The Journal of Finance and Data Science 10 (2024) 100132

Contents lists available at ScienceDirect

The Journal of Finance and Data Science

journal homepage: www.keaipublishing.com/en/journals/jfds

Research Article

# CPC-SAX: Data mining of financial chart patterns with symbolic aggregate approXimation and instance-based multilabel classification

Konstantinos Nikolaou

*Department of International & European Economic Studies, Athens University of Economics and Business, Patision 76, 10434, Athens, Greece*

ARTICLE INFO

*Keywords:*
SAX
Chart Patterns
Classification
Multilabel
Time Series

ABSTRACT

In order to be able to classify financial chart patterns through machine learning, we introduced and applied a novel classification algorithm on time series data of different financial assets through SAX (Symbolic Aggregate approXimation), a transformation algorithm. After applying a linear regression model on the features of a dataset to reduce the number of parameters needed, converting real valued data to strings of characters through Piecewise Aggregate Approximation (PAA) and labelling each level increasingly with Latin alphabets characters, the new algorithm called CPC-SAX (Chart Pattern Classification) compares vectors describing the ASCII value changes along the string and classifies them using already labelled SAX-transformed data. The results show satisfying accuracy scores on data of different time windows and types of assets. We also obtain information on the appearance of said patterns. By reaching our goal of properly classifying chart patterns as they appear, we can have a better indication of the future price trend, allowing the investor/trader to make better informed decisions.

## 1. Introduction

In the field of finance, chart patterns play an important role as the building blocks of technical analysis. They are a visual representation of the forces of supply and demand behind stock price movements, helping traders identify if more buying or selling is happening, which can help make entry and exit decisions. Finance chart patterns have been thoroughly examined and their research has expanded throughout the years thanks to efforts such as of Bulkowski (2022), whose *Encyclopedia of Chart Patterns* has been a main source of knowledge and inspiration for our research. This material shows how the appearance of chart patterns can be linked with a probability of either an upward or downward trend, therefore assisting the investor in making a more sound active short-term strategy.

The world of Machine Learning on the other hand is still a relatively new field of science, which came of age in the last decade or so, and has revolutionised the way scientific research and analytical procedure is executed. As Machine Learning can be quite invasive, it has made itself a powerful tool in many aspects of financial analysis, such as asset price forecasting, and the extraction/classification of chart patterns is no different.

This paper looks to expand on the research of financial chart pattern classification, a method that can significantly improve the way we invest by assisting the financial analyst. It uses dimensionality reduction in order to process a large amount of data multiple times,

*E-mail address:* k.nikolaou@aueb.gr.

Peer review under responsibility of KeAi Communications Co., Ltd.

https://doi.org/10.1016/j.jfds.2024.100132
Received 13 July 2023; Received in revised form 24 November 2023; Accepted 16 May 2024
Available online 3 June 2024

and while previous research was focused on feature extraction on real-valued data, either using shapelets, as did Ye L. (2011) or Perceptually Important Points (PIPs), shown in Chung et al. (2002), this paper, uses the tried and tested method of Symbolic Aggregate approXimation transformation (SAX), with a sliding window (Lin et al., 2007).

SAX uses a Piecewise Aggregate Approximation (PAA) (Zhang et al., 2018) with indexing, to transform a time series snippet into a string of characters with varying length and set of indexes, eg. 'aabcba'. A sliding window means that the algorithm starts transforming from the first index, index 1 to index k. Then it repeats but moves one step forward, from index 2 to index k+1, and so on and so forth. SAX is the brainchild of Eamonn Keogh, who has been behind its creation and adaptations/improvements over the last 25 years, and it gives us an alternative to the PIP method, where we still keep the principal components of the time series and turn this problem from real values to language processing.

For the purpose of having enough labelled data to use in a supervised learning manner, an extraction pseudocode was developed that goes over an already SAX transformed train set and classifies each string, using a rule-based approach for each pattern.

While there are numerous established methods for solving natural language problems concerning classification, this one has a particularity; it doesn't revolve around a corpus of words that are meaningfully combined to form sentences, but around what seems to be incomprehensible. The usefulness of similarity metrics like Hamming distance and neural network models dissipates in such outlandish features.

Therefore a new algorithm was developed, which instead of comparing the similarity of SAX strings, it compares the movement of PAA levels (from 'a' to 'b', or from 'b' to 'd' etc.). In this way, it accounts for the symmetry of strings that describe the same pattern, as for example 'aba' and 'bcb' represent the same up-down pattern.

After movement vectors are created out of test SAX strings, they are compared one-on-one with the training data using cosine distance, by picking a test string and scanning all labelled strings in all bags. If their vectors aren't identical, then their difference (element-by-element) is added in a sum. This iterates for the entire bag, or until an identical vector has been found. If it is not, an average difference is derived (using harmonic mean) for the bag in question. This iterates for all bags. In the end, the test string is matched with the pattern whose bag has the least average distance. The whole algorithm is explained thoroughly in Section 2.

Of course, the existence of a cosine distance implies the existence of a distance threshold, which would become an extra parameter of the algorithm. This makes sense, since even if a test sample string has the least mean cosine distance, for example, being 1.8 out of 2 (the maximum cosine distance) that doesn't make it a valid match since it is objectively quite dissimilar. We need a threshold after which a test sample string doesn't fit any of the patterns.

In order to account for this threshold parameter which can have different values per dataset, we formulated a linear regression model with the threshold parameter as dependent variable and different attributes of a dataset as independent variables. Thus automatically giving us estimations of the threshold parameter, and ridding us of the need to adjust it for every dataset.

Then, we go over the appearance of chart patterns by type of asset and time frame, using the pseudocode that detects patterns using SAX transformation. After the training bags have been filled, we test our algorithm on test datasets.

The data used are oldest available historical prices of daily frequency of Stocks, ETFs, Foreign Exchange Rates (FOREXs), Mutual Funds and Cryptocurrencies, from Yahoo Finance.

Overall our findings show great accuracy for most datasets, for two time windows of 30 and 60 days, even when compared to existing machine learning methods. Therefore showcasing how effective SAX transformation with mathematically-simple methods can be, while still allowing room for improvement over current model architecture.

Our goal was to develop a new approach to chart pattern classification that is as efficient, if not more, than the existing literature. Justifiably, since such a method of feature extraction has not been used before on classification of financial chart patterns, and deserves to be compared to already highly developed existing methods. Furthermore, we are troubled by a certain lack of literature about machine learning on financial chart pattern classification, especially at the time the research question was posed. Considering the excess of research on finance using Artificial Intelligence, this sector, while quite open to applications, appears to be overlooked. We aspire to encourage more Machine Learning-oriented research on the topic with this paper.

The rest of the paper is structured as follows: Section 2 contains a review of existing literature concerning the main academic subject. Section 3 describes the necessary background on Financial Chart Patterns and SAX transformations. Section 4 describes the CPC algorithm created for this research. Section 5 presents the research methodology and data used. Section 6 describes the results and findings, while Section 7 concludes.

## 2. Relevant literature

First, Bagnall et al. (2017) greatly summarized time series classification algorithms for general problems. When it comes to pattern extraction, a part of the procedure followed in this paper, the rule-based approach used in our data sets is inspired by Yuqing Wan (2017), who created rules for real-valued data. Also, Chung et al. (2002) introduced the concept of Perceptually Important Points (PIPs) that Chen CH. (2013) used to solve various time series procedures including classification. Liu and Kwong (2007) used a novel algorithm

called PXtract to identify chart patterns from real-valued data using wavelet multi-resolution analysis and radial basis function neural networks. Recently, Zheng Y. (2021) introduced a different approach to extract chart patterns, based on hierarchical shape-features rather than rules. Cartwright et al. (2021) introduced the Matrix Profile algorithm for time series applications.

When it comes to artificial intelligence, most recent advances applied Deep Learning Neural Networks. Liu et al. (2021) reviewed basic machine learning algorithms for various time series tasks, including classification. Karmelia et al. (2022) used Feedforward Neural Networks to classify candlestick chart patterns, while Liu and Si (2022) used 1D Convolutional Neural Networks (CNNs), and so did Jearanaitanakij and Passaya (2019) and Kusuma et al. (2019). Lin et al. (2021) implement a pattern recognition method for informed stock trading using one-day candlestick patterns and a set of simple machine learning methods. Hu et al. (2019) used another ensemble of machine learning methods for candlestick pattern classification. Xu (2021) used a similar method of machine learning models and neural networks for candlestick pattern classification. Finally, Hung and Chen (2021) used a CNN and an Autoencoder to extract features from candlestick patterns and a Recurrent Neural Network (RNN) to make prediction on stock prices.

## 3. Theoretical background

### 3.1. Financial chart patterns

Chart Patterns are distinct motifs found in price charts of financial assets, able to assist professionals on predicting the future trend of an asset's price, as explained in the Encyclopedia of Chart Patterns where each pattern is associated with a probability of an upward or downward breakout. In this way, they are separated into continuation (the price continues with its current trend) or reversal (the price reverses to the opposite direction) patterns. They can also be separated into bullish (upward) or bearish (downward) behaviours.

When analysing a chart pattern, we look for specific graphical features that help us distinguish them amongst the chaotic, Brownian-esque movements observed in financial time series. One of these features is called the *neckline*, which is a horizontal line that signifies a level of support or resistance, and connects two or more swing lows or highs. Inclining lines do indicate levels of support or resistance in chart patterns, and in a way funnel the chart between them.

In this paper, we will include most of the basic chart patterns, those that can be distinguished when transformed into strings of characters through SAX. Keep in mind that some patterns do look very similar to others, making them hard to include into this first attempt. Out of 53 chart patterns we will be classifying 32 of them, although the SAX approximation does combine some variants into one, specifically in the case of double bottom and double top patterns.

Below we showcase the patterns included in this paper. Another form of representation of financial charts are candlestick charts. Candlestick charts, unlike typical bar or point and figure charts, display the size of change in an object's price, and the direction of the change (up or down) in a green or red colour, respectively. They are very common for day-to-day trading, and have distinct patterns of their own. In a candlestick, the bar signifies the open and close, while the sticks signify the high and low price.

It is important to note that the complex version of the Head and Shoulders patterns are not explicitly included in the approach, but some of them might get classified as well, depending on their shape. The candlestick chart representations are taken from the *Encyclopedia of Chart Patterns* by Bulkowski (2022).

### 3.2. SAX transformation

Initially, we have to transform all of our data into strings of characters using the Symbolic Aggregate approXimation algorithm with a sliding window. A short explanation of its function is in order (see Fig. 1).

For a sequence **S** of real-valued data, the algorithm performs a Piecewise Aggregate Approximation (PAA), essentially breaking **S** into a number of approximately equally sub sets on the x-axis defined by input, and computes the aggregate of each sub set. Then, it denotes a character on each sub set, depending on each value level on the y-axis. Usually the characters used are from the Latin alphabet (a being the lowest, then b, then c etc) and their range, meaning the value levels, are defined by input. Therefore, sequence **S** is now a string of words, reducing dimensionality and noise. This procedure is visualised in Fig. 2.

## 4. The CPC-SAX algorithm

The CPC-SAX algorithm was created specifically for this paper, but follows a very simple procedure.

We start by using SAX with a sliding window function. For time series **T**, with $N$ points, we define a window of size $M$. Starting in the beginning of **T**, the window transforms a set $S_1$ with SAX, and then "slides", or moves by a number of positions $A$ into set $S_2$ which has dropped the first $A$ points of $S_1$, and added the next $A$ points of time series **T**. This goes on for the entire time series, which reduces its length from $N$ to $N\text{-}M + A$. In this paper, the move size of the window is 1, and the window size will be 30 and 60, representing days, since our data points are Adjusted Close Prices for each day. We will look for chart patterns in different time windows, for a position trading standpoint, though chart patterns can appear in day-to-day, and in buy-and-hold trading.

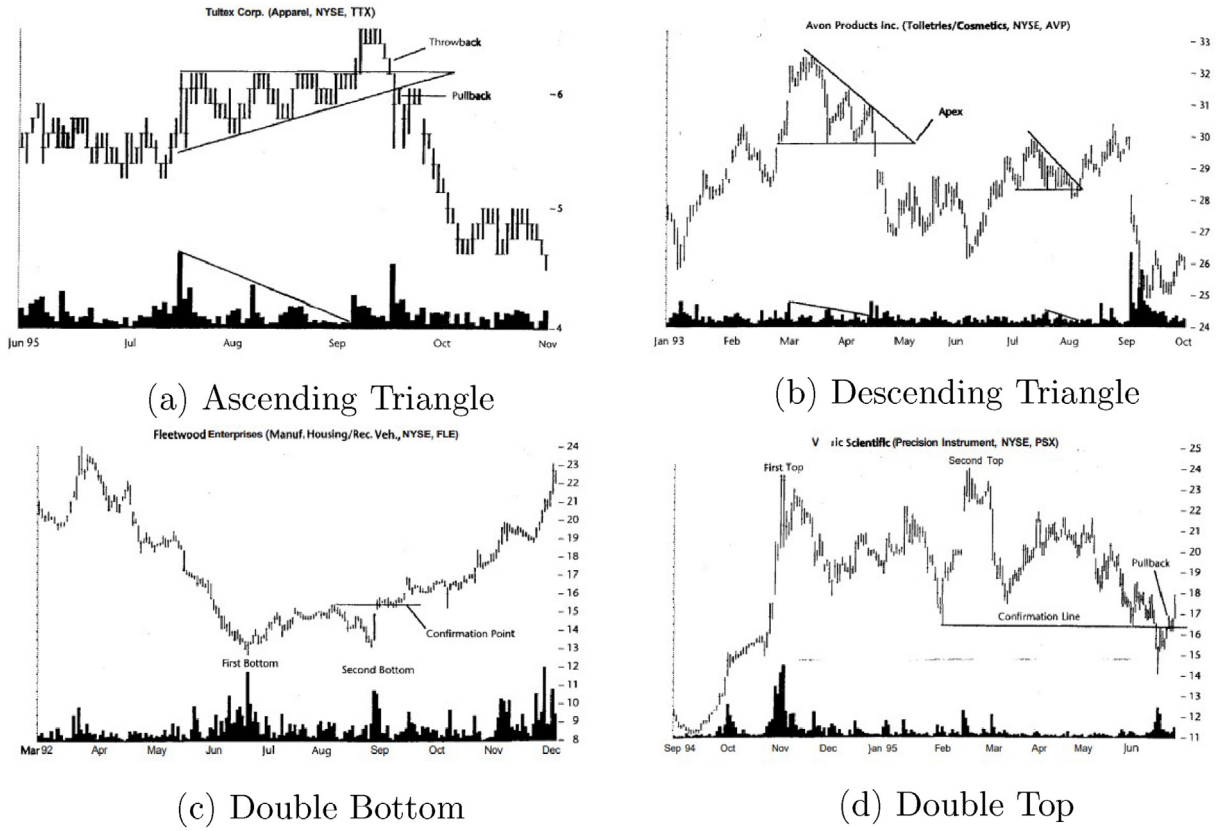



**Fig. 1.** Part of the patterns included.

For transforming the datasets, a GitHub repository called saxpy (Nathan Hoffman, 2018) was used.

Afterwards, the novel algorithm calculates the difference of ASCII values for corresponding characters in a string, creating a 'movement' vector, as displayed in Algorithm 1. Since some position are more important for pattern discovery, eg. what happens in the middle of the window for set $S_i$ is more important than what happens at the edges, we have assigned a normalized weight vector, with a different weight value for each position.

The code created has different weight vectors. In this paper the "centroid" vector was used, where $\mathbf{W} = [0.10, 0.15, 0.25, 0.25, 0.15, 0.10]$, for strings of length seven.

Assume a problem with $N$ different bags of words $B_1, B_2, \ldots, B_N$, all with a different label, and $M$ unlabelled strings $W_1, W_2, \ldots, W_M$. The algorithm takes the first word $W_1$, and performs a "scan" for each string of each bag $B_i$.

When comparing two vectors (one labelled and another unlabelled), the weighted cosine distance is calculated, as in Algorithm 2. If it is zero, the strings display the same pattern & the unlabelled string is given the same label as the labelled one. That can also mean that they express the same pattern for different levels of prices on the y-axis. For example, *aba* and *bcb* are symmetrical (they express the same up-down movement), so are *bcbd* and *cdce* and *abac*. Using movement vectors, we disregard the price level of each string. If the above does not occur, then the cosine distance is added up to a sum, and the next string in the bag is examined. If a string isn't labelled until the end of the bag $B_i$, the harmonic mean of the sum of cosine distances is calculated. If the mean is less than a given threshold $Z$, then the unlabelled string is given the label of bag $B_i$. If not, the algorithm proceeds to the next bag.

This continues for all bags. If at the end, the string remains unlabelled, then it is labelled as None, meaning it does not represent any pattern, and the algorithm continues to the next word. This happens for all unlabelled words $W$.

**Algorithm 1** Creating the movement vector of a SAX string

**Require:** *string*
**Ensure:** Return a list type containing the movement vector of input

```
function MOVE VECTOR
    for all characters in string: do
        add order(string[i+1])-order(string[i]) to list
    end for
    return list
end function
```

A function was created that extracts patterns with a bag of words as an input.

After transforming real-valued data with SAX, they are split into a training and a test set. The training set is simply put through the pattern extracting pseudocode which uses the rule based approach to put each string to an array (or bag of words), if it fits the criteria for a pattern, and returns these bags-of-words, one for each pattern. We give priority to more complex variants, which have more strict restrictions. For example, an ascending triangle is more likely to be mistaken for a rising wedge since the latter's conditions are more lax, and their movement is similar. Therefore triangles go first. It is very unlikely for the opposite to happen, with the rules/conditions in place.

If we input the training part of each dataset, the output are labelled bags-of-words that shall be used to make predictions on the test set. For datasets of over 1500 samples, the train/test split was 80/20, while in the other case it was 65/35.

The prediction function takes as input the labelled bags, the unlabelled test set, already transformed by SAX, while the threshold hyperparameter, the word and alphabet size were part of the initialisation of the Python 3.x class CPC-SAX, which contains all the algorithms listed in this paper.

It is important to note that the prediction process has two features. One is called *priority* which, if stated as true, sorts the labelled bags of words by length, in order to speed up the prediction process (by probability). The other is called *make up*. If a bag has no words, which means that a pattern wasn't extracted at all during training, it is extended with three template strings that fit the corresponding rules. In this way, we make sure the prediction algorithm has the samples it needs to find all patterns. If a bag was empty, it would be impossible for any algorithm to detect a corresponding pattern, since it (the “machine”) would not have “learned” anything relevant.

In our case we have thirty three labels therefore thirty three bags-of-words. We take the output of the pattern extraction algorithm as a tuple, and use it as an input on the prediction algorithm. The output of the prediction function is a list, containing the name of the pattern found in the respective position of a string in the test set (if the string doesn't correspond to a pattern, in its place is the string “None”). These are the predicted labels, with the procedure displayed in Algorithm 3.

Then the test set is used as an input in the pattern extraction algorithm, but the output is again a list with the corresponding labels of each string. These are considered the true labels.

Finally, we use different metrics such as an accuracy score to judge our results.

**Algorithm 2** Calculate Distance between Two Words

**Require:** *word 1: string*, *word 2: string*
**Ensure:** Return valid weighted cosine distance between vectors

```
function DistanceCalculate
    w ← [0.10, 0.15, 0.25, 0.25, 0.15, 0.10]
    if len(word1) ≠ len(word2) then
        raise Error
    end if
    v1 ← move vector(word 1)
    v2 ← move vector(word 2)
    d ← cosine distance(v1, v2, w)
    return d
end function
```

**Algorithm 3** Predict Labels for Test Data

**Require:** *train data: list*, *test data: list*, *threshold: float*
**Ensure:** Return test labels in valid format

```
function PREDICT
    ytest ← test data
    ytrain ← train data
    testlabels ← []
    if make up == True then
        for y in ytrain do
            make up(y)
        end for
    end if
    for i in range(0, length(ytest)) do
        a ← 0
        min ← threshold
        lab ← 0
        for traindata in ytrain do
            d ← []
            for j in range(0, length(traindata[1])) do
                if move vector(ytest[i]) == move vector(traindata[1][j])
then
                    test labels.append(traindata[0][0])
                    a ← a + 1
                    break
                else if traindata[0][0] ≠ None then
                    d.append(distance calculate(ytest[i], traindata[1][j]))
                end if
            end for
            if a ≠ 0 then
                break
            else if length(d) == 0 then
                continue
            else if harmonic mean(d) < min then
                min ← harmonic mean(d)
                lab ← traindata[0][0]
            end if
        end for
        if lab ≠ 0 and a == 0 then
            test labels.append(lab)
        else if lab == 0 and a == 0 then
            test labels.append(None)
        end if                          11
    end for
    return test labels
end function
```

In this section only the most basic functions were included. All of the code is stored in this GitHub repository.

## 5. Methodology

We have to explicitly state the rules used for extracting patterns from the transformed strings of words. These rules compare the characters of each string in a boolean fashion, looking for "rough" representations of each pattern. This is why more complex patterns were hard to include in this paper. We set the size of each string to 7, since most patterns are composed of 7 or 5 PIPs, and the alphabet size to 6. This was done mostly intuitively, as a way to reduce noise but not over-aggregate and lose features for the patterns. By reducing the alphabet size to 3, we received very high prediction accuracy, but this probably attributed to the patterns being oversimplified.

For string **S** of length seven, examples of the rules used are in Algorithms 4, 5, 6 and 7.

**Algorithm 4** Bump-Run-Reversal-Tops

**Require:** A list $S$ of length 7
**Ensure:** True if $S$ satisfies certain conditions, False otherwise

```
function BUMP_RUN_REVERSAL_TOPS(S)
    if S_0 ≤ S_1 and S_1 < S_2 and max(S) = S_4 and S_3 > S_2 and S_6 ≤ S_1 then
        return True
    else
        return False
    end if
end function
```

**Algorithm 5** Cup with Handle

**Require:** A list $S$ of length 7
**Ensure:** True if $S$ satisfies certain conditions, False otherwise

```
function CUP_WITH_HANDLE(S)
    if min(S) = S_2 and S_0 > S_1 and S_0 > S_3 and S_4 > S_5 and S_6 > S_5 then
        return True
    else
        return False
    end if
end function
```

**Algorithm 6** Rounding Bottom

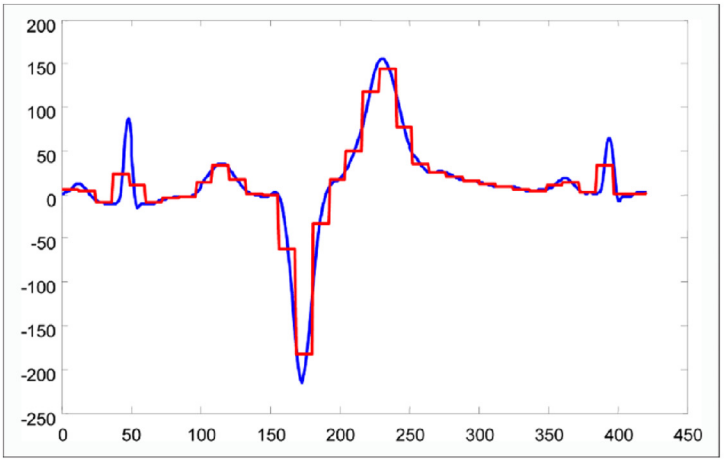
**Require:** A list $S$ of length 7
**Ensure:** True if $S$ satisfies certain conditions, False otherwise

```
function ROUNDING_BOTTOM(S)
    if S_5 = S_1 and min(S) = S_3 and S_2 = S_4 and S_0 ≥ S_1 and S_1 > S_2 and S_6 ≥ S_5 then
        return True
    else
        return False
    end if
end function
```

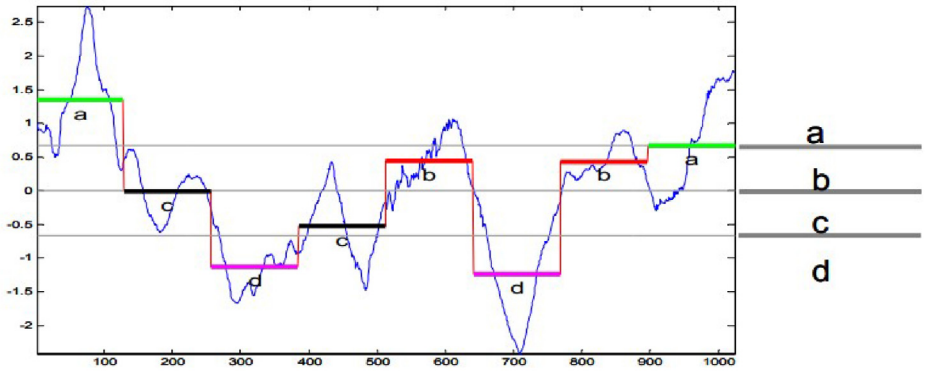
**Algorithm 7** Head and Shoulders-Up

**Require:** $S$, a list of 7 numerical values representing prices
**Ensure:** True if the pattern is a "head and shoulders" pattern, False otherwise

```
if max(S) == S[3] and S[1] == S[5] and S[2] == S[4] and S[1] > S[0] and S[1] > S[2] and S[6] < S[5] then
    return True
else
    return False
end if
```




(a) Example of PAA on a time series. This is the first step of any SAX transformation.




(b) A time series dataset of electrical consumption (of length 1024) is converted into an eight-symbol string "acdcbdba." Note that the general shape of the time series is preserved, in spite of the massive amount of dimensionality reduction.

**Fig. 2.** Visualisation of the SAX transformation.

We start the procedure by creating a linear regression model to estimate the threshold parameter. The value of the parameter is the target variable, and is dependent on the size, mean, and standard deviation of each data set. We used the Bayesian Information Criterion (BIC) to judge the model. This was performed on 63 data sets, of which 24 were cryptocurrencies, 5 were FOREXs and the rest were stocks. After using the CPC algorithm on the training sample of the datasets in hand, where the sample is used to create bags of strings, we use our testing sample to check the accuracy of the CPC algorithm. Meanwhile, we test for changes in percentages of train/test splitting, and we obtain the runtime of the algorithm and the error rate by dataset size.

Finally, we extract all patterns from the same entire datasets using the label pseudocode, to examine the appearance of patterns for different time periods.

## 6. Results

First of all, applying a linear regression model on all assumed variables, $y \sim size + mean + variance$, we obtain that not all variables fit well with the target sample. This is why by using a stepwise selection method for the linear regression model, we obtain the following:

$$y = 6.959151e-02 + 4.885355e-06*size$$

with a BIC score of $-426.42$ and a Residual Sum of Squares of 0.063485.

The main assumptions need to hold, and indeed we observe that the mean of the residuals of the model is $5.31e-19$ which approximates to zero. Also the variance of the model is 0.001023. The model's residuals do pass the Jarque-Bera normality test ($p = 0.17$), as well as multiple Goldfield-Quandt heteroskedasticity tests for all 0.1 intervals ($p = 0.11$–$0.17$), therefore it is a suitable enough as a regressor. It is worth mentioning that the deviations in terms of BIC score between this and the null model is slim. It would make little difference if we used either of them. However, we need the optimal solution and thus we will proceed.

The above model is used for the classification of chart patterns of different sizes. By applying the algorithm for 30 and 60 day time windows, we obtain the following average metrics (Accuracy Score and Matthews' Correlation Coefficient-MCC):

By observing Table 1 and Appendix I, we notice over 85% accuracy for nearly all datasets and categories, and satisfying values for Matthews' Correlation Coefficient, or $R_{33}$ statistic since our problem has multiple labels, in most cases. The reasoning behind choosing the latter metric is its ability to rank our classifier ($+1$ as perfect prediction, 0 as no better than a random guess, $-1$ as total disagreement between prediction and observation).

We notice stocks have the highest level of accuracy, followed closely by FOREXs, then Mutual Funds, with ETFs and Cryptocurrencies standing at the lowest levels. This has to do with what patterns are more likely to appear for each asset (more/less complex), as well as over what time periods these are formed.

Also, we notice that the 60-day time window has slightly better accuracy scores. This could be attributed to the quantization (length) of the window. Which means that different string lengths would be able to capture patterns differently, affecting accuracy for different time windows.

We obtain the error over dataset size plot for the 30-day time frame, shown in Fig. 3. Likewise, for the 30-day time window, we display the runtime for each dataset, in Fig. 4.

Due to the many comparisons and calculations the algorithm makes, it becomes computationally expensive for larger datasets.

Other than performance metrics, it is important to display characteristics of the algorithm. The most important procedure before measuring performance, was regressing the threshold parameter with dataset size and maximum accuracy. The intuition behind this is displayed in Fig. 5.

We notice how the accuracy is optimal at a specific range and then begins to drop off. The difference for parameter value zero (meaning no leniency in predictions) is small but leads to optimal results in nearly every case. This confirms the existence of patterns similar but not identical to the target, and the use of our algorithm. These similar patterns definitely outweigh any loss in accuracy from faulty pattern classification due to leniency in distance measurements.

Another important and arbitrary parameter was the train/test split percentage (since the splitting is done randomly) and how the random states affect accuracy. The results for two different datasets are in Fig. 6.

There are noteworthy fluctuations per state, which increase in amplitude as size decreases, since smaller datasets are more sensitive to these changes.

**Table 1**
Resulting Average Metrics for each Category of Asset, per time window. We observe overall satisfying scores, and a distinct difference in performance between assets, that changes when the time window widens. The MCC is also not tightly correlated to the Accuracy Score, showing a different ranking in the case of the 60-day window.

| Category | Acc. Score-30 | MCC-30 | Acc. Score-60 | MCC-60 |
|---|---|---|---|---|
| Stocks | 0.8950 | 0.8010 | 0.9115 | 0.8333 |
| FOREX | 0.8895 | 0.7916 | 0.9160 | 0.8296 |
| Mutual Funds | 0.8748 | 0.7499 | 0.9218 | 0.8430 |
| ETFs | 0.8626 | 0.7312 | 0.9196 | 0.8470 |
| Cryptocurrencies | 0.8397 | 0.6419 | 0.8953 | 0.7686 |

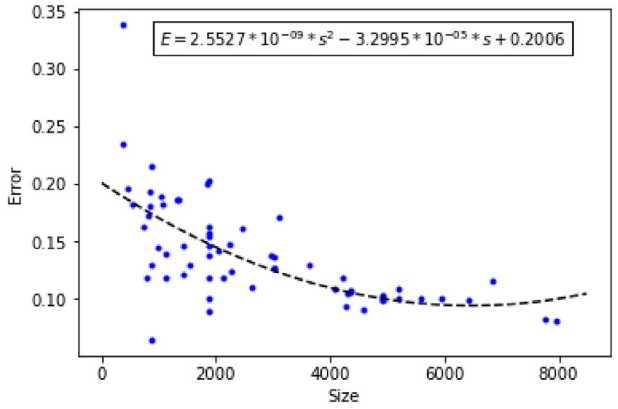



**Fig. 3.** Error for Data Size, with a polynomial fit of the second order displayed. We notice the error has negative correlation with the size of data, that begins to plateau at larger values.

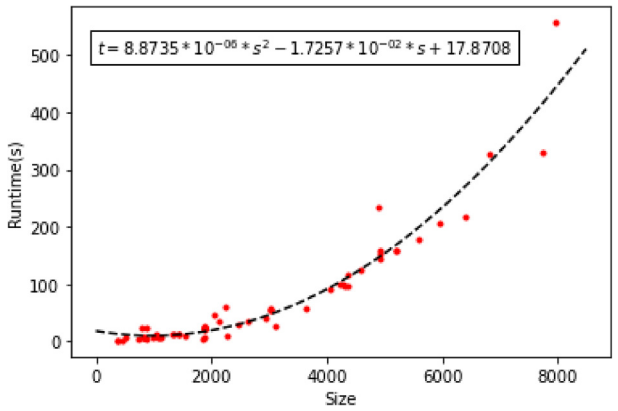



**Fig. 4.** Runtime over Data Size, with a polynomial fit of the second order displayed. Runtime exponentially increases over time, reducing scalability of the algorithm.

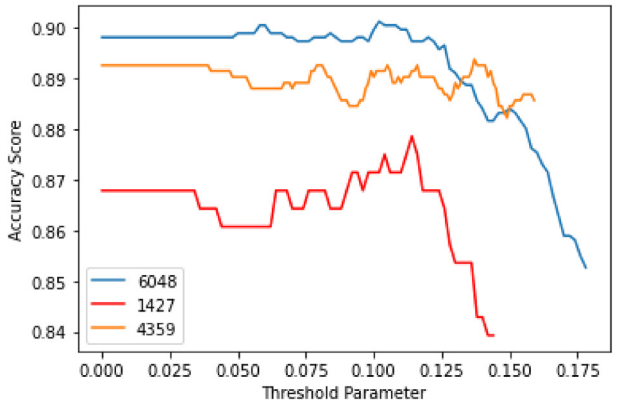



**Fig. 5.** Accuracy Score per Threshold Parameter for Different Dataset Sizes. The use of a threshold parameter gives out at least slightly optimal results. However, the high accuracy at near-zero range indicates many vectors are reappearing.

Last but not least we need to observe the appearance of patterns for different time windows, displayed in Table 2.

We can observe a quality of information: First, that more patterns generally appear on 30-day than 60-day periods, both as an absolute number and as a percentage. Actually Stocks and Mutual Funds display the appearance of more patterns (34,92%/33,62% and 36,00%/34,68%). This can be interpreted in many ways. In reality, it is mainly about the ability of the pseudocode to perform depending on asset and time window, since chart patterns appear on every time frame. It may have to do with how discernible patterns are in each case.

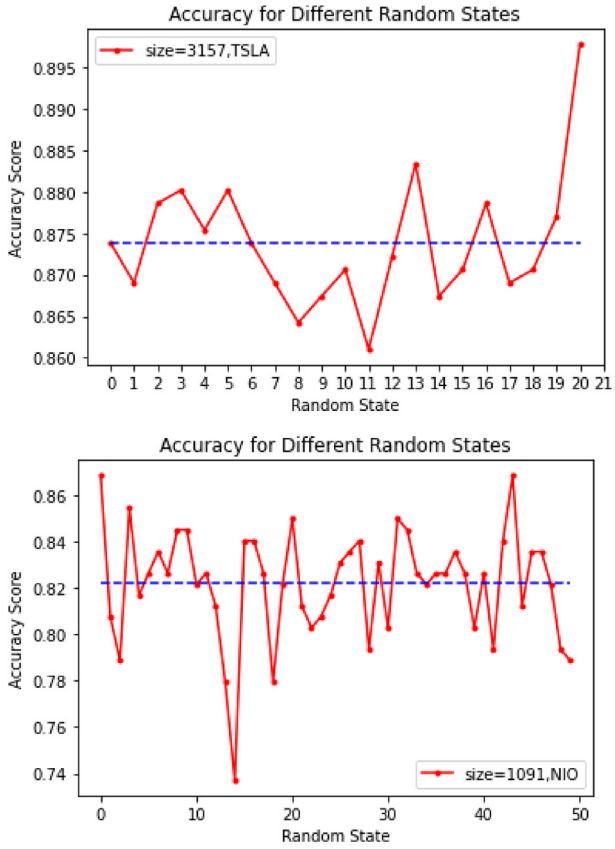



**Fig. 6.** Accuracy fluctuations for changing random states, for two different data sizes. The smaller the data set, the smaller the fluctuations as expected. Different random states do have a noticeable on accuracy.

**Table 2**
Number of Patterns per Asset, per Time Window. It is more important to look at the percentage of patterns per Asset. This displays how capable the extraction pseudocode is for different scenarios.

| Time Window | 30 | 60 |
|---|---|---|
| **Stocks** | | |
| Total Patterns | 43,081 | 41,275 |
| None | 80,285 | 81,490 |
| **ETFs** | | |
| Total Patterns | 23,966 | 23,190 |
| None | 43,196 | 43,370 |
| **Mutual Funds** | | |
| Total Patterns | 21,054 | 20,125 |
| None | 37,429 | 37,907 |
| **FOREXs** | | |
| Total Patterns | 17,969 | 16,176 |
| None | 34,311 | 35,804 |
| **Cryptos** | | |
| Total Patterns | 5947 | 5640 |
| None | 11,714 | 11,721 |

Finally we can compare our model's results with existing literature. Each model's potential is displayed in Table 3, and we must note that all of them use historical daily stock price data from different markets and all used candlestick patterns.

The above and our average metrics show that our method can achieve similar or superior results, while using far more simple methods and being one of the few research models to use charts and not candlestick patterns. It may be not the optimal method in this comparison, however Hu et al. (2019) uses synthetic datasets on candlestick patterns to measure accuracy scores, meaning close comparisons are harder to make. Considering its novelty and potential, it is at the very least a worthy start for future research.

**Table 3**
Summary of research results for existing methods, displaying similar performance against far more advanced methods.

| Authors (Year) | Method(s) | Best Result |
|---|---|---|
| Jearanaitanakij and Passaya (2019) | Convolutional Neural Network | Accuracy: 65.62% |
| Kusuma et al. (2019) | Convolutional Neural Network | Accuracy: 92.2% |
| Hu et al. (2019) | Bagging, Random Committee, Random Sub Space, PART, Random Forest, Artificial Neural Network, Support Vector Machine | Accuracy: 95.3% (Random Forest) |
| Xu (2021) | AdaBoost, Random Forest, XGBoost, Multi Layer Perceptron, Convolutional Neural Network | Accuracy: 90.4% (XGBoost) |
| Lin et al. (2021) | Ensemble of Machine Learning methods (Random Forest, Gradient Boosting Decision Tree, Logistic Regression, k-Nearest Neighbors, Support Vector Machine, Long Short-Term Memory) | Accuracy: 91% (k-Nearest Neighbors) |
| Hung and Chen (2021) | Convolutional Neural Network, Autoencoder, Recurrent Neural Network | Accuracy: 82.78% (TX dataset)<br>Accuracy: 67.08% (NI225 dataset) |
| Karmelia et al. (2022) | Feedforward Neural Network | Accuracy: 93%, F1-score: 72% |
| This paper | CPC-SAX | Accuracy: 93.23% (30 days-Stocks), 93.62% (60 days-Stocks), 94.88% (60 days-Overall) |

## 7. Conclusion

All in all, we have created an algorithm capable of taking an input of SAX strings in bags representing chart patterns, as well as unlabelled SAX strings, and running a multilabel classification.

This algorithm should be applied to datasets labelled perhaps more accurately and definitely more completely, with the use of different algorithms like the PXtract algorithm that was cited, or even by human hand if feasible. In that way, our algorithm can be even better tested, since the extracting process is crucial to learning and verification, and the pseudocode implemented has restrictions and conditionalities on the forming of rules and in the order the bags are processed. Even still, since the pseudocode is used on creating both training and test labels, it puts verification on equal grounds of scrutiny and should not affect the efficiency of our model.

Another shortcoming that should be explored is the procedure's inability to include patterns with more intricate details, such as symmetric triangles and gaps/islands, that have breaks of prices that cannot be expressed in approximations easily. Perhaps these gaps could be flagged before the SAX procedure and carried through classification. The quantification of SAX strings could be redefined, if we need to include more intricate patterns.

Also, the model assumes fixed time windows, which can be an obstacle when wanting to process multiple time windows. Finally, computation time increases exponentially for higher numbers of data points. This could be addressed with the use of Graphics Processing Units (GPUs), faster compilers such as Numba or even Cython, a superset of Python with comparable performance to C.

Overall, chart pattern classification still has research topics left to be explored. For example, by predicting the appearance of a specific pattern, one can use the chart pattern theory to have an indication (probability) of whether the price will rise, fall or hold. In another example through Machine Learning, clustering methods can be used to identify patterns.

Concluding, we restate our hope that more researchers will attempt and without a doubt succeed in making advancements on chart pattern classification and Machine Learning.

## Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

## Acknowledgments

The author would like to thank George Tzagkarakis of FORTH Group of Signal Processing for posing the original research question for the author's undergraduate thesis, as well as Fotis Papailias of the Athens University of Economics & Business and King's College London, for refereeing the original form of this paper. The author would also like to thank the Editor and Reviewers assigned to this article, as its current form could not be achieved without their contributions.

## Appendix I. Full Results

**Table 4**
Results of Stocks for different time windows.

| **Title** | 30 days | | 60 days | |
|---|---|---|---|---|
| | Accuracy Score | MCC | Accuracy Score | MCC |
| **TSLA** | 0.8898 | 0.7922 | 0.9097 | 0.8356 |

(*continued on next page*)

**Table 4** (*continued*)

| Title | 30 days | | 60 days | |
|---|---|---|---|---|
| | Accuracy Score | MCC | Accuracy Score | MCC |
| **AAPL** | 0.9121 | 0.8422 | 0.9252 | 0.8642 |
| **AMC** | 0.8625 | 0.7528 | 0.9056 | 0.8104 |
| **F** | 0.9172 | 0.8438 | 0.9272 | 0.8609 |
| **BBD** | 0.8955 | 0.8034 | 0.9047 | 0.8286 |
| **LCID** | 0.7748 | 0.6170 | 0.8857 | 0.7371 |
| **AMD** | 0.9048 | 0.8290 | 0.9362 | 0.8808 |
| **NIO** | 0.8357 | 0.6506 | 0.8696 | 0.7535 |
| **BAC** | 0.9323 | 0.8741 | 0.9290 | 0.8591 |
| **NVDA** | 0.9134 | 0.8336 | 0.9264 | 0.8614 |
| **LU** | 0.8857 | 0.7970 | 0.8586 | 0.7470 |
| **WBD** | 0.8870 | 0.7874 | 0.8897 | 0.7820 |
| **RIG** | 0.8958 | 0.8012 | 0.9224 | 0.8534 |
| **INTC** | 0.9267 | 0.8541 | 0.9246 | 0.8555 |
| **T** | 0.9177 | 0.8311 | 0.9179 | 0.8355 |
| **MSFT** | 0.9130 | 0.8379 | 0.9214 | 0.8554 |
| **AAL** | 0.9017 | 0.8053 | 0.9022 | 0.8023 |
| **SOFI** | 0.8144 | 0.5873 | 0.8242 | 0.7242 |
| **NFLX** | 0.8986 | 0.7966 | 0.9164 | 0.8452 |
| **AMZN** | 0.8982 | 0.8147 | 0.9329 | 0.8731 |

**Table 5**
Results for FOREXs for different time windows.

| Title | 30 days | | 60 days | |
|---|---|---|---|---|
| | Accuracy Score | MCC | Accuracy Score | MCC |
| EURUSD=X | 0.8854 | 0.7784 | 0.8857 | 0.7730 |
| JPY=X | 0.9010 | 0.8127 | 0.9258 | 0.8463 |
| GBPUSD=X | 0.8898 | 0.7847 | 0.9095 | 0.7992 |
| AUDUSD=X | 0.8978 | 0.8142 | 0.9298 | 0.8522 |
| NZDUSD=X | 0.8945 | 0.8102 | 0.9041 | 0.8041 |
| EURJPY=X | 0.8846 | 0.7701 | 0.9268 | 0.8373 |
| GBPJPY=X | 0.8958 | 0.8127 | 0.9003 | 0.7958 |
| EURGBP=X | 0.8941 | 0.7923 | 0.9016 | 0.8040 |
| EURCAD=X | 0.8774 | 0.7587 | 0.9174 | 0.8200 |
| EURCHF=X | 0.8885 | 0.7967 | 0.9317 | 0.8630 |

**Table 6**
Results for Mutual Funds for different time windows.

| Title | 30 days | | 60 days | |
|---|---|---|---|---|
| | Accuracy Score | MCC | Accuracy Score | MCC |
| FELAX | 0.8840 | 0.7784 | 0.9262 | 0.8572 |
| FELTX | 0.8803 | 0.7706 | 0.9216 | 0.8476 |
| FELIX | 0.8794 | 0.7689 | 0.9243 | 0.8534 |
| FIKGX | 0.7799 | 0.6204 | 0.9113 | 0.8269 |
| FSELX | 0.9038 | 0.8278 | 0.9488 | 0.9039 |
| FELCX | 0.8821 | 0.7756 | 0.9289 | 0.8608 |
| BCSFX | 0.8726 | 0.7831 | 0.9211 | 0.8736 |
| BCSVX | 0.8726 | 0.7817 | 0.9155 | 0.8656 |
| FSERX | 0.8274 | 0.6536 | 0.8699 | 0.7058 |
| FREEX | 0.9059 | 0.8329 | 0.9255 | 0.8626 |
| MYIMX | 0.8815 | 0.7481 | 0.9225 | 0.8175 |
| MRIMX | 0.8571 | 0.7256 | 0.8983 | 0.8038 |
| MCIMX | 0.8065 | 0.5815 | 0.9195 | 0.7888 |
| MAIMX | 0.8780 | 0.7361 | 0.9102 | 0.7932 |
| NOMIX | 0.8855 | 0.7890 | 0.9311 | 0.8618 |

**Table 7**
Results of ETFs for different time windows.

| Title | 30 days | | 60 days | |
|---|---|---|---|---|
| | Accuracy Score | MCC | Accuracy Score | MCC |
| BLCN | 0.7959 | 0.6621 | 0.9289 | 0.8742 |
| BLOK | 0.8320 | 0.7150 | 0.8950 | 0.8336 |
| XSD | 0.8772 | 0.7733 | 0.9310 | 0.8559 |

(*continued on next page*)

**Table 7** (*continued*)

| Title | 30 days | | 60 days | |
|---|---|---|---|---|
| | Accuracy Score | MCC | Accuracy Score | MCC |
| KBND | 0.8756 | 0.7536 | 0.8990 | 0.8418 |
| SMH | 0.8887 | 0.8044 | 0.9183 | 0.8437 |
| PSI | 0.8861 | 0.7826 | 0.9243 | 0.8309 |
| CNRG | 0.8068 | 0.6470 | 0.9304 | 0.8714 |
| CHIS | 0.8100 | 0.6118 | 0.8866 | 0.8003 |
| SOXX | 0.8941 | 0.7978 | 0.9280 | 0.8568 |
| EWD | 0.9086 | 0.8255 | 0.9201 | 0.8498 |
| FLTW | 0.8273 | 0.6668 | 0.9177 | 0.8672 |
| EWT | 0.8840 | 0.7767 | 0.9083 | 0.8168 |
| PTF | 0.8855 | 0.7713 | 0.9007 | 0.8158 |
| KCE | 0.9079 | 0.8146 | 0.9237 | 0.8583 |
| PDBC | 0.8719 | 0.7776 | 0.9050 | 0.8173 |
| SPHB | 0.8628 | 0.7303 | 0.8787 | 0.7485 |
| IYW | 0.8995 | 0.8111 | 0.9344 | 0.8799 |
| LIT | 0.8424 | 0.7037 | 0.9269 | 0.8573 |
| KARS | 0.8115 | 0.6315 | 0.9286 | 0.8526 |
| QLD | 0.8973 | 0.8120 | 0.9282 | 0.8735 |

**Table 8**
Results for Cryptocurrencies for different time windows.

| Title | 30 days | | 60 days | |
|---|---|---|---|---|
| | Accuracy Score | MCC | Accuracy Score | MCC |
| BTC-USD | 0.8688 | 0.7666 | 0.9329 | 0.8814 |
| ETH-USD | 0.6912 | 0.4881 | 0.8443 | 0.8314 |
| USDT-USD | 0.8468 | 0.4988 | 0.8934 | 0.5470 |
| BNB-USD | 0.8790 | 0.7655 | 0.8934 | 0.7973 |
| XRP-USD | 0.8737 | 0.7731 | 0.8913 | 0.7877 |
| BUSD-USD | 0.8686 | 0.4415 | 0.8853 | 0.5497 |
| ADA-USD | 0.8629 | 0.7675 | 0.9262 | 0.7893 |
| DOGE-USD | 0.8763 | 0.7790 | 0.8765 | 0.8500 |
| MATIC-USD | 0.8000 | 0.6000 | 0.8421 | 0.7423 |
| SOL-USD | 0.8010 | 0.6550 | 0.9049 | 0.7407 |

## References


Bagnall, A., Lines, J., Bostrom, A., et al., 2017. The great time series classification bake off: a review and experimental evaluation of recent algorithmic advances. Data Min. Knowl. Discov. 31, 606–660. https://doi.org/10.1007/s10618-016-0483-9.

Bulkowski, T., 2022. Encyclopedia of Chart Patterns, third ed. John Wiley & Sons.

Cartwright, E., Crane, M., Ruskin, H., 2021. Financial time series: market analysis techniques based on matrix profiles. Eng. Proc. 5, 35. https://doi.org/10.3390/engproc2021005045.

Chen, C.H., Tseng, V.S., Y, H.e.a., 2013. Time series pattern discovery by a pip-based evolutionary approach. Soft Comput. 17, 1699–1710. https://doi.org/10.1007/s00500-013-0985-y.

Chung, F.L., chung Fu, T., Luk, R., Ng, V., 2002. Evolutionary time series segmentation for stock data mining. In: 2002 IEEE International Conference on Data Mining, 2002. Proceedings, pp. 83–90. https://doi.org/10.1109/ICDM.2002.1183889.

Hoffman, Nathan, Joshua Southerland, S.M., 2018. saxpy-python implementation of symbolic aggregate approximation. https://github.com/nphoff/saxpy.

Hu, W., Si, Y.W., Fong, S., Lau, R.Y.K., 2019. A formal approach to candlestick pattern classification in financial time series. Appl. Soft Comput. 84. https://doi.org/10.1016/j.asoc.2019.105700 doi:10.1016/j.asoc.2019.105700.

Hung, C.C., Chen, Y.J., 2021. Dpp: deep predictor for price movement from candlestick charts. PLoS One 16, 1–22. https://doi.org/10.1371/journal.pone.0252404 doi:10.1371/journal.pone.0252404.

Jearanaitanakij, K., Passaya, B., 2019. Predicting short trend of stocks by using convolutional neural network and candlestick patterns. In: 2019 4th International Conference on Information Technology (InCIT), pp. 159–162. https://doi.org/10.1109/INCIT.2019.8912115.

Karmelia, M.E., Widjaja, M., Hansun, S., 2022. Candlestick pattern classification using feedforward neural network. Int. J. Advance Soft Compu. Appl 14.

Kusuma, R.M.I., Ho, T.T., Kao, W.C., Ou, Y.Y., Hua, K.L., 2019. Using deep learning neural networks and candlestick chart representation to predict stock market. arXiv:1903, 12258.

Lin, J., Keogh, E.J., Wei, L., Lonardi, S., 2007. Experiencing sax: a novel symbolic representation of time series. Data Min. Knowl. Discov. 15, 107–144. https://api.semanticscholar.org/CorpusID:979006.

Lin, Y., Liu, S., Yang, H., Wu, H., Jiang, B., 2021. Improving stock trading decisions based on pattern recognition using machine learning technology. PLoS One 16, e0255558. https://doi.org/10.1371/journal.pone.0255558 doi:10.1371/journal.pone.0255558.

Liu, J.N., Kwong, R.W., 2007. Automatic extraction and identification of chart patterns towards financial forecast. Appl. Soft Comput. 7, 1197–1208. https://doi.org/10.1016/j.asoc.2006.01.007. https://www.sciencedirect.com/science/article/pii/S1568494606000123 (soft Computing for Time Series Prediction).

Liu, L., Si, Y., 2022. 1d convolutional neural networks for chart pattern classification in financial time series. J. Supercomput. 78, 14191–14214. https://doi.org/10.1007/s11227-022-04431-5 doi:10.1007/s11227-022-04431-5. accepted on March 6, 2022, Published on March 30, 2022, Issue Date: August 2022.

Liu, H., Huang, S., Wang, P., Li, Z., 2021. A review of data mining methods in financial markets. Data Science in Finance and Economics 1, 362–392. https://doi.org/10.3934/DSFE.2021020. https://www.aimspress.com/article/doi/10.3934/DSFE.2021020.

Xu, C., 2021. Image-based candlestick pattern classification with machine learning. In: Proceedings of the 2021 6th International Conference on Machine Learning Technologies. Association for Computing Machinery, New York, NY, USA, pp. 26–33. https://doi.org/10.1145/3468891.3468896 doi:10.1145/3468891.3468896.

Ye, L.,K.E., 2011. Time series shapelets: a novel technique that allows accurate, interpretable and fast classification. Data Min. Knowl. Discov. 22, 149–182.

Yuqing Wan, Y.W.S., 2017. A formal approach to chart patterns clas-sification in financial time series. Inf. Sci. 411, 151–175.

Zhang, C., Chen, Y., Yin, A., Qin, Z., Zhang, X., Zhang, K., Jiang, Z.L., 2018. An improvement of paa on trend-based approximation for time series. In: Vaidya, J., Li, J. (Eds.), Algorithms and Architectures for Parallel Processing. Springer International Publishing, Cham, pp. 248–262. https://doi.org/10.1007/978-3-030-05054-2_19.

Zheng, Y.,S.Y.W.R., 2021. Feature extraction for chart pattern classification in financial time series. Knowl. Inf. Syst. 63, 1807–1848. https://doi.org/10.1007/s10115-021-01569-1.