level should help prevent CPF from incorrectly assuming model similarity without fair evidence *i.e.* at least 95% similarity in behaviour as per classification errors. Lower values of m allow model reuse and representation with less evidence of similarity.

When two classifiers behave similarly, we assume that both classifiers describe the same concept. The older classifier is generally more valuable to keep, as it has often been trained on more instances and been compared to other classifiers in our collection more frequently. The older classifier *represents* the newer classifier as follows: if it was the current classifier, the older model becomes the current classifier; the older model receives the newer model's fading points (described below); and the newer model is removed from our collection. The older and newer model behave similarly, so we lose little by retaining only the older model.

3.3   Fading

Data streams can be of infinite length, and over time, the classifier collection in our framework may continually grow, risking an ever-expanding performance overhead. To avoid this, we use a fading mechanism to constrain the size of our classifier collection. Our fading mechanism prefers recent and frequently-used classifiers, and penalises others. At any stage, we maintain an array of fade points F. Fade points of a given classifier c_a, F_{c_a} can be expressed as follows:

$$F_{c_a} = \begin{cases} 0 & \text{if represented by older model} \\ (r+1) \times f + \sum(F_{m_{c_a}}) - (d-r) & \text{otherwise} \end{cases}$$

Here, r is drift points that a model is reused, f is a user-set parameter for points to gain on creation and reuse, d is the number of drifts the classifier has existed for (excluding the drift at which it is created) and $\sum(F_{m_{c_a}})$ is the sum of points for any models represented by c_a. Every drift in which a model is not reused, it loses a point, but when it is reused it gains f points. When newer models are represented by an older model, the older model inherits the newer model's points. When $F_{c_a} = 0$, the model is deleted. The user can control the size and number of models by selecting f. Removing models through fading gives our technique less opportunity to identify similarity with previous models, so the user can lose some information about recurrent behaviour through this step.

In our experiments, we set f to be 15. This is based on RCD's maximum of 15 models. This will constrain the total number of models to around that number unless models are regularly reused. Where there is a zero reuse and representation rate, we cannot have more than 15 models, and only have more models when we have reused models that may represent recurrent concepts.

3.4   Model management in practice

Figure 3 illustrates how our technique maintains its collection of classifiers. For the sake of simplicity, we have set $f = 3$ and $m = 0.95$.
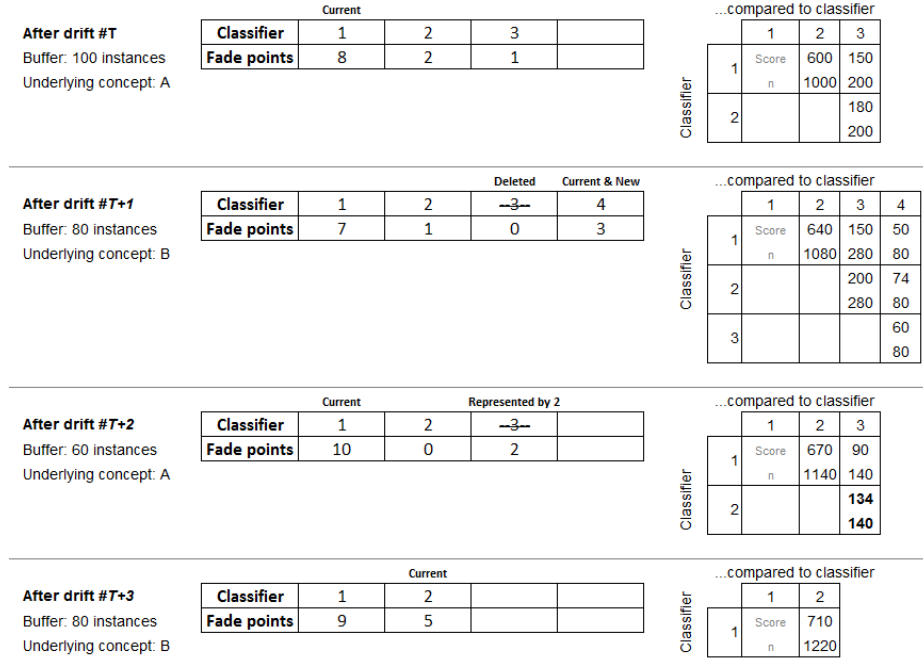

Fig. 3. Example to illustrate reuse, representation and fading

1. After drift point T in a stream, in which our warning buffer had 100 instances, we have three models. Classifier c_1 was reused for concept A, and has just gained f points, while c_2 and c_3 lost a point each. Pairwise comparisons in the similarity matrix have been updated with the *Score* from 100 new instances seen in the warning buffer.
2. After drift $T+1$, no existing classifiers match the buffer or the new classifier, so a new classifier becomes c_4 and gains f points, while the others lose a point each. Classifier c_3 now has zero points and is deleted, so $c_4 \to c_3$.
3. After drift $T+2$, concept A recurs, and c_1 matches the buffer through accuracy so gains f points and becomes the current classifier. The other two models lose a point each. Classifier c_2 would be deleted, but now $Sim(X, c_2, c_3) \geq m$ so c_3 is deleted and c_2 represents c_3, so gets its points.
4. After drift $T+3$, concept B recurs, and c_2 matches the buffer through accuracy so gains f points while c_1 loses a point.

4   Experimental Design and Results

To test CPF against RCD, we used the MOA API (available from `http://moa.cms.waikato.ac.nz`). We used the authors' version of RCD from `https://sites.google.com/site/moaextensions`, keeping default parameters (including limiting total models to a maximum of 15 at any given time). Experiments were run on an Intel Core i5-4670 3.40 GHz Windows 7 system with 8 GB of RAM. RCD and CPF maintain a buffer of instances if a detector is in a drift state. This can cause variations in memory requirements over a small space of instances so we excluded this buffer in our memory measurements for both. CPF was run with $f = 15$, $m = 0.95$ and a minimum buffer size of 60.

4.1   Datasets

We used eight synthetic datasets for testing. Each had 400 drift points with abrupt drifts and 10000 instances between each drift point. Concepts recurred in a set order. Data stream generators used are available in MOA apart from CIRCLES, which is described in [9]; we used a centre point of (0.5, 0.5) and a radius $\in \{0.2, 0.25, 0.3, 0.35, 0.4\}$ to represent different concepts for this. We repeated experiments 30 times, and varied the random number seed. We present the mean result and a 95% confidence interval; where these overlap, we have no statistical evidence of a difference between results. EDDM for RCD and CPF was run with its suggested settings of $\beta = 0.95$ and $\alpha = 0.9$ on these datasets. Settings used to generate our datasets are available online `https://www.cs.auckland.ac.nz/research/groups/kmg/randerson.html`.

Our five real-world datasets are commonly used to test data stream analysis techniques. Electricity, Airlines and Poker Hand are available from `http://moa.cms.waikato.ac.nz/datasets`; Network Intrusion is available from `http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html`; and Social Media [14] is the Twitter dataset from `https://archive.ics.uci.edu/ml/datasets/`

`Buzz+in+social+media+`. EDDM for RCD and CPF was run with settings of $\beta = 0.9$ and $\alpha = 0.85$ on these datasets. Real-world datasets tend to be shorter and noisier; making EDDM less sensitive avoids overly reactive drift detection.

4.2 CPF fading mechanism

For this experiment, we compared CPF with the fading mechanism and CPF without. We compared accuracy, memory use, runtime, mean models stored and mean maximum models stored in the classifier collection over the synthetic datasets. As per Table 1, our fading mechanism caused slight losses in accuracy for two datasets and a slight gain in another. Runtime and memory use were never significantly worse for the fading approach and were significantly better in most cases. Mean models and mean max models for the experiments show that the fading mechanism significantly constrains the number of models produced. This provides evidence that the fading mechanism works as an effective constraint for models considered by CPF and loses little accuracy.

Table 1. Effect of fading mechanism on CPF

Dataset	Approach	Mean accuracy	95% CI	Mean memory	95% CI	Mean time	95% CI	Mean models	95% CI	Max models	95% CI
Agrawal	CPF	78.18%	0.08%	**9.91E+05**	**6.45E+04**	**2.60E+04**	**1.77E+02**	16.94	0.12	21.83	0.23
	CPF - no fade	**80.00%**	**0.77%**	2.70E+08	1.67E+07	1.54E+06	1.14E+05	1183.04	51.30	2153.47	95.67
CIRCLES	CPF	96.39%	0.17%	**9.42E+05**	**3.25E+04**	**5.15E+03**	**6.38E+01**	9.75	0.45	18.93	0.82
	CPF - no fade	**97.18%**	**0.12%**	2.05E+06	1.35E+05	7.43E+03	3.10E+02	44.52	2.55	55.50	3.90
Hyperplane	CPF	**93.61%**	**0.02%**	**5.15E+05**	**2.82E+04**	**2.13E+04**	**8.36E+01**	17.04	0.14	24.37	0.35
	CPF - no fade	92.90%	0.03%	1.25E+08	1.47E+06	5.74E+05	1.10E+04	813.68	6.41	1500.77	9.16
LED	CPF	69.10%	0.31%	**5.90E+05**	**1.01E+05**	**6.92E+04**	**2.05E+03**	16.80	0.09	18.40	0.20
	CPF - no fade	68.95%	0.43%	6.19E+07	4.61E+06	1.15E+06	9.35E+04	491.48	28.94	971.43	57.29
RandomRBF	CPF	80.32%	0.90%	**1.32E+06**	**8.01E+05**	**2.99E+04**	**1.32E+03**	16.21	0.18	21.60	0.48
	CPF - no fade	81.13%	1.08%	1.64E+08	2.62E+07	9.43E+05	1.55E+05	797.00	106.02	1477.77	197.04
SEA	CPF	84.62%	0.14%	**1.76E+06**	**2.66E+05**	**9.02E+03**	**5.87E+02**	13.20	0.42	20.10	0.71
	CPF - no fade	84.46%	0.13%	5.06E+06	1.28E+06	1.31E+04	2.78E+03	39.87	5.78	60.87	8.85
STAGGER	CPF	99.33%	0.00%	4.30E+05	1.67E+03	3.05E+03	4.22E+01	4.05	0.07	4.83	0.31
	CPF - no fade	99.33%	0.00%	4.32E+05	3.62E+03	2.98E+03	2.55E+01	4.27	0.23	4.87	0.35
Waveform	CPF	78.15%	0.40%	**1.15E+06**	**3.81E+05**	**1.00E+05**	**4.49E+03**	16.47	0.41	26.47	0.61
	CPF - no fade	78.36%	0.07%	9.92E+07	4.48E+06	1.17E+06	4.39E+04	303.87	8.46	565.87	14.72

4.3 CPF similarity margin

We tested CPF with different values of m on synthetic and real datasets and ranked accuracy by m as per Table 2. For synthetic datasets, this was mean accuracy across 30 trials. We compared how often a setting for m did worse than most others *i.e.* was ranked 4th or 5th, as we wanted a setting that would perform consistently over a variety of datasets.

We chose $m = 0.95$ for our experiments, as it received few bad rankings for either synthetic or real datasets. Higher settings for m generally worked well on synthetic datasets except for Agrawal and RandomRBF. Choosing $m = 0.975$ would be justifiable based on real-world datasets. These had no guaranteed
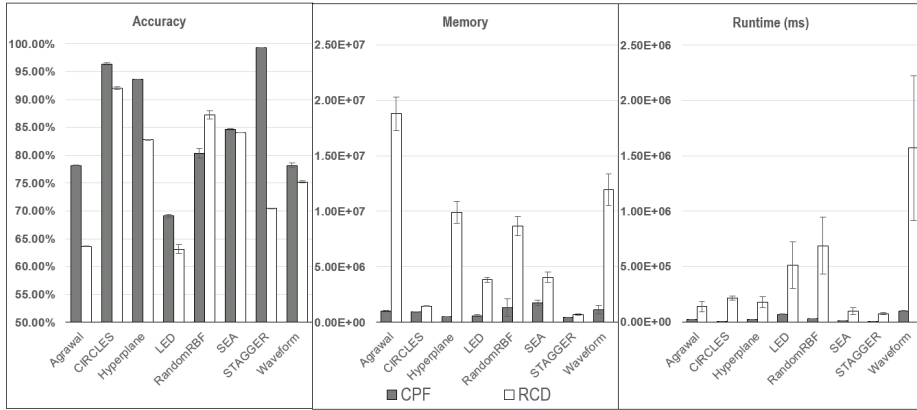
Table 2. Comparison of CPF accuracy across different levels of m

Similarity margin (m)	Rank on synthetic datasets (lower is better)									Rank on real datasets					
	Agrawal	CIRCLES	Hyperplane	LED	RandomRBF	SEA	STAGGER	Waveform	Freq rank > 3	Electricity	Poker	Intrusion	Airlines	Social Media	Freq rank > 3
0.85	2	5	5	5	1	5	5	5	6	5	5	4	5	5	5
0.90	1	4	4	4	2	4	4	3	5	3	4	1	1	4	2
0.95	3	1	1	3	3	3	3	4	1	4	1	3	3	3	1
0.975	4	2	2	1.5	4	2	2	1	2	2	2	2	3	2	0
0.99	5	3	3	1.5	5	1	1	2	2	1	3	5	3	1	1

recurrence, so a higher setting for m could reduce spurious reuse of inappropriate models, avoiding resulting drops in accuracy.

We also tested CPF with differing minimum buffer sizes (30, 60, 120, 180, 240). For these approaches, we compared accuracy, memory use, runtime and drifts detected over the synthetic datasets. Our results showed some variation in accuracy, memory and runtime based upon buffer size but showed no clear discernable pattern across datasets.

4.4   Comparison with RCD on synthetic datasets

Fig. 4. Comparison of CPF and RCD on synthetic datasets

For this experiment, we compared CPF's accuracy, memory and runtime against RCD's on eight synthetic datasets, as per Figure 4. CPF consistently outperformed RCD in terms of memory usage and runtime. RCD stores a set

of instances for each model, increasing memory usage, and runs k-NN to compare these instances to the warning buffer which increases runtime. CPF was always much faster, and reliably required less memory than RCD. CPF significantly outperformed RCD's accuracy on all datasets except for RandomRBF. RandomRBF creates complex problems through assigning classes to randomly placed overlapping centroids: k-NN is well-suited for identifying the current concept in this complex situation, while CPF relies on similarity between models' behaviour which may be too inexact to identify the current concept.

4.5   Comparison with RCD on real-world datasets

For this experiment, we compared CPF's performance against RCD's on our five real-world datasets in terms of accuracy, memory and runtime. As per Figure 5, CPF consistently used less memory and was faster than RCD, taking less than 10% of the time of RCD for all datasets except for Airlines. In terms of accuracy, CPF performed similarly with Airlines and Intrusion, better with Poker and Social Media and worse with Electricity than RCD. CPF's accuracy is closer to RCD's than seen in our tests with synthetic data. These real datasets are shorter, often tend to be noisier and may not genuinely feature recurring concepts. This is likely to impact the techniques' relative performance.
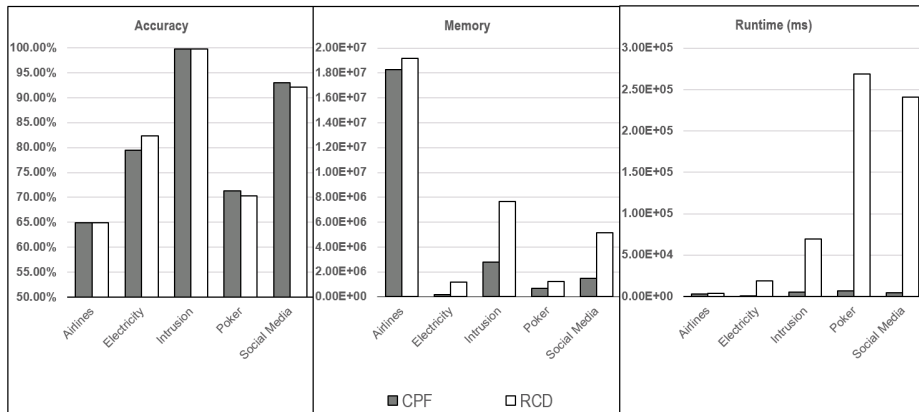

Fig. 5. Comparison of CPF and RCD on real-world datasets

5   Conclusion and further work

We have proposed CPF as a fast, memory-efficient, accurate framework for analysing streams with recurring concept drift. Through tracking similarity of model behaviour over the life of our stream, we can regularly reuse suitable

models that have already been trained when a drift occurs. This avoids the need to train from scratch, and provides valuable insight on the recurrent patterns in drift for our data stream. Through our experiments, we have shown that our proposed approach is often more accurate than a state-of-the-art framework, RCD, while consistently requiring less memory and running much faster than this framework, on both synthetic and real-world datasets.

Future work to develop this technique could include: developing our similarity measure so that it considers classification and not just error in multi-class problems; finding ways to prioritise model-comparisons as part of our model similarity approach; investigating ways of combining or merging models where they are similar; and utilising our similarity measure/representation approach to simplify and speed up ensemble approaches to recurring drifts.

References

1. Tsymbal, A.: The problem of concept drift: definitions and related work. Computer Science Department, Trinity College Dublin **106** (2004)
2. Widmer, G., Kubat, M.: Learning in the presence of concept drift and hidden contexts. Machine learning **23** (1996) 69–101
3. Yang, Y., Wu, X., Zhu, X.: Mining in anticipation for concept change: Proactive-reactive prediction in data streams. Data mining and knowledge discovery **13** (2006) 261–289
4. Gomes, J.B., Sousa, P.A., Menasalvas, E.: Tracking recurrent concepts using context. Intelligent Data Analysis **16** (2012) 803–825
5. Gonçalves Jr, P.M., De Barros, R.S.M.: RCD: A recurring concept drift framework. Pattern Recognition Letters **34** (2013) 1018–1025
6. Gama, J.a., Žliobaitė, I., Bifet, A., Pechenizkiy, M., Bouchachia, A.: A survey on concept drift adaptation. ACM Comput. Surv. **46** (2014) 44:1–44:37
7. Domingos, P., Hulten, G.: Mining high-speed data streams. In: Proceedings of the sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM (2000) 71–80
8. Hoeffding, W.: Probability inequalities for sums of bounded random variables. Journal of the American statistical association **58** (1963) 13–30
9. Gama, J., Medas, P., Castillo, G., Rodrigues, P.: Learning with drift detection. In: Advances in artificial intelligence–SBIA 2004. Springer (2004) 286–295
10. Baena-Garcıa, M., del Campo-Ávila, J., Fidalgo, R., Bifet, A., Gavalda, R., Morales-Bueno, R.: Early drift detection method. In: Fourth international workshop on knowledge discovery from data streams. Volume 6. (2006) 77–86
11. Gama, J., Kosina, P.: Tracking recurring concepts with meta-learners. In: Progress in Artificial Intelligence. Springer (2009) 423–434
12. Brzezinski, D., Stefanowski, J.: Reacting to different types of concept drift: The accuracy updated ensemble algorithm. Neural Networks and Learning Systems, IEEE Transactions on **25** (2014) 81–94
13. Hulten, G., Spencer, L., Domingos, P.: Mining time-changing data streams. In: Proceedings of the seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM (2001) 97–106
14. Kawala, F., Douzal-Chouakria, A., Gaussier, E., Dimert, E.: Prédictions d'activité dans les réseaux sociaux en ligne. In: 4ième conférence sur les modèles et l'analyse des réseaux: Approches mathématiques et informatiques. (2013) 16–28