Data Stream Clustering Applied to Preprocessed Yugioh Data

Vincent Allen

5/8/2020

cs6405: Clustering Algorithms

About the Author:

After graduating from the Missouri University of Science and Technology with an Electrical Engineering B.S degree in Dec 2014, Vincent joined Boeing as a Software Engineer II. He has since started a Computer Science M.S. degree and plans to graduate in Dec 2020. During his into to AI course, Vince's AI (Volcanic Counter) placed 3rd in the chess tournament that semester. Later on, in his free time he created a UI in python to go with it. Growing up Vince enjoyed playing the trading card game Yugioh.

Table of Contents

Executive Summary

Before proceeding, we'll give a brief but relevant introduction to Yugioh. It is a trading card game with thousands of cards and variety of card types. Imagine two children saying back and forth "My alien shoots your dinosaur with a lazer gun", then "Well my dinosaur uses an anti lazer shield", then "My alien eats shields" and you have a good idea of what a game of Yugioh is like. There is enough variety in the card effects to make analysis of them difficult. There will always be outliers.

The goal of this project is to use stream clustering to predict the level of a Yugioh monster card based on its attack, defense, and card text. Most monster types have a level/rank between 0 and 12. There isn't a rule that says higher level monsters should be stronger than lower level monsters (i.e. have higher attack and defense or better effects), but that's the general trend. In most cases low level monsters are sacrificed to summon high level monsters, so this makes sense. In any case, the interesting aspect of the clustering should be translating the card text into numerical attributes.

With stream clustering we are able to make a prediction as we process every single point of data, instead of having to wait for a training phase to finish. Thus, we are able to watching metrics update dynamically as the algorithm walks the data.

Introduction

Preprocessing

Before discussing any data mining algorithms, we should focus briefly on preprocessing data. In real life there will be missing data and decisions will have to be made about how to deal with it. One common approach, called listwise deletion, just means deleting any datapoints with missing attribute data. This is only acceptable if the population is sufficiently large and the data is MCAR. Substitution just means replacing the sample with another sample from the population. Imputation means inserting a new value.

MCAR, or Missing Completely at Random, means that the probability of missing data is equal in all cases.

MAR, or Missing at Random, means that the probability of missing data is equal in certain partitions of the data. I.e. we know why it's missing, and the reason isn't one of the attributes we're recording.

MNAR, or Missing Not at Random, means that the probability of missing data varies for reasons we don't know.

Clustering

Clustering is one of many approaches to machine learning. More specifically, it is a method of unsupervised learning – meaning it can be used to analyze data that hasn't already been labeled/ classified. Clustering algorithms can be applied to a dataset to just 'see what happens'. Perhaps it will label the data the way we expect it to, or perhaps it will find a new pattern we humans didn't notice.

As for the approach, similar data points are grouped together in groups called clusters. This similarity is measured by calculating the distance between the attributes of each data point. A good cluster will have less variance. Ideally, a clustering algorithm would minimize the number of standard deviations a point is from the cluster centroid. Note that the centroid is the mean of all the points in the cluster. Consider Figure 1. The data points are colored according to the labels assigned to them when generated.

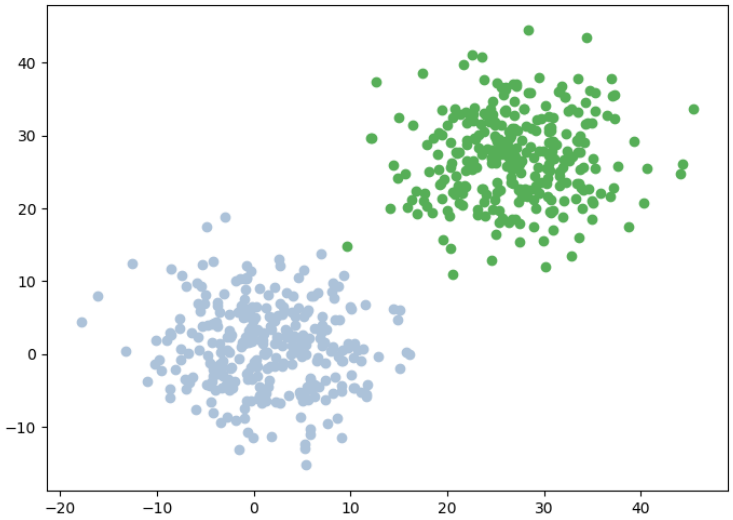

Figure 1

The points are grouped together essentially how we as humans would group them. There are two ‘blobs’ of points, so we would think there should be two clusters. We don’t have to know how many clusters there are, because our minds come up with ‘two’ at a glance. Clustering algorithms can also figure it out without being told by setting a certain threshold for variance. If the points are ‘x’ distance from the centroid, they must not be part of this cluster. See Figure 2 for an example of that distance threshold being set too low. It was too restrictive, so what should have been two clusters turned into several. These figures were generated using a demo file in our repository (see link in Appendix A).

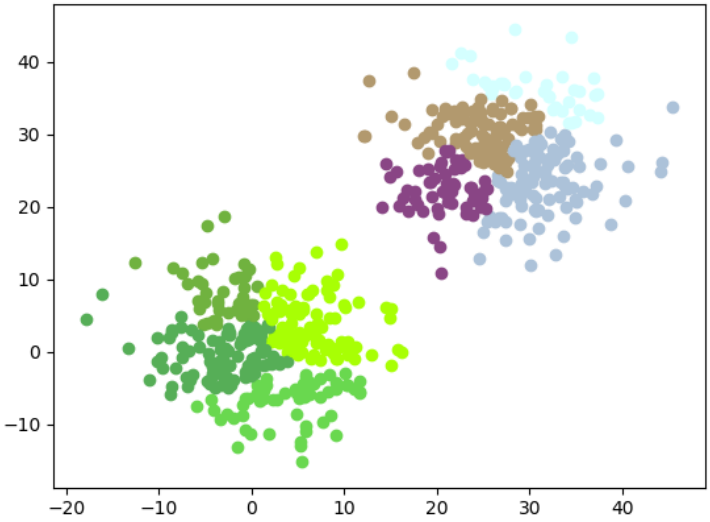

Figure 2

Hierarchical Clustering

Hierarchical clustering is one of many approaches to clustering. In particular, the relationship between the clusters forms a tree. The leaf clusters merge together to form their parent cluster, and so on. See the dendrogram in Figure 3 (also generated using the demo file in our repo (see link in Appendix A).

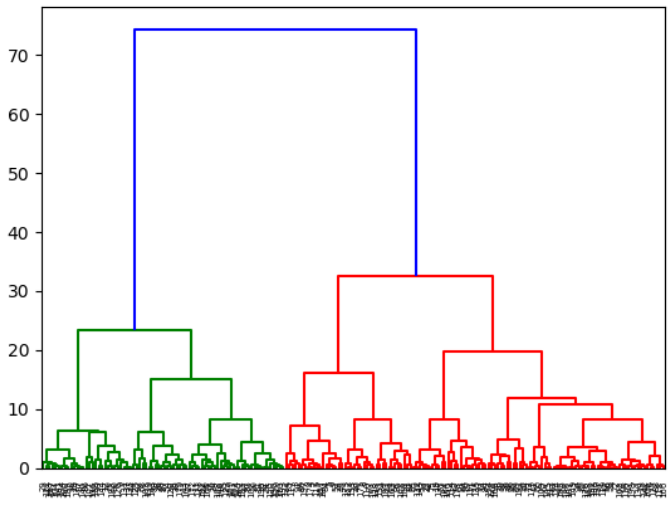

Figure 3

Data Stream Algorithms

Data stream algorithms in general are useful for analyzing either big data or data that changes or arrives frequently. The concept is that data is received one entry at a time instead of having access to the entire data set from the beginning. This gives us the huge advantage of being able to pause at any moment and have a viable model. It also means that data stream algorithms can attempt to analyze an enormous data set without having to hold the entire thing in memory at once to succeed.

At any point the algorithm is able to either accept a data for training or for an attempt at prediction. Note that Figure 4 assumes that the data being analyzed is labeled, but of course clustering algorithms could be used with unlabeled data. In that case the prediction would try to assign the new point to one of the clusters identified by the algorithm. All the clusters could of course would be assigned an arbitrary label.

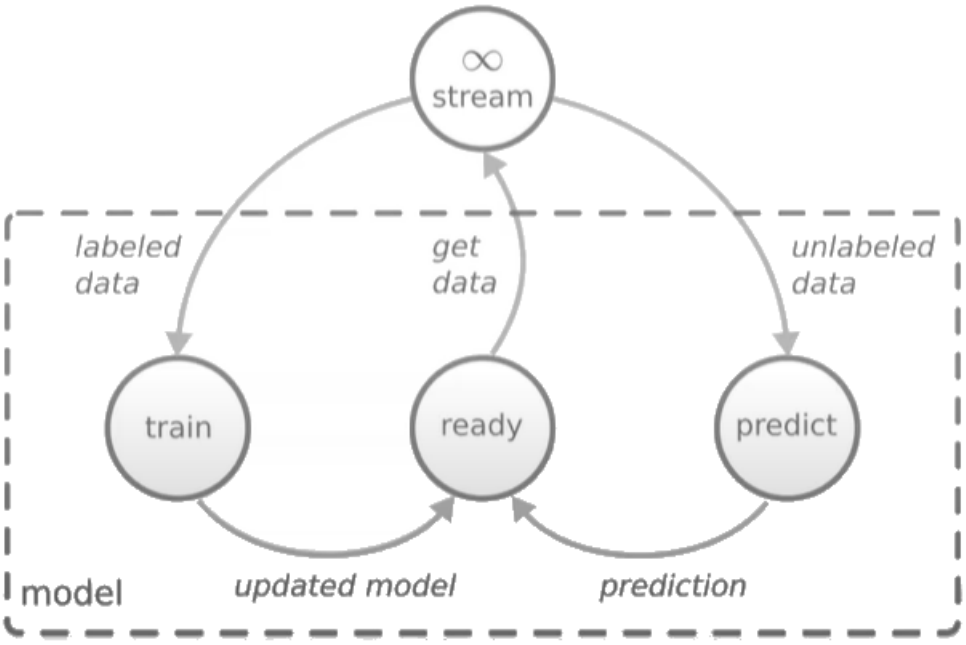

Figure 4 (taken from a scikit-multiflow video)

Project Specifications

Why Yugioh, and why data stream clustering? This particular data set offers a chance to experiment with textual data preprocessing. If we use only the inherently-numeric attributes there is an accuracy of roughly 50%, so it is obvious whether any additional attributes derived from card text have negative, positive, or zero impact. The goal is to improve the accuracy above 50%, and the tool to be used for evaluation will be a data stream clustering algorithm. This type of algorithm allows us to see the accuracy fluctuate as the algorithm parses the data set. It gives us a way to granularly observe the impact of re-ordering our data or including/omitting attributes.

Detailed Design

Preprocessing

In our case, a few data points were (1) labeled incorrectly and (2) missing data. Both issues were fixed manually by looking up the correct information.

Next, we had to deal with numerical data that wasn't missing but couldn't be assigned a 'correct' value either. Several monsters in the game have placeholder values (e.g. '?','???', or 'X000') for attack and defense values, and their effects specify the value. The problem is that they are all situationally dependent. E.g. monster X has 600 attack per card in the opponent's hand. Given that they are all special cases, manual imputation could be an option for a person who knows the game. Though it would be time consuming and extremely biased. The simplest solution is probably to omit them entirely.

As mentioned in the introduction, this approach (listwise deletion) is only acceptable if the dataset is sufficiently large (over 6000 in our case) and MCAR. This 'missing' data is definitely MCAR, as there isn't a specific subset of monsters with variable attack and defense values. Any type of monster with any level could have such an effect.

The original approach was to simply set the attack and defense to 0. This is because several cards in the game actually have a value of 0 instead of something like '?', yet they still have an effect that replaces the 0 value. An example is *Relinquished*, which captures an opponent's monster and uses its attack and defense values as its own. The question is, how do we tell those cards apart from the ones which actually have a value of 0? Is it worth the time to try and detect these cards by parsing the card text? Not to mention that there are other cards with typical attack and defense values which can be increased or decreased via their effect. If two different people started down that same rabbit hole, they would emerge with two completely different data sets. Thus, we believe the best/least biased action is to not alter the attack and defense data based on card text.

Speaking of card text, it will actually contribute the bulk of our preprocessing logic. Since it isn't inherently numeric, we'll need to translate the text into one or more numeric attributes. The goal is to group monsters based on level, and monsters with good effects are comparable with ones that have high attack or defense stats. I.e. we need find a way to detect and measure 'good' effects.

There is one thing to note before we dive in. One type of monster called 'Normal' has text but it serves no functional purpose. It's just lore. Thus, Normal monster text will be omitted during the preprocessing. Now, on to translating card text to numeric values.

The first attempt was a simple word count. Let's assume that the effects with the most words are strongest. This is definitely incorrect. Consider *Exodia the Forbidden One* which has the 40-word effect "If you have [...] in addition to this card in your hand, you win the Duel." This is a very good effect for being so short, as there are just a handful of cards in the game that allow a player to win outright without using monsters to attack their opponent.

Next, we tried a variation of the word count, specifically searching for words like 'destroy' and 'damage' which typically involve harming the opponent's cards or their life points (when their life points hit 0, they lose). This resulted in a decrease in accuracy when paired with the normal word count.

As an interesting twist we tried inverting the word count attribute later in the project. The idea was the opposite – that lower star monsters would have longer effects. This was accomplished simply changing the equation to 1 – word count. A value of 1 was used because the all data is normalized as it is written.

For future work, if we continue to analyze the card text it would be recommended to focus solely on effects that pertain to altering the ATK and DEF stats of the monster. Perhaps more intelligent parsing of this data would yield the results we were hoping for.

Clustering

Several data stream clustering algorithms were considered for use in this project. Originally ClusTree was the most intriguing given that it was hierarchical and entirely online / an anytime algorithm.[3] Unfortunately no repository was found in the survey papers or in the ClusTree article. One repository with the name ClusTree was found, but it made no reference to the paper. The suspicion is that it is something else entirely.

gStream was also considered, but unfortunately, its API didn't allow us to interact with each data point one at a time. This was in complete contrast to the scikit-multiflow library. See Appendix A for more details.

Figure 5 is an example Data Stream Clustering algorithm schematic, with sample inputs. The training/test data blocks are a reminder that even though clustering is unsupervised it can still be used with labeled data. In fact, the labels it generates could be correlated with the already existing labels and used to measure accuracy. The serializer would just read one element at a time from the training data set, since the whole set wouldn't be passed all at once to a Data Stream algorithm. The web crawler input is an example of a data stream type that would be especially well suited for input to one of these algorithms.

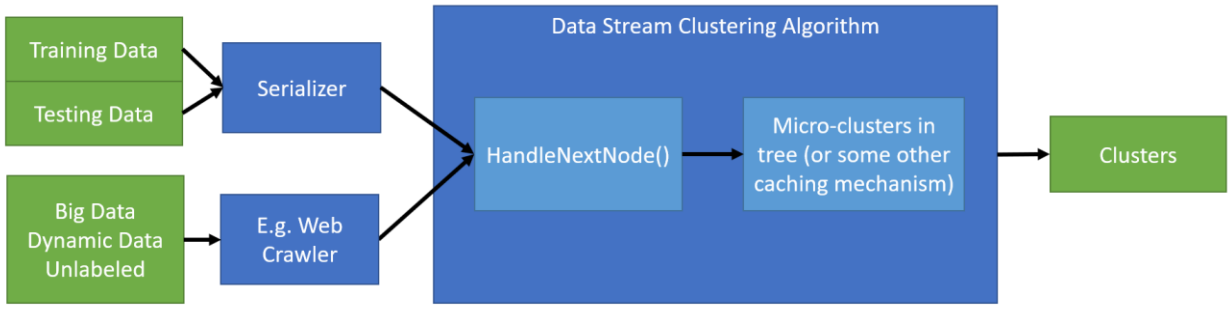

Figure 5

Though the original intent was to use a hierarchical data stream clustering algorithm, but technical limitations prohibited customization of the scikit-learn and scikit-multiflow libraries. As an alternative to combining them both, each algorithm was tested individually. The hierarchical clustering was considered for the fast access for prediction the hierarchy would provide. However, in the end we went with the kNN-based stream clustering algorithm instead for more granular control of the model.

Experimental Results

Ase a baseline, see Figure 6. In these results only the attack and defense values were used as attributes. Note that these values stay roughly the same if values of 0s are used to replace the ?, ???, and X000 attack and defense values for some monsters. As a reminder those datapoints were removed instead.

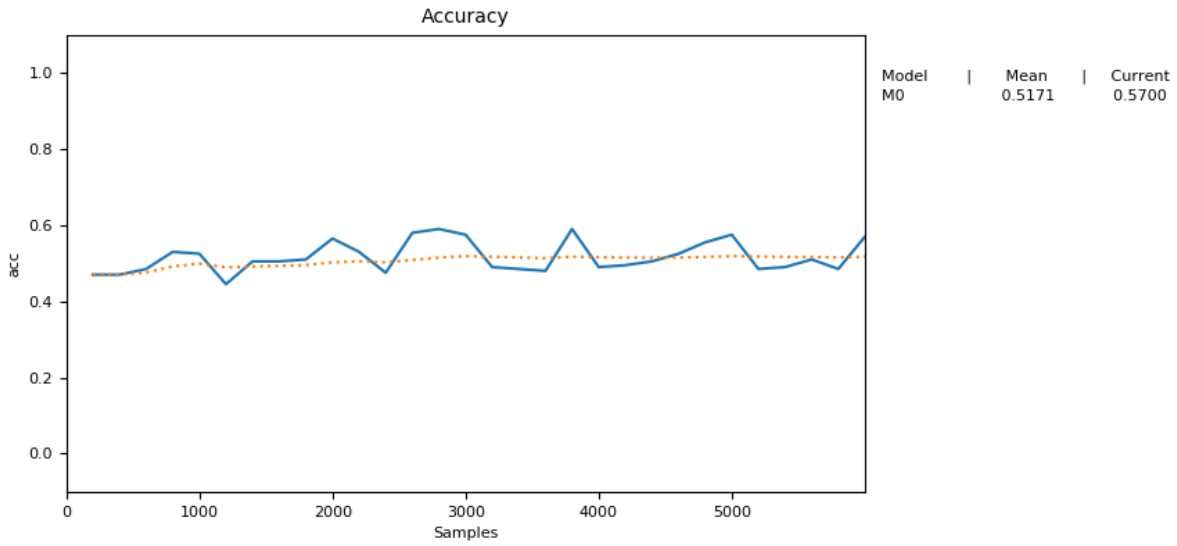

Figure 6

Now, in Figure 7 we tried the word count measurement in addition to the attack and defense stats, but saw about a 1-3% drop in accuracy. Note that between each run the data is shuffled randomly, and each run is attempted 3-5 times.

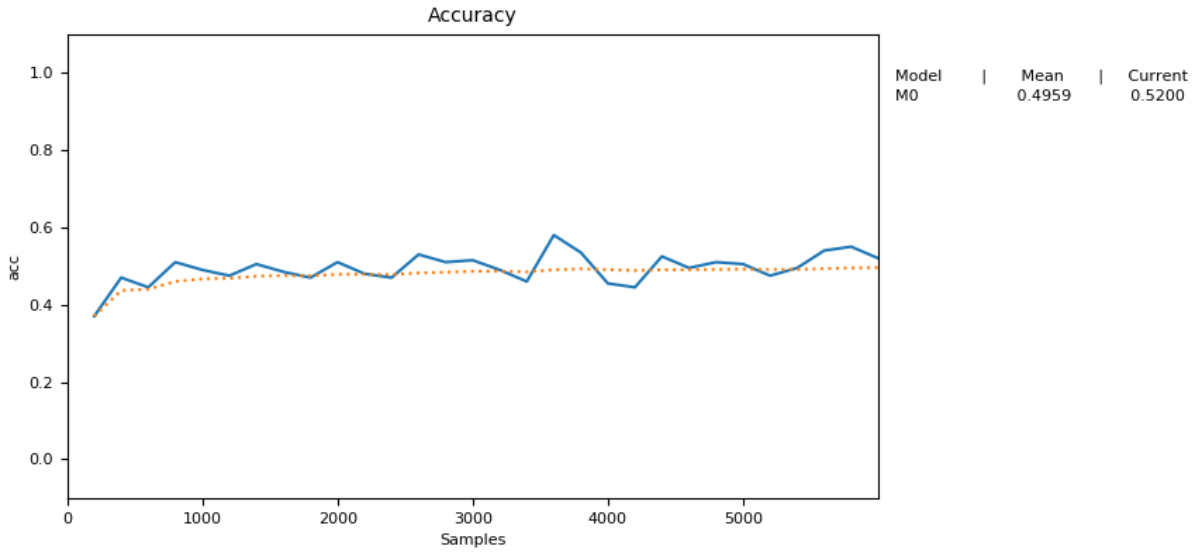

Figure 7

In Figure 8 we tried adding another attribute that counted only the words ‘destory’ or ‘damage’, but the average decreased by another few percent. Note that the normal word count attribute was also present for these runs.

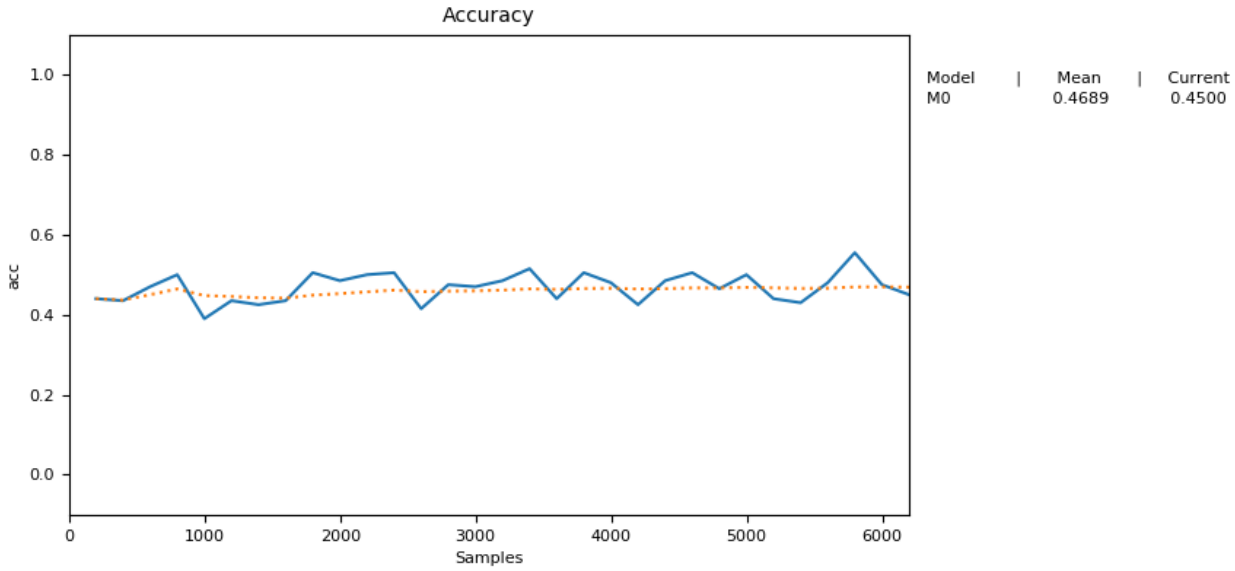

Figure 8

Next, we took a step back and tried inverting the word count, so that higher word counts equated to lower ranks. The interesting thing is that the accuracy was nearly identical to that of the non-inverted word count run. The mere presence of the word count attribute in either direction decreases accuracy. In hindsight this should have been obvious, because distances would remain the same even with inverted values.

At this point it became obvious that unless we did something drastically different the word count approach wouldn’t get us closer to a more accurate estimate. So, it was concluded that the best approach would be to predict using only the ATK and DEF attributes.

One potential consideration for future work could be to further divide the monster data into types (YYZ, fusion, normal, etc) using the same attributes. The advantage would be increased accuracy for some sections (e.g. pure ATK and DEF for normal) but a decrease in reliability because of the smaller sample sizes.

References

Works Cited

[1] M. Ghesmoune, M. Lebbah, and H. Azzag, "State-of-the-art on clustering data streams," Big Data Analytics, vol. 1, no. 1, 2016. (link)

[2] S. Marsland, J. Shapiro, and U. Nehmzow, "A self-organising network that grows when required," Neural Networks, vol. 15, no. 8-9, pp. 1041–1058, 2002. (link)

[3] P. Kranen, I. Assent, C. Baldauf, and T. Seidl, "The ClusTree: indexing micro-clusters for anytime stream mining," Knowledge and Information Systems, vol. 29, no. 2, pp. 249–272, 2010. (link)

[4] S. Mansalis, E. Ntoutsi, N. Pelekis, and Y. Theodoridis, "An evaluation of data stream clustering algorithms," Statistical Analysis and Data Mining: The ASA Data Science Journal, vol. 11, no. 4, pp. 167–187, 2018. (link)

[5] A. Amini and T. Y. Wah, "Density Micro-Clustering Algorithms on DataStreams: A Review," Lecture Notes in Engineering and Computer Science, vol. 1, 2011. (link)

[6] L. Liu and T. Peng, "Clustering-based topical Web crawling using CFu-tree guided by link-context," Frontiers of Computer Science, vol. 8, no. 4, pp. 581–595, 2014. (link)

Works Consulted

Articles / Websites:

- https://www.guru99.com/supervised-vs-unsupervised-learning.html
- https://www.quora.com/Is-unsupervised-learning-a-better-approach-than-supervised-learning-models
- https://www.researchgate.net/post/what_is_micro-cluster_in_data_stream_clustering_algorithm
- http://www.personal.psu.edu/pum10/tois-tan.pdf
- http://www.rroij.com/open-access/an-approach-to-build-a-web-crawler-using-clustering-based-kmeans-algorithm-14-22.pdf
- https://en.wikipedia.org/wiki/R-tree
- https://nces.ed.gov/training/datauser/PSS_02.html?dest=PSS_02_S0170.html
- https://www.theanalysisfactor.com/causes-of-missing-data/ *(broken link since original visit)*

Source Code:

- https://github.com/Spark-clustering-notebook/coliseum/wiki/G-Stream
- https://github.com/huawei-noah/streamDM
- https://cran.r-project.org/web/packages/clustree/index.html

Videos:

- https://www.youtube.com/watch?v=B-NN-weo5e4

Appendix A

You can find the source code for this project here:

https://github.com/vtad4f/stream_clustering/tree/master/src

Some experimentation was performed with an R implementation of gStream. However, with just one look at the API and the demo it is obvious that data cannot be fed into the algorithm one node at a time.

In contrast, this python implementation of several streaming algorithms explicitly defines data stream classes with next_sample() functions. See this demo script which uses kNN to predict one element of data at a time. Here are a few lines from the file:

```
while n_samples < max_samples:
    X, y = stream.next_sample()
    my_pred = knn.predict(X)
    if y[0] == my_pred[0]:
        my_corrects += 1
    n_samples += 1
```

Now, we are free to take this same granular approach with our implementation, but there is a higher-level import from the same package which displays a graph with the metrics of our choice. The following is an excerpt from our implementation, with all the optional parameters removed for brevity:

```
from skmultiflow.data import FileStream
from skmultiflow.lazy.knn import KNN
from skmultiflow.evaluation import EvaluatePrequential

stream = FileStream('data/stream1.csv')
stream.prepare_for_use()
mdl = KNN()
evaluator = EvaluatePrequential(metrics=['accuracy'])
evaluator.evaluate(stream=stream, model=mdl)
```

This is straightforward, and we can watch the accuracy update gradually as the algorithm walks the dataset. That being said, we've done some preprocessing work to translate the card text into numeric attributes, and to normalize the data. It was all trivial to implement, so we will only include the high level functions here:

```
if __name__ == '__main__':
    """
        BRIEF  Parse and assemble a single csv with star columns
               https://www.yugiohcardguide.com/level/0.html
               all the way through .../12.html

               Note that the page contents have been copied manually
    """
    header_row, rows = ReadAll()
    Write(header_row, rows, 'stars.csv')

    ProcessText(rows)
```

```
FixNumeric(rows)
random.shuffle(rows)
Write([
    'attrib1', # atk
    'attrib2', # def
    'attrib3', # word ct
    'class'    # level
], rows, 'stream1.csv')
```

Appendix B

See additional file for the annotated bibliography.