Pattern Recognition and Matching in Ice Core Data

Nathan Dunn
University of Maine - Main, nathan.dunn@umit.maine.edu

Follow this and additional works at: https://digitalcommons.library.umaine.edu/honors

Part of the Computer Sciences Commons, and the Mathematics Commons

Recommended Citation
https://digitalcommons.library.umaine.edu/honors/224

This Honors Thesis is brought to you for free and open access by DigitalCommons@UMaine. It has been accepted for inclusion in Honors College by an authorized administrator of DigitalCommons@UMaine. For more information, please contact um.library.technical.services@maine.edu.
PATTERN RECOGNITION AND MATCHING IN ICE CORE DATA

by

Nathan Dunn

A thesis submitted in partial fulfillment of the Requirements for a Degree with Honors (Mathematics, Computer Science)

The Honors College

University of Maine

May 2015

Advisory Committee:
Sudarshan Chawathe, Associate Professor of Computer Science and Cooperating Associate Professor of the Climate Change Institute
Thomas Bellsky, Assistant Professor of Mathematics & Statistics
Andre Kurbatov, Associate Research Professor of the School of Earth and Climate Sciences
Sharon Tisher, Instructor and Lecturer of the School of Economics and Assistant Adjunct Professor in Honors
Roy Turner, Associate Professor of Computer Science
Abstract

The purpose of this research is to investigate the potential of applying concepts from machine learning, such as pattern recognition and matching, to detect climatic signals in ice core data. The main components of this project are the development of a pattern language for expressing relationships between chemical signals over time, a method of tokenizing ice core chemistry data into an easily manageable form, a method of matching specific instances of climatic signals to a specific pattern string, and a method to recognize and evaluate patterns within ice core chemistry data. While there are weaknesses in each of these components, this research serves as a successful proof of concept for the feasibility of applying machine learning techniques to ice core analysis.
This thesis is dedicated to Mom, Dad, Jon, Andrew, Nick, Elizabeth, Catherine, and the first member of the next generation, Skylar.
Acknowledgements

I would like to thank Professors Bellsky, Kurbatov, Tisher, and Turner for serving on my advisory committee. Your feedback was invaluable to this project.

I would like to thank my family, to whom this thesis is dedicated, for their continuous love and support.

I would like to thank my fellow friends in Honors. Our commiserations were just the thing to keep me going when feeling overwhelmed.

Thank you to the grad students involved in this research, particularly Bjorn Grigholm and Mark Royer.

Finally, I would like to thank my advisor, Professor Chawathe. I wouldn’t have gotten half as far as I did without your calm encouragement and wealth of insights. Moreover, thank you for giving me a chance to prove myself. Without that chance, I couldn’t have become a Computer Science major, and this thesis surely wouldn’t exist.
# Contents

1 Introduction ........................................... 1

2 Background and Motivation ................................ 3

3 Related Work ........................................... 4

4 Methods .................................................. 6
   4.1 Pattern Language .................................... 6
   4.2 Tokenization ........................................ 8
      4.2.1 Fully Qualified Token Structure .................. 8
      4.2.2 Chunking ....................................... 10
      4.2.3 Behavior Classification .......................... 11
      4.2.4 Evaluation ...................................... 16
   4.3 Pattern Matching .................................... 16
   4.4 Pattern Extraction and Completion .................... 19
      4.4.1 Pattern Evaluation .............................. 19

5 Implementation ......................................... 22

6 Results .................................................. 27
   6.1 Behavior Classification .............................. 27
   6.2 Seasonal Sample Run ................................ 29
   6.3 Volcanic Sample Run ................................ 31
   6.4 Belukha Pattern Extraction .......................... 31
   6.5 GISP2 Pattern Extraction ............................ 36

7 Discussion .............................................. 39

8 Conclusion .............................................. 41

References ................................................. 42
List of Figures

1  Tokenization Example ................................. 8
2  Fully-Qualified Token Structure ......................... 9
3  Match Example ........................................ 17
4  Anti-Match Example .................................... 17
5  Indeterminate Match Example .......................... 18
6  Classifier Training Pairs Plot .......................... 27
7  Accuracy vs. $k$ ............................................ 28
8  Accuracy vs. $\gamma$ ....................................... 28
9  Accuracy vs. Maximum Weight ........................... 28
10 Tokenized Seasonal Data .................................. 30
11 Tokenized Volcanic Data .................................. 31
12 Tokenization Partition Sensitivity ....................... 40

List of Tables

1  ITASE Sulfate Patterns ................................... 30
2  Belukha Output ............................................ 35
3  GISP2 Output ............................................... 37
1 Introduction

Ice cores provide climatologists with time series data about past climates by trapping various particles or chemical species in the atmosphere in yearly ice layers. These quantities, that is, concentrations of chemical species or other numerical values derived from an ice core, are sometimes associated with specific climatic conditions, such as the δ18O proxy’s association with ambient temperature [22]. The focus of this research is to provide a system that automates a portion of the work done by climatologists in analyzing ice cores, and potentially other time series data sets. The work in question is that of looking for trends and patterns within the data. The intention is to allow scientists to keep up with the rapidly increasing resolution and volume of data produced with new sampling methods [24]. Another potential benefit of this approach is the ability to detect trends that are too subtle to detect manually. One feature of the completed form of this project is the ability for a researcher to specify a data set and a pattern and have the system output time periods where the data exhibit that pattern. Another feature is the ability of the system to take in raw input and then output patterns that are present in the data. This may prompt the discovery of archetypal patterns, that is, patterns whose occurrences are not limited to any particular point in time. It is the hope that these patterns correspond to general climatic processes and will help elucidate the underlying driving forces of climate change. However, at this stage in the research we only hope to demonstrate a proof of concept by implementing a system that will support these features and perhaps return some candidate values, some of which will merit further attention from climatologists. This pattern detection will be achieved through a combination of computer science concepts, such as the creation of a pattern language and the use of machine learning. These techniques will allow climatologists to more easily analyze the patterns within ice cores.

The development of a pattern language is an essential part of this research. The language gives researchers a formal framework to define and specify certain climatic patterns. The language will be very expressive in order to describe a wide variety
of possible characteristics of a climatic pattern. Although this allows for the creation of complicated strings, each string is built with the underlying principle of expressing relationships between events using clauses that specify precursor and successor events. That is, a precursor event occurs, and then a successor event happens within a certain future time frame. For this research, the climatic events of interest are primarily the behaviors of certain quantities in ice cores. We stress that pattern strings make no claims of causation; precursors may or may not have a direct effect on successors, but that is not the focus of the language.

Tokenization is used to simplify and discretize raw ice core data by partitioning time series into fixed-width sub-series, or chunks. These chunks are converted into tokens, which capture high-level features of the sub-series. For our purposes, the most essential feature to extract is the qualitative behavior of the quantity within the sub-series. To this end, we consider 5 types of behaviors: Spikes and Dips (large, rapid increases or decreases, respectively, of a quantity, followed by a return to typical behavior), Increasing and Decreasing trends (steady and significant changing of a quantity over time), and Flat (essentially noise with no discernible trends). We will also use a special indicator value Unknown (UNK) for chunks that are missing significant amounts data, or are otherwise unclassifiable.

Pattern matching is the ability of the system to find specific instances of a pattern string in the data. Tokenization puts the data into a form that closely mirrors the form of a pattern string. The method of matching a string is essentially to look for a token that matches a precursor, and then look for a token that matches a successor within the specified time window. Pattern matching is an essential part of this research, not only because it provides valuable output in its own right, but because it provides a basis to evaluate the quality of a pattern.

Pattern extraction is the ability of the system to look at raw data and find patterns that are present, with relatively little human guidance. A stepping stone to pattern extraction is pattern completion, where one portion of a pattern string is left out and the
system determines which value creates the most valid pattern. Pattern completion could also be a useful standalone feature in that it would allow a researcher to specify a partial pattern based on some knowledge of climatic processes. An essential part of both of these functions is the ability to evaluate patterns quantitatively. This evaluation will be based on a number of components, one of which is how accurately the precursor event predicts the successor event.

2 Background and Motivation

Climate change is a pressing problem that has gained significant awareness in recent years. Many claim that irreversible destruction to the environment will occur without intervention [12]. Paleoclimatologists play an essential role in assessing these threats to the environment. Examining the climate of the past provides important insights into understanding the climate of the present and future [11]. Ice cores are an essential tool used by paleoclimatologists.

Ice cores are cylindrical sections of ice, sometimes miles in length, drilled from glaciers, mountaintops, or any other location permanently in snow. The ice that composes these cores is built up from layers and layers of compressed annual snowfall. Various soluble and insoluble species remain archived in the snowfall composing the layers. The snow is initially porous, which allows it to trap atmospheric gases. This provides atmospheric data that could not be obtained from other paleoclimatic archives [15]. Ice cores are processed by melting small cross sections at a time and analyzing the water for dissolved chemical species and particles.

Climatologists usually receive ice core data in the form of data series composed of (concentration, depth) pairs. Initial processing is needed to transform the depth of a sample to its age (this is not a straightforward linear transformation as the snow layers become thinner deeper in the core due to compression and horizontal flowing) [22]. Climatologists then perform various operations on the data to draw conclusions about climatic conditions of the past.

Another recent factor motivating this project is the advancement of the technology that collects data from ice cores. New
laser ablation techniques allow ice cores to be sampled with sub-millimeter thick layers [24]. This will allow for very large data sets to be gathered from ice cores. These data sets will likely be unwieldly for researchers to process using their traditionally manual methods. The intention is that this project will lay the groundwork for a new tool to help researchers more effectively analyze data

3 Related Work

While ice core analysis and climatology are thriving fields of studies, and machine learning is a technique that has been applied to an incredibly diverse set of tasks, we believe this research is a rather novel union of the two. However, there is much previous work that is closely related to various aspects of this project.

Bringing automation to part of a traditionally “hands-on” process is the ultimate goal of this project. In a similar fashion, other tasks related to ice core analysis have been automated in recent years. Calculating the age of a layer of snow from its position in an ice core is an important part of mapping depth-value pairs to time-value pairs. This is typically done by observing the fluctuations in isotopes of hydrogen and oxygen, indicating periods of high and low temperatures (i.e. seasonal shifts). Traditionally, this has been done through manual inspection and counting of these oscillations. This is now generally an automated process [22].

Machine learning is an incredibly diverse field that has been used to perform many “human” tasks that were previously believed to be impossible for a computer to perform. One such application of machine learning is the development of a system to gauge the physical attractiveness of human faces [14]. Beyond just using machine learning, this work is similar to ours in that it uses feature extraction to detect patterns in a chaotic domain. Even more to the point, the work of Yip uses machine learning to detect patterns in genomes, a domain which is much more akin to the sequential nature of ice core data [27].

One operation that is used in this project is that of spike detection, that is, separating a large impulse from a noisy signal.
Quiroga outlines a technique for detecting spikes in neural impulse signals using statistical techniques closely related to machine learning [21]. Although not as sophisticated, for our purposes of classifying the behavior of a chunk as a Spike or otherwise, we use the K-Nearest Neighbors (KNN) algorithm. KNN is a simple, but powerful supervised machine learning classifier [7]. It works by taking in some training data set consisting of (Feature Vector, Category) pairs corresponding to objects that have been classified by some process. Then, given a new object to classify, the algorithm determines its feature vector, and finds the k objects (for some positive integer k) in the training data that are closest (with respect to some metric) to the new object in feature-space. The k neighbors then each cast a “vote” for their own category. The new object’s classification is determined as the category with the highest number of votes. Votes can be weighted so that neighbors that are closer to the new object have a stronger say. The behaviors we use to describe trends in ice core data, i.e. Spike, Increasing trend, etc. are motivated by similar types of signals defined in *Ice Core Studies of Global Biogeochemical Cycles* [19]. Furthermore, the methods used to process data in order to detect these signals are similar to the features we use in our classification system, such as calculating z-scores and the correlation coefficient [19].

The pattern language used in this work is a domain-specific language used to greatly simplify a problem. This technique is strongly endorsed by Bentley [4].

One crucial assumption that this project relies on is that certain climatic events strongly indicate that another kind of event will follow. This type of behavior is called a precursor event and has been observed for a variety of climatic disturbances and events. For instance, a particularly large volcanic eruption creates high levels of sulfate in the surrounding area as an immediate effect. The eruption also releases much ash and dust that obscure the sunlight, causing a slight cooling period that leaves a measurable impact on ice cores [3]. Iron-rich dust that is blown into the ocean has been associated with explosions in populations of phytoplankton. The increased rate of photosynthesis causes $O_2$ levels to rise and $CO_2$ levels...
to drop, leading to cooling periods [16]. Another, more apparent demonstration of this concept is the existence of trends that vary seasonally, such as the value of the $\delta^{18}O$ proxy [9]. By the nature of the oscillation, a low value preceds a high value, and vice versa.

There are currently system available for processing ice cores. P301dx is one such application that focuses on giving researchers the ability to manipulate data in an intuitive way by providing instant visual feedback. [6] It is the hope that this application will ultimately be integrated into P301dx.

One technique that is used extensively in ice core data analysis is Principle Component Analysis, whereby a signal is decomposed into distinct components. In this context, a time series is additively decomposed into components that each correspond to a different source. For instance, the total sodium concentration series can be broken up into an ocean component (amount of sodium from seawater) and a wind component (amount of sodium from dust deposition). Empirical Orthogonal Function (EOF) Analysis performs a similar task whereby multiple correlated signals are decomposed into orthogonal basis vectors. We mention this here to give some impression of the processing techniques that researchers perform on data. [26] This pre-processing will be an essential first step to perform on data that will be processed by this application.

4 Methods

4.1 Pattern Language

In its simplest form, a pattern string has 3 parts: a precursor clause, a successor clause, and a time window. A clause specifies the attributes of a particular event. We will primarily be concerned with clauses describing a certain behavior (such as an Increasing trend) of a certain quantity (such as calcium ion concentration). This closely links the pattern language and the tokenization process, as clauses and tokens are very similar in form. This allows us to define a token as either matching or not matching a clause. The third attribute, a time window, describes the amount of delay between the precursor clause and the successor clause. As there may in general be a variable amount of time
between occurrences of precursor and successor events, the time window is specified as an interval. Since the tokenization process creates tokens that are equal-width, we can speak of lengths of times in terms of the time span covered by each token. That is, with a token width of 5 years, a timespan of 10 to 20 years would correspond to 2 to 4 token widths.

\[
\text{Precursor } SO_4^{2-} \sim \delta^{18}O / : [5, 10]
\]

\[
\text{Successor } \delta^{18}O / : [5, 10]
\]

Window

The example string above describes a precursor relationship that may be associated with volcanic eruptions: a spike in sulfate immediately follows the eruption, and then there is a cooling period (indicated by a steady increase in \(\delta^{18}O\)) 5 to 10 token widths later.\(^1\)

Although this simple form of a pattern string is the form we will primarily use within this project, the fully qualified pattern language is much more expressive. Clauses can be formed from the conjunction or disjunction of two simpler clauses. A conjunctive clause would be satisfied if a token satisfied both of the simpler clauses, and a disjunctive clause would be satisfied if a token satisfied either of the simpler clauses. A clause can even encompass an entire pattern string, allowing the specification of a cascade of effects. Note that because a pattern string refers to multiple events over time, we must alter our assertion that a token either matches or doesn’t match a clause. To clarify this notion, we will refer to a clause such as \(A \land\) as a simple clause, and a clause composed of a complete pattern string as a complex clause.

As will be seen in the following section, a token captures many features of a subseries beyond just the behavior of the quantity. A clause could specify a range of possible values for any of these features.

\[
\{ Fe \sim (O_2 / \cap CO_2) : [0, 1] \} \sim \delta^{18}O / : [5, 10]
\]

The above string exhibits a number of advanced features. It represents a cascade of successive events by having a full pattern as a precursor. The successor of the first pattern string is a compound clause made from the conjunction of two clauses. Both of the constituent clauses will have to be satisfied at the same time for the compound clause to be satisfied.

\(^1\)Recall that we cannot know the width of the window \([5, 10]\) in real time without knowing the token width.
clause as a whole to be satisfied. This pattern string represents the iron fertilization event described in Section 3. A spike in iron is quickly succeeded by an increase in oxygen and a decrease in carbon-dioxide (as a result of the phytoplankton blooms), which then more gradually leads to a cooling period.

4.2 Tokenization

As mentioned before, Tokenization is the process of putting ice core data into terms of high-level features to make it more applicable to a pattern string. A time series is tokenized by splitting it up into smaller, fixed-length time intervals and then analyzing the subtrends. The sub-series within a certain interval will be referred to as chunks. The subtrends for a token are the same as for a clause in a pattern string. This allows us to say that a token either matches or doesn’t match a simple clause.

![Graph of tokenization](image.png)

Figure 1: A synthetic data set that has been converted into a stream of tokens capturing the qualitative behavior of the quantity over time. From left to right, the tokens produced are: Flat, Spike, Decreasing, Increasing, Dip, Decreasing.

4.2.1 Fully Qualified Token Structure

The high-level behavior of a quantity within an interval has so far been presented as the defining feature of a token. In the fully qualified model, a token has many more qualities. For instance, it is desirable to keep track of time attributes, such as the time within the ice core that the token occurred, as well as the type of quantity that is being measured in the interval. In addition to the sake of bookkeeping, this information could also be relevant for constructing more detailed pattern strings. It is conceivable that the token could maintain all of the values of the time series such that the entire data series could be reconstructed from the combination of all the tokens, although that would be contrary to the spirit of tokenization, i.e. distilling complicated data into the most relevant information.

Certain information about a token is only
A Token Stream is the highest level Token. It represents all the information contained within an ice core. Each ice core is unique and holds information beyond its time series data. The location at which an ice core was drilled is an important piece of contextual information for interpreting climatic information from the core [15]. Many ice cores are uniquely identified by name, such as the GISP2 core [18]. Other identifying information could be included under this subtree. A researcher could use this in a pattern string to specify a pattern concerning only a specific climatic record. For instance, a researcher could use geographic location to create pattern strings describing climatic phenomena that apply only to a specific re-
region of the earth.

A Composite Token holds the features for the actual data over a specific time interval. The tree uses the regular expression-like notation of “*” to indicate that there could be 0 or more of these features. One sub-feature of a Composite Token is its Time Data, which summarizes the time interval of the data the Composite Token describes. This in turn includes the sub-features of Start, End, and Span (the width of the time interval). Note that this includes redundant information, as Span = End - Start. One of the three would be a calculated field in order to maintain integrity of the information.

A Composite Token contains 0 or more Quantity Tokens, which contain information about a sub-series of a certain quantity. Quantity Label indicates the quantity that the values are actually describing, such as “Concentration of Ca$^{2+}$”, “Electrical Conductivity”, or some other numerical quantity. Behavior is the feature of Tokens that is most essential to this project. As we have seen, Behavior is divided into Sporadic events (Spike/Dip), steady Trends (Increasing/Decreasing), no discernable behavior (Flat), and information unavailable (Unknown). Sign describes whether a sporadic event is a Spike or a Dip, or similarly, whether a Trend is Increasing or Decreasing. A steady trend can be described by the correlation coefficient of the data within the interval with respect to time (R), or by the slope of the regression line. R gives more insight into the statistical significance of the trend and Slope gives more insight into the practical significance of the trend.

4.2.2 Chunking

The first step in tokenization is splitting a time series up into a Composite Chunk List. Each composite chunk represents a sub-series of all the quantities of concern over a certain time span, that is, if a chunk has a Start Time of $t_1$ and an End Time of $t_2$ (with $t_1 < t_2$), it contains all the data points corresponding to the time values between $t_1$ and $t_2$. The Chunk List contains all the chunks from a particular ice core, so that the complete data set can be recovered from the list. The length of the timespan ($t_2 - t_1$) is referred to as the Chunk Width, and is constant for all chunks in the list. A user can choose
how to break up the data, either by specifying the number of chunks to split a time series into or by specifying the width of the chunks. If specifying the number of chunks, the chunk width is calculated (as one might suspect) as

\[
\frac{(\text{Latest time}) - (\text{Earliest time})}{\text{#Chunks}}.
\]

Although the width is constant for each chunk in a list, the number of data points is not. One essential reason for this is the aforementioned compression that takes place in ice cores at great depths. Fewer samples per time are available for older sections of the core, making for more sparse data. It may be the case that a chunk contains missing values, that is, time values with no accompanying quantity value. This could be due to errors when processing the ice cores, or other reasons. We simply remove these values using a “cleaning” process so that our future calculations will be unaffected. It is even possible that a chunk will contain 0 time-value pairs.

4.2.3 Behavior Classification

As previously mentioned the most essential aspects of a token is its Behavior component, that is, a qualitative description of the trend exhibited by the time series encapsulated by the corresponding chunk. If a chunk has too few values (less than 5 for our purposes\(^2\)), we make no claim about its behavior and classify it as Unknown (or UNK). For non-trivial cases, we turn to the K-Nearest Neighbors (KNN) algorithm, a machine learning algorithm with applications in classifying objects [7].

As the KNN algorithm is a supervised classification algorithm, the implementation requires the manual classification of chunks to form a training data set, but after this initial investment, the classifier would not need further human intervention. However, it is possible that two different researchers would classify the same chunk differently depending on the nature of their work. Therefore, it is desirable to allow a user to train a new classifier and to give researchers the ability to specify a predefined classifier to be used.

---

\(^2\)As will be seen, some of the statistical attributes we compute are undefined for data sets of 3 or fewer values. We set our threshold at 5 for some extra leeway.
We use a variety of statistical values that reasonably characterize the behavior of the series to serve as our Feature Vectors. These are maximum z-score, minimum z-score, coefficient of variation, sample skewness, and a value we call the Fisher-score.

Each of these has the property that it is scale invariant. This is essential, as we expect our results to be immune to a change of units from, say, measuring concentration in parts per million to molarity. With the exception of the coefficient of variation, each of these values is also translation invariant, which can be undesirable.\footnote{Consider a data-set consisting of points lying on a line segment from (0, 1) to (1, 1.01). While this is certainly a statistically significant increase, it may be questionable whether this has practical significance.}

A set of values may be converted to a set of z-scores by subtracting from each value the mean and dividing each value by the standard deviation. Thus, the z-score is:

$$z_i = \frac{x_i - \bar{x}}{s_x},$$

where $\bar{x} = \frac{\sum_{i=1}^{n} x_i}{n}$ is the sample mean and $s_x = \sqrt{\frac{\sum_{i=1}^{n} (x_i - \bar{x})^2}{n-1}}$ is the standard error. This standardizes the data so that the resultant data has a sample mean of 0 and a standard error of 1. The z-score of a value indicates how strongly (and in what direction) it deviates from the group, irrespective of the scale of the underlying data. Since a Spike, by definition, occurs when a small cluster of points is dramatically larger than the rest of the group, we expect the maximum z-score of a chunk exhibiting a Spike to be quite high. Similarly, we expect the minimum z-score of a chunk exhibiting a Dip to be low. Since these features are based on the minimum and maximum values of some distribution, they do have the unfortunate property that they are sensitive to $n$, the size of the data set. For instance, we would expect a set with 1000 values to have a more extreme maximal element than one with only 10. However, we do not expect this to be a very significant factor, and the presence of a Spike should indicate a high maximum z-score, even for meager $n$. In contrast, the z-scores for random scatter (as with a Flat trend) modeled with Gaussian noise would almost certainly be expected to be less than 4 in magnitude, even for $n$ in the hundreds. Similarly, data following a typical linear trend such as $x = at + \epsilon$ (where
\( \varepsilon \) is a Gaussian error term) would have z-scores not much more than the error term alone, so Increasing and Decreasing trends should likewise have a much lower maximum z-score. Detecting extreme or outlier values is a traditional method of identifying spikes in climatology, albeit using more sophisticated methods, such as robust spline residuals [19].

The coefficient of variation is the ratio of the standard deviation to the mean, that is,

\[
c_v = \frac{\sigma}{\mu}.
\]

We calculate the sample coefficient of variation using the biased estimator \( \hat{c}_v = \frac{s_x}{\bar{x}} \), where \( \bar{x} \) and \( s_x \) are defined as above. It indicates how much variability is present in the set compared to the magnitude of the values. While this is scale invariant, it is the only feature that is not translation invariant. This is important because it can indicate if occurrences that “look like trends from close-up” should really be regarded as noise. To see this, consider, say, a set of data exhibiting a spike. The spike could be 10 times as large as any other point in the set, but if there is another data set, identical to this except translated by a very large constant, all data points would be approximately equal in magnitude (i.e. having a ratio close to 1). In this case we would qualitatively regard the trend as Flat. Unlike our other features, the coefficient of variation gives us a way to distinguish between the two data sets since its value would be closer to 0 for the second.\(^4\) However, the coefficient of variation should only be used with ratio data, and in particular, it should not be used with values that are negative [8]. When classifying a chunk representing data from a quantity that can take on negative values, such as \( \delta^{18}O \), only the other 4 features will be used, for both the featurization and the distance calculation.

The Skewness of a distribution is defined as the third standardized moment:

\[
\gamma_1 = \frac{\mu_3}{\sigma^3},
\]

where the numerator is the third centralized moment, \( \mu_3 = E[(x - \mu)^3] \), and the denominator is the cube of the standard deviation, \( \sigma = \sqrt{E[(x - \mu)^2]} \). To calculate the sample

\(^4\)Because \( s_x \) is unaffected by a translation and \( \bar{x} \) is increased by the constant of translation.
skewness, we use the biased estimator
\[ b_1 = \frac{\sum_{i=1}^{n} (x_i - \bar{x})^3}{s_x^3} / n, \]
where \( s_x \) is the standard error, as defined above. Skewness is an indication of whether
the distribution “leans” right or left [2]. The
motivation behind this feature is that ex-
treme values will create a “bump” in the tails
of the distribution. Extreme high values will
cause a positive skewness, and extreme low
values will cause negative skewness. For
Flat and linear trends, we expect skewness
to be close to 0. This is similar to the case of
Min and Max Z, and serves as a supplemen-
tary metric for classifying these occurrences.

Our Fisher-Score statistic is:
\[ \sqrt{\frac{n-3}{2}} \log \left( \frac{1+r}{1-r} \right). \]
This essentially measures how significant a
positive or negative trend exists in the data.
At the heart of the Fisher-Score is the corre-
lation coefficient, or more specifically, Pear-
son’s \( r \):
\[ r = \frac{\sum_{i=1}^{n} x_i t_i - n\bar{x}\bar{t}}{(n-1)s_x s_t}. \]
\[ ^5 \text{For our purposes, } t_i \text{ is time and } x_i \text{ is the value of some quantity, but in general these can be any values that occur in pairs.} \]

Pearson’s \( r \) is bounded between -1 and 1
and gives an indication of how closely corre-
lated the paired data \((t_i, x_i)\) are. This alone
can be a good indicator of behavior: values
close to 1 and -1 are likely to be Increasing and Decreasing, respectively, and other
trends would likely have \( r \) closer to 0. But
this value is sensitive to small \( n \) (for \( n = 2, \)
\( r \) is likely equal \( \pm 1 \)), and the boundedness
of \( r \) is not ideal since there is a larger dif-
ference (in terms of the relative statistical
significance) between \( r = 0.95 \) and \( r = 0.8 \)
than \( r = 0.5 \) and \( r = 0.35 \). Thus, we use the
Fisher Transform:
\[ W = \frac{1}{2} \log \left( \frac{1+r}{1-r} \right) \]
to give the feature an infinite range, which
also emphasizes small, but important, differ-
ences near the boundary values \( r = \pm 1 \).

The Fisher Transform also has the prop-
erty that when the underlying joint distribu-
tion of \((x_i, t_i)\) is bivariate normal with cor-
relation parameter \( \rho = \rho_0 \), then for large
\( n \), \( W \sim N(\frac{1}{2} \log \frac{1+\rho_0}{1-\rho_0}, \frac{1}{n-3}) \) [23]. We
will assume this property holds for our
data. Finally, we scale by $\sqrt{n-3}$ to obtain our Fisher-score, making our statistic independent of sample size. Since the resulting statistic is approximately normally distributed with standard deviation 1, the Fisher-score implicitly acts as a test statistic for a null hypothesis $H_0 : \rho_0 = 0$. Thus, moderately large magnitudes of the Fisher-Score (greater than 2) indicate high confidence that time and the quantity of interest have an underlying correlation (within the specified time interval).

We expect that the particular distance metric we choose will not be very consequential to the performance of the classifier, so we somewhat arbitrarily choose the Euclidean distance as our metric because of its intuitive nature. That is the distance, $d$, between two objects $o_1$ and $o_2$ is

$$d(o_1, o_2) = \sqrt{\sum_{i=1}^{5} (f_i(o_1) - f_i(o_2))^2},$$

where each $f_i$ is one of the feature statistics previously defined. Of more consequence is our method of scaling within the feature space; in general it is important that each feature have a similar range of values so that the distance calculation is not dominated by one feature [7]. We scale using the standardization transformation where each feature is converted to its z-score. This has the advantage that the transformation is expected to stabilize with more observations (because the sample mean and standard error are unbiased estimators and will converge to their expected values due to the Central Limit Theorem). This is in contrast to another common technique where the data are scaled using the range of values so that every value is within the range [-1,1], which does not stabilize with more observations because the distribution of the maximum (or minimum) order statistic is dependent on sample size. The standardization transformation is also desirable if the underlying features have a similar distribution (such as normal), since the range of values would be similar for each feature, thus ensuring each feature has roughly the same bearing on the distance calculation. Conversely, if a particular feature is very skewed (prone to extreme values), we would expect that feature to have a

---

6One can see why this is true by considering a feature space of, say, height in feet and age in years. Without scaling these values, the “distance” between two people can easily be dominated by their age difference.
negligible contribution to the distance calculation for typical values, or the feature may dominate the distance calculation for the occasional extreme value.

Another design choice is the particular value of $k$ we use, that is, the number of neighbors that vote when classifying. We would like to devise a method where the voting power of a neighbor depends on its distance from the object, partially so that we can avoid the situation where there are ties, and also because it is reasonable to assume that the closer an object is to a neighbor, the more likely it is to be of the same category. A reasonable approach is to introduce a non-negative exponent parameter $\gamma$, so that a neighbor $N$'s vote when classifying object $o$ receives a weight of:

$$\frac{1}{d(N,o)^\gamma}.$$

Note that for $\gamma = 0$, we obtain the equal-weight voting method.

However, with the above method, the contribution of a vote is unbounded, as the distance could be arbitrarily small. Thus, we introduce a Maximum Weight parameter so that the voting process is not necessarily completely decided by an object that happens to fall very close to a point in the training set.

4.2.4 Evaluation

Once a training set of data has been established, we will use Leave-one-out cross-validation (LOOCV) to assess the accuracy of the classifier, wherein each object in the training set is removed and then classified with respect to the remaining data [5]. The output from this is the proportion of objects that are correctly re-classified. We will determine the optimal values of $k$, $\gamma$, and Maximum Weight by performing a parameter sweep to maximize the accuracy, as determined by LOOCV. Another evaluation criteria is to test whether the behavior types predicted by the classifier (for chunks not used to create the model) match a manual classification.

4.3 Pattern Matching

Given a pattern string, the basic procedure for finding matches within a token stream is quite straightforward. Simply scan through
the token stream generated from a set of raw data until a token matching the precursor clause is found. From that point, check tokens that fall within the time window for one that matches the successor clause. For evaluative purposes, we are also interested in when a precursor clause is satisfied, but there is no matching successor token within the time window. We refer to this as an anti-match.

\[
A \not\sim E \land : [1,3]
\]

Figure 3: A pattern string and a token stream exhibiting a match.

In Figure 3, the depicted token stream exhibits a match because an Increasing token occurs in the \(A\) sub-stream at time 2, which matches the precursor clause, and a Spike token occurs in the \(E\) sub-stream at time 4, which matches the successor clause. The delay between these occurrences is \(4 - 2 = 2\), which is contained in the interval \([1,3]\).

\[
A \not\sim E \land : [1,3]
\]

Figure 4: A pattern string and a token stream exhibiting an anti-match.

In Figure 4, the depicted token stream exhibits an anti-match because an Increasing token occurs in the \(A\) sub-stream at time 2, which matches the precursor clause, but there is no token that matches the successor clause 1 to 3 time units after the precursor.

A complication to this procedure arises when there is an UNK token within the time window of the pattern string. We cannot be sure whether the occurrence as a whole is a match or an anti-match. We refer to this occurrence as an indeterminate match, and it will be disregarded during evaluation, that is, it does not increase or decrease the calculated power of a particular pattern string. A similar situation occurs when the time window goes outside the range of the token stream\(^7\). We also define this as an indeterminate match. This effectively means that a token stream has an infinite number of UNK tokens.

\(^7\)Consider when a precursor token is the last token in the stream.
tokens on either side.

\[
A \backslash \sim E \land : [1,3]
\]

In Figure 5, the depicted token stream exhibits an indeterminate match because an Increasing token occurs in the A sub-stream at time 2, which matches the precursor clause, but there is an UNK token within the interval 1 to 3 time units after the precursor. If there had been sufficient information to classify the chunk that produced this token, it may have been classified as a Spike, or some other behavior. Had it been a Spike, we would have a match, as in Figure 3, but otherwise we would have an anti-match, as in Figure 4. Since we cannot say which of these might have occurred, we have an indeterminate match.

One case we have not considered is when there are both successor tokens and UNK tokens within the time window following a precursor token. For instance, consider if we had the situation as in Figure 5, except we were considering the pattern string \(A \backslash \sim E \lor : [1,3]\). If a researcher was simply using the system to find matching patterns, this would be a match since we have a token matching the precursor clause at time 2, followed by a token matching the successor clause at time 5, a delay of \(5 - 2 = 3\) time units, which is in the time window \([1,3]\). But if the system was performing matching so as to evaluate the pattern string, this would be considered an indeterminate match. If this were not so, a token stream with many UNK tokens could inflate the accuracy of a string by masking the potential anti-matches.

It may be the case that there are several precursor tokens and several successor tokens within a small time interval such that each precursor could be matched with each successor. Rather than allow this many-to-many matching relationship, we match a precursor only with the earliest successor. One benefit of this is approach is that each precursor token will be associated with exactly one of a match, anti-match, or indeterminate match. Note that the same successor tokens could still be matched with a number of different precursor tokens, making a
many-to-one relationship.

4.4 Pattern Extraction and Completion

As Pattern Extraction relies heavily on the process of Pattern Completion, we will discuss these processes in tandem. Given the brute-force nature of these algorithms, we limit our domain to basic 3-component pattern strings. Pattern Completion works by trying all possible values for the missing component of a string and finding the one that yields the string with the highest evaluation score. For a missing clause, this entails trying all clauses formed by each combinations of quantities available in the token stream with the 5 non-UNK behaviors. If the time window is missing, the system forms all intervals \([t_1, t_2]\) with \(t_1, t_2\) integers such that \(0 \leq t_1 \leq t_2 \leq L - 1\), where \(L\) is the length of the stream, that is, the number of tokens for each quantity in the stream.

Pattern Extraction operates very similarly. The system forms all possible pairs of all possible clauses and then uses Pattern Completion to find the time interval that yields the most powerful string. The system returns a predetermined number\(^8\) of the most powerful strings. In terms of extracting the most powerful patterns out of all possible patterns, this system is not optimal because the output will contain at most one string composed of a particular combination of precursor and successor clauses. That is, if the two most powerful strings differ only by their time window, only the more powerful of the two will be returned. However, this is desirable when we assume that for any combination of precursor and successor clauses, there is one time interval that best describes the relationship between the two. Multiple strings that differ only in their time windows would be inferior and extraneous results that would only prevent other precursor, successor relationships from being considered.

4.4.1 Pattern Evaluation

Essential to the success of Pattern Completion and Extraction is a way to evaluate and compare pattern strings based on their power, that is, based on how useful they

\(^8\)15 by default, so as to not overwhelm the researchers.
are in explaining climatic phenomena. At a glance, we would like a pattern string to be accurate in that whenever a precursor token occurs, the successor token should usually occur within the indicated time window. This leads us to a first approximation of power as

\[
\frac{\text{#Matches}}{\text{#Matches} + \text{#Anti-Matches}}.
\]

While this is an important component of power, and we denote it as \textit{accuracy} for future reference, this quantity alone suffers from some weaknesses. Notice that we can always increase this quantity by expanding the time window of a string since this will only lead to more matches (assuming the token stream is very long compared to the width of the time window, and that there are few UNK tokens, so that the effect of indeterminate-matches is negligible). In general we would like to penalize long time windows in some way. Also, consider the case where the successor token is naturally very frequent. In the extreme case, if the, say sulfate, token stream consisted solely of Flat tokens, a pattern string with a successor clause of $SO_4^{2-}$ could never experience an anti-match since there would always be a successor token within the window following a precursor token (or the window would be out of the range of the stream, in which case there would be an indeterminate match, and not an anti-match).\footnote{One may ask if there is a corresponding problem when the precursor token is very common. This is essentially already accounted for by the current definition of \textit{accuracy}, because more precursor tokens can lower accuracy if the pattern described by the string is ineffective and doesn’t successfully predict all the corresponding successors.}

Both of these factors, long time windows and frequent successors, are problematic in that they increase the likelihood that a successor token will follow a precursor by sheer chance, and not by any true relationship between the precursor and successor clauses, thus inflating the \textit{accuracy} of a string. We will use the value \textit{ubiquity} to refer to the probability that a string will match a token stream by sheer chance. We will refer to the reciprocal of \textit{ubiquity} as \textit{rarity}. Once a method for calculating this value is derived, we will use it to augment the evaluation metric. For a given pattern string and token that matches the precursor clause, let $X$ be the number of successor tokens that appear in
the time window following the precursor token. There will be a match if there are any successor tokens in the window, that is if \( X > 0 \). This leads us to the definition:

\[
ubiquity = P(X > 0).
\]

We can find this probability by modeling \( X \) with a binomial distribution, \( X \sim B(n, p) \), where \( n \) is the width of the time window, that is, for time window \([t_1, t_2]\), \( n = t_2 - t_1 + 1 \), and \( p \) is the probability that a randomly selected token will match the successor clause. We can estimate this probability with

\[
p = \frac{\text{#successor tokens}}{\text{#non-UNK tokens in token stream}}.
\]

Note that the assumptions of the binomial distribution rely on each token in the stream corresponding to an independent Bernoulli trial with constant probability of success [10]. This may be reasonable if there is only one successor token in the stream, but this is certainly false for multiple tokens, as the values cannot be independent because there is a fixed number of successors within the token stream. Furthermore, by the very nature of time series, we do not expect behaviors to be independent. However, using a binomial model still provides a reasonable heuristic for our purposes. With that disclaimer in mind, we may finally use our model to calculate the desired probability:

\[
P(X > 0) = 1 - P(X = 0) = 1 - \binom{n}{0} p^0 (1 - p)^n \quad (1)
\]

All other things being equal, that is, with constant accuracy, a string with a higher rarity is more powerful than a string with lower rarity because, informally, it is more “impressive” to predict a rare event. This suggests the final definition:

\[
power = accuracy \times rarity = \frac{accuracy}{ubiquity}.
\]

Note that this value is undefined precisely when \( p = 0 \), corresponding to the situation where there are no successor tokens anywhere in the token stream. This would cause accuracy to be either 0 or take on the indeterminate form \( \frac{0}{0} \), as well as implying ubiquity = 0. In this situation we will define power = −1 to indicate that there is no...
suitable evaluation for this pattern.

However, using some foreknowledge, we will see that it is desirable to add some parameters to give us more control over how these factors contribute to the power. One particular alteration is to introduce a non-negative Buffer parameter to the definition of accuracy. As inspired by the Agresti-Coull method of calculating proportions where one adds 4 “bogus” observations, 2 successes and 2 failures [1], we effectively recalculate accuracy by increasing both the number of matches and the number of anti-matches by an amount equal to Buffer:

\[
\text{accuracy} = \frac{\text{#Matches} + \text{Buffer}}{\text{#Matches} + \text{#Anti} + 2\times\text{Buffer}}.
\]

This has the effect of shifting this proportion closer to $\frac{1}{2}$, thus lessening the effects of “lucky” matches; Under the previous definition, a string with 10 matches and 0 anti-matches would have the same (perfect) accuracy as a string with 1 match and 0 anti-matches, but intuitively it seems that the string with 10 matches should be evaluated as “more powerful”.\(^{10}\) The Buffer parameter can be set to a large value so that only patterns that frequently appear in the data have a high accuracy, or alternatively, Buffer can be left at a low value (even 0, as in our first definition) to give those patterns that are by their nature infrequently observed a “fighting chance”. Despite its interpretation as an integer-valued number of observations, we allow Buffer to take on real values.

We also introduce non-negative exponent parameters $\alpha$ and $\beta$ (not both equal to 0) to give us a final definition:

\[
\text{power} = \frac{\text{accuracy}^\alpha}{\text{ubiquity}^\beta}.
\]

Since accuracy is a value between 0 and 1, a high $\alpha$ can penalize even small deviations from perfect accuracy, or conversely, a small $\alpha$ can de-emphasize them. Similarly for $\beta$ and ubiquity.

5 Implementation

This project was implemented in Java, in part because of its strong typing, which

\[^{10}\text{There would not necessarily be a distinction between the two strings under the current definition of power, given that they could have the same ubiquity.}\]
allows easier management of various data types, as well as for ease of integration with P301dx, which is also Java based.

An important class that eased the process of accessing tabular data was the DataTable class. We modeled our interface after R’s data-frame objects. These DataTable objects contain 2-dimensional arrays of real values (stored as a 2-d array of doubles) following the database model where a column is called a field and a row is called a record. Each record corresponds to one data object, and the fields correspond to attributes of that object. Each field is associated with a String-valued label, or header. These tables are implemented in column-major order. This design choice is for the sake of consistency and ease of integration with P301dx’s architecture, as well as for the sake of efficiency in that many of the operations performed on these tables operate on the set of values for a particular field rather than for a particular record. Columns can be indexed both numerically and by the name of the column header. Rows can be indexed numerically. In all of these cases, the result is an array of doubles. The dimensions of these tables are immutable, but the entries themselves are not. Rows can be set as well as swapped to allow sorting based on a column. A subtable can be created from a table based on either rows or columns. That is, a table that contains only the specified rows (or columns) of the original table. A subtable based on columns can be specified either with an array of column headers, or with an array of integer column indices. A subtable based on rows can be specified with an array of integer row indices. This DataTable class is not directly used in the application, rather it is used as a base class for FeatureTable and RawTimeSeriesTable.

A FeatureTable is the basis of our K-Nearest Neighbors classifier, as it is used to hold training data. Although the architecture is general enough for any type of object that can be featurized and classified, in this project we only use it for chunks. \footnote{Recall: a chunk is a sub-time series.} Each record corresponds to a chunk, where each field in the record is a different feature. The FeatureTable also contains a supplementary String-valued field: Classification. As the
A `RawTimeSeriesTable` is used to hold the raw data from an ice core. This table requires a particular column to be specified as the Time field. In this context, each record holds the values of all the quantities at particular point in time, and each field corresponds to the values of a particular quantity throughout entire duration of time series (with the exception, of course, of the Time field itself, which encompasses all of the time measurements for each record throughout the duration of the series). The table can return summary information, such as the time corresponding to the most recent measurement, least recent measurement, and the span of time covered by the data. A subtable can be created by specifying a start and end time. The resultant table will only contain records from within that time period.

A `MathUtil` class was used to encapsulate many vector and statistical operations. These include vector arithmetical operations such as addition, subtraction, multiplication by a scalar, Euclidean norm, etc. (using arrays of doubles as vectors) and computation of summary statistics, such as mean, variance, minimum value, maximum value, etc. (again, using arrays of doubles as data sets). There is also a method to transpose a 2-d array of doubles, which can be useful for constructing `DataTable`'s.

Currently, CSV files are the only type of external data format that can be used to create a `RawTimeSeriesTable`. These files must be in the form of a table, that is, the first row of a file must contain the headers for the corresponding fields in all subsequent records, and each line after the first corresponds to a record, each containing the same number of values as there are headers. When parsing the file, any non-numeric values are evaluated as NaN.\(^{12}\) The rationale behind this is that certain formats use “NA” or some other such `String`-valued indicator for missing data. Additionally, P301 uses the convention that missing values are represented...
with -999, so this value too is evaluated as NaN if it is encountered.

As the time of a measurement can be specified with respect to time after or time before a certain event, larger values do not necessarily indicate a more recent time. The application assumes the most recent records are at the top of the file, which is the convention for these data sets. The table stores this information, that is, whether the times are specified before or after a certain event, so that it can accurately provide the previously mentioned summary data, such as earliest time, latest time, etc.

To correctly construct the `RawTimeSeriesTable`, it is important that the application be able to determine which column of the file holds the Time values. The user is able to specify this, but if this information is omitted, the application is able to fairly accurately deduce this information. Since the files are typically organized so that records are in order, the basic strategy is to look for a column that is sorted. However, since there are cases where a record small number of records are out of place, we instead look for the column that is most sorted by counting inversions.\(^{13}\) Since some records maintain the depth data for the measurements, there may be multiple columns that are monotone.\(^{14}\) This “tie” is broken by returning the column that is farthest to the right. This is because the time column is typically the rightmost column.

When the application performs pattern matching, it makes use of a `MatchDataObject`, which performs all the matching essentially in one operation. It first filters through the token stream to find all tokens matching the precursor clause, or *precur-\(^{15}\)sor tokens*, and then filters the stream for all tokens matching the successor clause, or *successor tokens*. It then forms a `DifferenceTable` composed of the difference in time occurrence of each precursor, successor pair. Then the time differences for a particular precursor token with respect to the successor tokens are examined to determine if there is a match, anti-match, or indeterminate match.

\(^{13}\) The number of times where adjacent records are not in order.

\(^{14}\) Either increasing or decreasing.

\(^{15}\) This is not a subclass of `DataTable`.
As mentioned before, pattern extraction works by generating all possible pattern strings and returning only the most powerful. Then regardless of the number of strings the algorithm will return, if \( n \) is the length of the token stream, and \( Q \) is the number of quantities in the stream, then there are \( O(n^2Q^2) \) patterns generated. Each of these must be evaluated, which requires finding all matches to a particular pattern within the data. In the worst case, all behaviors would be approximately equally present in the stream, leading to a \( DifferenceTable \) with \( O(n^2) \) elements, a fixed proportion of which must be checked to determine whether a precursor token constitutes a match, anti-match, etc. So while we can save computation time by reusing the same \( DifferenceTable \) for patterns that differ only by their time window, the runtime of the process is still \( O(n^4Q^2) \) overall. This runtime could be improved by optimizing the matching algorithm, or by creating a more sophisticated extraction algorithm that doesn’t rely on brute-force methods.

The functionalities implemented in this project are made available to a user through a command line application. This application offers an environment where users can read and tokenize raw data, create variables, display results visually, and save interesting data objects for later use. The application uses a combination of Lisp-like and Perl-like syntaxes. The basic data type supported by the application is an \( AppVar<T> \), essentially a wrapper for the type \( T \). The application supports several primitive types, such as, Boolean values, integers, reals, and strings.\(^{16}\) There are also data types that are more specific to this pattern domain, such as \( AppVar \) wrappers for tokens, token streams, patterns, time series tables, matches, and several others. Each of these data types has an associated \( static \) \( final \) instance that is used only for type checking and type conversion.

The command syntax is Lisp-like in that it is of the form \( (\text{fun args}^*) \), that is, a function call is surrounded in parentheses, with the first word being the name of the function, followed by 0 or more space-separated arguments. The application currently does not support nested functions, so an args.

\(^{16}\)Strings composed of characters, rather than pattern strings.
ment may only be either a literal or a variable. The syntax is Perl-like in that variable names begin with the sigil ‘$’ and are followed by 1 letter or underscore, followed by any number of letters, underscores, or numbers. Since function calls cannot be nested, meaning it would otherwise be impossible to set the output of a function to a variable, the output from the most recently executed command is cached in the special variable ‘$_’ (another convention taken from Perl).

Some of the basic functions offered in the application are set, print, run, quit, and ls. There are more pattern-oriented functions offered such as parsePattern, readTable, subTable, tokenize, match, extract, completeTime, as well as others.

6 Results

6.1 Behavior Classification

In total, 556 chunks were manually classified to form a training set. These chunks were formed from data taken from the GISP2 ice core [18]. The raw data was converted into 100 composite tokens and various quantities, such as sulfate and calcium were used.

Figure 6: A pairs plot showing each classification in feature space. The orange, purple, green, blue, and red points correspond to Spikes, Increasing trends, Flat, Decreasing trends, and Dips, respectively. Across the diagonal is a histogram for each feature taken over the entire data set.

Figure 6.1 shows the resulting training set plotted in feature space. Note that each different classification group is clustered together, indicating that they have similar features. This gives credibility to our choice for the feature set.

In order to establish optimal values of the parameters in the KNN model, we performed a parameter sweep, with $k$ varying through each integer from 1 to 30, $\gamma$ varying from 0 to 3 in increments of 0.1, and Max Weight varying from 0.0001, to 1000.
by powers of 10, as well as $+\infty$, which effectively removes the limit imposed by *Max Weight*. The parameters of the model were set to each combination of values in turn, and then the accuracy of the resulting model was determined using leave-one-out cross-validation. The results are displayed in the following sunflower plots.\(^{17}\)

Figure 7: A sunflower plot graphing $k$, the number of neighbors that vote in the classification process, vs. accuracy.

Figure 8: A sunflower plot graphing $\gamma$, the exponent determining the weight of distance when voting, vs. accuracy.

Figure 9: A sunflower plot graphing *Maximum Weight*, the highest weight a vote can be assigned, on a logarithmic scale vs. accuracy. Note: the value of 4 on the x-axis actually corresponds to a *Maximum Weight* of $+\infty$ rather than 10,000.

The maximum accuracy achieved was 96%, which occurred with $k = 9$, $\gamma = \text{any one of 0.4, 0.5, or 0.6}$, and *Max Weight*.

---

\(^{17}\)A sunflower plot is a modified scatterplot that allows visualization of clustering when there is much repetition in the data. This is essential here as each parameter as well as the accuracy itself can only take on a fixed number of discrete values.
equal to any of the test values. We use these results to select parameters for the final model. We use $k = 9$, $\gamma = 0.5$ because it is in the middle of the 3 equivalently evaluated parameters, and $Maximum Weight = +\infty$. As Figure 9 suggests, we found that the $Maximum Weight$ had no effect on the accuracy for this data set. Since it appears that this parameter has no bearing on the model, we choose a value of $+\infty$ for this parameter, which is effectively the same as if there is no maximum weight.

However, this value of accuracy should be taken with a grain of salt for a number of reasons. First off, some researchers, such as Hirsch, raise objections to the validity of cross-validation as an evaluation technique for classifiers [13]. Secondly, there is some sampling bias in that only observations that with prominent behavior in the first case made it into the model. That is, chunks that do not readily conform to one of the behavior types have been classified as UNK, and so have been omitted from the model.

We will further evaluate the accuracy of $k = 9$, $\gamma = 0.5$, and $Maximum Weight = +\infty$.

6.2 Seasonal Sample Run

For our first sample run, we form a token stream from the US_ITASE-00-2013 ice core [17]. This core has been chosen because of the prominent seasonal trend of sulfate concentration. The data encompasses a time span from 1888 to 2001. We preprocessed the data by discarding all quantities besides the sulfate concentration, and then we smoothed the data by taking a 10-point acausal moving average.\textsuperscript{18} The chunk width of the data is specified as 0.5 years, so that the token window is synchronized to one half the period of the seasonal events. The classification parameters were set to the experimentally determined values

\textsuperscript{18}As the data is not equally spaced in time, a time-weighted moving average would have been more appropriate here. However, the 10-point average is simpler and produces reasonable output.
Figure 10 is a truncated portion of the entire token stream generated from the ITASE data. It is fairly representative of the rest of the stream in that the chunks that indicate large changes are classified as either Increasing or Decreasing trends, and the chunks containing intervals where the data did not change as drastically are typically classified as Flat tokens. We see that the classification algorithm assigns tokens to chunks reasonably well in this case. At a glance it may seem that the three Flat tokens in the middle of the stream are misclassifications as they do not follow the “Increasing, Decreasing, etc.” pattern. However, at least for the second two Flat tokens, these could be reasonable descriptions of the behavior of the quantity. The amount of change of the quantity over the span of the sub-series is not as dramatic in terms of relative change as that in the other chunks.

Table 1: The complete set of patterns extracted from the ITASE Sulfate data.

<table>
<thead>
<tr>
<th>Pattern String</th>
<th>[1,1]</th>
<th>[12,12]</th>
<th>[1,1]</th>
<th>[4,4]</th>
<th>[1,1]</th>
<th>[73,73]</th>
<th>[106,106]</th>
<th>[95,95]</th>
<th>[89,89]</th>
<th>[14,17]</th>
</tr>
</thead>
</table>

We now turn our attention to the pattern extraction feature. Table 1 shows all the patterns extracted from the ITASE token stream. The parameter settings for this extraction were $\alpha = 1, \beta = 0.5,$ and $Buffer = 4$. As we will extensively examine the supplemental information associated with extracted patterns in coming results, we omit the evaluation values and focus our attention on only the content of the strings produced. Two patterns in particular stand out: $SO_4^{2-} \sim SO_4^{2-} /: [1,1]$ and $SO_4^{2-} / \sim SO_4^{2-}: [1,1]$. These strings mutually describe a signal that alternates Increasing and Decreasing every time period, which accurately describes this oscillatory behavior.

---

19Recalling that pattern extraction only returns one string for each combination of possible clauses, we can see this table is exhaustive because only three types of tokens are present in the data, so only 9 pattern strings are possible.
data. This is reassuring in that our pattern extractor was able to detect a fairly simple but prominent pattern.

6.3 Volcanic Sample Run

For our second sample run, we form a token stream from the US.ITASE-00-1 ice core [17]. This core was chosen because its time span covers the Tambora eruption of 1815 [25].\(^{20}\) We pre-processed the data by discarding all data besides the sulfate concentration, and the new smoothed the data by taking a 21-point acausal moving average.

The parameters for the chunking portion of tokenization were as follows: we limit the time span to the years 1715.5 to 1917.5, and the chunk width is 10 years. The classification parameters were set to \(k = 10\), \(\gamma = 1\), \(\text{Maximum Weight} = 100\).

![Figure 11: A token stream formed from the sulfate concentration from the US.ITASE-00-1 core.](image)

We see from Figure 11 that the tokenization process successfully detects the large Spike around 1810, but it misclassifies the even larger Spike at 1815, which corresponds to the Tambora eruption. Instead of a Spike, the system views it as a Decreasing trend, most likely because the tip of the spike is close to the edge of the chunk window. This makes it appear to the classifier that the quantity started out high and then dropped. This indicates a flaw in the classifier, as ideally we would hope that the most prominent spikes would be the most unambiguously classified.

6.4 Belukha Pattern Extraction

We now turn our attention to the process of pattern extraction. We will tokenize and extract data from ice core data sets, (using multiple quantity types) so as to evaluate what kind of output one could expect to obtain from this system. Despite the apparent flaws in the tokenization process, we make the assumption that our tokenization process is robust so that we can continue forward with the analysis. The rationale is that even though the system is currently rather

\(^{20}\)Sometimes called the year without a summer
crude, these tests provide valuable information about the benefits a more completed version of this system could provide.

We format our extracted patterns in tables that are effectively expanded versions of Table 1. We include three more columns: “Matching Info.” (Matching Information), “Quant. Eval.” (Quantitative Evaluation), and “Qual. Eval.” (Qualitative Evaluation).

The Matching Information column contains 4 integer values describing how many instances of the pattern were found, as well as supplementary information used to evaluate the pattern. More specifically, the first value is the number of Matches that were found in the token stream. The second value is the number of Anti-Matches that were found. The third value is the total number of tokens in the stream that matched the precursor clause. The final value of the column is the total number of tokens in the stream that matched the successor clause. As every precursor corresponds to exactly one of a Match, Anti-Match, or Indeterminate Match, this data implicitly contains the number of Indeterminate Matches that were found, given by

\[
\text{#Indeterminate Matches} = \text{#Precursors} - (\text{#Matches} + \text{#Anti-Matches}).
\]

The Quantitative Evaluation column contains three real values, which give information about how the power of the string was calculated. The first value is the adjusted accuracy, which, recalling from Section 4.4.1, is

\[
\text{accuracy} = \frac{\text{#Matches} + \text{Buffer}}{\text{#Matches} + \#\text{Anti} + 2*\text{Buffer}}.
\]

The second value is ubiquity, that is, the probability that a successor token will occur in the time window by random chance. The third value is the final value of power, that is,

\[
\text{power} = \frac{\text{accuracy}^\alpha}{\text{ubiquity}^\beta}.
\]

The Qualitative Evaluation, contains two codes, the first describing how valid the pattern string is given its performance with respect to the generated data, and not with respect to the actual meaning of the quantities and behaviors, and the second describes how valid the string is in terms of the cli-
matic interactions it supposedly represents.

We evaluate a string based on the empirical results using a letter to indicate its class. These classes are A, W, X, Y, Z, and M. Classifying a string as one of type W, X, Y, or Z indicates that the string is most likely lacking in terms of predictive power. These are essentially false positives in that they appear powerful by the quantitative methods, but they would likely have little use to a researcher. Strings of Type A are more likely to correspond to actual correlations between climatic signals, and so are more useful. We use Type M (miscellaneous) as a catchall for strings that don’t fit cleanly into one of the other categories, but these are likely to also be false positives. The strings in each category have certain characteristics in common with regard to their structure and the context of the token stream. There could certainly be other categories describing other types of strings, but these are the most apparent in the results obtained thus far.

We call a pattern string Type W if both the precursor and the successor tokens are extremely rare within the token stream (only 1 or 2 occurrences apiece), and the time window is very narrow (most often with equal endpoints). However, the actual values of these endpoints can be large (and often are). This type of string almost always represents an overfitting, and describes a relationship that only occurs once. However, it may be evaluated as powerful because it is completely accurate and appears to “predict” a rare event.

We call a pattern string Type X if the precursor token is very common in the stream (roughly, 75% or more), the successor token is moderately common, and the time window is medium in length. This pattern string tends to exhibit many matches because there will likely be a successor within the time window.

We call a pattern string Type Y if the precursor token is very common in the stream (roughly, 75% or more), the successor token is moderately uncommon to very rare, and the time window is rather wide. This pattern string tends to exhibit many matches because most of the precursors tokens can be matched to the same successor token since the window gives them a long “reach”. This string can be evaluated as powerful be-
cause the rarity of the successor helps offset the long time window when calculating the \textit{ubiquity} component, and the string will tend to be very accurate, since a precursor token will tend to be either within range of one of the few successors, or the window will go out of bounds, creating an indeterminate match.

We call a pattern string Type Z if both the precursor and successor tokens are very common in the stream. These tend to have a small window width. These strings exhibit matches simply because a successor token will tend to be very close to a precursor token because they are both so plentiful. Despite the \textit{ubiquity} of this string being quite high, it will also have a very high accuracy because the many matches will help overwhelm the \textit{Buffer} parameter.

We call a pattern string Type A if both the precursor and the successor tokens are moderately uncommon to moderately common within the token stream (roughly 10\% - 50\% each), and the time window has a short width (roughly 2 or less token widths in length). If this type of pattern string is evaluated as powerful, it has most likely “earned” that distinction because we wouldn’t expect it to experience many matches by pure chance.

We evaluate a string based on it’s feasibility as a descriptor of climatic events by assigning it an integer value between 1 and 5. These numerical codes correspond to different levels of confidence that a pattern could conceivably represent a climatic process. These evaluations were based off of both the structure of a string and the context of the string, that is, with regards to its performance in the token stream. An earth sciences researcher aided in this evaluation process by lending knowledge of climatic processes and ice core signals.

A score of 5 indicates that a pattern string appears to correspond to a known climatic process or driver. A 4 indicates that the pattern string could reasonably correspond to some climatic process. A 3 indicates that the pattern string appears to be of a form that one might expect of a climatic signal, but there is no such known climatic process. A 2 indicates that there is no real basis to believe the string corresponds to a climatic process, but a small change, either in
the structure of the string, or in the empirical
evidence, would make it significantly more
plausible that there could be a climatic pro-
cess associated with the string. A 1 indicates
that the string bears no resemblance to any-
thing that would be expected to correspond
to a climatic process. In some cases, our
methods of considering climatic processes
are admittedly quite speculative, but this is
meant to fully explore the potential bene-
fits of such an application, one of which is
spurring new ideas about what signals may
be detected in ice core data.

For this analysis, we used data from the
Belukha ice core [20]. We used the Ca, Cl,
K, Mg, Na, NO₃, and SO₄ quantities be-
tween the years of 1800.5 to 2002.5. We per-
formed no pre-processing on the raw data.
We split the raw series into 30 chunks, which
corresponds to a token width of ≈6.7 years.

Our settings for the classifier were k = 9,
γ = 0.5, and Maximum Weight = ∞. The set-
ing for the pattern extractor were α = 1.5,
β = 0.2, Buffer = 4.

Table 2: The 15 most powerful patterns extracted from the Belukha core.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>NO₃~Mg [1,11]</td>
<td>19 0 28 2</td>
<td>0.85 0.53 0.89</td>
<td>Y 1</td>
</tr>
<tr>
<td>NO₃~Ca [1,8]</td>
<td>21 0 28 4</td>
<td>0.86 0.68 0.86</td>
<td>Y 2</td>
</tr>
<tr>
<td>NO₃<del>Mg</del> [0,0]</td>
<td>17 0 17 18</td>
<td>0.84 0.60 0.85</td>
<td>A 3</td>
</tr>
<tr>
<td>NO₃<del>Cl</del> [0,3]</td>
<td>26 0 28 14</td>
<td>0.88 0.92 0.84</td>
<td>X 1</td>
</tr>
<tr>
<td>NO₃<del>NO₃</del> [1,2]</td>
<td>27 0 28 28</td>
<td>0.89 1.00 0.83</td>
<td>Z 2</td>
</tr>
<tr>
<td>NO₃<del>K</del> [0,9]</td>
<td>20 0 28 4</td>
<td>0.86 0.76 0.84</td>
<td>Y 1</td>
</tr>
<tr>
<td>NO₃<del>Ca</del> [0,3]</td>
<td>26 0 28 17</td>
<td>0.88 0.96 0.83</td>
<td>X 1</td>
</tr>
<tr>
<td>NO₃<del>Mg</del> [0,3]</td>
<td>26 0 28 18</td>
<td>0.88 0.97 0.83</td>
<td>X 1</td>
</tr>
<tr>
<td>NO₃<del>K</del> [4,5]</td>
<td>24 0 28 22</td>
<td>0.88 0.93 0.83</td>
<td>Z 1</td>
</tr>
<tr>
<td>NO₃<del>Mg</del> [1,9]</td>
<td>20 0 28 5</td>
<td>0.86 0.81 0.83</td>
<td>Y 1</td>
</tr>
<tr>
<td>NO₃<del>Na</del> [0,5]</td>
<td>24 0 28 12</td>
<td>0.88 0.95 0.83</td>
<td>X 1</td>
</tr>
<tr>
<td>NO₃<del>Cl</del> [0,7]</td>
<td>22 0 28 8</td>
<td>0.87 0.92 0.82</td>
<td>Y 1</td>
</tr>
<tr>
<td>K<del>NO₃</del> [0,0]</td>
<td>22 0 22 28</td>
<td>0.87 0.93 0.82</td>
<td>Z 2</td>
</tr>
<tr>
<td>K<del>NO₃</del> [5,5]</td>
<td>1 0 1 1</td>
<td>0.56 0.03 0.82</td>
<td>W 1</td>
</tr>
<tr>
<td>K<del>SO₄</del> [5,5]</td>
<td>1 0 1 1</td>
<td>0.56 0.03 0.82</td>
<td>W 1</td>
</tr>
</tbody>
</table>

The majority of these patterns scored a
1 because they had very unlikely precursor,
successor relationship, such as a Flat trend
preceding a Spike, and they had long time
windows. The token width of over 6 years makes it very unlikely that a trend would be correlated with another trend more than 1 or 2 token units later. This is especially true for a Flat precursor, since Flat is a benign signal and usually indicates the lack of any significant events. Various strings scored a 2 due to a close relation with some more interesting results. For instance, string 2 could be modified so that the precursor clause was a Spike in nitrate, rather than a Flat trend. One cause of a nitrate spike could be a forest fire. The aftermath of such a fire could leave soil unanchored due to the destruction of root systems and wind shields, leading to an increase in the amount of dust and soil picked up by the wind. This could lead to a rise in quantities associated with dust deposition, such as calcium.

String 5 describes a pattern where a Flat trend in nitrate levels tends to precede more Flat trends in nitrate. This could make intuitive sense in that a system in one state may tend to stay close to that state in the near future.

String 13 describes a potentially interesting relationship between potassium and nitrate. Perhaps it could indicate that events that tend to affect potassium and nitrate levels are absent together. However, taking into account the empirical analysis of the string, we again see that sheer frequency of the Flat nitrate tokens (and Flat potassium tokens for that matter) make it unlikely this has any real basis.

The strongest pattern in this set is String 3, which may not be surprising considering it is the only string that was empirically classified as Type A. The structure of the string suggests Flat behavior in calcium and magnesium quantities tend to coincide. This could be indicative of a covariance between these values, but there is no immediately obvious climatic driving force that would lead to this.

6.5 GISP2 Pattern Extraction

In this portion of the analysis, we analyze data from the GISP2 ice core. We used the $Ca$, $Cl$, $K$, $Mg$, $Na$, $NH_4$, $NO_3$, and $SO_4$ quantities between the years of 14.74 to 110,395 (years before 2000). We split the raw series into 100 chunks, which corresponds to a token width of $\approx 1100$ years.
Table 3: The 15 most powerful patterns extracted from the GISP2 core.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Cl~Na:[0,0]</td>
<td>15 0 15 17</td>
<td>0.89 0.17 1.21</td>
<td>A 5</td>
</tr>
<tr>
<td>2 NO3~NO3:[37,68]</td>
<td>32 0 95 1</td>
<td>0.94 0.28 1.19</td>
<td>Y 1</td>
</tr>
<tr>
<td>3 K~NO3:[37,68]</td>
<td>32 0 72 1</td>
<td>0.94 0.28 1.19</td>
<td>Y 1</td>
</tr>
<tr>
<td>4 Cl~NO3:[37,68]</td>
<td>31 0 79 1</td>
<td>0.94 0.28 1.19</td>
<td>Y 1</td>
</tr>
<tr>
<td>5 Mg~NO3:[37,68]</td>
<td>31 0 70 1</td>
<td>0.94 0.28 1.19</td>
<td>Y 1</td>
</tr>
<tr>
<td>6 SO4~NO3:[37,68]</td>
<td>31 0 64 1</td>
<td>0.94 0.28 1.19</td>
<td>Y 1</td>
</tr>
<tr>
<td>7 Na~NO3:[37,68]</td>
<td>31 0 74 1</td>
<td>0.94 0.28 1.19</td>
<td>Y 1</td>
</tr>
<tr>
<td>8 NH4~NO3:[37,68]</td>
<td>30 0 87 1</td>
<td>0.94 0.28 1.18</td>
<td>Y 2</td>
</tr>
<tr>
<td>9 K~SO4:[0,0]</td>
<td>15 0 15 19</td>
<td>0.89 0.19 1.18</td>
<td>A 3</td>
</tr>
<tr>
<td>10 Ca~NO3:[37,68]</td>
<td>28 0 61 1</td>
<td>0.94 0.28 1.18</td>
<td>Y 1</td>
</tr>
<tr>
<td>11 SO4~Ca:[0,0]</td>
<td>14 0 14 19</td>
<td>0.89 0.19 1.17</td>
<td>A 3</td>
</tr>
<tr>
<td>12 K~Ca:[0,0]</td>
<td>15 0 15 20</td>
<td>0.89 0.20 1.17</td>
<td>A 3</td>
</tr>
<tr>
<td>13 K~NO3:[99,99]</td>
<td>1 0 72 1</td>
<td>0.60 0.01 1.17</td>
<td>M 1</td>
</tr>
<tr>
<td>14 K~Cl:[64,64]</td>
<td>1 0 15 1</td>
<td>0.60 0.01 1.17</td>
<td>M 1</td>
</tr>
<tr>
<td>15 K~NO3:[66,66]</td>
<td>1 0 15 1</td>
<td>0.60 0.01 1.17</td>
<td>M 1</td>
</tr>
</tbody>
</table>

Our settings for the classifier were $k = 9$, $\gamma = 0.5$, and $\text{Maximum Weight} = \infty$. The settings for the pattern extractor were $\alpha = 1.5$, $\beta = 0.2$, $\text{Buffer} = 2$.

String 1 was the most strongly evaluated string by all metrics. String 1 describes a pattern where a drop in chloride levels tends to coincide with a drop in sodium levels. Because both token types that match the clauses of the pattern string are rather uncommon (both with a frequency of $\approx 15\%$), this string is strong in the empirical sense that the matches most likely aren’t due to just random chance. Furthermore, two trends coinciding on a Decreasing trend is more interesting than Flat trends coinciding, as we saw several times in the previous set of determinate matches, making them irrelevant.

In regards to strings 13, 14 and 15, they are most like strings of category W in terms of their empirical behavior, but they received a classification of M because they do not strictly fit the definition. The successor tokens are singleton events in all cases, and the time windows are overly specialized, but the precursor tokens aren’t exceedingly rare in the series. But these behave like W strings because the long windows cause all except the earliest precursor token to experience indeterminate matches, making them irrelevant.
results. This pattern is also significant in that there is a climatic mechanism behind this that is immediately apparent, and that is seawater deposition. Seawater contains significant concentrations of dissolved ionic compounds, one of which is sodium chloride, or regular table salt. Although there could be several ways for these sodium and chlorine to be deposited in an ice core, this pattern suggests that the values tend to strongly correlated with one another, most likely indicating that one source is responsible for most of these values. The most likely source that could supply both of these particles is seawater.

Beyond this correlation, there were three other patterns, strings, 9, 11, and 12 that were of a similar form, where two distinct quantities experienced the same behaviors at the same time. These were all classified as Type A, indicating that these apparent patterns probably aren’t due to random chance. Although there is no immediately obvious climatic process that would cause these quantities to be strongly correlated, the patterns exhibit that such a climatic process may exist.

Many of the strings that received a score of 1 were of Type Y, indicating that common precursors were being matched with a rare successor. In fact, strings 2-8 and 10 all used the same Increasing trend nitrate tokens to produce their matches. All of them also had the same time window of [37,68]. This long time window alone shows that these patterns are erroneous because there is very little chance two events could be correlated over the course of more than 40,000 years. Upon further inspection of the data, it appears that these time windows could be artifacts of the increased resolution of the more recent values.21 Another consequence of this is that the older values have less variability, so the older chunks are more likely to be classified as Flat trends. This is one of the factors that allows these patterns, which all have a Flat trend as their precursor and a time window covering more than half of the record, to have a highly rated power. This demonstrates the problem with very long records due to compression that was not as apparent in the previous runs.

21 Recall, this is due to the compression and horizontal flowing of the ice at lower depths, which leads to fewer samples per unit time. [22]
However, we gave one of these patterns, string 8, a score of 2 because there could be a related pattern that is more valid. We single out this string because the quantity associated with its precursor is ammonium, $NH_4$. Since both ammonium and nitrate are nitrogen based, it is possible there could be some process underlying their respective behaviors, although we still make no claims about what that may be.

7 Discussion

Our test runs showed many promising results, as well as revealed weaknesses in the system. Our results revealed that the tokenization classification system has success with certain types of data, such as the simple Seasonal data run, and is less successful with data that is more complex and has not been formatted specifically for tokenization, such as the Tambora data. Our results also showed that pattern extraction tends to produce a significant amount of “junk” strings, but also is capable of finding some potentially climatically sound, albeit basic, patterns.

Arguably, one of the most significant weaknesses of this system is the tokenization process. Inaccurate tokens cause inaccurate results in turn from higher order features, such as pattern matching and pattern extraction. While it is possible to manually correct misclassified tokens, the failure rate of the classifier would likely make this a comprehensive process. This would be in conflict with the original intention of the tool: to correct the occasional failure of the system. As the ultimate goal of this research is to reduce the amount of human attention needed to process ice cores, it is essential to improve the accuracy of the tokenization process.

It may be possible to solve some of these problems with a larger and more precise training data set for the K-Nearest Neighbors classifier, or by redesigning the feature-space of the classifier, but this approach is unlikely to alleviate the inherent problem of sensitivity to partitioning during the chunking process. Changing either the chunk width or the initial point of partitioning can create very different tokens due to the local nature of the tokenization process.
Figure 12: A synthetic seasonal time series that has been chunked with two different chunk widths. The top series has a chunk width of 100, which is equal to half the period of the signal. The bottom series has a chunk width of 200, which equals the full period of the signal.

Figure 12 shows a case where doubling the chunk width drastically alters the resultant token stream. In the top series, the partition boundaries coincide with the extrema of the signal. This creates a token stream of alternating Increasing and Decreasing tokens, accurately capturing the oscillatory nature of the data. The bottom series uses a chunk width twice as wide, which causes the partition boundaries to coincide only with the maxima of the signal. Then each sub-series appears as a horseshoe shape, which is interpreted as Flat by the system, since that is the token that best describes the behavior. This causes the token stream to fail to capture the underlying signal.

One possible area for future work is the exploration of a generalized system that can process variable-width tokens. If the system could automatically produce a partition that optimizes the explanatory power of the tokens, this could both solve the sensitivity problem and save a researcher from having to manually determine how to partition a data set.

Beyond tokenization, the sample runs also reveal potential weaknesses in the pattern extraction methods, specifically with regards to pattern evaluation, and even pattern matching. Although the evaluation method is supposed to take into account several competing factors that affect string quality, there are still ways that strings can “beat the system” and have a low explanatory power, but a high power rating, as demonstrated by the need to define 5 types of erroneous string categories. Note that most of these categories described strings with successor tokens of one frequency or
another interacting with precursor tokens that appear very frequently. At least part of the reason that these strings appear to have their power inflated is due to the many-to-one model for forming matches, where multiple precursor tokens can each “share” the same successor and each count as a match. Furthermore, it doesn’t appear that the issue of inaccurate evaluations can be solved solely by tuning the evaluation parameters $\alpha, \beta$, and Buffer. We tried many different settings for these parameters outside of the trials recorded, and it appears that each approach produces strings of the different false positive categories, just in different proportions of each.

Perhaps one area for future work could be to allow a user to create a blacklist of string types he or she doesn’t want to generate. Since, in our runs, the token types that caused these problems by being very common were Flat trends, this blacklist could include any string with a Flat trend type. This would likely be an agreeable solution since the main behaviors of interest are sporadic and steady trends, and it is unclear if a pattern involving a Flat behavior is meaningful.

8 Conclusion

Despite the shortcomings in the system, the ultimate goal of this research, to demonstrate a proof of concept for applying pattern matching techniques to ice core data, was achieved. We have successfully implemented a command-line application that allows users to interact with and perform operations on climate data. Users are able to create and test patterns, visualize matches within a token stream, and use pattern utilities such as pattern completion and pattern extraction.

Several of the flaws in the system, such as the many-to-one model of pattern matching, could be altered without substantially disrupting the rest of the system. Similarly, there may be ways of improving the tokenization method within the architecture of the current system, such as by providing a larger and more varied training set to the classifier.

Even though the current implementation is still rough, we are hopeful for the future of this application. This research represents a significant step towards the ultimate goal of helping researchers process more data.
References


