

# Putting the Human in the Time Series Analytics Loop

Shima Imani

University of California, Riverside  
siman003@ucr.edu

Sara Alaei

University of California, Riverside  
salae001@ucr.edu

Eamonn Keogh

University of California, Riverside  
eamonn@cs.ucr.edu

## ABSTRACT

Time series are one of the most common data types in nature. Given this fact, there are dozens of query-by-sketching/ query-by-example/ query-algebra systems proposed to allow users to search large time series collections. However, none of these systems have seen widespread adoption. We argue that there are two reasons why this is so. The first reason is that these systems are often complex and unintuitive, requiring the user to understand complex syntax/interfaces to construct high-quality queries. The second reason is less well appreciated. The expressiveness of most query-by-content systems is surprisingly limited. There are well defined, simple queries that cannot be answered by any current query-by-content system, even if it uses a state-of-the-art distance measure such as Dynamic Time Warping. In this work, we propose a natural language search mechanism for searching time series. We show that our system is expressive, intuitive, and requires little space and time overhead. Because our system is text-based, it can leverage decades of research text retrieval, including ideas such as relevance feedback. Moreover, we show that our system subsumes both motif/discord discovery and most existing query-by-content systems in the literature. We demonstrate the utility of our system with case studies in domains as diverse as animal motion studies, medicine and industry.

## KEYWORDS

Time Series; Similarity Search; NLP

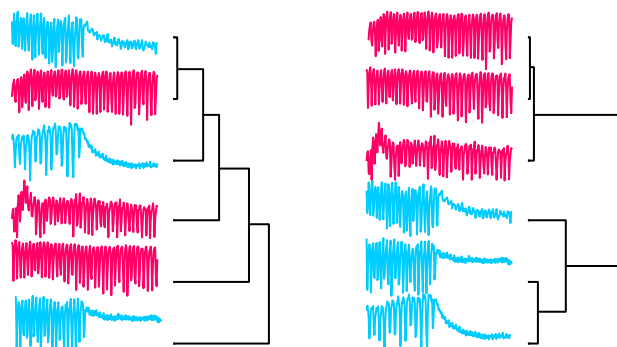
## 1 INTRODUCTION

One of the fundamental questions a data scientist can ask is, “Are there examples of this pattern in my dataset?” [35]. In some cases, the analyst may have a query pattern (query-by-example [9]) taken from a different dataset. In other cases, the analyst may be able to *draw* a pattern of interest (query-by-sketching [39]) or specify it in a query algebra. While there are dozens of systems that implement one of the above ideas, none has seen widespread adoption. One reason may be the complexity of these systems.

For example, time series algebra systems require the user to express their query in formal language that may not be intuitive to a user who is an expert in their domain, but may have little knowledge of SQL or other formal query languages.

In addition, the systems may simply not be *expressive* enough to be useful. For example, almost all query-by-sketching/example systems use some distance measure, typically Dynamic Time Warping (DTW) or a variant thereof as the underlying search mechanism [39]. However, there are simple intuitive patterns that can be easily specified in informal English, but not with any known query-by-sketching/example or time series query algebra

paradigm. Figure 1 shows such an example in an entomological dataset.



**Figure 1: Six snippets from a dataset of behavior of a Beet Leafhopper (*Circulifer tenellus*), an insect pest for many commercially important crops. The behavior shown in cyan is known as Non-Ingestion-D1. *left*) If we attempt to cluster these behaviors with Euclidean distance the result is effectively random. *right*) A clustering based on natural language features is more intuitive (details later in this work).**

If we use any of the cyan patterns as our query, query-by-content will *not* return a semantically similar pattern. However in the natural language space, the term `periodic followed-by constant` produces correct results.

The rest of this paper is organized as follows. In the next section we take the time to develop the reason *why* natural language search is such a promising technology for time series. In Section 3 we introduce the necessary notation and definitions and we introduce our search system in Section 4. We highlight the utility of our system in Section 5 by showing how we exploit the classic text retrieval idea of relevance feedback. In Section 6 we perform an extensive empirical evaluation. We will revisit related work in Section 7. Finally, we summarize our results and outline the future work in Section 8.

## 2 Why Natural Language for Time Series?

Given that there are dozens of query-by-sketching (QbS) [39][37], and hundreds of query-by-example (QbE) [31] systems for time series [13], it is natural to ask why we might also need a natural language query mechanism.

Consider the analogues that Google offers for image search. Google offers both natural language image search and query-by-example (*reverse image search* [40]). In Figure 2 we show the results of a query “barefoot girl riding bike”.



**Figure 2: The top-four results for a Google image search of “barefoot girl riding bike”.**

By any standards, this is an extraordinary successful result. Of the (conservatively) trillions of images that Google has indexed, a vanishingly small fraction of them feature barefoot girls riding bicycles. Nevertheless, this query achieved almost perfect precision (upon careful inspection of the video from which the second image in Figure 2 was culled, the young lady was wearing a pair of incredibly gossamer sandals).

In contrast, Google’s query-by-example is much less impressive. One can upload a photograph as a query and find exact *bit-for-bit* matches. In addition, because its main use is finding violations of copyright, it is also somewhat invariant to the obvious obfuscations someone might do to frustrate such a search, adding/removing a watermark, cropping or slightly changing the aspect ratio or color balance, or as shown in Figure 3.*left*, reversing the image.<sup>1</sup>



**Figure 3: *left*) The top result of a reverse search of the rightmost image shown in Figure 2. Note that the images differ by the addition of a watermark, and by being reversed. *right*) A screen capture from [36] showing a query-by-sketching system.**

While Google image search is able to find slightly corrupted version of images, it cannot find *semantically* similar images. For example, the last image shown in Figure 2 would never retrieve the *other* images shown in Figure 2, even though they are clearly related.

One could imagine a more sophisticated query-by-sketching system. Figure 3.*right* shows one of the most highly cited examples [36]. However, it is not clear that this system would retrieve either the second image in Figure 2 (occluded wheel) or the third image in Figure 2 (the bike is viewed from the rear). Moreover, while most people could draw a bicycle in its canonical side view, clearly drawing and indexing the concept “barefoot girl” is beyond the intended scope of [36], and beyond the artistic ability of most people.

There has recently been an explosion of interest in natural language querying of relational databases. We refer to such work as *QbV* (Query-by-Voice) [14][38][42]. Our proposed work shares a little of the motivation of *QbV*. We are not particularly motivated by the possibility of hands-free access [14] or providing access for individuals with disabilities (while recognizing that as a laudable goal). Our core claim is that in many circumstances, natural language time series search is more expressive than any *QbE* or *QbS* system.

<sup>1</sup> Here the archivist did not seem to realize that if one flips a bicycle image, the chain and chainring appear on the wrong side of the bike.

For example, if a user wants to find a noisy subsequence, she cannot simply draw a noisy query. Figure 1 shows that even if the user carefully selects one of the D1 behaviors as an example, she should not expect it to retrieve the other examples of that behavior. Likewise, no *QbE* or *QbS* systems that we are aware of can retrieve patterns based on a combination of local and global features, for example, retrieving a pattern that is high (relative to the *global* data), and has falling trend (a *local* feature).

### 3 Definitions and Notation

We begin by describing the necessary definitions and notation. The data type of interest is *time series*:

**Definition 1 (Time series):** A time series  $T$  of length  $n$  is a sequence of real-valued numbers  $t_i$ :  $T = t_1, t_2, \dots, t_n$ .

Time series are often *multidimensional*:

**Definition 2 (MTS):** A multidimensional time series **MTS** consists of  $k$  time series  $T$  of length  $n$ , where  $k \geq 2$ .

$$\begin{aligned} \text{MTS} &= \{T_1, T_2, T_3, \dots, T_k\}, \\ \text{where } T_1 &= t_{11}, t_{12}, t_{13}, \dots, t_{1n} \\ T_2 &= t_{21}, t_{22}, t_{23}, \dots, t_{2n} \\ &\dots \\ T_k &= t_{k1}, t_{k2}, t_{k3}, \dots, t_{kn} \end{aligned}$$

$t_{ji}$  and  $t_{ki}$  are two points cooccurring and  $j \neq k$ .

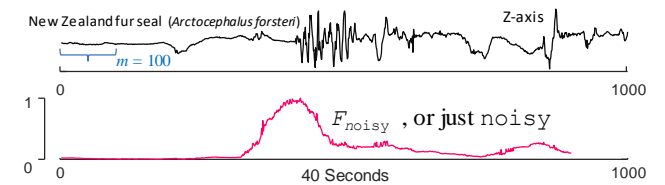
In many cases, the **MTS** and all the constituent time series have mnemonic names, for example a Body Area Network might be:

$$\text{BAN} = \{\text{ECG}, \text{Temperature}, \text{Glucose}, \text{O}_2\text{Saturation}\}$$

Where appropriate, we will use such mnemonic names in our examples. In many types of data analytics, in both the single and multidimensional cases, we do not care about *global* properties of the time series, but instead we are interested in the behavior of *local* regions. A local region of time series is called a *subsequence*:

**Definition 3 (Subsequence):** A subsequence  $T_{i,m}$  of a time series  $T$  is a continuous subset of the values from  $T$  of length  $m$  starting from position  $i$ .  $T_{i,m} = t_i, t_{i+1}, \dots, t_{i+m-1}$ , where  $1 \leq i \leq n - m + 1$ .

Our basic plan is to define a set of intuitive features that map each subsequence to a value. As shown in Figure 4, we do this for every subsequence, producing a *word feature vector* or “meta-time series” that describes each time series.



**Figure 4: *top*) A snippet of seal behavior, captured with an accelerometer. *bottom*) A word feature vector  $F_{\text{noisy}}$  corresponding to **noisy** that annotates the raw data. Note that the value of  $F_{\text{noisy}}$  rises as it encounters the noise in the raw time series and falls as it encounters smoother data.**

We call such time series *word feature vectors*:

**Definition 4 (Word feature vector):** Given a time series  $T$  and a function  $X$ , a *word feature vector*  $F$  is meta-time series of

length  $n - m + 1$  that annotates  $T$ , reporting the value of  $X$  for every subsequence of  $T$ .

Each of these word feature vectors corresponds to an intuitive property of a time series. For example, we have  $F_{\text{noisy}}$ ,  $F_{\text{smooth}}$ ,  $F_{\text{linear}}$  etc. To allow a more intuitive use of our system, we will use the English terms such as noisy, smooth, linear etc to refer to the vector, as in noisy, smooth, linear etc.

We need to review one issue that all time series subsequence ranking systems must deal with [29][33], *trivial matches*:

**Definition 5 (Trivial matches):** Given a time series  $T$ , containing a subsequence  $T_{p,m}$ , if  $T_{p,m}$  scores highly on any scoring function, then  $T_{j,m}$  which  $j \in [-m/2, m/2]$  will almost certainly score high on the same function. These spurious high scoring subsequences are *trivial matches*.

To avoid counting trivial matches when finding the top- $k$  matches to a query, we discard some of the patterns using the concept of an *exclusion zone*, a standard practice [29][33].

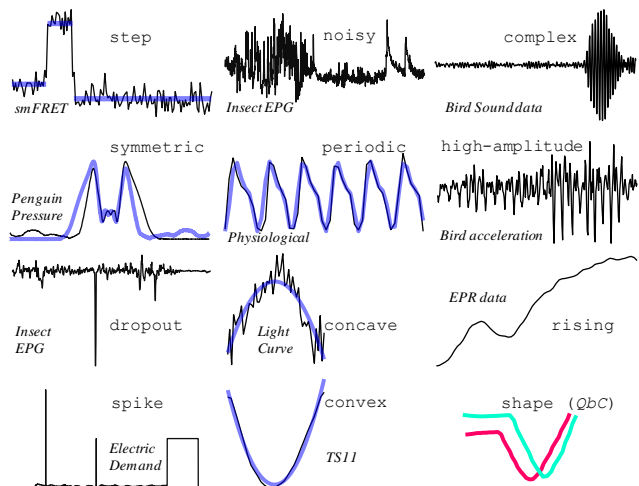
We list and briefly define below a subset of initial set of features we have implemented. Note that in every case the feature vectors are normalized to be between zero and one. Where features come in natural pairs, we define only one. Unless otherwise stated, the negative feature  $X$  can be obtained by simply computing  $X = 1 - Y$ , where  $Y$  is the positive version of the feature.

The formal definitions and code for each feature are relegated to [16] for brevity. We begin with the *local* features:

#### • Local Features

- o rising (falling): The degree to which the slope of the best fit line is positive [10].
- o concave (convex): The degree to which there is a low residual error when modeled by 2<sup>nd</sup> degree polynomial.
- o linear (non-linear): The degree to which there is a low residual error when modeled by 1<sup>st</sup> degree polynomial.
- o smooth (noisy): The degree to which there is a low residual error when modeled by a moving average.
- o complex (simple): Complexity [3] rewards a subsequence for having more peaks, valleys and features.
- o spiky (dropout): Rewards a subsequence for having a small (but non-zero) number of data points much greater than (less than) the mean.
- o periodic (aperiodic): The degree to which there is a low residual error when modeled by two FFT coefficients[13][21].
- o symmetric (asymmetric): Rewards a subsequence for having an axis symmetry about its midpoint.
- o step (no-step): Rewards a subsequence for being modeled well by a small number of segments using a Piecewise Constant Approximation [4].
- o high-amplitude (low-amplitude): Rewards a subsequence for having a high variance.

Figure 5 shows some examples of subsequences from various real datasets that have a high score on one of our features. Note that some of the features can come in two “flavors”, normalized and unnormalized [15][29].



**Figure 5: Subsequences from various datasets that illustrate one of our features. Note that in general, scoring high on one feature does not preclude scoring high on others. For example, Penguin-pressure would also score high on smooth.**

The local features can be computed by looking only at the relevant subsequence. However, there are also some features that can only be computed relative to a global context:

#### • Global Features

- o high (low): Rewards a subsequence for having a relatively high value compared to the overall time series.
- o typical (unusual): Rewards a subsequence for being unusual [6] (a low value in the Matrix Profile [29]).

One issue we may face is the lack of agreement on the correct wording for the features. These differences may be idiosyncratic, or domain informed. For example, the oil and gas community often refers to low-amplitude regions as *flat* or *level*. We can address this with a synonym list. Thus we have:

low-amplitude: flat, level, constant,...

Our thesaurus is currently handcrafted, but there is rich literature in learning thesauruses automatically from the web or from user interactions [20]. Finally, as illustrated in Figure 6 and Figure 7 we support two *special* features:

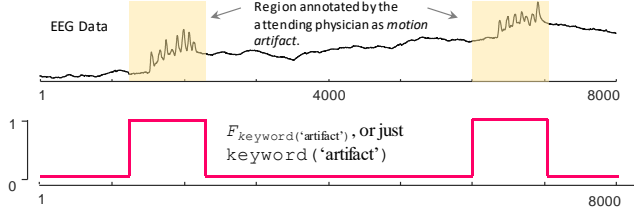
#### • Special Features

- o keyword: For time series data which are annotated by text we reward a subsequence for being covered by that keyword. For example, in medical datasets, keyword(“sleeping”) would reward a subsequence that falls in a region covered by a nurse’s free text annotation “patient is sleeping, skin pale.”
- o shape: For specialized domains we can define a dictionary of shapes and reward a subsequence for resembling them under any given distance measure. For example, shape(ValveFailure) could reward a subsequence which resembles a shape that a user associates with a previous encountered problem [27].

In Figure 6 we show an example of a word feature vector created using a keyword. Here we again support the use of synonyms. For example, in a medical domain we set the value of

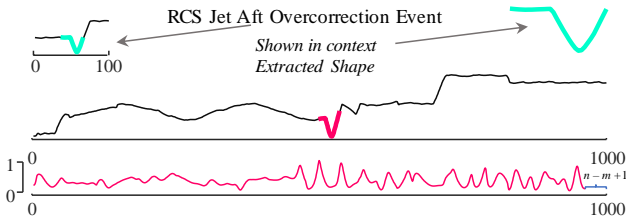
$F_{\text{keyword}}(\text{"jaundice"})$  to be "1" if we encounter "jaundice" or "yellow" or "icterus".

In medical domains, specialized thesauruses have existed for over one hundred years [28], and can now be accessed with APIs. For more general domains, we may have to learn or construct an appropriate mapping [5].



**Figure 6: top) A snippet of EEG data [24] that was hand annotated by a physician. bottom) The word feature vector that was created to reflect the doctor’s annotation.**

Our final feature is shape. Inclusion of this feature allows our system to subsume most existing *QbS* and *QbE* systems for time series [37][31][39]. Figure 7 illustrates the basic idea.



**Figure 7: top) Extracting a specific shape from the STS-57 space mission time series. middle) The time series  $T$  with just the best matching location highlighted. bottom) Computing the shapes distance to all subsequences in  $T$  allows us to compute the *shape*(“overcorrection”) feature which annotates all the subsequences as to how similar they are to the target pattern.**

Here an engineer found an “overcorrection” pattern in one trace that she believes is important enough to be a named primitive in future searches. We can use the MASS algorithm efficiently compute its *distance profile* [15] relative to  $T$ , which is simply its z-normalized distance to every subsequence in  $T$ . Because we use high values to indicate the presence of a feature, we “flip” the distance profile before normalizing it.

In summary we call these word features a *feature-set*. We also provide a dictionary of synonyms for each feature within the feature-set called *dictionary-set*. While we precompute and store these word features, the MASS algorithm to compute the distance profile is so efficient (under a second for  $n = 2^{22}$ ) that we could also support ad-hoc queries[37][31].

### 3.1 The Temporal Operator

In addition to the features mentioned in Section 3, we allow a temporal operator *followed-by*. This operator can be parameterized to include the *maximum* number of data points the user expects feature  $B$  follows feature  $A$ . Suppose we have a time series sampled at 60 Hz. If the user is looking for the highest increase in value that happens in under two seconds, she

could query `low followed-by(120)high`. With no parameter given, it is defaulted to the subsequence length.

Our single operator may seem limited relative to the expressiveness of Allen’s interval algebra [1]. However, consider the following: Google search supports forty-eight advanced search operators. However, even the three simplest and most intuitive Boolean operators are very rarely used.<sup>2</sup> People expect to be able to answer almost any query with just an (implicit) AND operator.

Moreover, in working with experts in domains as diverse as entomology and petrochemical production, we were struck at how universal and natural the *followed-by* descriptor was. When we asked “*How would you recognize an event of type X in your data?*”, almost everyone said something like, “*You will see an A, followed by a little bit of B, followed by a lot of C.*”

The synonyms for our temporal operator are:

*followed-by*: *succeeded-by*, *then*, *next*, ...

The taxonomy above is by no means meant to be complete. The reader may have ideas for additional features which could be added to our framework. However, as we will show, this vocabulary is expressive enough to allow us to find targeted patterns in dozens of diverse domains.

### 3.2 A Worked Example of Defining a Feature

The formal definitions/code for each feature are relegated to [16]. We show some of the introspection and effort that went into defining one feature, *spiky*. Even for a simple feature such as *spiky*, “*the best method to identify spikes in time series is not known.*” [7].

Our first attempt to define the *spiky* feature used the intuition that a spike is an abnormally high value. This can be captured by smoothing the time series, then calculating the residual error between the original and smoothed versions. The high residual error indicates an abnormally high value in the time series. This can be obtained by:

```
ResidualError = T - smoothing(T);
spiky = sum(ResidualError); % Version 1
```

There are several smoothing functions that can be used for this purpose but after testing several smoothing functions we chose `medfilt1`, the built-in function in Matlab.

We realized this definition has several weaknesses. Sequences that most people would judge as noisy but not spiky can achieve a high score. This is because summing the `ResidualError` in one noisy subsequence can have the same effect as the *spiky* subsequence. The simple solution to this problem is taking the *maximum* of `ResidualError` instead of summing them. In Matlab this can be captured by:

```
spiky = max(ResidualError); % Version 2
```

Empirically, we find that this definition is much more robust. Many of the features we implemented required multiple iterations to produce acceptable results. We do not claim that any of our definitions are objectively *correct*. As [7] points out, even for something as simple as *spike*, there may be different definitions that are best for epidemiologists vs. economists etc.

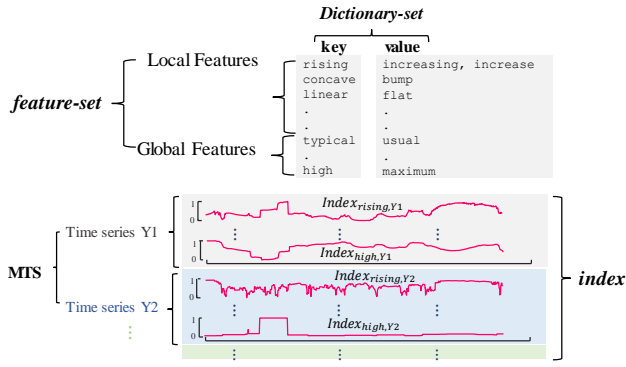
<sup>2</sup> For Google search, AND is implicit, OR needs to be explicitly added, and NOT is represent as -.



#### 4 NLP Time Series Search

Recall that the *feature-set* is a set of local and global word features, and the *dictionary-set* are key-value pair synonyms for each feature within the *feature-set* as shown in Figure 8. For each feature in the *feature-set* there is a corresponding function which takes a multidimensional time series *MTS* and the subsequence length *sub* as inputs and returns a meta-time series *index-feature* (e.g. *index<sub>rising</sub>*).

We are now in a position to explain our system. At a high-level, for each time series, we precompute an *index*, which is an array containing all of the *index-features*, as shown in Figure 8. This *index* can be stored on a disk to await future query sessions. At query time, we begin by *parsing* the query to ensure that it is in the correct format. Then we use a search algorithm to find the top-*k* subsequences that are best matches for the query. Below we explain these steps in more detail.



**Figure 8: top) An illustration of the *feature-set* and *dictionary-set*. bottom) The *index* for the given time series *Y1*, *Y2* etc.**

The inputs to the algorithm *Build-Index* are a multidimensional time series *MTS* of size  $p \times q$  and a user-defined subsequence length *sub*. The output of the algorithm is the *index*, which is a  $p \times (q - sub)$  array normalized between zero and one. The algorithm *Build-Index* is outlined in Table 1 and its output is hinted at in Figure 8.

In line 1, the algorithm initializes an *index* array *Idx* to an empty array. In lines 2 to 5 we calculate the *index-feature* for each feature in the *feature-set*, then add it to the *index* array *Idx*. The *index* array *Idx* is returned on line 6.

**Table 1: The *Build-Index* Algorithm**

<b>Algorithm:</b> <i>Build-Index</i> ( <i>MTS</i> , <i>sub</i> )
<b>Input:</b> Multidimensional time series <i>MTS</i> , subsequence Length <i>sub</i>
<b>Output:</b> Index <i>Idx</i>
1: <i>Idx</i> $\leftarrow$ empty array
2: <b>for</b> <i>feature</i> <b>in</b> <i>feature-set</i> <b>do</b>
3: <i>Index-feature</i> $\leftarrow$ <i>feature</i> ( <i>MTS</i> , <i>sub</i> )
4: <i>Idx</i> $\leftarrow$ <i>Index-feature</i>
5: <b>end</b>
6: <b>return</b> <i>Idx</i>

After building the index we store it on the disk (discussed in Section 4.2). Now we need a valid query to search the index. However, user input can be imperfect, containing spurious

punctuation or misspelled words. Thus, we need to parse the text to produce a valid query. For the ease of explanation, we initially ignore the *followed-by* term in the query.

At query time we break a sequence of characters into pieces called *tokens*. In order to tokenize a multidimensional time series the user needs to specify *which* dimension(s) she is interested in. We resolve this by asking the user to specify the name of the time series and features of interest all in uppercase and lowercase letters, respectively. For example, for the query: TEMPERATURE high spike PRESSURE linear, we parse it into six tokens, with the uppercase letters TEMPERATURE and PRESSURE indicating the time series name and the features [high, spike] and [linear] being the corresponding features for each time series respectively.

For any token, if we cannot find the corresponding feature we search the *dictionary-set* to find the key indicating its synonym as shown in Figure 8. If we cannot find the token in the *dictionary-set* we simply ignore it but echo a warning to the user i.e. “*missing feature bpik*”. Now we are ready to find the top-*k* results.

In the spirit of information retrieval from which we draw inspiration [2], each subsequence can be considered a *document*. After parsing, for each document *j* we calculate the *score<sup>j</sup>*, similar to text mining, as the sum of all the *index-features* within the query.

$$\text{score}^j = \sum_{i \in \text{Query}} \text{Index}_i^j$$

The algorithm *Top-K-Results* for finding the best matches to a query is outlined in Table 2. The inputs include a query *Q* and a user-chosen number of *k* matches. The output is the top-*k* subsequence within the time series of length *sub*.

We initialize an array *score* to an empty array in line 1. We iterate over each feature in the query *Q* and add the *index* for that specific feature to the *score* array in lines 2 to 4. Then we store the total score in *s* which is the element wise summation of the *score*. In lines 6 to 9 we sort *s* in the descending order and find the *k* subsequences, *top-k*, within time series as we are taking care of the exclusion zone. Finally the *top-k* results (i.e. *k* subsequences) are returned.

We can now explain how we handle the *followed-by* case. If the query includes any *followed-by<sub>value</sub>* terms, we truncate the *index-feature* by the desired parameter amount and shift the *index-feature* relative to all other *index-features* in the *index*. If the user does not provide a value, we default to the subsequence length *sub*.

$$\text{followed-by}_{\text{value}} \text{Index} = \text{Index}(\text{value: end} - \text{value} + 1)$$

In order to keep the same size *index-feature* as the original one, we pad zeros to the end of the just calculated *Index-feature*. We then give this modified *index* as an input to Table 2 to calculate the top-*k* best matches.

**Table 2: An Algorithm to Find the Top-K-Results**

<b>Algorithm 2:</b> <i>Top-K-results</i> ( <i>Q</i> , <i>k</i> )
<b>Input:</b> The user defined query <i>Q</i> , number of results <i>k</i> , build-Index <i>index</i> , subsequence length <i>sub</i> , Multidimensional time series <i>MTS</i>

**Output:** Top-k subsequence  $top-k$

```

1:  $score \leftarrow \emptyset, top-k \leftarrow \emptyset$ 
2: for  $q$  in  $Q$  do
3:    $score \leftarrow index(q)$ 
4: end
5:  $s \leftarrow \text{sum}(score)$  // element-wise summation of  $score$ 
6:  $id \leftarrow \text{sort}(s, 'descend')$ 
7: for  $i$  in  $k$  do
8:    $top-k \leftarrow MTS[id_i: id_i + sub]$  //subsequence of length  $sub$ 
9: end
10: return  $top-k$ 

```

## 4.1 Cold Starting our System

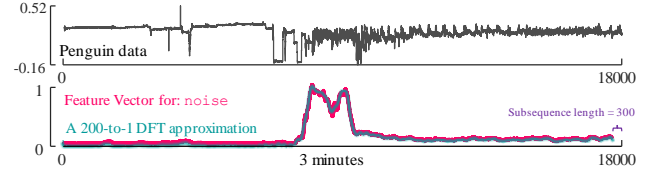
Our system is simply a *text* search, and 54% of the US population performs several text searches each day [22]. Nevertheless a novice user may not have the vocabulary to create an effective query. To solve this cold start problem, we can switch our system into *suggest* mode, in which the user can use her pointer to indicate any location in a time series. Then, the tool instantly displays a sorted list of the feature words that would have (individually) scored the relevant subsequence highly [16]. After a short amount of time interacting with this system, the user can gain intuition as to what *compound* queries may satisfy her query needs.

## 4.2 Indexing for NLP Time Series Search

We have optimized our system for simplicity and scalability. We claim that we do not need to create sophisticated indexes for our representation (for most users). Recall our insect example, recorded at 100 Hz. The longest example is eleven hours long, or 3,960,000 data points. Suppose we extracted all our feature vectors and stored them on disk, and at query time, a user queries “concave spiky”. Including the time needed to load the data from the disk, we can answer this query in under one second using nothing more than the optimized brute force search outlined in Table 2.

However, the *memory* overhead could become a bottleneck. The space overhead is  $O(nk)$ , which for longer time series becomes untenable. There is one piece of low hanging fruit that we can exploit. Recall that most<sup>3</sup> of our features come in complementary pairs. That is to say  $featureA = 1 - featureB$ . Thus, we can store only one of the pair, and when needed, invert its value, effectively halving our memory requirement.

To achieve further compression of the feature vectors we can exploit the rich literature on time series dimensionality reduction [30][23][26]. Most time series are amiable to a significant dimensionality reduction with little error. Many techniques have been proposed, including Discrete Fourier transform (DFT) [30], the Discrete Wavelet Transform (DWT) [23], Piecewise Aggregate Approximation (PAA) etc. Figure 9 show an example of a feature vector approximated with DFT.



**Figure 9: *top*) A 3-minute snippet of time series. *bottom*) The Noise feature vector for the penguin data (red), and its DFT 200-to-1 approximation (blue)**

While different representations can better suit different kinds of data [23], empirical comparisons suggest that on average, there is no reason to favor any approach in terms of reconstruction error [26]. However, different representations have other properties that can be useful depending on how they are deployed. For our system we use the DFT. The DFT representation has the property that it can be very quickly cast back from its reduced representation to the original dimensionality. To concretely ground this with some real numbers, consider the following. Recall our eleven hours/3,960,000 data points of insect data. We compute one feature vector for it, we compress it at a twenty-to-one ratio, and then we save it to a disk. If we are given a query that requires this feature vector, we can retrieve it from a disk, and upsample it (i.e., perform an inverse FFT) in 0.12 seconds. So even if our natural language query involved five keywords, and all data is disk-resident, we can comfortably return the answer in under a second.

Since the number of keywords we currently support is about thirty, and our complimentary observation reduces the number of vectors we must save to about half that, a 15-to-1 compression gives us a space overhead of just  $O(n)$ .

To test the scalability of this scheme, we performed the following experiment. We created, then saved to a disk, all the feature vectors for an increasing longtime series. We used EOG data concatenated from different individuals, because it is a highly variable data source, which typically features a “bit of everything”.

We created queries by randomly selecting four words from our vocabulary, for example {concave, spiky, rising, symmetric}, and measured how long it took to find the single best match. As a point of comparison, we also measured the time needed to perform a one-nearest Euclidean distance neighbor query on the original data, assuming all data is in main memory. For this we used MASS, which is the optimally fast algorithm<sup>4</sup> [15]. We used random queries of length 500, although MASS is essentially invariant to the query length and the data’s structure. Figure 10 shows the results.

<sup>3</sup> Most, but not *all*, spiky  $\neq (1 - \text{dropout})$

<sup>4</sup> This claim needs two qualifications. If we know the query length ahead of time (and its intrinsic dimensionality is small), and we are given a large memory budget, we could build an index that might support faster nearest neighbor queries. In the more general case, for ad-hoc queries whose length is not known ahead of time, MASS is optimal if (as widely believed) FFT is optimal [15].

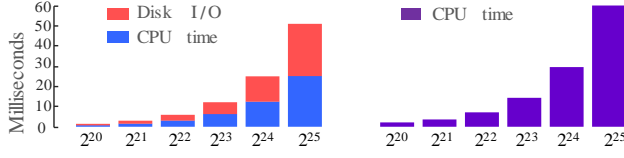


Figure 10: *left*) The time taken to answer natural language queries, with a disk resident index, for increasing long time series. *right*) For the same time series, the time taken to answer a one-nearest Euclidean distance neighbor query on the original data, using the optimal algorithm [15], and assuming that disk access take zero time.

These results show that even if we use a level of dimensionality reduction to bring our space complexity down to  $O(n)$ , we can answer queries for very large datasets in an *interactive* time. Moreover, the time needed to answer queries compares very favorably with the state-of-art query-by-content methods [15].

Given that we can support interactive time queries for datasets with millions of data points, our incentive to further optimize is diminished, as we have a tool that can support the entomologists, medical doctors, marine biologists etc. that motivated this work. Clearly there are potential “customers” for natural language systems that may desire scalability to hundreds of millions of data points and beyond. For data sizes just beyond our current grasp, the low-hanging fruit of parallel computation and/or GPUs offer the possibility for an easy order of magnitude improvement. Beyond that, novel representations and data structures will be needed, perhaps augmenting and adapting work on indexing intervals [11]. We leave such considerations to future work.

## 5 User Feedback: A worked example

User feedback can be incorporated in text search to improve the percentage of the relevant document displayed to the user [2]. One feature of our system is that it is amiable to most text retrieval techniques, including relevance feedback [2][18].

Consider the time series in Figure 11, which represents the behaviors *walking*, *running*, and *skipping* from PAMAP datasets.

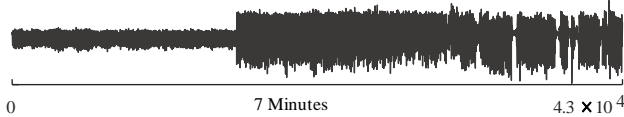


Figure 11: The hand-mounted X-axis acceleration of an individual walking, running, and then skipping rope [34].

Imagine our user wants to find a particular pattern that she believes will discover *walking* behaviors, and she initially searches with: X-ACCELERATION, symmetric, concave, examining just the first top-6 results, as shown in Figure 12.

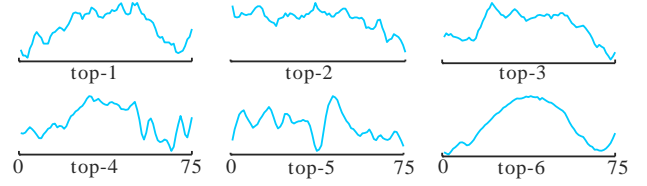


Figure 12: The top-6 results of the user query, X-ACCELERATION, symmetric, concave, on the dataset shown in Figure 11.

In this case, the results are something of a mixed bag. Can we improve them? One way to achieve this is to use Rocchio’s Algorithm [18], exploiting our text-based view of time series [19]. This algorithm revises the original query based on the feedback that the user gives about the relevance of the returned results. The modified query moves closer to the centroid of relevant documents. The algorithm has proven convergence properties, given only the assumption that the relevant and non-relevant documents form clusters and the notion of their centroid is meaningful in certain senses [18][19]. However, empirically it often works even if those assumptions are violated. The formula for Rocchio’s Algorithm is:

$$\vec{Q}_j = (a \cdot \vec{Q}_0) + \left( b \cdot \frac{1}{|D_r|} \cdot \sum_{\vec{D}_j \in D_r} \vec{D}_j \right) - \left( c \cdot \frac{1}{|D_{nr}|} \cdot \sum_{\vec{D}_k \in D_{nr}} \vec{D}_k \right)$$

$\vec{Q}_j$  is the modified query,  $\vec{Q}_0$  is the original query and  $|D_r|$  and  $|D_{nr}|$  are the size of relevance and non-relevance documents, respectively.  $\vec{D}_j$  and  $\vec{D}_k$  are sets of vectors containing the coordinates of relevance and non-relevance documents.  $a$ ,  $b$ , and  $c$  are the hyper-parameters. For instance, setting “ $a$ ” to zero will exclude the original query from the next result. Assume our user chooses just result number 6 as a relevant document. The result of the modified query is shown in Figure 13, with  $a = 1$ ,  $b = 0.8$ , and  $c = 0$ , typically recommend values in the literature [18][19].

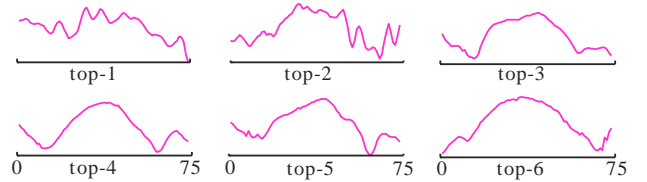


Figure 13: The top-6 results using Rocchio algorithm.

Beyond the subjective visual superiority of these patterns, by checking with the contemporaneous annotations provided with this data, we discover that 4 of 6 results now *correctly* reflect the query purpose i.e. *walking* behavior.

As impressive as this result is, we can improve it even further by using Euclidean distance in our measurement. Recall that an interesting feature of our system is that we subsume most query-by-sketching [39][37], and query-by-example [31] systems by incorporating the shape feature.

In text mining, the Cosine similarity is commonly used to measure the similarity between two documents. An important property of Cosine similarity is its independence from a document’s length. In our system we can compare subsequences

in the same manner as a comparison between documents in text mining. The subsequences are the same length which allows us to use Euclidean distance as a similarity measure [3]. In this case we revise the top- $k$  results by computing the Euclidean distance between a word feature vector of each subsequence and a word feature vector of the centroid of the relevant documents. We modify the result as follows:

$$S^k = (a. \sum_{w_i \in Q_0} FV_{w_i}^k) - b. \sqrt{\sum_w (FV_w^k - FV_C)^2}$$

$k$  is the subsequence index and  $S^k$  is the score for the  $k^{\text{th}}$  subsequence.  $\sum_{w_i \in Q_0} FV_{w_i}^k$  is the sum of feature vector for each word within the original query in subsequence  $k$ .

We calculate Euclidean distance between the feature vector for each word in our dictionary,  $FV_w^k$ , and the average feature vector for the relevant documents,  $FV_C$ .  $a$  and  $b$  are the hyper-parameters for modifying the score. To produce the top- $k$  result we sort  $S^k$ s in descending order, producing the top-6 result shown in Figure 14.

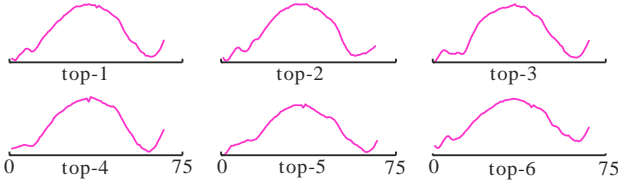


Figure 14: Top-6 results of user defined query using Euclidean distance as a similarity measure.

Not only are the results subjectively improved, but a post-hoc analyses confirms that now *all* these subsequences belongs to the same *walking* behavior.

## 6 Experimental Evaluation

To ensure that our experiments are reproducible, we have built a website which contains all data/code/raw spreadsheets for the results, in addition to many additional experiments that are omitted here for brevity [16]. This commitment to reproducibility extends to all the examples the previous sections. Because the ability to search time series with natural language is new, we first conduct short case studies to help the reader appreciate the utility of this ability. Except where otherwise stated, in the below examples, we set the top- $k$  to reflect the number of true positives in the dataset, to make the numerical evaluation of accuracy easier.

### 6.1 Marine Mammals

We consider a seal behavior dataset [12] which contains twenty-six distinct behaviors of seventy-two seals, belonging to four species. As Figure 15 shows, this dataset contains data from a wearable accelerometer mounted on the seals back.

We must immediately disclaim, none of the authors has any experience or knowledge in this domain. However, as we shall show, *in spite of* our naivety, our system does allow us to ask and answer useful queries. To give the reader some appreciation of

the structure of the data, Figure 15 shows the Y-acceleration Australian fur seal AFS (*Arctocephalus pusillus*).

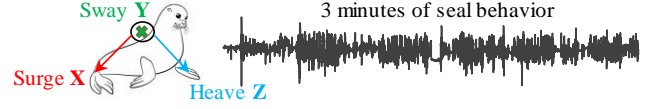
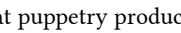


Figure 15: A sample of seal behavior hints at the complex structure of the data.

One of the behaviors observed by the seal’s handlers is *scratching*. Scratching in seal species is described as “perched on fore-flippers with rear on the ground, head down towards rear and one rear flipper scratching neck or head” [12]. As non-experts, it is hard for us to translate this into a query, but we can bypass this issue with the following simple idea.

We watched videos to see examples of this behavior [41] and try to emulate it by “puppetry”, while holding a phone to record the motion (Matlab has an iPhone app that allows you to capture all the phone’s sensors as time series).

After watching some videos of seals *scratching*, our first attempt at puppetry produced the pattern  (~3.5 seconds). We decided to covert this to the text query SWAY periodic and low-amplitude.

The result for this query is shown in Figure 16. The green color is a binary vector encoding the ground truth for the scratching behavior appearing in the time series. The red regions within time series is the result of the scratching query.

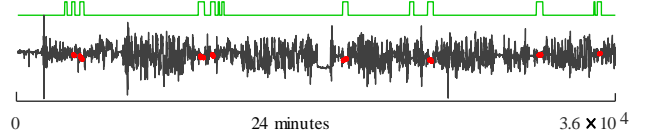


Figure 16: *top*) A Boolean vector representing the scratching behavior (green bar). *bottom*) The time series annotated (red) with the top-13 scratching query results.

Visually the results are very promising. We can objectively score them as follows. We call each separated labeled behavior a *block*. A *success-block* is a block that our algorithm correctly identifies as having the behavior. The accuracy is the sum of all the success-block length divided by the total blocks.

$$\text{Accuracy} = \frac{\sum_j \text{success-block length of labeled behavior}_j}{\sum_i \text{block length of labeled behavior}_i}$$

The accuracy of our method is 78.3%. In contrast, the default rate (random sampling) is just 0.66%

Another behavior in the seal dataset is *shaking*. Shaking is described by the experts as a “short and sharp movement of the head left and right to remove water from the fur” [12]. For this behavior a seal sits on the ground and after a couple of seconds of sudden movement, the seal sits at rest again.

The shaking query can be transcribed as SWAY constant followed-by complexity followed-by constant. Figure 17 shows a search for this within a concatenation of Y-acceleration from eight New Zealand fur seals (*A. forsteri*).



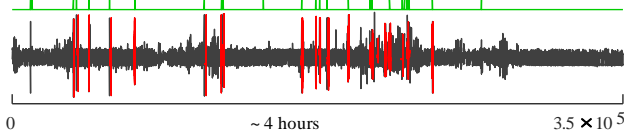



Figure 17: *top*) A Boolean vector representing the shaking (green). *bottom*) A time series annotated with red regions are the result for the top-26 shaking query.

The default rate for finding shaking behavior is just 0.01%, but the accuracy of our proposed method is 80.7%. As this behavior was perhaps the most amenable to shape-based search, we also compared the Euclidean distance similarity measure to our result (i.e. query-by-example [31]), however Euclidean distance achieved just 37.0%.

The final behavior we consider for the seal dataset is a terrestrial gait called *moving*. Moving is described as “*locomotion out of water which four flippers used as legs*”. Here we wish to test the following hypothesis: Given a concrete example of a behavior in one animal, can we construct a query that will generalize to a different member of the same species? We examined the annotated time series of Australian sea lion ASL (*Neophoca cinerea*) to extract this pattern . Using this pattern, we describe the Moving behavior as SWAY constant followed-by rising followed-by falling followed-by constant. Figure 18 shows the result of this query on a different seal.

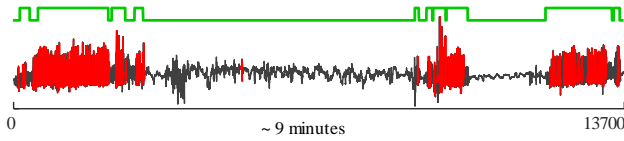


Figure 18: *top*) A Boolean vector representing the moving behavior (green bar). *bottom*) Time series of Y-acceleration of Abbey (sea lion) annotated with red regions are the result for the top-230 movement query.

While the previous experiments use *accuracy* as a metric, here we use the *success-rate*. In the previous experiments each behavior happens once or twice per block. For this experiment each block contains a lot of moving behavior, so observing just one does not explain the whole block. We define success-rate as a number of times we correctly predict a query within the green region divided by the total number of trials. The success-rate for our system is 95.6%. Visually we can see that most of our results are true positives. We also compare our result with two similarity measures: Euclidean distance *ED* and *DTW* [39]. The success-rate of these methods is 20.1% and 30.0%, respectively. Figure 19 visually summarizes this.

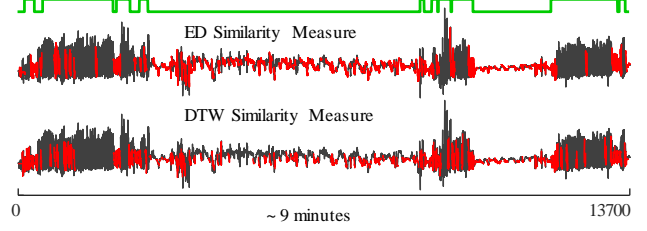


Figure 19: *top*) A Boolean vector representing the moving behavior (green bar). *bottom*) The time series shown in Figure 18. The red subsequences show the top-230 results for ED and DTW are plagued by false positives.

## 7 Related Work

In [17] the authors introduce a shape description language (SDL) with a formal semantics for searching time series. Like our approach, it has an alphabet of primitives, including up, Up, down, Down, Stable etc. However, each primitive is Boolean. That is to say, each subsequence either has the Up property or it does not. In contrast, our approach allows *continuous* degree of membership to Up (we call “rising”), which we argue is more robust and expressive. For example, a query to SDL can return the empty set, whereas we rank every possible subsequence in the dataset.

Beyond the brittleness of SDL, the language itself may be too precise and complex to be useful in many contexts. For example, to find a region with two peaks, our system uses peak followed-by peak, whereas SDL uses [23]:

```
Shape doublespeak (width ht1 ht2) (in width (in
order spike (ht1 ht1) spike (ht2 ht2))))
```

The original SDL paper did not test on real data, and while it is highly cited, none of the citing papers directly implements these ideas. However, a recent paper does implement a similar system called ShapeSearch [25]. The ShapeSearch system has a similarly expressive query language, for example to find “an overall upward trend between  $x$  values 1 to 10 with a peak between 3 to 5.” The user can type:

```
[p{up},xr{1,10}] AND [p{peak},xr{3,5}]
```

Again, we see such precision as being very useful in certain limited domains, but too complex and unforgiving for many.

There are many other interesting (*Qbs*) [39][37], and (*Qbe*) [31] systems for time series [13], but to our knowledge there are none that directly compete with our system.

## 8 Discussion and Conclusions

We have introduced a natural language search system for time series data. Our system is intuitive and easy to use, yet arguably more expressive than any existing query-by-sketching or query-by-example system [37][31][39]. Future work includes *learning* the mappings between the data and the words [5][32] and summarizing the time series [8] using words.

## REFERENCES

- [1] J.F. Allen (1983). Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26(11), 832-843.

- [2] R. Baeza-Yates, and B. Ribeiro-Neto, 1999. *Modern information retrieval* (Vol. 463). ACM Press, New York, NY.
- [3] G.E. Batista, E.J. Keogh, O.M. Tawaf, and V.M. De Souza, (2014). CID: an efficient complexity-invariant distance for time series. *Data Mining and Knowledge Discovery*, 28 (3), 634-669.
- [4] K. Chakrabarti, E. Keogh, S. Mehrotra, and M. Pazzani, (2002). Locally adaptive dimensionality reduction for indexing large time series databases. *ACM Transactions on Database Systems (TODS)*, 27(2), 188-228.
- [5] M. De Rijke. 2017 Apr 3-7. Learning to Search for Datasets. *26<sup>th</sup> International Conference on World Wide Web*. Perth, Australia.
- [6] Anastasia Giachanou and Fabio Crestani. 2016. Tracking Sentiment by Time Series Analysis. In *Proceedings of the 39th International ACM SIGIR conference on Research and Development in Information Retrieval (SIGIR '16)*. ACM, New York, NY, USA, 1037-1040.
- [7] D.E. Goin and J. Ahern. 2018. Identification of Spikes in Time Series. arXiv: preprint arXiv:1801.08061.
- [8] S. Imani et al., "Matrix Profile XIII: Time Series Snippets: A New Primitive for Time Series Data Mining," in *IEEE Int. Conf. on Data Mining (ICBK2018)*, 2018.
- [9] Eamonn J. Keogh, Michael J. Pazzani. 1999. Relevance Feedback Retrieval of Time Series Data. *SIGIR*, 183-190.
- [10] Shoubin Kong, Qiaozhu Mei, Ling Feng, Fei Ye, and Zhe Zhao. 2014. Predicting bursts and popularity of hashtags in real-time. In *Proceedings of the 37th international ACM SIGIR conference on Research & development in information retrieval (SIGIR '14)*. ACM, New York, NY, USA, 927-930.
- [11] O. Kostakis and P. Papapetrou. 2017. On searching and indexing sequences of temporal intervals. *Data mining and knowledge discovery*, 31(3), 809-850.
- [12] M.A. Ladds, A.P. Thompson, D.J. Slip, D.P. Hocking, R.G. and Harcourt. 2016. Seeing it all: evaluating supervised machine learning methods for the classification of diverse otariid behaviours. *PloS one*, 11(12), p.e 0166898.
- [13] Zitao Liu, Yan Yan, and Milos Hauskrecht. 2018. A Flexible Forecasting Framework for Hierarchical Time Series with Seasonal Patterns: A Case Study of Web Traffic. In *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval (SIGIR '18)*. ACM, New York, NY, USA, 889-892.
- [14] G. Lyons, V. Tran, C. Binnig, U. Cetintemel, and T. Kraska. 2016, June. Making the case for query-by-voice with echoquery. In *Proceedings of the 2016 International Conference on Management of Data*. ACM, New York, NY, USA, 2129-2132..
- [15] A. Mueen, K. Viswanathan, C.K. Gupta, and E. Keogh. 2015. The fastest similarity search algorithm for time series subsequences under Euclidean distance. url: [www.cs.unm.edu/~mueen/FastestSimilaritySearch.html](http://www.cs.unm.edu/~mueen/FastestSimilaritySearch.html)
- [16] Project Website: <https://sites.google.com/site/nlptimeseries>
- [17] R.A.G. Psaila, Mohamed Wimmers and E.L. It. 1995. Querying shapes of histories. *Very Large Data Bases. Zurich, Switzerland: IEEE*.
- [18] J.J. Rocchio. 1971. Relevance feedback in information retrieval. *The SMART retrieval system: experiments in automatic document processing*, 313-323.
- [19] G. Salton, A. Wong, and C.S. Yang. 1975. A vector space model for automatic indexing. *Communications of the ACM*, 18(11), 613-620.
- [20] S. Shekarpour, E. Marx, S. Auer, and A.P. Sheth. 2017. RQUERY: Rewriting Natural Language Queries on Knowledge Graphs to Alleviate the Vocabulary Mismatch Problem. In *AAAI*, 3936-3943.
- [21] Milad Shokouhi. 2011. Detecting seasonal queries by time-series analysis. In *Proceedings of the 34th international ACM SIGIR conference on Research and development in Information Retrieval (SIGIR '11)*. ACM, New York, NY, USA, 1171-1172.
- [22] Statista: The statistics Portal. Retrieved from <https://www.statista.com/statistics/250948/frequency-of-google-search-usage-by-us-users/>
- [23] S. Sumathi, and S.N. Sivanandam. 2006. *Introduction to data mining and its applications* (Vol. 29). Springer.
- [24] K.T. Sweeney, S.F. McLoone and T.E. Ward. 2013. The use of ensemble empirical mode decomposition with canonical correlation analysis as a novel artifact removal technique. *IEEE transactions on biomedical engineering*, 60(1), 97-105.
- [25] T. Siddiqui, P. Luh, Z. Wang, K. Karahalios, A. Parameswaran. 2018. ShapeSearch: Flexible Pattern-based Querying of Trend Line Visualizations (Demo). *Vldb'18: 44th Int'l Conf on Very Large Data Bases*, Rio De Janeiro, Brazil.
- [26] X. Wang, A. Mueen, H. Ding, G. Trajcevski, P. Scheuermann, and E. Keogh. 2013. Experimental comparison of representation methods and distance measures for time series data. *Data Mining and Knowledge Discovery*, 26(2), 275-309.
- [27] Sheng Wang, Zhifeng Bao, J. Shane Culpepper, Zizhe Xie, Qizhi Liu, and Xiaolin Qin. 2018. Torch: A Search Engine for Trajectory Data. In *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval (SIGIR '18)*. ACM, New York, NY, USA, 535-544.
- [28] M.B. Wilfred. 1903. A Thesaurus of Medical Words and Phrases. *The Laryngoscope*, 13(9), 736.
- [29] C.C.M. Yeh, et. al. 2016. Matrix Profile I: All Pairs Similarity Joins for Time Series: A Unifying View that Includes Motifs, Discords and Shapelets. *IEEE ICDM 2016*, 1317-1322.
- [30] R. Agrawal, C. Faloutsos, and A. Swami. 1993, October. Efficient similarity search in sequence databases. In *International conference on foundations of data organization and algorithms*. Springer, Berlin, Heidelberg, 69-84.
- [31] H. Hochheiser, and B. Shneiderman. 2002, April. A dynamic query interface for finding patterns in time series data. In *CHI'02 Extended Abstracts on Human Factors in Computing Systems*. ACM, 22-23.
- [32] L. Soldaini, E. Yom-Tov. 2017 Apr 3-7. Inferring individual attributes from search engine queries and auxiliary information. In *26<sup>th</sup> International Conference on World Wide Web*. Perth, Australia, 293-301.
- [33] B. Chiu, E. Keogh, and S. Lonardi. 2003, August. Probabilistic discovery of time series motifs. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 493-498.
- [34] A. Reiss and D. Stricker. 2012, June. Introducing a new benchmarked dataset for activity monitoring. In *Wearable Computers (ISWC), 2012 16th International Symposium*. 108-109.
- [35] Haggai Roitman, Sivan Yogev, Yevgenia Tsimmerman and Yardena Peres. 2013. Towards Discovery-Oriented Patient Similarity Search Health Search and Discovery (HSD) workshop (@SIGIR).
- [36] C. Xiao, C. Wang, L. Zhang, and L. Zhang. 2015, June. Sketch-based image retrieval via shape words. In *Proceedings of the 5th ACM on International Conference on Multimedia Retrieval* (pp. 571-574). ACM, 571-574.
- [37] M. Correll, and M. Gleicher. 2016, October. The semantics of sketch: Flexibility in visual query systems for time series data. In *Visual Analytics Science and Technology (VAST), 2016 IEEE Conference*. IEEE. 131-140.
- [38] D. Chandarana, V. Shah, A. Kumar, and L. Saul. 2017, May. SpeakQL: Towards Speech-driven Multi-modal Querying. In *Proceedings of the 2nd Workshop on Human-In-the-Loop Data Analytics*. ACM. 11.
- [39] M. Mannino and A. Abouzied. 2018, May. Qetch: Time Series Querying with Expressive Sketches. In *Proceedings of the 2018 International Conference on Management of Data*. ACM. 1741 - 1744.
- [40] Google Image Search. Retrieved July 1, 2018, from <https://images.google.com/>
- [41] Walrus Scratching. Video. (31 July 2013). Retrieved July 30, 2018 from <http://www.youtube.com/watch?v=zReIGuJR1J8>
- [42] G. Ren, X. Ni, M. Malik, and Q. Ke. 2018. Conversational Query Understanding Using Sequence to Sequence Modeling. In *WWW '18. International World Wide Web Conferences*. 1715-1724.