Evaluation and Comparison of DBA and SAX K-means Clustering on NASDAQ Stocks

CSCI E-82 - Final Project - Fall Semester 2017

Authors: Srini Katakam and Jonathan McNutt

*Abstract—***The problem of time series clustering is not a trivial one, and financial data such as stock market prices are no exception. Without the "ground-truth" of target class labels, the standard comparison metrics for supervised classification techniques (KNN, Neural Networks, etc.) such as accuracy, specificity, recall or sensitivity, and the harmonic mean cannot be applied. This paper will use inertia, or the within-clusters-sum-of-squares, to assess and compare several K-means clustering techniques, including two newer approaches, DBA and SAX.**

I. INTRODUCTION

According to a research paper from UCR, entitled "Dynamic Time Warping Averaging of Time Series allows Faster and more Accurate Classification"[9], KNN (Nearest Neighbor) algorithm using DTW as the distance metric was the most accurate classifier for time series, even beating neural networks! This could be due to the auto-correlated nature of time series structures themselves. The authors of this paper came up with Nearest Centroid Classifier, which employs DTW Barycenter Averaging (DBA) in order to represent the "average" time series of a particular class. This average time series representation can be considered the "centroid", which brings us to the critical topic of clustering (when we do not have the ground-truth class labels and cannot do classification).

The advantages to clustering using DBA were first discussed in detail in the paper "A global averaging method for dynamic time warping, with applications to clustering"[10], where the authors discussed DTW as the best distance metric for time series classification, and DBA as a novel strategy for finding the average sequence. The K-means algorithm was previously unable to utilize the full power of DTW distance, since K-means is really meant for Euclidean distance, but Euclidean distance is not accurate when the different series are out of sync or of different lengths. The DBA clustering paper proved for the first time that DTW combined with the new DBA can be effectively used with K-means for clustering time series with impressive results.

Our project was motivated by the curiosity for discovering new ways to find groups or classes of stocks from the NASDAQ. We asked ourselves these questions: Could stocks be clustered in such a way as to gain a competitive edge over the market? What can we learn from different clustering techniques? What is the best way to cluster stock price time series? To try to find answers to these questions, we embarked on an exploratory analysis to test and compare two state-of-the-art methods: DBA DTW K-means clustering and SAX (**S**ymbolic **A**ggregate Appro**X**imation). The latter approach is a relatively new which is more of an approximation of a time series represented as symbols rather than exact time series for performing analysis. It is proven that this approximation is lower bounding to original time series. We will show how SAX can be used for clustering time series data in a smarter and efficient way and at the end we will close with a comparison of the results between these two methods on a smaller dataset size.

II. PREPROCESSING

We obtained NASDAQ stock closing prices from a public Kaggle[4] data set and gathered only the prices from 2012-2016 for SAX clustering, but for DBA DTW clustering we used data from the years 2016 and 2017, for the practical sake of time constraint in running the kernels. We pulled in over 7,000 different stock ticker symbols, and then cleaned the data set by backfilling the missing values (probably due to those stocks not existing on the market at the time). The cleaned data frame was then written to a csv file for use in separate iPython notebooks per clustering method.

III. DBA K-MEANS CLUSTERING

We decided to use the tslearn[7] package due to its robust time series clustering functions and data normalization tools built-in. One of the functions is called TimeSeriesKMeans, which can take different distance metrics as a hyperparameter and use them for K-means clustering. In order to show the superiority of DBA, we ran three distance metrics: Euclidean, DBA, and Soft-DTW. We initially tried to use all 6,399 stock tickers for the full 2012-2016 data set, but DBA ran overnight and did not complete, so for practical reasons we reduced the data set to only 100 stock tickers for 2016, and then after some success tried again with 500 stock tickers for 2017. To prepare the data sets for clustering we first transposed the data frames so that the tickers were row indexes, then transformed the data frame into a matrix, and then used the tslearn function TimeSeriesScalerMeanVariance() to normalize the time series so that all had a mean of zero and standard deviation of 1, and all were the same size (same number of trading days in each series, as otherwise we could not compare results to Euclidean distance).

The metric of comparison we used between the three K-means clustering results was “inertia”, or the within-cluster sum of squares criterion, which “can be recognized as a measure of how internally coherent clusters are”.[11] In general, the lower the inertia, the better the clustering.

TABLE I: COMPARING INERTIA SCORES ACROSS DISTANCE METRICS, SAMPLE SIZES AND K-VALUES

	100 stocks from 2016		**500 stocks from 2017**	
Distance Metric	*K=3*	*K=6 (elbow)*	*K=3*	*K=5 (elbow)*
Euclidean	114.281	87.786	115.052	96.034
DBA	**24.965**	**18.502**	**27.010**	**23.597**
Soft-DTW	1059.793	529.802	1328.836	916.803

Clearly, the results show that DBA K-means was able to obtain the minimal inertia (but using the maximum amount of processing time to do so). We were surprised to see that Euclidean distance “beat” Soft-DTW, but we suspect that if we had the time to do further trials, we would have found that the Euclidean results received a huge boost due to the normalization of the time series data. If the time series were of different sizes (as they were before our initial preprocessing), then we postulate that the Soft-DTW inertia would be lower (better) than the Euclidean score. Also of note, we can see that choosing a better K-value (K=5 and 6) using an “elbow plot” resulted in much better inertia scores for all distance metrics tested.

The following three representative figures show the differences between the K-means clustering results for the three distance metrics used. Cluster 1 is visualized for each metric, and it is easy to see that all three results have the same upward trend from the beginning of trading year 2017 to the end (up until November). This upward trend can be interpreted as a steady increase on average in stock value over the course of the year. In other words, knowing which stocks are grouped into this cluster would be very helpful in putting together a portfolio that maximizes returns, or at least would minimize losses. In the iPython notebooks we show the rest of the plots for all clusters per K-value per year. It is also helpful to know which clusters are downward trends and flatline trends, so as to avoid risk or knowing when to sell certain stock.

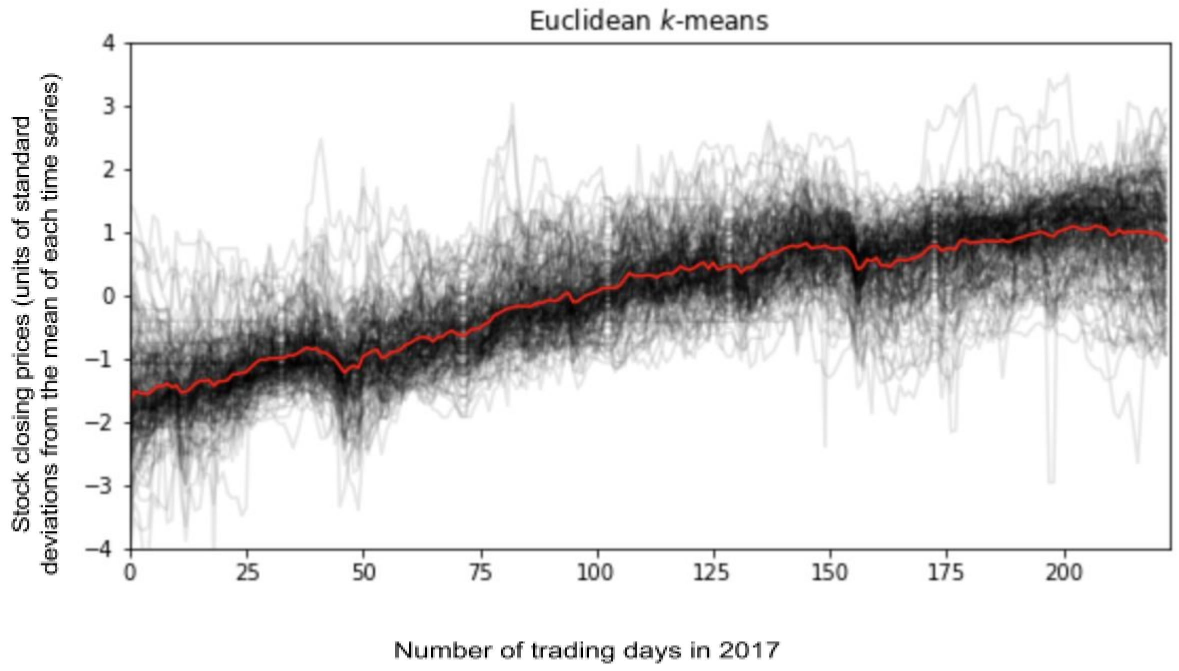

Figure 1: (best viewed in color) The time series in cluster 1 using Euclidean distance as the metric for K-means clustering with K=5. The red line is the centroid, or average time series for this cluster.

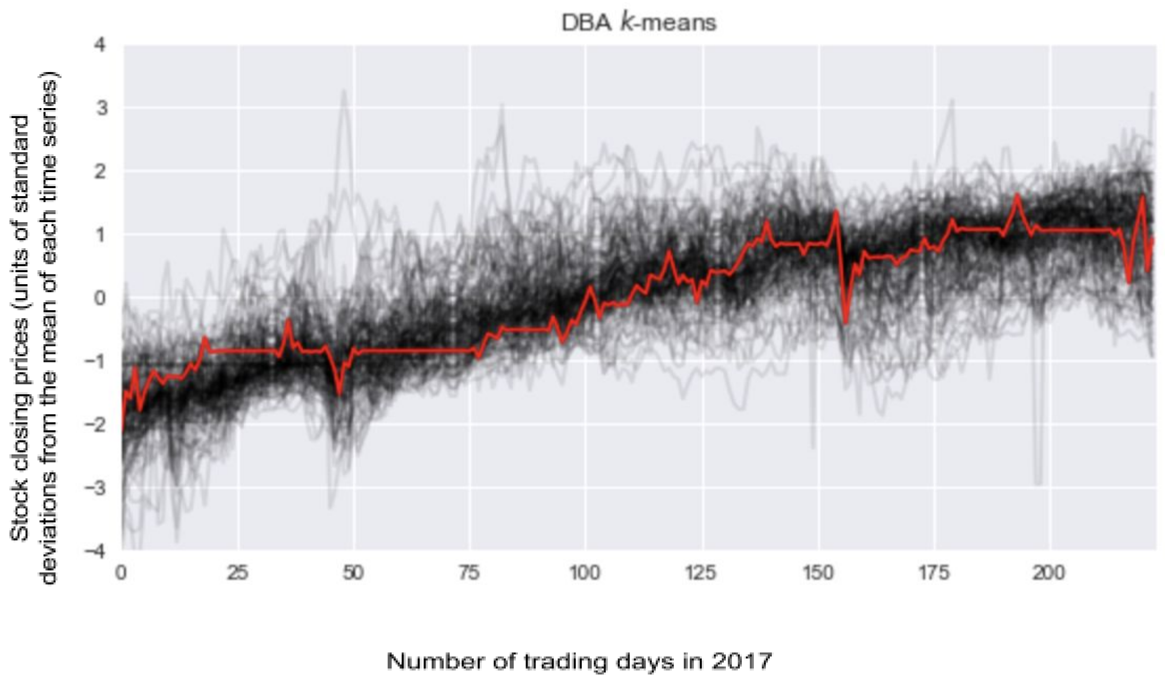

Figure 2: (best viewed in color) The time series in cluster 1 using DBA and DTW distance as the metric for K-means clustering with K=5. The red line is the centroid, or average time series for this cluster.

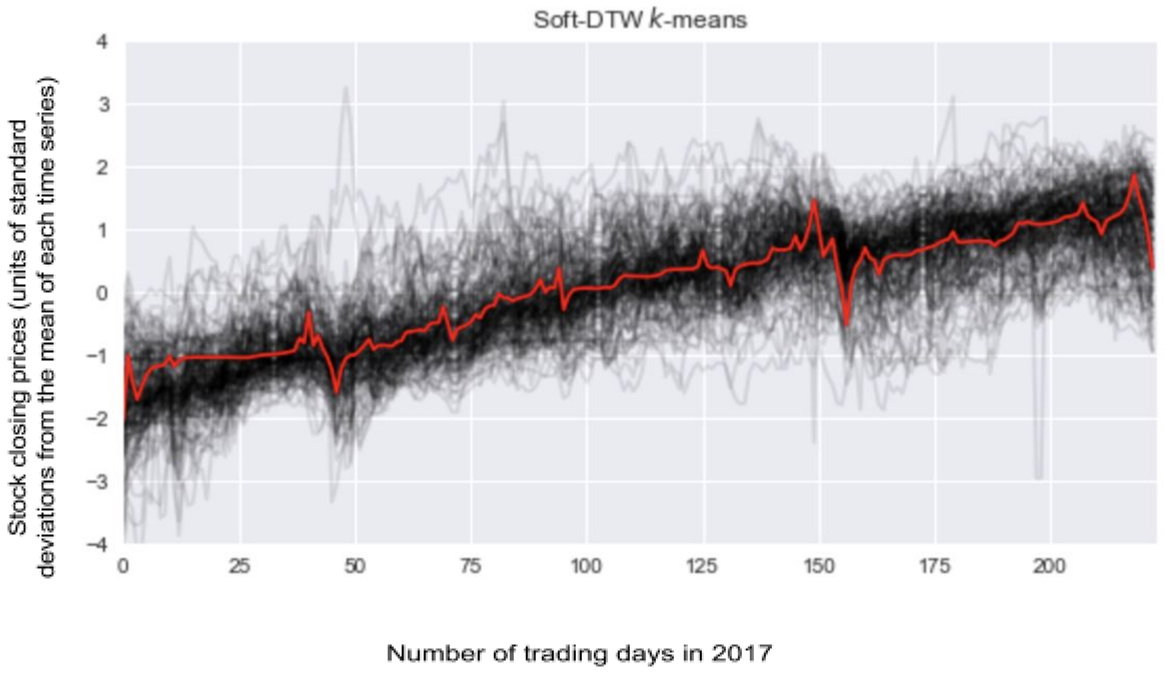

Figure 3: (best viewed in color) The time series in cluster 1 using Soft-DTW distance as the metric for K-means clustering with K=5. The red line is the centroid, or average time series for this cluster.

On the practical side, Euclidean computation was near-immediate for this many stocks (100 or 500) and at this size of time series (one year), whereas Soft-DTW took several minutes, and

DBA took over an hour to complete! On the one hand, DBA K-means seems to be “king of the hill” when it comes to clustering time series scored by inertia, but on the other hand, the time, energy and hardware (CPU/GPU) resources available can present difficulties for real-time application at scale, such as for wearable devices. Hence, we also examined another, novel approach to time series clustering, SAX.

IV. SYMBOLIC AGGREGATE APPROXIMATION (SAX)

This is a new algorithm that we haven't covered in our class, but apparently is a well-known technique for time series approximation and analysis. Detailed explanation of the algorithm can be found in the links under references. In short, SAX can represent a long time series into character symbols called Bag-of-words (BOW). It has been proven that this representation allows lower bounding of Euclidean distance in the original space. As we learned more about SAX, it was very clear that we could use NLP techniques like TF-IDF to cluster the stock data by converting each stock time series into BOW representations.

We found a Python implementation of SAX called pysax[2] online which had all the functions that are required to obtain SAX strings from time series. We had to fix some issues with the library as it was developed for Python 2.x (comments can be seen in the pysax.py file). As you can see from the notebook, the process of obtaining SAX words from time series is very simple and fast. The stock data is first Z-normalized, followed by SAX and the final dataframe is represented with stock tickers as indices with a single column *sax_bow* with list of word representations of corresponding time series. This dataframe is converted into TD-IDF vectorizer matrix and then used for K-Means clustering technique to perform clustering. As the data size was huge, the optimal K value using knee plot was giving us 80+ clusters which was way too many. We set K=15 for simplicity and for improving the visualization of the clusters when we plot them. Using the TF-IDF matrix, we computed the distance matrix and then converted the distance matrix into a 2D array using Multidimensional Scaling[3] for plotting the clusters.

V. TUNING SAX HYPERPARAMETERS

Parameters like window, stride sizes, number of bins and alphabets needed to be tuned for proper clustering of the tickers in this dataset. Since we don't have the ground truth of actual classes in this dataset, the clustering performance is verified visually and kind of manually by checking the stocks that end up together and their sector on NASDAQ for a possible pattern matching. Here is a brief description of these parameters:

Window size: sliding window length to define the number of words
Stride size: stride of sliding
Number of Bin’s: number of bins in each sliding window
Alphabets: alphabet for symbolization

Note: Please refer to notebook for plots. In this paper, we will be including the final tuned clustering plots.

Trial 1:
Window = 130, Stride = 130, Bins = 10, Alphabets = ABCD
Observation:
With a large window and stride size, we get fewer bag of words. As a result, the model runs fast but the clusters produced had lot of overlaps.

Trial 2:
Window = 65, Stride = 65, Bins = 10, Alphabets = ABCDE
Observation:
In order to improve the clustering, we lowered the window and stride sizes to half the previous trial and added one more alphabet for more granular symbolization of time series. The clustering is better than before. We can see the black and gray clusters very well. Upon some manual verification of tickers that ended up together, we found that most of the black stocks were either Financial or Pharma sectors. But again, there was lot of overlap with other clusters. K-means inertia for this trial was 5215.43.

Trial 3:
Window = 35, Stride = 35, Bins = 10, Alphabets = ABCDE
Observation:
The window and stride sizes are now set to 1/4th of the original settings thereby increasing BOW size 4 times. Here again we noticed that the clusters are now even more apparent than before. K-means inertia for this trial was 4992.94 better than previous trial.

Trial 4:
Window = 10, Stride = 10, Bins = 10, Alphabets = ABCDE
Observation:
Further reduction of window and stride size did help with the clustering a bit. But the model took more time and the results were not that different compared to previous trial. So at this point, it's clear that further reduction will not gain us much other than impacting the model performance. We will be going back to 35 as the window and stride sizes and start tuning number of bins param. K-means inertia for this trial was 4611.36, again improvement over previous trial.

Trial 5:
Window = 35, Stride = 35, Bins = 20, Alphabets = ABCDE
Observation:
Increasing the number of bins from 10 to 20 and setting the window and stride sizes to 35 resulted in longer BOW representation (each word is now 20 characters long). These params yielded in an impressive clustering plot. As we can see below, most of the clusters are well separated with few exceptions. The clustering borders are now more clear than any trial that we did before. Below are the final plots, the plot in Figure 4 is generated on the entire dataset, and

the plot in Figure 5 is generated on a sample of 1000 tickers after clustering was performed. Again, I verified few of the stocks manually and found that few of them kind of belong to similar sector as listed by NASDAQ. There are still some mis-clustering happening a lot. It's quite possible that these clusters could represent stocks part of a different mutual funds or large cap, medium cap or low cap. People with more financial back ground can better interpret these results. K-means inertia for this trial went up a little to 5077.80.

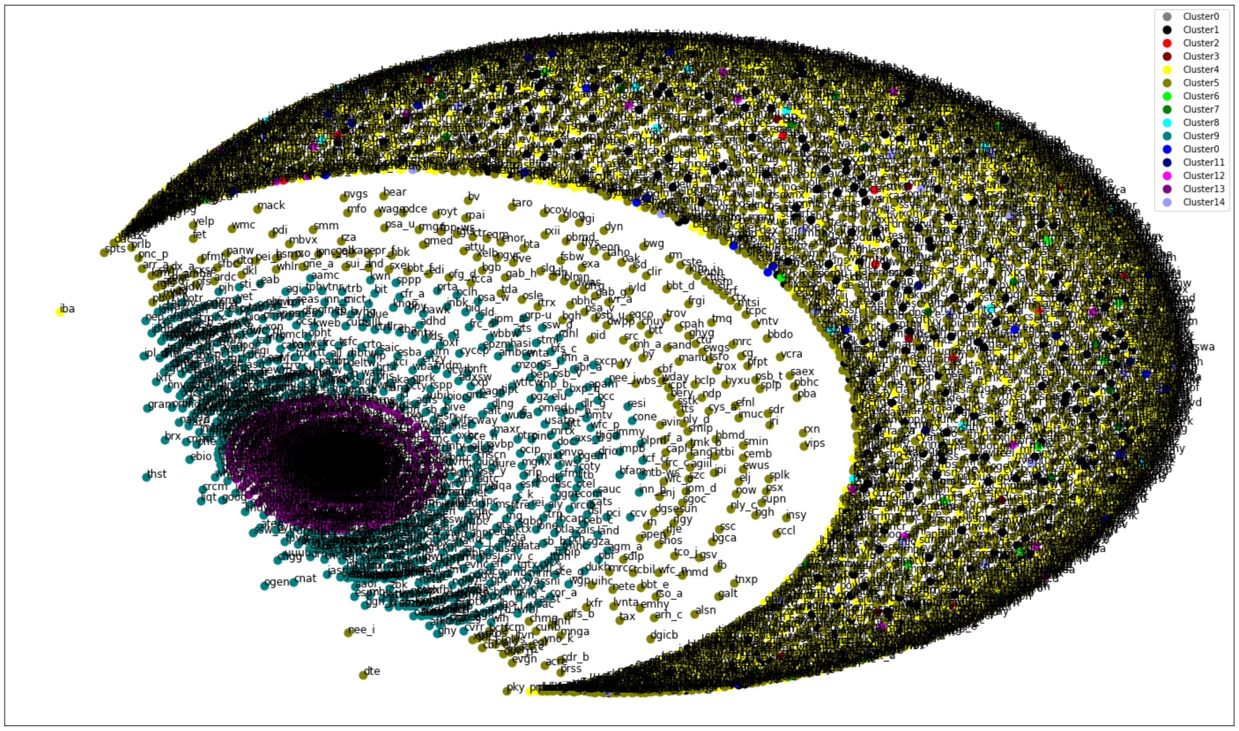

Figure 4: K-means clustering plot using 15 clusters on the complete stock dataset of around 6400 tickers based on the close prices from 2012 to 2016. As the dataset is huge, its hard to see the ticker labels. But overall, the clustering pattern is visible.

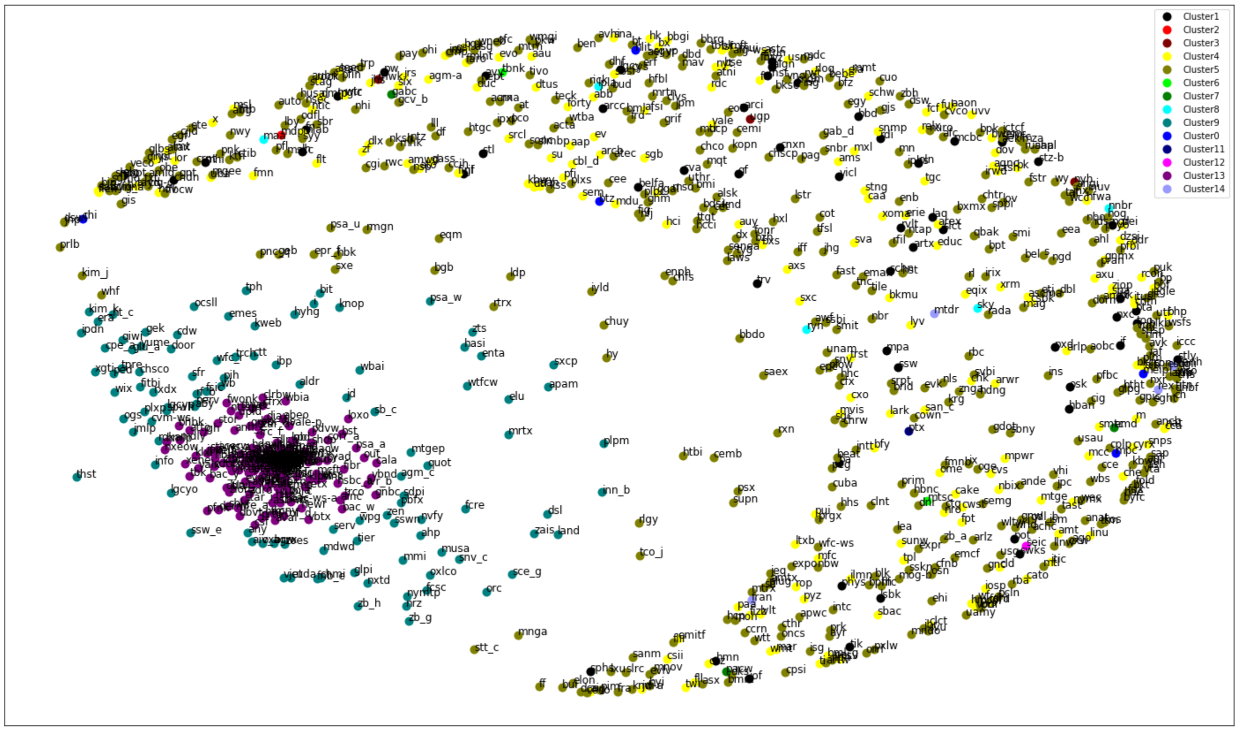

Figure 5: K-means clustering plot using 15 clusters on a sample of 1000 tickers based on the close prices from 2012 to 2016. When you zoom in, we can see the ticker labels.

Overall, proper tuning of SAX is critical for clustering performance based on above results. As we keep decreasing the window and stride sizes, the Inertia keeps going down but at some point the model takes long time to cluster and we have even noticed that the generated BOW's seem to deviate from some basic rules of symbolization (perhaps the way pysax.py is implementing needs to be revisited).

VI. CONCLUSION

DBA and SAX K-means clustering techniques have been proven to be superior methods but for different reasons. Overall, we have shown that these two new techniques can successfully cluster stock data time series and help guide investment strategies by knowing which stocks belong to which clusters with their corresponding trends. The important takeaways from this analysis are as follows:

- DBA seems to have the lowest inertia of the techniques we assessed, but at the cost of greater time complexity.
- SAX is very fast, but is an approximation.

Future research could explore the time improvement gained with parallelization of the DBA algorithm, something we did not have time to explore. In addition, SAX results could be further improved with more tuning of the hyperparameters.

We also explored the option of trying out Matrix Profile on this dataset. The idea was to identify motifs for each stock and use them as features for clustering. But because of the time constraint we couldn't get to that part.

References

[1] http://www.cs.ucr.edu/~eamonn/SAX.htm
[2] https://github.com/dolaameng/pysax
[3] http://brandonrose.org/clustering
[4] https://www.kaggle.com/borismarjanovic/price-volume-data-for-all-us-stocks-etfs
[5] https://jmotif.github.io/sax-vsm_site/
[6] http://www.cs.ucr.edu/~eamonn/MatrixProfile.html
[7] http://tslearn.readthedocs.io
[8] http://perso.eleves.ens-rennes.fr/~xshan192/doc/time_series.pdf
[9] http://www.cs.ucr.edu/~eamonn/ICDM_2014_DTW_average.pdf
[10] http://lig-membres.imag.fr/bisson/cours/M2INFO-AIW-ML/papers/PetitJean11.pdf
[11] http://scikit-learn.org/stable/modules/clustering.html