A News Recommender Engine with a Killer Sequence

Pieter Bons[1], Nick Evans[1], Peter Kampstra[1], and Timo van Kessel[1]

[1] ORTEC - Houtsingel 5, 2719 EA Zoetermeer, The Netherlands
pieter.bons@ortec.com, nick.evans@ortec.com,
peter.kampstra@ortec.com, timo.vankessel@ortec.com

Abstract. Our submission to the CLEF NewsREEL 2107 News Recommendation Evaluation Lab attempts to apply the concept of storytelling to the participating news domains. The goal is to guide the user through a series of related articles where each transition from one article to the next provides an opportunity to steer the storyline in a certain direction using recommendations. The approach employs collaborative filtering to discover an optimal sequence of articles – a killer sequence. The choices that were made by users reading two or more successive articles were stored in the graph database Neo4J. The recommendations were generated by querying this database for the most popular historic sequences containing the concerning article. For articles that were not yet part of any sequence we generated the recommendations from a dynamic set of the most recently changed publications for that domain. The performance of our combined algorithm was approximately 79% better for Click Through Rate (CTR) than the competition baseline. We investigated whether this top performance was due to unwanted behavior of the recommender, such as only answering on certain domains or times, but could not conclude that this was the case.

Keywords: Recommender Systems, News Recommender Engine, Graph Database, Sequence, Evaluation, Collaborative filtering, Real-time recommendations.

1 Introduction

The holy grail for (online) publishers is reader engagement. The more engaging the user's experience, the more likely that they will come back, increasing their loyalty to the domain during each visit until they may even become brand ambassadors. Once the consumer has crossed the barrier to enter the domain, the goal is to keep him there as long as possible. One of the important strategies to achieve this is storytelling. On the content side this requires rich and interactive media to retain the reader's attention. This can be reinforced by artificial intelligence and automation tools providing relevant personalized content to individual readers.

Many types of algorithms could be considered for this job. However, traditional news recommendation algorithms rarely consider the time sequence characteristic of user browsing behaviors. Also, using strategies from other domains such as movies or music offers no solution as in these cases the order in which items are consumed is

hardly relevant, while this is crucial for news articles which are much less independent from each other. Text mining may be employed to classify the articles into topics or to calculate similarities but this approach is also symmetric under the order in which articles are read.

To tackle this subtle issue, we have chosen to go with a collaborative filtering approach where the wisdom of the crowds will provide the information in which order the articles are best presented. The collective behavior of the users will uncover dominant sequences of articles that together form a powerful story. This sequence can then be suggested to new users via the recommendations causing them to stay more engaged and consume more content from the domain.

There is some related work which uses the timestamps of user-item interaction events [11, 13]. We have not based our work specifically on these papers but followed a pragmatic approach based on our experience in the online news industry.

2 Approach

The Living Lab Evaluation [2] was performed using the open recommendation platform (ORP) provided by plista. By redirecting some of the live traffic, the ORP allows participants to test algorithms in a real environment. There are four types of events that are sent from ORP: Recommendation requests, Impressions, Item Updates, and Error Messages (such as timeout).

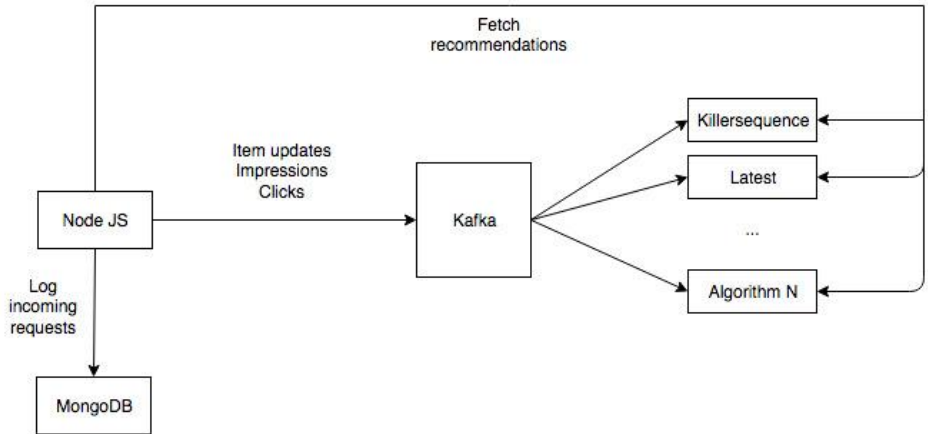

Fig. 1. Data processing infrastructure

Our infrastructure choices were dictated by two main requirements: 1) the system should answer recommendation requests as fast as possible, and 2) the system should be extensible and support many different algorithms running in parallel. In order to satisfy both these requirements it was necessary to decouple the processing of recommendation requests from the processing of impressions, clicks and item updates. We settled on Apache Kafka [1] to implement this decoupling. As can be seen in

Errore. L'origine riferimento non è stata trovata. all requests are first processed by a Node.js [9] server. All requests are stored in MongoDB [3] for tracking and debugging. If the request is a recommendation request, the recommendations are gathered and returned. If the request is an impression, a click or an item update, the request is published to a Kafka topic. The recommendation algorithms can subscribe to this topic and process the necessary information as it becomes available. This decoupling ensures that the time to answer a recommendation request is not affected by the number or nature of the recommendation algorithms. We found that a single Node.js process was enough to adequately handle all the recommendation requests (note that Node.js processes are always single threaded).

The setup we used for this contest relied only on two recommendation algorithms: most recent (the baseline) and killer sequence. Recommendations were taken from killer sequence if available and otherwise from most recent. In the future, once more recommendation algorithms have been implemented, we would like to experiment with a meta-algorithm for selecting the algorithm from which recommendations should be used for a particular request. Therefore, our infrastructure already takes multiple algorithms into account.

2.1 Killer sequence algorithm

For each new item update event an article node is generated in the Neo4J graph database [10]. A flag is also set to indicate whether the article is available for recommendations. This flag can be altered by later item updates with the same id.

Based on the impression events we keep track of user sessions. When an impression event is received, a new session is created containing the user ID and the article ID in the REDIS [4] database with an expiration time of 60 seconds. When a second impression with the same id is handled within these 60 seconds, the session is updated with the new article ID and a relation is set in Neo4J between the first and the second article nodes signifying that these two articles were read in this specific order by the user. The sequence can grow to as many articles as the user consumes as long as there is less than 60 seconds between the events so the session does not expire.

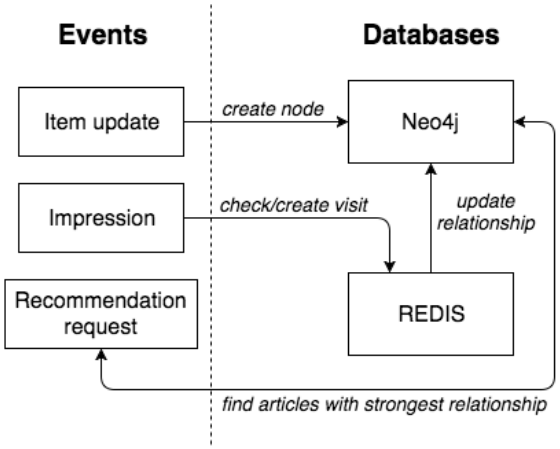

Fig. 2. A schematic overview of the event handling for the killer sequence algorithm.

The replies to the recommendation requests are generated by querying the Neo4J graph database for the strongest sequence containing the concerning article. The top X results are returned and the ID's are sent to the ORP. We employ a breadth-first search of only one layer deep because it could be undesirable to skip an article in the sequence that forms a news story. To make the recommendation more personalized, we would like to incorporate the historical sequence of a user into the search but in the current implementation this information is no longer available since the we store the sum of all interactions and not the individual actions.

When the graph query does not return any recommendations because the article is not yet part of a sequence, a fallback recommendation is used randomly taken from a dynamic set of the most recently changed articles per domain, as implemented in the example code provided for the challenge.

The expiration time of 60 seconds was chosen arbitrarily and we expect that optimizing it will improve the performance of the algorithm.

3 Results and analysis

By participating in the CLEF NewsREEL 2107 Task 1 we were able to test our algorithm in a real-life environment and compare our algorithm to other algorithms. Originally, multiple test periods were planned and a single evaluation period was available at the end. However, due to technical issues on both sides, given by corporate environments, we started straight into the final evaluation period.

In Table 1 we show the overall results of CLEF NewsREEL 2107 Task 1. Our algorithm is ORLY_KS and is shown in bold, while the baseline algorithm is shown as BL2Beat. Our algorithm performs 79% better than the baseline, and scores the highest CTR of all algorithms with more than 1000 widget impressions. Overall, the CTRs are higher than during CLEF NewsREEL 2016, where the highest CTR reported was 1.23% [6]. One recommendation widget could contain multiple items, with the num-

bers of items not always being the same. Only one recommendation item per page could be clicked for a refresh to trigger, hence the CTR for items is always lower than for widgets. Because the item/widget ratio for our recommender is the lowest of all recommenders, the per item CTR is even slightly higher being 82% better than the baseline. However, the amount of widgets impressions for our recommender is quite a bit lower than the baseline and some other algorithms.

From previous years of CLEF NewsREEL (e.g. [8]) it is known that CTR ratios can differ hugely between different times and different domains. Therefore, answering a selective number of recommendation requests or having a different sampling of the available requests could make the CTR ratios incomparable. We will therefore focus our analysis on finding out if we inadvertently achieved a high CTR by answering selectively.

Table 1. Overall results of algorithms in CLEF NewsREEL 2107 Task 1 sorted by CTR.

Name	Clicks	Widgets shown	CTR	Items shown	ctrItem	Items / Widget	% CTR baseline
Riadi_NV_01	12	443	2.709%	1380	0.8696%	3.1151	232%
ORLY_KS	**896**	**42786**	**2.094%**	**130221**	**0.6881%**	**3.0435**	**179%**
ody4	1139	72601	1.569%	230896	0.4933%	3.1803	134%
IRS5	58	3708	1.564%	11856	0.4892%	3.1974	134%
ody5	1268	81245	1.561%	255663	0.4960%	3.1468	133%
ody3	813	59227	1.373%	184052	0.4417%	3.1076	117%
ody2	875	63950	1.368%	199547	0.4385%	3.1204	117%
IT5	925	68582	1.349%	214922	0.4304%	3.1338	115%
Eins	817	61524	1.328%	191647	0.4263%	3.1150	114%
yl-2	747	60814	1.228%	192207	0.3886%	3.1606	105%
WIRG	600	49830	1.204%	154419	0.3886%	3.0989	103%
ody1	810	68768	1.178%	214406	0.3778%	3.1178	101%
BL2Beat	**726**	**62052**	**1.170%**	**193014**	**0.3761%**	**3.1105**	**100%**
RIADI_pn	879	77723	1.131%	244334	0.3598%	3.1437	97%
IL	813	79120	1.028%	249492	0.3259%	3.1533	88%
RIADI_nehyb	764	75535	1.011%	236322	0.3233%	3.1286	86%
Has logs	6	816	0.735%	2610	0.2299%	3.1985	63%
ody0	166	23023	0.721%	72599	0.2287%	3.1533	62%
RIADI_hyb	2	349	0.573%	1146	0.1745%	3.2837	49%

3.1 Response time analysis

One of the properties of a good real-time news recommendation system is having a fast response time, preferably within 100 milliseconds. Also, since the ORP might drop recommendations after 100 ms, we decided to log the performance of our response times. The results are shown on a density plot in Fig. 3. In 99.9% of the cases

we responded within 90 milliseconds, and on average we responded in 7 milliseconds. Given that our recommender was located at the same hosting provider (Hetzner Online GmbH), 10 milliseconds is more than enough for transfer times. Therefore it is very unlikely that response time problems played a role in having less widget recommendations.

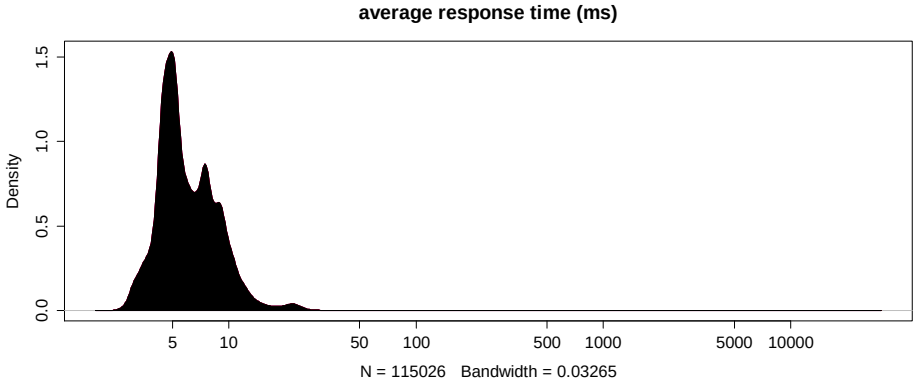

Fig. 3. Density plot of response times for our recommender during evaluation on a log-scale. Data gathering starts during the contest at 2017-04-28.

3.2 Weekly results and error reports

ORP provided us with an API to query the weekly results during the contest. The results are shown in Table 2. While our error ratios (in bold) for both weeks are higher than most recommenders, other recommenders like RIADI_nehyb have even higher error ratios but are able to produce more impressions.

The most likely explanation for the connection problems was that our corporate policy asked for a firewall of the TCP port in case an insecure connection was used. As the connection was insecure, the port was firewalled and whitelisted only certain IP addresses. However, in week 1 certain IP addresses were not whitelisted, while in the second week servers were added by plista, which were whitelisted only after a small delay.

In Table 3 we show the error notifications received by our server. The amount of content errors received exactly matches with 1415 instances. Probably, these content errors were caused by a deploy on 2017-06-28. Strangely, connection errors also seem to match somewhat, while it is usually impossible for a server to communicate such an error in case of a firewall blocking the connection. The number of timeouts errors is negligible.

Overall, the number of errors is still quite low and does not prevent other recommenders from achieving high numbers of impressions. Also, the errors are not only related to recommendation requests, but also to other requests. We therefore assume that the impact from these errors was quite low.

Table 2. Weekly results of algorithms in CLEF NewsREEL 2107 Task 1 sorted by overall CTR. Note that counts for both weeks are almost, but not precisely matched with overall results.

Name	Evaluation week 1					Evaluation week 2				
	clicks	impressions	CTR	Connect error	Content error	clicks	impressions	CTR	Connect error	Content error
Riadi_NV_01						12	443	2.709%	7204	4576
ORLY_KS	**367**	**18930**	**1.939%**	**3836**	**1415**	**529**	**23931**	**2.211%**	**3355**	**0**
ody4	571	36917	1.547%	583	0	560	35404	1.582%	465	0
IRS5	58	3708	1.564%	902	31					
ody5	608	41515	1.465%	630	0	655	39449	1.660%	994	0
ody3	392	29983	1.307%	1113	0	417	29035	1.436%	531	0
ody2	457	32751	1.395%	624	0	416	30995	1.342%	489	0
IT5	438	33766	1.297%	498	12	482	34534	1.396%	466	8
Eins	405	32207	1.257%	867	0	411	29024	1.416%	109	0
yl-2	401	32572	1.231%	331	84	343	27891	1.230%	63	84
WIRG	316	27031	1.169%	4001	55	284	22882	1.241%	2595	1
ody1	413	35382	1.167%	590	0	397	33141	1.198%	499	0
BL2Beat	**322**	**29535**	**1.090%**	**1740**	**402**	**405**	**32586**	**1.243%**	**548**	**0**
RIADI_pn	409	37488	1.091%	334	13	470	40184	1.170%	93	0
IL	388	40345	0.962%	752	8	422	38526	1.095%	765	7
RIADI_nehyb	326	36414	0.895%	24847	578	437	38858	1.125%	969	46
Has logs						6	816	0.735%	12	0
ody0	103	12982	0.793%	17815	0	61	9828	0.621%	0	0
RIADI_hyb	2	349	0.573%	82787	0					

Table 3. Error codes received. Unfortunately, these can also occur outside of recommendation requests.

Error code	Constant	Count
455	ERRCODE_CONNECTION_FAILED	6658
442	ERRCODE_FORMAT_INVALID	1415
408	ERRCODE_CONNECTION_TIMEOUT	76
Total		8149

3.3 Returning the wrong (number of) recommendations

A possible cause for not having the lowest number of recommendation items per widget is that our recommender simply does not return enough recommendations. As we stored almost all recommendations our recommender did in MongoDB, it was possible to investigate these. We also record which algorithm did the recommendation. In Table 4 the results of this analysis are shown. It is clearly visible that something went wrong during a deploy with a re-implementation of the baseline recommender (Most recent) in Redis. The idea of this re-implementation was to be able to restart the recommender without losing state, however it contained an off-by-one error. Therefore, it usually returned a recommendation more than requested.

We however expect that returning one recommendation too many is not a huge problem, as ORP usually requests more recommendations than it needs anyway. Most likely the extra recommendation was simply ignored.

It is also shown that our killer sequence recommender more frequently gives less recommendation than requested. In some cases it handed out 0 recommendations which was fixed later on and probably explains the difference between outgoing and incoming recommendations. In case both recommenders cannot obtain a result, no result was returned. In case the killer sequence recommender had fewer than requested recommendations, we did not add results from the baseline recommender. This might explain why our algorithm had the lowest item/widget ratio.

Table 4. Number of recommendations versus recommendation limit for different algorithms. Returning 7 recommendations is the result of an off-by-one bug.

# rec's	0	1	2	3	4	5	6	7	Total	%
K. Sequence	526	2358	1384	1140	1114	929	66061		73512	42.93%
Most Recent		24	85	30	12	23	35141	62415	97730	57.07%
All Outgoing	526	2382	1469	1170	1126	952	101202	62415	171242	100.00%
All Incoming				8		8	172144		172160	

In case an article is received from ORP it also contains a flag indication whether this article is recommendable. Articles with a flag other than 0 are not considered recommendable, however the baseline implementation ignores this flag when recommending, in both the Java and Node.js versions of the example code supplied by the organization. Also it does not check whether it recommends the same article multiple times.

In Table 5 we show an analysis of whether a recommendation contains an unrecommendable article or a duplicated recommendation. Especially the baseline recommender almost always does this. This is no surprise, as it does not check the commendable flag and returns from a set of recently updated articles. Out of 698340 received article updates, 460743 (66%) contain an non-recommendable article. If selecting 6 or 7 from these updates the probability of getting a non-recommendable article is quite high. Also, it is likely that the same article is updated multiple times within a short period, resulting in duplicate recommendations in 22% of the cases.

While our killer sequence never returned duplicates, it also failed to check the status flag which we did store in Neo4J. Therefore, it did return non-recommendable articles in 23% of the cases.

All in all there is a huge room for improvement in returning valid recommendation results. However, given that the baseline recommender makes the same mistakes a lot of the time, we do not believe this is the cause for having less widget impressions.

Table 5. Different situations for different sub-algorithms, percentages are for total recommendations given by the sub-algorithm. A recommendation is considered ok if it has only recommendations for articles with status=0 and no duplicates. Ok <6 means that there are less than 6 recommendations returned, while Ok >6 means there are more than 6 recommendations returned. Given that almost all requests were for 6 recommendations, Ok 6 with 6 returned recommendations, and no invalid or duplicate articles is probably preferable.

	With status != 0	With duplicates	Ok <6	Ok 6	Ok >6
KillerSequence	16954 (23%)	0 (0%)	4055 (6%)	52503 (71%)	0 (0%)
Most recent	96812 (99%)	21102 (22%)	6 (0%)	122 (0%)	85 (0%)
All Outgoing	113766 (66%)	21102 (12%)	4061 (2%)	52625 (31%)	85 (0%)

3.4 So why did we do less widget impressions?

The above analysis did not yield a conclusive result into the cause of having fewer impressions. Also, some recommenders in Table 1 do significantly more impressions than the baseline recommender does. We therefore assume that the variation of impressions between different recommenders is probably also the result of some non-random preference within the ORP platform.

4 Discussion

Currently the killer sequence algorithm behaves the same for every user. Given that not all users will have the same preferences, it is probably beneficial to personalize the algorithm towards certain user behavioral patterns. For example, instead of giving all follow relations the same weight, it is likely beneficial to give follow relations of similar users a higher weight. This would of course require a metric of similarity between users or a grouping of the users, where relations from the same user group are given more weight for the current user. We expect that a personalized version of the killer sequence will increase the performance.

It is also likely that trends are not static over time. Currently we do not take into account temporal trends in the data, but it is likely that it would be better to take the time when paths occurred into account. For example, it would be possible to use a decay function which gives less weight to paths that have taken place a long time ago.

In many machine learning problems, an ensemble multiple algorithms performs better than a single algorithm [6]. Our infrastructure already takes into account the possibility of using an ensemble of algorithms and using a meta-algorithm, however currently it uses just two algorithms and a very simple meta-algorithm. This of course offers many possibilities for using a better ensemble.

Due to the test setup, it was unfortunately impossible to track which sub-algorithm leaded to which results. In the world of Real Time Bidding [12], it is commonplace to have a unique identifier for a certain impression, which is used throughout the whole interaction, for example also when an impression is clicked. In this case it would have been quite beneficial for the evaluation to have both a unique identifier, and the recommender used to give the impression as attributes in the protocol. As this was not the case, we can only estimate the performance of the Killer Sequence sub-algorithm. Fortunately our algorithm did not require a direct feedback loop of how well the recommendations where doing, as such would have been impossible. Debugging and click attribution would be greatly improved if a future version of the ORP supported attributes for unique identifiers.

5 Conclusion

The CLEF NewsREEL 2107 Task 1 provided a unique opportunity to test our killer sequence recommender in a real-life situation, and compare the results with other recommenders. It turned out that our killer sequence recommender performed 79%

better than the baseline recommender. This resulted in the top position among the recommenders with more than 1000 recommendations. That could have been caused due to bad behavior, such as only answering requests only for certain high CTR domains or times. We investigated whether this higher than expected performance was due to unwanted behavior of the recommender, but could not conclude that this was the case. Therefore, it appears that our algorithm performs very well in the test environment.

There are still many ideas left for improvement of the results. A future version of the recommender should avoid non-recommendable articles, duplicate articles within the recommendation, or returning fewer results than needed. Also, personalization of the killer sequence recommender and ensemble recommendation is expected to yield better results.

References

1. Apache Kafka, https://kafka.apache.org/, last accesed 2017/06/01.
2. Azzopardi, L., & Balog, K.: Towards a living lab for information retrieval research and development. In: International Conference of the Cross-Language Evaluation Forum for European Languages, pp. 26-37. Springer, Heidelberg (2011).
3. Banker, K,: MongoDB in action. Manning Publications Co. (2011).
4. Carlson, J. L.: Redis in Action. Manning Publications Co. (2013)
5. Corsini, F., Larson, M., CLEF NewsREEL 2016: Image based Recommendation. In: Working Notes of CLEF 2016 – Conference and Labs of the Evaluation forum, Évora, Portugal, 5-8 September, 2016, pp. 618-827 (2016).
6. Dietterich, T. G.: Ensemble methods in machine learning. In: International workshop on multiple classifier systems, pp. 1-15, Springer, Heidelberg (2000).
7. Kille, B., Lommatzsch, A., Gebremeskel, G. G., Hopfgartner, F., Larson, M., Seiler, J., Malagoli, D., Serény, A., Brodt, T., Vries, A. P. de: Overview of NewsREEL'16: Multi-dimensional Evaluation of Real-Time Stream-Recommendation Algorithms. In: Experimental IR Meets Multilinguality, Multimodality, and Interaction – 7th International Conference of the CLEF Association, CLEF 2016, Évora, Portugal, September 5-8, 2016, Proceedings, pp. 311-331 (2016)
8. Kille, B., Lommatzsch, A., Turrin, R., Serény, A., Larson, M., Brodt, T., Seiler, J., Hopfgartner, F.: Overview of CLEF newsreel 2015: News recommendation evaluation lab (2015)
9. Tilkov, S., & Vinoski, S.: Node.js: Using JavaScript to build high-performance network programs. IEEE Internet Computing, 14(6), 80-83 (2010).
10. Webber, J.: A programmatic introduction to neo4j. In: Proceedings of the 3rd annual conference on Systems, programming, and applications: software for humanity, pp. 217-218, ACM (2012).
11. Xiang, L., Yuan, Q., Zhao, S., Chen, L., Zhang, X., Yang, Q., & Sun, J.: Temporal recommendation on graphs via long-and short-term preference fusion. In: Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining, pp. 723-732, ACM (2010).
12. Yuan, S., Wang, J., & Zhao, X.: Real-time bidding for online advertising: measurement and analysis. In: Proceedings of the Seventh International Workshop on Data Mining for Online Advertising, p. 3. ACM (2013).

13. Zhou, B., Hui, S. C., & Chang, K.: An intelligent recommender system using sequential web access patterns. In: Cybernetics and Intelligent Systems, 2004 IEEE Conference on, Vol. 1, pp. 393-398, IEEE (2004).