SPECTRAL TOP-N: A COMPUTATIONALLY EFFICIENT
TOP-N ALGORITHM FOR BIG DATA ANALYTICS

A Dissertation in
Industrial and Manufacturing Engineering

by

Kenneth Hutchison

© 2015 Kenneth Hutchison

Submitted in Partial Fulfillment
of the Requirements
for the Degree of

Doctor of Philosophy

August 2015
The Dissertation of Kenneth Hutchison was read and approved* by the following:

Soundar Kumara  
Allen E./Allen M. Pearce Professor of Industrial and Manufacturing Engineering  
Professor of Computer Science and Engineering  
Dissertation Adviser  
Chair of Committee

Paul Griffin  
Virginia C., Joseph C. Mello Chair of Industrial and Systems Engineering at Georgia Tech  
Professor of Industrial and Systems Engineering at Georgia Tech

Guodong Pang  
Assistant Professor of Industrial and Manufacturing Engineering

Kamesh Madduri  
Assistant Professor of Computer Science and Engineering

Daniel Finke  
Research Associate, Applied Research Lab  
Special Member

Harriet B. Nembhard  
Professor of Industrial and Manufacturing Engineering  
Interim Department Head

*Signatures on file in the Graduate School.
Abstract

Recommendation systems and their algorithms have been a rich research area of research since the first appearance of collaborative filtering as a data categorization tool. The wealth of available information is growing, and also the research striving to relate this ether of information to the individual. Today, this ether of information has taken on the visage of Big Data; where the individual plays the part of the consumer to make recommendation a seductive research area. Classically, recommendation systems have focused on analyzing relationships between user pairs or user product pairs, which makes a sensible first-cut at many Big Data problems. For instance, given a mass of historical user or purchase data, Top-N can be used to isolate common attributes among groups of users or the groups themselves given $N$ items, which provides a sane default for additional analyst intervention.

Recommender analysis is not without its caveats. Foremost, without implementation of the recommender system, one must resort to models which rely on probabilistic or economic theory [and corresponding assumptions] to compare competing methodologies. In addition, by conventional wisdom, the type of recommendation system chosen will also dictate the type of data analyzed and vice versa. That is to say, in a very diverse ecology of items such as Amazon.com, a recommendation algorithm which is computationally inefficient likely will not thrive. In this train of thought, one of the more robust algorithms developed for recommendation systems has been the Top-N algorithm, in that both user and item similarity can be assessed.
The Top-N criteria, though powerful, suffers from an exponential component which can make calculation untenable when item sets are large. Specifically, the calculation of similarity in a user-item system is $O(\text{users}^2 \times \text{items})$ since we have to perform \text{users}(\text{users} - 1) calculations of a possible \text{items} length. In this thesis, we combine ad-hoc calculation of the Top-N criteria with cluster computing and the relics of signal processing to produce a computationally amenable algorithm for real-time recommendation generation. [Recommender systems by design make assumptions to extrapolate user preference across a dataset of items, and the theory used to measure them also rely heavily on this assumption]

In this thesis, we propose an alternative similarity metric based on spectral analysis and benchmark its computational efficacy by comparing it to traditional or “vanilla” Top-N Algorithms using the Dotproduct, Cosine, and Hamming measures. The method itself stems from a paradigm shift into the frequency domain: where we leverage the computational efficiency of the Fast Fourier transform in order to obtain a similarity metric suited for Big-Data or real-time analysis. We benchmark the Spectral similarity metric by: comparison to traditional algorithms, through full-factorial experimentation, Big-Data case-studies, and extensions to other areas of data mining which struggle with combinatorial complexity; in this case: the Apriori Algorithm.

Our method allows for a small loss in accuracy, but significant benefits in terms of computational speed and random access memory consumption when dealing with the large binary matrices found in Top-N problems. We gain this advantage by using spectral analysis to compress and summarize the data, which decreases the computational effort and space required to compute the item or customer distance matrix.
# Table of Contents

List of Tables ................................................................. vii

List of Figures ............................................................... ix

Acknowledgments ............................................................. xi

Chapter 1. Introduction and Problem Context ........................................ 1
  1.1 Research Objectives ..................................................... 3
  1.2 Methodology ............................................................. 4
  1.3 Uniqueness and Contributions .......................................... 4
  1.4 Organization of the Dissertation ....................................... 5

Chapter 2. Literature Review of Recommendation Systems ......................... 6
  2.1 Collaborative Filtering .................................................. 6
     2.1.1 Memory or Information Based ..................................... 7
     2.1.2 Content/Feature Based ............................................ 12
  2.2 Major Challenges ...................................................... 13
  2.3 Dimensionality Reduction .............................................. 15

Chapter 3. Why Spectral Analysis? ............................................... 24
  3.1 History of Spectral Analysis ........................................... 24
  3.2 Nonparametric Method .................................................. 27
  3.3 Parametric Method ...................................................... 28

Chapter 4. The Hutchison-Kumara Measure and its Ecology ......................... 33
  4.1 Introduction ............................................................. 33
  4.2 The Hutchison-Kumara Measure of Similarity ......................... 33
  4.3 Properties of the H-K Similarity Measure ............................ 35
     4.3.1 Principal of Nominal Increase .................................. 36
        4.3.1.1 Evaluating the Cases ........................................ 39
     4.3.2 Another Good Property .......................................... 41
  4.4 Derivation of Computational Complexity ............................... 41
     4.4.1 Dotproduct Distance .............................................. 42
     4.4.2 Cosine Distance .................................................. 43
     4.4.3 Hutchison-Kumara ............................................... 44
     4.4.4 Hamming Distance ............................................... 45
  4.5 Storage Requirements: Observations and Discussion ...................... 46
     4.5.1 Assumptions ...................................................... 46
     4.5.2 Distance Matrix Analyses ....................................... 46
     4.5.3 Absolute Upper Limits for Tenable Calculation .................. 47
     4.5.4 H-K Measure: Row and Column Permutation ....................... 48
## List of Tables

2.1 A Literary Summary ........................................... 23
4.1 The Possible Cases for the Binary Variables and Constants .... 39
5.1 Factor Levels for Experiments 1-3 ............................... 57
5.2 Average Cosine Runtime in Microseconds In Five Separate Trials . 75
5.3 Dotproduct Runtime in Microseconds Across Five Separate Trials . 75
5.4 Hamming Runtime in Microseconds Across Five Separate Trials . 76
5.5 Spectral Runtime in Microseconds Across Five Separate Trials . 76
5.6 Cosine Memory Consumption in KB Across Five Separate Trials . 77
5.7 DotProduct Memory Consumption in KB Across Five Separate Trials . 77
5.8 Hamming Memory Consumption in KB Across Five Separate Trials . 78
5.9 Spectral Memory Consumption in KB Across Five Separate Trials . 78
6.1 Factor Levels for Filtering Experiment .......................... 83
6.2 Wilcoxon Rank Sum Test with Continuity Correction: Filter Window against Hamming-Spectral Agreement .......................... 88
6.3 Mean Memory Consumed in KB by Filter Span ................. 89
6.4 Mean Runtime in Milliseconds by Filter Span .................. 89
6.5 Factor Levels for Sparsity Experiment ........................... 91
6.6 Kruskal-Wallis Rank Sum Test: Hamming Spectral Agreement against \( P_{00} \) .......................... 93
6.7 Kruskal-Wallis Rank Sum Tests \( P_{00} \) against Hamming and Spectral Run-times and Memory Consumption ...................... 95
6.8 Hardware Thread Topology of this SandyBridge Processor [CPU Stepping: 7] ............................................. 97
6.9 Cache Topology of this SandyBridge Processor ................. 98
6.10 NUMA Topology of this SandyBridge Computer ............... 98
6.11 Clock and Power Consumption of this SandyBridge Processor . 99
6.12 FLOPS Performance of this Processor by Event on \textit{Likwid} Benchmark ................................................. 99
6.13 FLOPS Performance of this Processor by Metric on \textit{Likwid} Benchmark ................................................. 100
6.14 L2 Performance by Event on \textit{Likwid} Benchmark .......... 100
6.15 L2 Performance by Metric on \textit{Likwid} Benchmark .......... 101
6.16 L2 Cache Performance by Event on \textit{Likwid} Benchmark .. 101
6.17 L2 Cache Performance by Metric on \textit{Likwid} Benchmark .. 102
6.18 Branch Performance by Event on \textit{Likwid} Benchmark .......... 102
6.19 Branch Performance by Metric on \textit{Likwid} Benchmark .... 103
6.20 Benchmark 1 for Computational Architectures ............... 103
6.21 Mean Runtime in Seconds by Number of Cores ............... 104
6.22 Quadratic Models Describing Runtime by Method ............. 106
7.1 DOE Factor Levels .................................................. 124
8.1 Top-N Identified by Method .............................................. 132
8.2 Run-Time and RAM Consumed .......................................... 132
# List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>Spectral Decomposition Example</td>
</tr>
<tr>
<td>3.2</td>
<td>Experimental Performance vs DotProduct</td>
</tr>
<tr>
<td>5.1</td>
<td>Agreement by Method</td>
</tr>
<tr>
<td>5.2</td>
<td>Density Plot of Accuracy by Customers</td>
</tr>
<tr>
<td>5.3</td>
<td>Density Plot of Accuracy by Items</td>
</tr>
<tr>
<td>5.4</td>
<td>Density Plot of Accuracy by Top-N Items</td>
</tr>
<tr>
<td>5.5</td>
<td>Density Plot of Accuracy by Probability</td>
</tr>
<tr>
<td>5.6</td>
<td>Aggregate Runtime in Microseconds by Method</td>
</tr>
<tr>
<td>5.7</td>
<td>Aggregate Runtime in Microseconds by Customers</td>
</tr>
<tr>
<td>5.8</td>
<td>Aggregate Runtime in Microseconds by Items</td>
</tr>
<tr>
<td>5.9</td>
<td>Aggregate Runtime in Microseconds by Top-N</td>
</tr>
<tr>
<td>5.10</td>
<td>Aggregate Runtime in Microseconds by Probability</td>
</tr>
<tr>
<td>5.11</td>
<td>RAM Consumption by Method</td>
</tr>
<tr>
<td>5.12</td>
<td>RAM Consumption by Customers</td>
</tr>
<tr>
<td>5.13</td>
<td>RAM Consumption by Items</td>
</tr>
<tr>
<td>5.14</td>
<td>RAM Consumption by Top-N</td>
</tr>
<tr>
<td>5.15</td>
<td>RAM Consumption by Probability</td>
</tr>
<tr>
<td>5.16</td>
<td>Average Runtime in Microseconds by Method</td>
</tr>
<tr>
<td>5.17</td>
<td>Average RAM Consumed in Kilobytes by Method</td>
</tr>
<tr>
<td>6.1</td>
<td>How Does Filtering Affect Accuracy?</td>
</tr>
<tr>
<td>6.2</td>
<td>Log(Cos and Spectral Agreement) with Filter</td>
</tr>
<tr>
<td>6.3</td>
<td>Log(Cos and Spectral) Memory Consumed (KB) with Filter</td>
</tr>
<tr>
<td>6.4</td>
<td>Spectral Runtime by Filter</td>
</tr>
<tr>
<td>6.5</td>
<td>Normality QQ of Responses</td>
</tr>
<tr>
<td>6.6</td>
<td>A Binary Markov Model</td>
</tr>
<tr>
<td>6.7</td>
<td>The Effect of Transition Probability on Proportion Zero</td>
</tr>
<tr>
<td>6.8</td>
<td>Agreement by Transition Probability</td>
</tr>
<tr>
<td>6.9</td>
<td>Runtime and Memory Consumed in Sparsity</td>
</tr>
<tr>
<td>6.10</td>
<td>Runtime in Benchmark 1: 1 Core</td>
</tr>
<tr>
<td>6.11</td>
<td>Runtime in Benchmark 1: 2 Cores</td>
</tr>
<tr>
<td>6.12</td>
<td>Runtime in Benchmark 1: 4 Cores</td>
</tr>
<tr>
<td>6.13</td>
<td>Runtime in Benchmark 1: 6 Cores</td>
</tr>
<tr>
<td>6.14</td>
<td>Runtime in Benchmark 1: 8 Cores</td>
</tr>
<tr>
<td>6.15</td>
<td>Runtime in Benchmark 1: 36 Cores</td>
</tr>
<tr>
<td>7.1</td>
<td>Noise Process Example</td>
</tr>
<tr>
<td>7.2</td>
<td>Panel plot of demand series</td>
</tr>
<tr>
<td>7.3</td>
<td>Experimental Noise Process Example</td>
</tr>
<tr>
<td>7.4</td>
<td>Quantile-Quantile Plot of Response</td>
</tr>
<tr>
<td>7.5</td>
<td>Factor Levels and Accuracy</td>
</tr>
</tbody>
</table>
7.6 Number Detected by Number of Items .......................... 128
Acknowledgments

I would like to thank Dr. Soundar Kumara, for his contributions to my academic ability and personal character, and for helping me hone the craft of writing throughout the years. I also owe to him the importance of careful thinking (and thus intellectual agility) in the presence of impending challenges. I would like to thank the dissertation committee for their contributions to my work as well as providing poignant questions in their fields of expertise. I would like to thank Dr. Daniel Finke for providing research opportunities throughout my degree which were interesting and challenging. I would also like to thank him for providing a source of funding for the degree.

I would like to thank Dr. Thomas Reed Willemain, for his mentorship throughout my entire educational career, and for instilling in me the fevered pursuit of professional excellence. I would like to thank my family for being understanding and supportive of my educational journey, and for allowing me to place academics first for the past decade. I would like to thank friends who have dealt with a hectic schedule and endured lengthy conversations about abstract and highly technical topics ad nauseam.

I would like to thank the professors, teachers, and mentors in my life, whom have dealt gracefully with a gratingly curious mind. It is a blessing the educators I have encountered have nurtured curiosity; I hope that we continue to allot space for curiosity and scientific literacy in future generations.

I would like to thank the engineers, scientists, mathematicians, and visionaries past and present who have provided me a wealth of knowledge and tools to solve prescient problems that arise in humanity today.

Lastly, I would like to thank any soul who has discussed an idea or curiosity excitedly with me during my tenure on planet earth, this kindling has been invaluably motivating. In these memories I find my most inspired, joyous, and humbling moments: I owe a great debt to these individuals and their companionship on less traveled roads. I am thankful for those who pursue formidable challenges with me, so that we may mine opportunity from failure and discovery: and embrace them all the same.
Alice sighed wearily. “I think you might do something better with the time,” she said, “than waste it in asking riddles which have no answers.”

“If you knew time as well as I do,” said the Hatter, “you wouldn’t talk about wasting it, I dare say you never even spoke to Time!”

“Perhaps not,” Alice cautiously replied, “but I know I have to beat time when I learn music.”

“Oh!, that accounts for it,” said the Hatter. “He won’t stand beating.”

—Lewis Carrol
Chapter 1

Introduction and Problem Context

As available information becomes ubiquitous in our daily lives, techniques which leverage it for knowledge discovery also become crucial. Fruitful use of the human data-exhaust has become the norm for corporate strategy and for effective use of social networking. Recommender systems (Resnick & Varian 1997) have gained notoriety in this area because they are among few methods with more than a decades tenure to triage Big Data phenomena. One of the more popular models in enterprise is the Top-N recommendation algorithm [in the class of memory-based algorithms]. Top-N retains the recommendation accuracy of memory-based models, but when used in conjunction with clever user/item clustering apriori: remains scalable. The basic definition of the Top-N problem and it’s corresponding algorithm are the following (Deshpande & Karypis 2004):

\[
\text{Given a user-item matrix } R \text{ and a set of items } U \text{ that have been purchased by each user, identify an ordered set of items } X \text{ such that } \|X\| \leq N \text{ and } X \cap U = \emptyset.
\]

Given an \(n \times m\) user-item matrix \(R\) and a parameter which specifies the number of item-item similarities to be stored for each item \((k)\), the output \(\rho_{i,j}\) demonstrates the degree of similarity between \(i,j\) or \(j,i\).

*The similarity metrics calculated in the Algorithm are varied; the most popular are: Dot-product, Cosine (eqn2.1) Pearson Correlation (eqn.2.2) and Adjusted Cosine (eqn2.3).*
Algorithm 1 Top-N Model Creation

for $j=1...m$ do
  for $i=1...m$ do
    if $i \neq j$ then
      $\rho_{i,j} =$ similarity($R(\ast, j), R(\ast, i)$)
    else
      Let $\rho_{i,j} = 0$
    end if
  end for
  for $i=1...m$ do
    if $\rho_{i,j} \neq$ the k largest values in $\rho_{*,j}$ then
      $\rho_{i,j} = 0$
    end if
  end for
end for
Return $\rho$

There are disadvantages to the model in that calculation of the similarity metric distance matrix is a computational feat, requiring a trade-off in accuracy as smaller clusters and user-neighborhoods must be explored to produce novel recommendation. It stands to reason that in large applications, user-item matrices can become very large [the reason for this is because the number of customers can increase without bound]. Further, in large applications, the number of items which are sold by a retailer can be prohibitively large (Amazon.com for instance). The items themselves can number in the thousands or hundreds of thousands. However, the customers do not often purchase the majority of available items. As a result: user-item matrices tend to be very sparse [often binary] matrices. If a new customer purchases a single item, in order to calculate his/her similarity with respect to pre-existing customers there will be a vector describing the purchase pattern that is approximately items length but contains $items - 1$ zeros. In this dissertation, we grapple with the sparse binary matrices that are in vogue for Top-N input in a novel manner.
In this dissertation, we use signal processing techniques to address this problem. Signal processing allows for a tune-able symmetry calculation which offers tremendous practical speed advantages. These advantages stem from the ability of signal processing to summarize the series. The model is offered in both parametric and nonparametric varieties: using the Danielli Kernel in order to approximate the Spectra and also Autoregressive Integrated Moving Average models [in the case of continuous series]. The method will be benchmarked by its accuracy and computational efficiency relative to standard implementations [given in Algorithm 1] employing a few of the popular similarity metrics in the literature. After these experiments, we will investigate a few factors which may alter algorithm performance and ultimately extend the model to a different problem area for testing, eventually vetting both methods in a Case Study paradigm.

The performance and accuracy of a contrived method can be characterized in the following way:

1. "Closeness" to classical Top-N solution in accuracy.

2. Storage requirements relative to classical Top-N at execution time.

3. Speed increase relative to classical Top-N with different similarity metrics.

1.1 Research Objectives

The purpose of this section is to outline the primary research objectives of this dissertation, as well as unique work, and its associated contributions to the field.

1. Design, develop, and implement a Top-N algorithm which has significant performance advantages as compared to the vanilla algorithm by the metrics above.
2. Theoretically validate the viability of the developed algorithm.

3. Experimentally determine the best scenarios in which to use the algorithm, and also scenarios in which to employ caution.

4. Since data compression will harm accuracy, we should characterize the accuracy/performance trade-off of the algorithm so that practitioners can make informed decisions.

1.2 Methodology

In order to get the desired data compression, we first transform the customer-item matrices into the spectral domain. We then develop and characterize a measure to quantify the similarity in the resulting transformed series. We characterize the loss of accuracy and the computational benefits using a rigorous set of parameter ranges and noise generation processes using full factorial computational experiments. We then attempt to explore possible caveats for the method itself, and extend it to the Apriori Algorithm. We conclude with a case study to characterize the difference in accuracy, runtime, and RAM consumption in a BigData scenario using continuous data.

1.3 Uniqueness and Contributions

The focus of this dissertation at a very high level is the development of a method by which we can speed up the calculation of the similarity measure. We do this in a way that has not been done before: using the spectral domain to compress existing data and reduce the number of operations required to compute the distance matrix. Though there is no change in the O(n) notation, we can see that reducing the number of
calculations by a factor of two is likely of practical use and benefit both for computation
time and RAM consumption. Further, we extend this method to the Apriori algorithm
using nonparametric methods in a way that has not been done to quickly detect frequent
item-sets simultaneously provided the data is collected at such high resolution that we
can qualify each demand series as a time series. We do posit that to the best of our
knowledge, the work reported in the thesis is the first to deal with this problem in this
manner.

1.4 Organization of the Dissertation

This dissertation is organized into nine chapters. The first chapter introduces
the problem and the motivation for solving it. The second chapter deals with the rele-
vant literature and the shortcomings encountered within it. The third chapter describes
the measure which we derive and its ecology relative to other methods, as well as the
theoretical properties which it possesses. The fourth chapter describes spectral similar-
ity itself and the various situations under which it can be employed. The fifth chapter
describes the preliminary analysis of the method and it’s performance relative to bench-
mark algorithms. The sixth chapter explores possible caveats for the method and their
effect on the performance of the method. The seventh chapter describes the extension
of the method to the Apriori algorithm. The eighth chapter concludes the analysis with
case studies using actual data; and the ninth chapter summarizes the conclusions and
uniqueness of the research, and points to future research.
Chapter 2

Literature Review of Recommendation Systems

In this chapter, we discuss the relevant literature on recommendation systems and the Top-N algorithm: beginning with the field of collaborative filtering associated methodologies.

Recommendation System Types

2.1 Collaborative Filtering

Collaborative filtering (CF) utilizes historical information based on a large group of users in order to generate items which appeal to the current user in question [with no loss of generality in describing items instead]. CF methods are generally classified into Memory and Model-Based (Lü et al. 2012). Further, users [or items] can be partitioned more thoughtfully into nested communities so that inference can be generalized strongly (Borden et al. 2005) and computational effort decreased. Intuitively, collaborative filtering is the reason we read Amazon reviews, or ask our friend his/her opinion on a recent purchase. Collaborative filtering is simply the evolution of word-of-mouth opinion sharing via data collection and analysis (Schafer et al. 2007).

To extrapolate a user’s utility function over the item-space: collaborative filtering relies on one of many rating systems. Maintaining peace with statisticians requires us to make sure many items and users are present in the recommendation environment because
these methods are predominantly parametric/probabilistic in nature. Intuitively, given
a sufficiently large user space, another user must have similar needs or preferences; and
as a result items can be recommended which a similar user has shown preference for.

General drawbacks for collaborative filtering are sparsity, cold start[the inability
to characterize a user or item without any ratings apriori in an unsupervised manner], and
temporal variability. Since we often want many more users than items, sparsity makes
computation difficult. Further, new items have not received ratings and are difficult to
recommend or categorize [an issue content-based recommendation does not share]. Also,
if a user once was a collector of baseball cards and now has taken to postage stamps,
we should no longer recommend baseball paraphernalia to this individual; such is the
nature of temporal bias.

2.1.1 Memory or Information Based

Memory based collaborative filtering (user-based Collaborative Filtering (Park
et al. 2007)) typically attempts to correlate users based on historical ratings or preferences. These methods are generally unpopular in that computation of a relative corre-
lation metric is poorly scalable. Often, there are many new computations which must
be performed with the addition of each new rating  (Karypis 2001). Though, we are
likely to see a resurgence in these techniques when computational power is at less of a
premium, much like the statistical bootstrap.

Simple Cosine and Correlation The cosine similarity metric  (Sarwar et al. 2001), (Jiang
& Wang 2010), (Sarwar et al. 2000a), (Cantador et al. 2010) treats items as vectors in the
user-space, and the similarity is measured by the cosine of the angle between respective vectors. The similarity between items or users \((i, j)\) is given by:

\[
similarity(i, j) = \cos(i, j) = \frac{i \cdot j}{\|i\|_2 \cdot \|j\|_2}
\]  

(2.1)

The cosine similarity metric and its variants are pivotal in both Google’s PageRank algorithm and also the rating engine used by Amazon (Linden et al. 2003). The vanilla algorithm does not escape the exponential complexity of the distance matrix. Since a complete computation of similarity requires the Dotproduct of all user-item pairs, this metric can certainly be computationally expensive without further refinement of the user sets. A different method utilizes the Pearson correlation coefficient to relate the similarity of two items \(i\) and \(j\). To do this, we must find all users who rated both \(i\) and \(j\) (defined as set \(\beta\)) and compute the similarity:

\[
similarity(i, j) = \frac{\sum_{b \in \beta} (R_{b,j} - \bar{R}_i)(R_{b,j} - \bar{R}_j)}{\sqrt{\sum_{b \in \beta} (R_{b,j} - \bar{R}_i)^2} \sqrt{\sum_{b \in \beta} (R_{b,j} - \bar{R}_j)^2}}
\]  

(2.2)

Adjusted Cosine

In the user-user similarity computation, we note that the item ratings will be scaled between an upper and lower value and assumes a consistent scale. In item-item similarity, there will be different rating scales per user that should be accounted for. The adjusted cosine (Sarwar et al. 2001) uses the \(b_{th}\) user’s rating average \((R_b)\) to standardize appropriately:
\[
similarity(i, j) = \frac{\sum_{b \in \beta}(R_{b,i} - \bar{R}_b)(R_{b,j} - \bar{R}_b)}{\sqrt{\sum_{b \in \beta} (R_{b,i} - \bar{R}_b)^2} \sqrt{\sum_{b \in \beta} (R_{b,j} - \bar{R}_b)^2}} \quad (2.3)
\]

**Conditional Probability**

Attempts to treat similarity as the conditional probability that one item is purchased given that another item has been purchased. The resulting conditional probability of \(P(i|j)\) is the number of of items customers who purchased both divided by the customers which bought \(j\).

\[
P(i|j) = \frac{\text{Card}(i \cap j)}{\text{freq}(j)} \quad (2.4)
\]

**Default Voting**

Default voting is the practice of imputing missing observations for the user in question or his matching user in question. The practice of assigning a default vote could be used to impute preferences within a product category given a user’s purchasing habits. This of course would require a very intimate understanding of the product topology and the user behavior en masse’ and thus is not necessarily fruitful here.

Default voting is often used in simpler contextual environments. For instance, a visit to a given website may imply a rating of 1 whereas never visiting said website would give a rating of 0. These observations can further hone the recommendation process, though it again should be noted that they require very specific contextual understanding (Breese et al. 1998).
Inverse User Frequency

In text analysis, there is an obvious notion of the tradeoff between relevance and frequency. For instance, very common words will be found in almost any document, whereas the slightly less common words may speak volumes towards topic similarity; because intuitively algorithms should not labor on articles and qualifiers of the language. Further, we should consider it a sin to relate similarity based on the grammatical construction of documents. In this spirit, inverse user frequency endeavors to downscale items which are universally common and value the items which are more unique to the user him/herself. The intuition is that, if we both visit the grocery store: our similarity is not bread and butter, but rather the more eclectic purchases we share. Further, the $f_j$ component tends to point towards a universal like or dislike of items. Let $f_j = \frac{n_j}{n}$ where $n$ is the total number of users and $n_j$ is the number of users rating item $j$. Also, let $R_{a,j}$ represent the rating of user $j$ given to item $a$ and $R_{b,j}$ represent the rating of user $j$ given to item $b$. (Breese et al. 1998).

$$w(a, b) = \frac{\sum_j f_j \sum_j f_j R_{a,j} R_{b,j} - (\sum_j f_j R_{a,j})(\sum_j f_j R_{b,j})}{\sqrt{\sum_j f_j (\sum_j R_{a,j}^2 - (\sum_j f_j R_{a,j})^2) \sum_j f_j (\sum_j R_{b,j}^2 - (\sum_j f_j R_{b,j})^2)}} \quad (2.5)$$

Model Based Methods

Model based methods have been developed for two reasons. First, in the absence of many ratings for a given item, it is more comforting to have theoretical models supporting guesses. Second, in environments where data is prohibitively large, parametric models can allow for fast computation of rating estimates in many cases. Or, at the very least,
provide quick approximations to the true quantity (e.g. for statistical models sampling is in vogue.)

**Bayesian Probability**

Bayesian probability analysis is generally interested in estimating whether or not the user or item belongs to a hidden class $c$. (Schafer et al. 2007) So then the probability of user $u$ rating item $i$ a value $r$:

$$P(r|u,i) = \sum_c P(r|i,c)P(c|u) \quad (2.6)$$

and the necessary rating prediction:

$$E(r|u,i) = \sum_r (r \ast P(r|u,i)) \quad (2.7)$$

The Bayesian methods have many advantages; among which is the panoply of optimization algorithms to refine the estimates. (Hofmann 2004) Also, the Bayesian probabilities allow for the computation of confidence metrics. (Melville et al. 2002)

**Graph and Cluster Methods**

Depiction of the problem as a graph has led to a wealth of similarity indices: the Preferential Attachment Index (Barabási & Albert 1999), the Resource Allocation Index (Zhou et al. 2009), the Adamic-Adar Index (Adamic & Adar 2003), the Leicht-Holme-Newman Index (Leicht et al. 2006), Hub Depressed Index and Hub Promoted Index (Ravasz et al. 2002), Sorensen Index (Sørensen 1948), Jaccard Index (Jaccard 1901), and Salton Index (Salton & McGill 1986) (Lü et al. 2012). It is difficult to say
which among these is best, and the choice of index is likely suited to the application area. Yet another way to take advantage of structure within the data is to use random graph methods. There are several diffusion based methods which use the random graph, though a general example using Markov Chains leads us more clearly to the intuition (Yildirim & Krishnamoorthy 2008). In addition to these, there are several methods which use properties of the random walk: A few of these are Average Commute Time (Fouss et al. 2007), SimRank (Jeh & Widom 2002), (Lizorkin et al. 2008), (Li et al. 2010), Random Walk (with superposed and local (Liu & Lü 2010) variants), and Matrix Forest Index (Chebotarev & Shamis 2006).

2.1.2 Content/Feature Based

Feature based methods assume similar features will cause items to be rated similarly. In content based recommendation systems, typically the user will have a set of interests or a profile, which will then be matched with a set of items based on the features inherent to the items. The advantage of these systems is that cold start is not an issue, since features can be determined apriori; however, the user must supply more input relative to other techniques (Pazzani & Billsus 2007). Naturally, many are trying to use unsupervised learning and social media to mine user preferences for advertising purposes. Here we save some room for the elephant: unstructured data. The lack of amenable structure in social media makes surveying user preferences a painstaking process which may deter the user or result in less powerful data to make predictions.
Tags

Tagging is becoming a more prevalent tool in recent times, as it was touted in conjunction with much literature praising Web 2.0 (Song et al. 2011), (Adrian et al. 2007), (Rendle & Schmidt-Thieme 2010). In its refined form though, tag generation is simply mining appropriate keywords and treating them as features. Here we find the term-document calculations of yesteryear quite handy (Song et al. 2008) though the determination of appropriate tags is as much a question of semantic language processing as it is statistical. Tags have been employed with such esoteric methods as dirichlet allocation (Krestel et al. 2009) and in unsupervised environments such as social networks and music playlisting (Rae et al. 2010; Bu et al. 2010). In short, tags are what we call words, when words are used as features for content based recommendation.

Relevant Information/Rocchios Algorithm

Rocchios algorithm (Rocchio 1971; Joachims 1996) is a support vector construct used to employ feature selection given a bag of words. For instance, the most frequently occurring words are likely not useful in text processing of feature selection, and so they are discarded. In contrast, highly infrequent words are also not useful. The objective of Rocchios algorithm is to develop a model by which to discover words of high mutual information with the target and utilize these for feature selection (Rocchio 1971).

2.2 Major Challenges

Scalability and sparsity are often two sides of the same issue as they apply to memory based recommendation systems. In large user-item matrices sparseness is often
inescapable. In fact, memory based models are well known to be difficult to scale (Kim & Kim 2003). This issue brings to life a new research area: one of dimensional reduction. In order to represent data more compactly and retain relevant information, researchers have resorted to singular value decomposition (Kim & Cho 2003) and latent dirichlet allocation (Hoffman et al. 2010; Blei et al. 2003), though several other [less frequently used] methods exist (Daoud et al. 2009; Hofmann 1999; Middleton et al. 2004; Cantador et al. 2008). It should also be noted that often, scalable algorithms which deal with sparsity well [i.e. nearest neighbor, EM] generally sacrifice coverage and accuracy of the resulting recommendations (Konstan et al. 1997).

Another challenge is the cold-start phenomenon. In the case of a memory-based recommendation system, the addition of a new item can be very difficult to categorize. Similarly, the recommendation system may require a long warm-up period before providing quality results. In the case of content based systems, this is largely avoided in that features about the item/merchandise can be easily categorized before entering the system. New users or unresponsive users however will suffer lower quality recommendation until more information about the user is gathered. Yet another issue in the memory-based recommendation system paradigm is the tendency for false positives when interests shift. In other words, an old hobby or gift for a friend should probably not affect the recommendations in a perfect world. This is a challenge often mitigated by allowing the customer to tag the item as a gift or by submitting reviews on old merchandise thereby encouraging methods like default voting to work in the absence of perfect information.

The development of reliable accuracy metrics for recommendation systems has been a difficult task; since without implementation, one can become trapped in a schema
of assumptions that is decidedly false. Classically, accuracy metrics used have been of the holdout sample variety, where the system is withheld from a small group and analysis post-hoc provides insight into the quality of the method. Though, the uses of some metrics have been debatable in terms of efficacy. (McNee et al. 2006) There are a wealth of literature which actively debates the use of Root Mean Squared Error vs Median Adjusted Percentage Error vs Absolute Percentage Error etc. The consensus in the time series domain is that these types of metrics are matters of preference and it is not reasonable to settle on one universally.

Also, there is an inherent artifact of multiobjectivity in recommender systems. At a high level, the recommender system in its perfunctory increases sales; and so it may, but this is not the whole story. Further, the company must consider whether the system must strive for diversity in the recommendations, accuracy relative to the user, or coverage of merchandise. That is to say, any one system can strive to introduce users to strategic choices of new items, or to be completely true to the customers desires, or to offer as many items in the store as possible across all customers. Such a choice is difficult to make and often requires pilot studies (Zhou et al. 2010; Ge et al. 2010).

2.3 Dimensionality Reduction

Among the most notable methods used to reduce the size of recommendation databases is singular value decomposition. SVD is a matrix factoring technique where large matrices are reduced to a more manageable size; and is often used in conjunction with collaborative filtering to great effect (Sarwar et al. 2000b). The decomposition can reduce the total amount of information, though the overall reduction is content specific.
For our purposes, SVD generally enables the compact storage of large databases of historical recommendations, and allows for the extraction of said information by applying local transforms. The research interest herein is to quicken the algorithm which generates the recommendation itself, which makes dimensionality reduction a good candidate for companionship in attacking the two separate but intrinsically related problems.

Singular value decomposition is not without its alternatives. Probabilistic latent semantic analysis (pLSA) for instance, is a statistical model often used in conjunction with SVD (Hofmann 1999) to compute associations between pairs of documents [or often in our case: reviews]. pLSA introduces a conditional class variable which allows for analysis of the conditional co-occurrence of words in a set of documents; then maximum likelihood probabilities are obtained by the use of an optimization routine [most often Expectation Maximization]. There are a number of refinements of pLSA which allow for analysis of different types of response variables. PLSA in its vanilla form is used for term/document matrices, whereas there have been developments which can forego the correlation measurements used in most memory-based systems and operate on numerical ranking data (Hofmann 2003). It is worth noting here that the Gaussian assumption of the underlying probability densities is something that feels good, but may not be true. Since we are estimating underlying probabilities in a system that is very difficult to test, a practitioner would likely have a difficult time accepting these assumptions. The original method though is an excellent accompaniment to content-based recommendation, in that it may allow for the intuition or automation of finding keywords which describe an item or features which are unique.
Latent Dirichlet Allocation (LDA) is yet another metric by which to mine optimal descriptive terms [the number of which is a parameter] from a corpus of documents, using conditional probability. The conditional probability [which assumed dirichlet priors] analyzes terms in a document and terms given a topic in a document (Krestel et al. 2009). LDA has several shortfalls which make it problematic for recommendation system usage. Firstly, the parameter estimation requires specification of a number of outputs; which implies that the method itself requires either supervision or faith. Further, the method depends upon a sampling routine [often Gibbs sampling (Hoffman et al. 2010; Porteous et al. 2008; Griffiths 2002; Wang et al. 2009)] until the parameters converge. In this way, the runtime cannot necessarily be estimated \textit{apriori}, nor the likelihood of parameter convergence guaranteed. Lastly, the method operates with a corpus of fixed size, which implies that the method would have to be run each time a new review occurs or at fixed time intervals, which brings to question long term stability in the outputs.

Often, automated systems represent user behavior as binary classes, which are amenable to many techniques [e.g. SVM] finding relationships between users and topics/items. The concept of user ontology is much more nuanced. Ontologies provide axiomatic and entity rich descriptions of users so that more complex analyses of the underlying systems is possible. Ontological user profiling is performed in many ways, from relevance feedback [user ratings of samples] to user supplied profiles and automatic text mining; though cold start is a glaring issue (Middleton et al. 2004). That said, it seems probable that clever binary representation may be able to incorporate many ontological features. In our application, ontological profiling may play an important part in deducing what initial filters to apply to users given a wealth of initial information. Ontological
user profiling is an ever-present phenomenon on the web today, though its representation may not be obvious. The ubiquitous option to “like” the majority of content on the Web allows for posterior analysis and production of user ontologies (Sieg et al. 2007).

There are several approaches which are difficult to immediately classify, but make strides towards solving some inherent issues in the recommender paradigm. Time aware (Baltrunas & Amatriain 2009; Campos et al. 2014) recommender systems (Tso-Sutter et al. 2008; Zhang et al. 2011) strive to reduce the threat of transient user interests. These methods generally contain temporal weighting/distributional schemes which allow for a rating half-life if you will. Since superimposing a sell-by date on user interest is a risky venture; there may be greater utility in properly tagging things like gifts or hobby related items and treating them appropriately. Tag aware recommender systems strive to bridge the area of research between attributes and the relatively new notion of tags. The details generally are nuances of text processing, but nevertheless can prove useful if tags or keywords are easily mined.

Context aware recommendation (Adomavicius & Tuzhilin 2011) strives to combat temporal and irrelevant information by relating recommendations in a multidimensional fashion. What is meant by this is that instead of conditioning ratings on users and items, ratings are conditioned on users, items, and context. Defining the relevant context is a difficult task, and the cold start phenomenon is more damaging, in that reliable subsets of context must be sufficiently large to gain inference. Nevertheless, with an adequate dataset, ontology, or subset routine, it seems probable that the introduction of context could be beneficial in terms of computation time and accuracy. Iterative Refinement (IR) is a theoretical construct which allows for the calculation of a rater’s reliability
by observing that which he/she rates. IR to date does not appear to be utilized in practice (Laureti et al. 2006).

The Slope One Smoothing/Algorithms are techniques by which ratings for a given item are adjusted by the pairwise differences in average ratings between similar items (Lemire & Maclachlan 2005). The method is useful in conjunction with other collaborative filtering techniques, though alone, appears naive. Yet another recommendation scheme is that of Trustwalker, where a rating is recommended by randomly walking a trust network and selecting similar items by more trusted users [i.e. those earlier in the walk]. Trustwalker (Jamali & Ester 2009) has several attractive features, such as confidence in the estimates, but correspondingly requires a trust network *apriori* as well as preexisting ratings. As a result, methods such as these are interesting to consider as a refinement, but not as a strategy.

No discussion on recommender systems would be complete without the discussion of hybrid systems. In hybrid systems, collaborative filtering methodologies are combined for benefits in computation or accuracy. An example of this would be Amazon.com, which uses content driven filtering [i.e. item department] which is determined predominantly by the features of each item, and within each department, historical data [i.e. memory based filtering] is utilized to guide the recommendation process. The advantage of this is clear, first: subsetting the users allows for a decrease in the storage and computational requirements of local searches, and also allows the sandboxing of possible temporal interests. Many of the algorithms in practice and research are complementary entities. There are a number of combinations in the literature such as memory and model-based
(Pennock et al. 2000) the aforementioned trust and item based [e.g. Trustwalker], content and memory (Balabanović & Shoham 1997; Burke 2000) and a host of others. The message here is that refining the user or item description is a job of the recommender system, and the aim is to further refine it always; as a result: often recommender systems will be meshed in order to retain complementary features if the combined errata are not prohibitive. Finally, we get to the method of this dissertation; which is the Top-N Recommendation algorithm. The Homo-Habilis of the Top-N algorithm appears in (Shardanand & Maes 1995; Guttman et al. 1998) as Ringo: recommending artists and songs to users. Each user supplies a rating profile, which is used to group the users into subsets. Afterwards, the most bought items by most similar users in the subset are recommended [assuming the recommendee has not bought the recommended-in-question.]

A testament to hybrid modeling methods, are the results of the Netflix Challenge, wherein the Grand Prize winner was awarded to a submission which utilized no fewer than eight hundred models (Feuerverger et al. 2012). The contest consisted of a training set of nearly 500,000 users and the corresponding ratings on 18,000 movies. The contest criteria was root mean squared error, and the holdout set was maintained by the corporate sponsor. Many algorithms were combined in the contest using Alternative Least Squares, Restricted Boltzmann Machines (Salakhutdinov et al. 2007), and K-Nearest Neighbor approaches (Bell et al. 2007). ANOVA, Empirical Bayes, (Lim & Teh 2007), and SVD techniques (Kurucz et al. 2007) also received much utility in the Netflix Challenge submissions.
The lessons from the Netflix challenge and its relationship to Collaborative Filtering are many, but chiefly we can acknowledge that in the accuracy constrained environment, no single model variant was sufficient to describe a complex user-item relationship. Rather, individual models and their theoretic variants keyed in on very important but different aspects of the user-item space, and the desired accuracy was only obtainable through a blend of such models to sufficiently describe the user space. Since, much theoretic and practical work has emerged from the challenge, it is informative in many ways that the literature prior did not frequently discuss. Namely, model parameterization, blending, and over-fitting. Arguably, this is due to the applied nature of the challenge.

Today, two types of Top-N subsets exist: user-based as in Ringo and model-based methods (Sarwar et al. 2001). Model based methods as we discussed in the general case above tend to be faster, but require assumptions, data to build, and due to those assumptions tend to be less accurate on a large scale. User-based methods [or item-based since it is simply an inversion of the user-item matrix] tend to be flexible and free of assumptions. Unfortunately though, the computational requirements become prohibitive when the system becomes very large. There have been nonparametric attempts at tackling this problem, such as tighter clustering of the users and better heuristics for nearest neighbor (Ungar & Foster 1998a,b) but these methods are difficult to tune and the recommendations suffer. There have also been a number of model-based approaches designed to tackle the computational and retrieval (Huang et al. 2004) complexity; for instance neural networks (Christakou et al. 2007; Billsus & Pazzani 1998) and Boltzmann machines (Salakhutdinov et al. 2007) have seen much action in the trenches for producing very good recommendations with minimal training data. The drawback though is that
neural networks are difficult to understand and untangle issues once trained; and the addition of new features is almost impossible without re-training the network/machine. These drawbacks make completely unsupervised application in large applications ill-advised. Supervised use in large applications would require many experts on hand. Also, association rules are often used for recommendation generation (Mobasher et al. 2000; Demiriz 2004; Lin et al. 2002) though these also suffer from the computational complexity of generating association rules [which can be as bad as $2^n$ in complexity (Zhang & Zhang 2002)]. The majority of the computational issue in standard Top-N is in calculating the similarity distance matrix (Deshpande & Karypis 2004) (generally using either the aforementioned cosine, correlation or a straightforward conditional probability) which in a user-based Top-N system is $O(users^2 \times items)$ since we have to perform $users(users - 1)$ comparisons, each of $items$ length. The focus of this dissertation at a very high level is merely a tune-able method by which to speed up the calculation of this Dotproduct (or its equivalent) common to other similarity metrics. In a complementary fashion then, we can apply the clustering techniques apriori and cluster-computing techniques to further reduce the computational effort, and hopefully the latency of calculation. Ultimately resulting in a system amenable for real-time use. Table 2.1 demonstrates some of the methods, the authors, a brief summary, and criticisms encountered in the literature review.
### Table 2.1: A Literary Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Authors</th>
<th>Summary</th>
<th>Criticisms</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Memory/ Information Based</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cosine</td>
<td>Sarwar et al. 2001; Jiang &amp; Wang 2010; Cantador et al. 2010; Linden et al. 2003</td>
<td>Computes the similarity as an angle between two vectors</td>
<td>Computationally expensive, a competitor.</td>
</tr>
<tr>
<td>Pearson Correlation</td>
<td>Pearson, 1880</td>
<td>Computes similarity as the correlation between two vectors</td>
<td>Computationally expensive, linear.</td>
</tr>
<tr>
<td>Adjusted Cosine</td>
<td>Sarwar et al. 2001</td>
<td>Scales by user in addition to computing Cosine</td>
<td>More computationally expensive, not always applicable.</td>
</tr>
<tr>
<td>Conditional Probability</td>
<td></td>
<td>Conditional probability one item is purchased given another</td>
<td>Expensive to compute</td>
</tr>
<tr>
<td>Default Voting</td>
<td>Breese et al. 1998</td>
<td>Imputes ratings based on user behavior</td>
<td>Only applicable to very specific circumstances</td>
</tr>
<tr>
<td>Inverse User Frequency</td>
<td>Breese et al. 1998</td>
<td>Attempts to account for universal like or dislike of items</td>
<td>Very expensive to compute, requires many ratings</td>
</tr>
<tr>
<td><strong>Model Based</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bayesian Probability</td>
<td>Schafer et al. 2007</td>
<td>Conditions on hidden class to obtain ratings</td>
<td>Probabilistic, but has good algorithms to refine estimates</td>
</tr>
<tr>
<td>Graph and Cluster Methods</td>
<td>Barabasi &amp; Albert, 1999; Zhou et al. 2009; Adamic &amp; Adar 2003; Leicht et al. 2006; Ravanau et al. 2002; Sorensen 1948; Jacard 1901; Salton &amp; McGill 1983</td>
<td>Uses a graphical model apriori to compute similarity measures, of which there are many.</td>
<td>Generally very computationally expensive, no generally accepted best.</td>
</tr>
<tr>
<td><strong>Content/ Feature Based</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tags</td>
<td>Song et al. 2011</td>
<td>Treating keywords as features</td>
<td>Cold start less an issue, requires more user surveying apriori, not relevant to our work</td>
</tr>
<tr>
<td>Relevant Information/ Rocchio’s Algorithm</td>
<td>Rocchio 1971; Joachims 1996</td>
<td>Feature selection given a ‘bag of words’</td>
<td>Not applicable for us, computationally expensive</td>
</tr>
<tr>
<td><strong>Reducing Dimensionality / Text Analysis</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Singular Value Decomposition</td>
<td>Sarwar et al. 2000</td>
<td>Compresses matrices so that columns or rows can be transformed out.</td>
<td>Good candidate for companionship, loses information in compression, does not attempt to hasten the speed of the calculations per se’</td>
</tr>
<tr>
<td>Probabilistic Latent Semantic Analysis</td>
<td>Hofmann 1999</td>
<td>Computes associations between pairs of documents</td>
<td>Not relevant for us, as it works with documents, not binary data.</td>
</tr>
<tr>
<td>Latent Dirichlet Allocation</td>
<td>Krestel et al. 2009</td>
<td>Analyses terms in a document and terms given a topic in a document, assuming Dirichlet priors</td>
<td>Not relevant due to document formats, convergence of Gibbs sampling routine not guaranteed; also, the assumption of parametric priors is unsanitary.</td>
</tr>
<tr>
<td><strong>Difficult to Classify</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ontological Profiling</td>
<td>Middleton et al. 2004</td>
<td>Attempts to characterize items or users with an ontology</td>
<td>Difficult to do without supervision</td>
</tr>
<tr>
<td>Time-Aware Recommendation</td>
<td>Baltrunas &amp; Amatriain 2009; Campos et al. 2014</td>
<td>Strives to reduce the effect of transient user interest</td>
<td>Not very mature as a research area, difficult to do without supervision</td>
</tr>
<tr>
<td>Context Aware Recommendation</td>
<td>Adomavicius &amp; Tuzhilin 2011</td>
<td>Relates recommendations multidimensionally based on context</td>
<td>Requires some semblance of item or customer context, difficult to obtain without supervision.</td>
</tr>
<tr>
<td>Slope-One Smoothing</td>
<td>Lenire &amp; Machlachlan 2005</td>
<td>Adjust by pairwise differences in average ratings between similar items</td>
<td>Assumes that ratings are available, not relevant for our application.</td>
</tr>
</tbody>
</table>
Chapter 3

Why Spectral Analysis?

3.1 History of Spectral Analysis

Spectral Analysis has a rich history, and is a technique generally used in signal processing which garnered attention in the time series field as early as 1968, (Jenkins & Watts 1968; Priestley 1981) and has been employed to gain inference on subjects ranging from astronomical artifacts (Scargle 1982) to the the less deterministic economic data (which is encouraging for our purposes) (Granger et al. 1964). It is safe to say that spectral analysis has a long and useful history in the decomposition of time series. A look into the literature reveals: spectral analysis is used most often for seismic phenomena (Castagna et al. 2003; Nazarian et al. 1983; Mahin & Bertero 1981) , medical phenomena (Nikias 1993; Cunningham & Jones 1993; Akselrod et al. 1985), (specifically EEG readings/heart behavior (Kamath & Fallen 1992; Saul et al. 1988; Sato et al. 1995; Yien et al. 1997)), economic phenomena (Granger & Morgenstern 1963; Granger 1966), and species composition (Henttonen et al. 1985; Ford & Renshaw 1984; Adam et al. 2000), (electrical signal analysis is not included because the related literature is very broad). However, we can say that spectral analysis has yet to be applied in the inventory domain explicitly. More generally, spectral analysis has yet to be applied to demand series specifically. Perhaps this is because signal analysis had fallen out of fashion, as the last relic of economic spectral analysis was conducted in the 60's (Granger 1966).
One of the more attractive features about spectral analysis when encountering a large mass of data is the computational benefits which it provides. First, we can use a computationally efficient algorithm in the estimation of the spectra (Welch 1967; Bergland 1969). Further, the use of Monte Carlo estimation (or even bootstrap) for the spectrum confidence interval is charted territory (Benignus 1969). Additionally, in the estimation of the periodogram, there are many robust methods which are developed to work around various data features. For instance, if there is missing data or the time series is unevenly spaced, the Lomb-Scargle Periodogram can estimate the frequencies (Schulz & Mudelsee 2002; Bretthorst 2001). In contrast, if the data is insufficient to estimate the sample spectra, it is possible to use a window function to make the estimator more consistent, a very common example of this is the Tukey (or Tukey-Hanning or Blackman-Tukey) Window (Tukey 1967). We should note that we can also calculate confidence Intervals for the spectrum. Though it should be noted that the distributional properties of the sample spectra are very dependent on the type of window which is used to smooth the spectrum.

For our purposes, the field of spectral analysis is primarily concerned with detecting hidden periodicities in observed data. Essentially, we parameterize a data series by decomposing it into individual frequencies of the form:

\[ R \cos (2\pi ft + \phi) = A \cos (2\pi ft) + B \sin (2\pi ft) \tag{3.1} \]

where \( R \) is the amplitude(\( \sqrt{A^2 + B^2} \)), \( f \) is the frequency, and \( \phi \) is the phase.
This implies that:
\[ A = R\cos(\theta); \ B = -R\sin(\theta) \]  
(3.2)

which additionally implies then that we can estimate A and B via least squares.

First, we can express any observable time series as an arbitrary number of cosine curves with arbitrary number of amplitudes and frequencies. We can express this as (Chatfield 2013):

\[ Y_t = A_0 + \sum_{i=1}^{m} [A_i\cos(2\pi f_i t) + B_i\sin(2\pi f_i t)] \]  
(3.3)

where \( A_0 \) is the coefficient of the cosine term at the zeroth frequency.

Since there is no error term, we can express any observed series explicitly as a linear combination of an arbitrarily large number of cosine curves [Wold’s Theorem]. When estimating these frequencies at the Fourier Frequencies, we know the sine and cosine predictor variables are orthogonal and the least squares estimates become (assuming the sample size is odd for convenience (Cryer & Chan 2008)):

\[ \hat{A}_i = \frac{2}{n} \sum_{t=1}^{n} Y_t \cos\left(\frac{2\pi f_i t}{n}\right); \hat{B}_i = \frac{2}{n} \sum_{t=1}^{n} Y_t \sin\left(\frac{2\pi f_i t}{n}\right) \]  
(3.5)

Suppose that a series of m ratings is observed; by examining the relative strength of the sine-cosine pairs at the fourier frequencies, one could detect which items are moving together by determining that n of the m items have a relatively strong spectral signature at any given frequency. One may want to estimate this process for various
reasons, first, representing the data this way can allow us to determine frequent itemsets simultaneously, thus liberating us from the combinatorial complexity of the frequent itemset problem and allowing the use of Fast Fourier Transforms instead. Second, the information derived from this observation is not only limited to the detection of items being bought together frequently, but also when they are sold together frequently by providing a temporal quality in the results.

The following methods provide quick and storage efficient estimates for detecting jointly occurring and unobserved periodicities (which is a proxy for similarity). The proof of concept seeks to empirically assess the value of the estimation procedure when we can explicitly express a demand series in terms of a sine-cosine pair. If this is true, we can expect the theory will carry us for the remainder of the climb (by virtue of eqn 3.3). That is to say that, if we can detect unknown and subtle synthetic periodicities by generating synthetic data consistently, we can assume that the method will work for any combination of sine-cosine frequencies, and that is to say: any observed series (once detrended) because we know that we can express any series explicitly as a linear combination of an arbitrarily large number of cosine curves without the introduction of an error term. Whether the Spectral approximations are advantageous to replace current similarity metrics in practice is the subject of this thesis and will be verified through experimentation and case studies.

3.2 Nonparametric Method

Estimation of the spectrum by the smoothed periodogram method includes the following steps:
1. Subtract mean and detrend time series [unnecessary in our case]

2. Compute discrete Fourier transform (DFT)

3. Compute the periodogram as described in eqn. 3.4 and 3.5

4. Smooth the periodogram to get the estimated spectrum using a Danielli filter.

Given series \( i \) and series \( j \) of length \( n \) the similarity will be the closeness of the cumulative sum of the spectral densities \( \beta \) and \( \gamma \) for each series with the same smoothing coefficient \( m \):

\[
Sim(i, j) = \left| \sum_{k=1}^{n} \gamma_k - \sum_{k=1}^{n} \beta_k \right| = \sum_{k=1}^{n} \left| \gamma_k - \beta_k \right| \quad (3.6)
\]

This method most certainly will be faster than the parametric method, in that we do not have to solve for an optimal AR model and further compute the spectra in addition.

### 3.3 Parametric Method

It is known that the sample spectra can be approximated by the approximation of an autoregressive model [Wold Theorem] (Box et al. 2013), so we could find the optimal AR coefficients by three methods:

1. The Yule-Walker equations

2. Levinson-Durbin Recursive Algorithm

3. Maximum Entropy Estimation
After acquiring the most appropriate autoregressive model, we can estimate the spectra from the model itself. Since this method requires several steps and will be slower, we will use nonparametric estimation. Also, since we focus on binary data for the majority of the thesis, we can say that an autoregressive model is likely not a very good fit. However, can be employed when the data we are analyzing is continuous in nature. The nonparametric method should provide a good launching point by which to compare binary series found in user-item matrices quickly. Computational experiments are necessary to assess validity as compared to the original method. Also, the method should be used in limiting cases to test size limitations.

Fig. 3.1: Example Series Generated on Left (Binary Signal) and the resulting Spectral Decomposition on the right using FFT.
A proof of concept is straightforward, we would hope that if two binary series of large number were identical, so too would be the FFT; we know this to be true. Second, we would hope that as the series begin to differ, observable differences in the FFT begin to be detected [the magnitude of which corresponding to the difference.] In Figure 3.2, we perform the experiment as proposed in Algorithm 2; wherein we permute a series in order to assess whether or not the absolute value of the difference in power densities can describe dissimilarity.

\textbf{Algorithm 2} Spectral Permutations

\begin{algorithm}
\begin{algorithmic}
\STATE Generate Binomial Series of Length 1000, with probability .4 and size 1
\STATE Generate Vector of Probabilities from .01 to .5 in increments of .01
\FOR {Prob in Probability Value}
\FOR {Observation in Binomial Series}
\STATE Generate a random uniform number [0,1]
\IF {random uniform number \neq current Prob}
\STATE Change state of x at current observation
\ENDIF
\ENDFOR
\STATE Store Absolute Value of Mod Spectra-Original Spectra(window=2 observations)
\ENDFOR
\end{algorithmic}
\end{algorithm}
We notice in Figure 3.2 that the spectra do diverge as incremental changes are made to binary series, which is encouraging for the measure. Also, on a Python 2.7 installation given a 1.5Ghz processor and 2GB of RAM, the calculation of a dot product for a Bernoulli series of length 1000000 averaged .3615438 seconds in 100 trials, whereas the calculation of the DFT averaged .1756310 seconds in 100 trials. It is important to note that the computational gains will be a factor greater than two in the multidimensional
case, since efficient algorithms exist for the DFT on matrices. (Press et al. 2002; Chu 1989)

We choose spectral analysis to derive the similarity measure for the following reasons.

1. The Discrete Fourier Transform is well-studied and is available in almost every programming language. In the rare cases in which it isn’t, trigonometric identities make the code straightforward.

2. The DFT allows for the compression of an observed data series with moving average, periodic, or autoregressive components very succinctly.

3. The DFT is estimable by FFT algorithms with $n \log(n)$ runtime. This is an attractive feature for compression apriori.

4. Since the Spectral Density is strictly positive and unique for the series analyzed, we can derive a similarity measure borrowing intuition from Hamming. We do this in Chapter four.

5. It was demonstrated in the literature review of the Netflix Challenge that multi-model paradigms fare well in recommender systems because each model detects distinct features in the historical recommendations. We posit then that in addition to the increase, we will be able to better characterize periodic, autoregressive, and moving average components.
Chapter 4

The Hutchison-Kumara Measure and its Ecology

4.1 Introduction

In this chapter, we derive our similarity measure (the Hutchison-Kumara Measure) and seek to make formal some of its properties as a similarity measure. Furthermore, we derive the computational complexity of some its competitors in the Top-N algorithm. We conclude with a discussion of the storage requirements and the practical implications the use of such a metric may engender. We note that the metric in this dissertation is primarily concerned with binary matrices since that is the general form of the user-item matrices found in Top-N problems. So, our derivations focus on the binary case.

4.2 The Hutchison-Kumara Measure of Similarity

H-K Measure of similarity:

\[
\frac{\sum_{i=1}^{n} |\delta_i - \beta_i|}{\max(\sum_{i=1}^{n} \delta_i, \sum_{i=1}^{n} \beta_i) - \min(\sum_{i=1}^{n} \delta_i, \sum_{i=1}^{n} \beta_i)}
\]

(4.1)
Where \( \delta \) is the Spectral Density of Series \( i \) and \( \beta \) is the Spectral Density of Series \( j \).

if \( n \) is a constant, and the data is binary:

\[
\max\left(\sum_{i=1}^{n} \delta_i, \sum_{i=1}^{n} \beta_i\right) = K, \quad (4.2)
\]

\( K \) here is a constant based on any pair of series analyzed, but in general it follows the strictly positive value of a Laplace distribution.

If we assume that \( \delta, \beta \) asymptotically follow a \( \chi^2(2) \) distribution:

We know that the \( \chi^2 \) distribution is a special case of the \( \gamma \) distribution:

\[
\chi^2(v) = \gamma\left(\frac{v}{2}, \beta = 2\right)
\]

So that:

\[
\chi^2(2) = \gamma\left(\frac{2}{2}, \beta = 2\right) = \gamma(1, 2)
\]

We also know that the exponential distribution is the a special case of the Gamma distribution when \( \alpha \) is 1.

\[
\chi^2(2) = \gamma\left(\frac{2}{2}, \beta = 2\right) = \gamma(1, 2) = \text{exponential}(2)
\]

And, we know that the difference of two identically distribution exponential distributions is Laplace.

\[
\chi^2(2) - \chi^2(2) = \gamma(1, 2) - \gamma(1, 2) = \text{exponential}(2) - \text{exponential}(2) = \text{Laplace}(0, \frac{1}{\lambda}) = \text{Laplace}(0, \frac{1}{2})
\]
The minimum possible value of the sum of the spectral density of a binary series is zero:

\[
\min\left(\sum_{i=1}^{n} \delta_i, \sum_{i=1}^{n} \beta_i \right) = \sum_{i=1}^{n} (0 + 0i) = 0
\] (4.3)

This leads to:

\[
\frac{\sum_{i=1}^{n} |\delta_i - \beta_i|}{\max(\sum_{i=1}^{n} \delta_i, \sum_{i=1}^{n} \beta_i) - \min(\sum_{i=1}^{n} \delta_i, \sum_{i=1}^{n} \beta_i)} = \frac{\sum_{i=1}^{n} |\delta_i - \beta_i|}{K - 0}
\] (4.4)

We retain the denominator, because it is desirable [and in fact a requirement] for a measure to scale between \((0,1)\).

\[
\frac{\sum_{i=1}^{n} |\delta_i - \beta_i|}{K} \propto \sum_{i=1}^{n} |\delta_i - \beta_i|
\] (4.5)

Where \(\delta\) is the Spectral Density of Series \(a\) and \(\beta\) is the Spectral Density of Series \(b\)

### 4.3 Properties of the H-K Similarity Measure

Essentially, there are two properties of similarity measures which should be valid (Egghe 2010).

1. The Principal of Nominal Increase

   Let \(X = (x_1, ..., x_n)\) and \(Y = (y_1, ..., y_n)\) s.t. \(x_i, y_i \geq 0 \ \forall \ i \ \text{s.t.} \ X \neq Y\).

   Let \(A\) be any constant vector s.t. \(a > 0\)

   Then our similarity measure satisfies the principal of nominal increase if:
$similarity(\vec{X} + \vec{A}, \vec{Y} + \vec{A}) > similarity(\vec{X}, \vec{Y})$ \hspace{1cm} (4.6)

2. “Another Good Property”

Let $\vec{X} = (x_1, ..., x_n)$ and $\vec{Y} = (y_1, ..., y_n)$ s.t. $x_i, y_i \geq 0 \forall i$ s.t. $X \neq Y$.

Then our similarity measure should satisfy:

$similarity(\vec{X} + \vec{Y}, 2\vec{Y}) > similarity(\vec{X}, \vec{Y})$ \hspace{1cm} (4.7)

4.3.1 Principal of Nominal Increase

To demonstrate that our measure satisfies the principal of nominal increase, we should constrain the problem to work within a binary paradigm. In other words, we can allow for the summation of constants only when the sum does not result in a vector which is not binary. That is to say, we can sum 1’s when we have 0. But, we should be more strict and allow for the summation of 0 when we have 1 [effectively allotting for a to be zero occasionally]. Actually, we propose two constants vectors greater than zero: $a$ and $b$, [as given in the definition for the Principal of Nominal Increase] and we evaluate the cases in each.

We begin with our similarity measure:

$$\frac{\sum_{i=1}^{n} |\delta_i - \beta_i|}{\max(\sum_{i=1}^{n} \delta_i, \sum_{i=1}^{n} \beta_i) - \min(\sum_{i=1}^{n} \delta_i, \sum_{i=1}^{n} \beta_i)}$$

Since the similarity is a unitary transform of the original values, we use the original values to assess cases [where $x, y$ are binary] for consistency with the definition of nominal
increase: $\bar{X} = x$ and $\bar{Y} = y$

$$\frac{\sum_{i=1}^{n} |x_i - y_i|}{\text{max}(x_i, y_i) - \text{min}(x_i, y_i)} = \frac{\sum_{i=1}^{n} |x_i - y_i|}{N - 0}$$

Because binary values are real:

$$= \frac{\sum_{i=1}^{n} \sqrt{(x_i - y_i)^2}}{N} = \frac{\sum \sqrt{x_i^2 - x_i y_i + y_i^2}}{N} = \frac{\sum \sqrt{x_i - x_i y_i + y_i}}{N}$$

Since the function derivative is equal to the sum of the individual derivatives, we work with the polynomial.

$$\frac{\sqrt{x_i - x_i y_i + y_i}}{N} \quad (4.11a)$$

We then allow for the addition of constants: $a, b$ in this case represent $\bar{A}$

$$\frac{\sqrt{x_i + a - (x_i + a)(y_i + b) + y_i + b}}{N}$$

Or the more simplified form:

$$\frac{\sqrt{a(-b - y + 1) + (1 - b)x + b + (1 - x)y}}{N}$$

Since the data are binary, and $a$ and $b$ depend on $x$ and $y$:

$$\frac{df}{da} = \frac{\delta f}{\delta a} \frac{da}{da} + \frac{\delta f}{\delta b} \frac{db}{da}$$
Multiplying through by $da$

$$df = \frac{\delta f}{\delta a} + \frac{\delta f}{\delta b} db$$

Taking partial derivatives with respect to $a$ and $b$:

$$= \frac{-b - y_i + 1}{2N \sqrt{-(a + x_i)(b + y_i) + a + b + x_i + y_i}} + \frac{-a - x_i + 1}{2N \sqrt{-(a + x_i)(b + y_i) + a + b + x_i + y_i}}$$

Simplifying

$$= \frac{-b - a - y_i - x_i + 2}{2N \sqrt{-(a + x_i)(b + y_i) + a + b + x_i + y_i}}$$

The possible cases are shown in Table 4.1.
Table 4.1: The Possible Cases for the Binary Variables and Constants

<table>
<thead>
<tr>
<th>Case</th>
<th>$x_i$</th>
<th>$y_i$</th>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1.2</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2.1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>2.2</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2.3</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>3.1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3.2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>4.1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

### 4.3.1.1 Evaluating the Cases

These cases refer to those which are shown in Table 4.1.

1.1

\[
\frac{-b - a - y_i - x_i + 2}{2N \sqrt{-(a + x_i)(b + y_i) + a + b + x_i + y_i}} = \frac{-0 - 0 - 1 - 0 + 2}{2N \sqrt{-(0 + 0)(0 + 1) + 0 + 0 + 0 + 1}} = \frac{1}{2N}
\]

1.2

\[
\frac{-b - a - y_i - x_i + 2}{2N \sqrt{-(a + x_i)(b + y_i) + a + b + x_i + y_i}} = \frac{-0 - 1 - 1 - 0 + 2}{2N \sqrt{-(0 + 0)(0 + 1) + 0 + 0 + 0 + 1}} = 0
\]
\[ \frac{-b - a - y_i - x_i + 2}{2N \sqrt{- (a + x_i)(b + y_i) + a + b + x_i + y_i}} = \frac{-1 + 0 + 0 + 0 + 2}{2N \sqrt{- (0 + 0)(1 + 0) + 0 + 1 + 0 + 0}} = \frac{1}{2N} \]

2.2

\[ \frac{-b - a - y_i - x_i + 2}{2N \sqrt{- (a + x_i)(b + y_i) + a + b + x_i + y_i}} = \frac{-1 - 1 - 0 - 0 + 2}{2N \sqrt{- (1 + 0)(1 + 0) + 1 + 1 + 0 + 0}} = 0 \]

2.3

\[ \frac{-b - a - y_i - x_i + 2}{2N \sqrt{- (a + x_i)(b + y_i) + a + b + x_i + y_i}} = \frac{-0 - 1 - 0 - 0 + 2}{2N \sqrt{- (1 + 0)(0 + 0) + 1 + 0 + 0 + 0}} = \frac{1}{2N} \]

3.1

\[ \frac{-b - a - y_i - x_i + 2}{2N \sqrt{- (a + x_i)(b + y_i) + a + b + x_i + y_i}} = \frac{-0 - 0 - 0 - 1 + 2}{2N \sqrt{- (0 + 1)(0 + 0) + 0 + 0 + 1 + 0}} = \frac{1}{2N} \]

3.2

\[ \frac{-b - a - y_i - x_i + 2}{2N \sqrt{- (a + x_i)(b + y_i) + a + b + x_i + y_i}} = \frac{-1 - 0 - 0 - 1 + 2}{2N \sqrt{- (0 + 1)(1 + 0) + 0 + 1 + 1 + 0}} = 0 \]

4.1

\[ \frac{-b - a - y_i - x_i + 2}{2N \sqrt{- (a + x_i)(b + y_i) + a + b + x_i + y_i}} = \frac{-0 - 0 - 1 - 1 + 2}{2N \sqrt{- (0 + 1)(0 + 1) + 0 + 0 + 1 + 1}} = 0 \]
At each point in the series, we can see that our distance increases if the values are different, and it does not change if the values are the same. Thus, the sum will reflect this, and our unitary transformation to approximate the series (spectra are greater than zero) will as well. Thus, we have shown that the H-K measure satisfies the principal of nominal increase for binary data.

4.3.2 Another Good Property

Is satisfied because the possible ways to evaluate this is covered in the Principles of Nominal Increase section; further, since our data is binary, there are restrictions on the addition of $2Y$ such that $Y$ must be zero. As a result, we do not formally fit this definition and only have trivial cases.

4.4 Derivation of Computational Complexity

In order to assess the H-K measure against others, we must first derive their computational complexities, this can give us insight into the difficulty of the methods computationally, though the number of operations is also of interest.
4.4.1 Dotproduct Distance

We infer the Computational Complexity of Dotproduct in Top-N (n items, m customers):

To Compute the distance measure:

\[ \vec{x} \cdot \vec{y} = x_1 \cdot y_1 + x_2 \cdot y_2 + \ldots + x_n \cdot y_n \]

\[ O(n - 1)\text{[additions]} + O(n)\text{[multiplications]} = O(n) \]

To Compute the similarity matrix (taking advantage of symmetry):

\[ (n)(n - 1)(m) = O(n^2 m)/2 \]

To sort using binary search (which gives us the Top-N items):

\[ mO(\log(n)) \]

So then the complexity is:

\[ O(n) + O(n^2 m) + mO(\log(n)) = O(n^2 m) \]
4.4.2 Cosine Distance

We should note that this popular Cosine metric is in use at Amazon.com.

\[
\frac{\vec{x} \cdot \vec{y}}{\|\vec{x}\|_2 \cdot \|\vec{y}\|_2} = x_1 \cdot y_1 + x_2 \cdot y_2 + \ldots + x_n \cdot y_n +
\]
\[
(x_1 \cdot x_1 + x_2 \cdot x_2 + \ldots x_n \cdot x_n) + (y_1 \cdot y_1 + y_2 \cdot y_2 + \ldots y_n \cdot y_n)
\]

For the numerator:

\[O(n - 1)[\text{additions}] \text{and} O(n)[\text{multiplications}] \quad (4.31a)\]

For the denominator:

We have O(n) summations for each (because binary data squared is the original data) plus one square root operation each.

\[O(n - 1) + 2O(n) + O(n) = O(n - 1) + 3O(n) = O(n)\]

To Compute the similarity matrix:

\[(n)(n - 1)(m) = O(n^2 m)/2\]

To sort using binary search (which gives us the Top-N items):

\[mO(log(n))\]
So then the complexity is:

\[ O(n) + O(n^2 m) + mO(\log(n)) = O(n^2 m) \]

### 4.4.3 Hutchison-Kumara

Chiefly, depends on the FFT to compress the data apriori which is \( n \log(n) \) (Farhang-Boroujeny & Lim 1992) in complexity.

The prior transform cost is:

\[ mO(n \log(n)) \]

And the sum of the absolute values is \( O(n) \):

To Compute the similarity matrix:

\[ (n)(n-1)(m) \approx O(n^2 m)/4 \]

To sort using binary search (which gives us the Top-N items):

\[ mO(\log(n)) \]

So then the complexity is:

\[ mO(n \log(n)) + O(n) + O(n^2 m) + mO(\log(n)) = O(n^2 m) \]
4.4.4 Hamming Distance

We will represent the Hamming as the Euclidean-squared metric since our data are in the Real domain.

\[
|\vec{x} - \vec{y}| = \left( (x_1 - y_1)^2 \right)^\frac{1}{2} + \left( (x_2 - y_2)^2 \right)^\frac{1}{2} + \cdots + \left( (x_n - y_n)^2 \right)^\frac{1}{2}
\]

(4.40a)

\(O(n)\) substractions, \(O(n)\) square operations and \(O(n)\) square roots. The square and square root operations are slightly more expensive than simple arithmetic (depending on the number of decimal places desired, but it is linear in the number of digits using Newton’s method, so is at worst \(O(n(d))\) but we approximate this at \(O(n)\).

\(3(O(n)) = O(n)\)

To Compute the similarity matrix:

\((n)(n - 1)(m) = O(n^2m)/2\)

To sort using binary search (which gives us the Top-N items):

\(mO(\log(n))\)
So then the complexity is:

\[ O(n) + O(n^2m) + mO(\log(n)) = O(n^2m) \]

In conclusion, though we do not see a difference in the Big O notation, we still see a practical decrease in the number of operations required for the H-K measure. The H-K measure performs approximately half as many calculations and storage operations as the other measures.

4.5 Storage Requirements: Observations and Discussion

We should first determine how much RAM is consumed by a distance matrix and also upper bounds on the computation of each method.

4.5.1 Assumptions

1. The distance matrix is symmetrical

2. The layout of the matrix can be reduced to a vector

3. Each distance matrix entry can be adequately represented by a float

4. The RAM consumed by each calculation is discarded upon completion

4.5.2 Distance Matrix Analyses

If we assume that each entry can be expressed by a float, we assume then that each entry is 32 bits, or four bytes. Since the matrix is symmetrical:

\[ \text{Total Entries} = \frac{nm^2}{2} \]
So then, for fixed \( n \) and fixed RAM in kilobytes(\( \theta \)), \( m \) cannot exceed:

\[
\left(\frac{2 \ast \theta}{n/250}\right)^\frac{1}{2} = \sqrt{\frac{500 \ast \theta}{n}}
\]

For fixed \( m \) and fixed RAM in kilobytes(\( \theta \)), \( n \) cannot exceed:

\[
\frac{2 \theta}{(n/250)^2}
\]

For example, if we have 10 customers (\( n \))[which we should note is 40 bytes or 1/25 kb] and 8 Megabytes of RAM (8000 kilobytes), we know \( m \) cannot be any larger than

\[
\sqrt{(500 \ast 8000)/10} = 632.45 = 632 \text{ items}
\]

### 4.5.3 Absolute Upper Limits for Tenable Calculation

In terms of absolute upper limits, we need to be able to fit both vectors into RAM to compute the distance value between them. If we have 8000 Megabytes of memory and each entry is a float (or an int) [.000004 Megabytes] we can fit a total of 2,000,000,000 entries into memory, which means each series must be 1,000,000,000 entries in length for calculation to be tenable for the Hamming, Dotproduct or Cosine Measures. Our spectral method summarizes each series by a series of half the original length. So then, we can use it in the event that each series is as great as 2,000,000,000 entries in length (in the case of 8 GB of memory.) Assuming that we read in each series individually and perform the FFT first.

For the spectral method, we note that for the distance matrix:

\[
\text{Total Entries} = \frac{nm^2}{4}
\]
4.5.4 H-K Measure: Row and Column Permutation

Before vetting the method against others, we will first ensure that the measure itself is consistent. That is to say, we will run an experiment wherein we a large binary matrix and permute the rows to determine whether or not the Top-N items change. In addition to the subroutine demonstrated in Algorithm 3 we will also make note of how often the indices change [if they do.]

<table>
<thead>
<tr>
<th>Algorithm 3 Consistency Check</th>
</tr>
</thead>
<tbody>
<tr>
<td>Generate Bernoulli Matrix(1000,10000) with P=.5 (M)</td>
</tr>
<tr>
<td>for Replication in 1...10000 do</td>
</tr>
<tr>
<td>Permute Rows of Matrix Set k=ncol(M)</td>
</tr>
<tr>
<td>for j=1...m do</td>
</tr>
<tr>
<td>for i=1...m do</td>
</tr>
<tr>
<td>if i ≠ j then ρ_{i,j}=\sum</td>
</tr>
<tr>
<td>else Let ( ρ_{i,j} = 0 )</td>
</tr>
<tr>
<td>end if</td>
</tr>
<tr>
<td>end for</td>
</tr>
<tr>
<td>for i=1...m do</td>
</tr>
<tr>
<td>if ( ρ_{i,j} \neq ) the k smallest values in ( ρ_{*,j} ) then ( ρ_{i,j} = 0 )</td>
</tr>
<tr>
<td>end if</td>
</tr>
<tr>
<td>end for</td>
</tr>
<tr>
<td>Retain the row indices of ( ρ )</td>
</tr>
<tr>
<td>if The indices of ( ρ ) have changed then</td>
</tr>
<tr>
<td>Halt</td>
</tr>
<tr>
<td>end if</td>
</tr>
<tr>
<td>end for</td>
</tr>
</tbody>
</table>

In Algorithm 3 we should note that we do not compare only the Top-N items, but rather the entire sorted distance vectors. This is a much more general condition. In other words, we are checking to see whether or not the Top-N items is the same where N is the number of items-1. In 10,000 matrix permutations, we failed to see an instance
where the sorted Top-N items differed. The results are intuitive because permuting the columns is the same as rearranging the distance measures [because the spectral densities are unique], and as a result: the sort operation will give us the same information as the unpermuted data. Further, this would also hold true for the columns since that this is performing the same experiment to the transpose of the binary matrix.
Chapter 5

Analysis of the Hutchison-Kumara Similarity Measure

5.1 Synthetic Validation of the H-K Measure: Methodology

Our measure can be considered a similarity measure, which provides a tradeoff between speed/computational requirements and accuracy relative to other similarity measures. If the similarity measure is demonstrated to be successful on synthetic data, we will integrate it into algorithms which utilize other, more well-established similarity measures and investigate the advantages in speed and storage.

In order to validate the measure initially, we will have to compare binary strings in a Factorial Experiment. The competitors for the initial phase will be the Cosine, Dotproduct, and Hamming similarity measures, in that they are widely utilized and accepted in practice. The experiment will be a full factorial design [since we care for accuracy more than speed] and the factors which will be held at various levels are:

1. Length of Binary Strings
2. Probability of flipping a bit in each Binary String [i.e. how many bits we change in the string for each pass of the similarity metric]
3. Probability of observing a 1 in the binary string: Presuming the Binary string is generated from a Binomial distribution of probability factor of length factor and size one.
4. Top-N Number

The H-K Measure of similarity:

$$\frac{\sum_{i=1}^{n} |\delta_i - \beta_i|}{\max(\sum_{i=1}^{n} \delta_i, \sum_{i=1}^{n} \beta_i) - \min(\sum_{i=1}^{n} \delta_i, \sum_{i=1}^{n} \beta_i)}$$  \hspace{1cm} (5.1)$$

Where $\delta$ is the Spectral Density of Series $i$ and $\beta$ is the Spectral Density of Series $j$.

if $n$ is a constant, and the data is binary:

$$\max(\sum_{i=1}^{n} \delta_i, \sum_{i=1}^{n} \beta_i) = K$$  \hspace{1cm} (5.2)$$

$$\min(\sum_{i=1}^{n} \delta_i, \sum_{i=1}^{n} \beta_i) = \sum_{i=1}^{n} (0 + 0i) = 0$$  \hspace{1cm} (5.3)$$

This leads to:

$$\frac{\sum_{i=1}^{n} |\delta_i - \beta_i|}{\max(\sum_{i=1}^{n} \delta_i, \sum_{i=1}^{n} \beta_i) - \min(\sum_{i=1}^{n} \delta_i, \sum_{i=1}^{n} \beta_i)} = \frac{\sum_{i=1}^{n} |\delta_i - \beta_i|}{K - 0}$$  \hspace{1cm} (5.4)$$

$$\sum_{i=1}^{n} |\delta_i - \beta_i| \propto \sum_{i=1}^{n} |\delta_i - \beta_i|$$  \hspace{1cm} (5.5)$$

Where $\delta$ is the Spectral Density of Series $i$ and $\beta$ is the Spectral Density of Series $j$.

The Cosine Metric will be as described in eqn(2.1).

The Dotproduct Metric will simply be dotproduct of the two binary strings which are compared.

The Hamming distance will be

$$\sum_{i} D_i$$, where:
\[ D_i = 1 \text{ if } x_i \neq y_i \]
\[ D_i = 0 \text{ o.w.} \]

**Response**  In each pass of the algorithm [when each similarity metric’s function is called to operate on the same set of binary strings to detect similarity], the system time of each metric will be stored and the profiles will be analyzed against the factor levels to see if there are blind-spots for any of the metrics in terms of series length etc, and also to profile the speed and RAM consumption of our method against competitors in a static computing environment.

5.1.1 **Accuracy and Time Validation**

In order to review the accuracy and runtime characteristics of the H-K measure, we will use descriptive and nonparametric analyses to describe our performance against other measures. The differences will be recorded in two separate experimental runs. The intuition being that if the results are close, there likely exists a transform from our method to the more traditional; and the models will help us find it.

5.1.2 **Storage Validation**

In order to validate the storage requirements, we will simply record the differential in RAM requirements as each function is called respectively and analyze the results. There are theoretical underpinnings which can be called upon here as well to use as a sanity check.
5.2 Comparing Algorithms

To illustrate the method, we will insert it into an algorithm which classically depends on other similarity metrics and use it to compare execution speed and accuracy of the algorithm. Here we will use ’vanilla Top-N’ as described in Algorithm 1 with the similarity computation removed, and ours inserted in its place. The competitors will be the very same algorithm in both the dotproduct and cosine variants. It is important to keep in mind, that in the preliminary experimentation, the raw periodogram spectra will be used. Because spectra can be noisy (especially when applied to Bernoulli series) we can expect that the use of parametric estimation or the Danielli Filter will improve the estimates, this will be explored in later chapters.

5.2.1 Competing Algorithms

In order to compete with existing Top-N, we will stay as true to form as possible to the traditional algorithm, and make changes only to the similarity metrics which are internal to their structure. The structure of the algorithms which will be used in the new experiment are outlined below in Algorithms 4 through 7. In these algorithms, we simply employ the traditional Top-N vehicle, but change the measures of similarity implicitly to the alternatively accepted counterparts to the H-K measure [i.e. the Dotproduct, Hamming, and Cosine measures]. These algorithms are simply visual companion to assist in building the intuition among the various measures.
Algorithm Preliminaries:

\( \rho \) is the Nonparametric Spectral Density in Algorithm 4

**Algorithm 4** Spectral Top-N [Parametric and NonParametric]

\[
\begin{align*}
\text{for } j=1\ldots m & \text{ do} \\
& \text{ for } i=1\ldots m \text{ do} \\
& \quad \text{ if } i \neq j \text{ then} \\
& \quad \quad \rho_{i,j} = \sum [\text{Danielli Kernel}((R(*, j), R(*, i)), \beta)] \\
& \quad \text{ else } \text{ Let } \rho_{i,j} = 0 \text{ Let } \xi_{i,j} = 0 \\
& \quad \text{ end if} \\
& \text{ end for} \\
& \text{ for } i=1\ldots m \text{ do} \\
& \quad \text{ if } \rho_{i,j} \neq \text{ the k smallest values in } \rho^{*},j \text{ then } \rho_{i,j} = 0 \\
& \quad \text{ end if} \\
& \text{ if } \xi_{i,j} \neq \text{ the k smallest values in } \xi^{*},j \text{ then } \xi_{i,j} = 0 \\
& \text{ end if} \\
& \text{ end for} \\
& \text{ end for} \\
& \text{Return } \rho
\end{align*}
\]

**Algorithm 5** DotProduct Top-N

\[
\begin{align*}
\text{for } j=1\ldots m & \text{ do} \\
& \text{ for } i=1\ldots m \text{ do} \\
& \quad \text{ if } i \neq j \text{ then } \rho_{i,j} = (R(*, j) \cdot R(*, i)) \\
& \quad \text{ else } \text{ Let } \rho_{i,j} = 0 \\
& \quad \text{ end if} \\
& \text{ end for} \\
& \text{ for } i=1\ldots m \text{ do} \\
& \quad \text{ if } \rho_{i,j} \neq \text{ the k largest values in } \rho^{*},j \text{ then } \rho_{i,j} = 0 \\
& \quad \text{ end if} \\
& \text{ end for} \\
& \text{ end for} \\
& \text{Return } \rho
\end{align*}
\]
Algorithm 6 Cosine Top-N

\[
\text{for } j=1...m \text{ do} \\
\quad \text{for } i=1...m \text{ do} \\
\quad \quad \text{if } i \neq j \text{ then } \rho_{i,j} = \cos^{-1}\left( \frac{R(\ast, i) \cdot R(\ast, j)}{\|R(\ast, i)\|_2 \|R(\ast, j)\|_2} \right) \\
\quad \quad \text{else Let } \rho_{i,j} = 0 \\
\quad \text{end if} \\
\text{end for} \\
\text{for } i=1...m \text{ do} \\
\quad \text{if } \rho_{i,j} \neq \text{ the k smallest values in } \rho_{\ast,j} \text{ then } \rho_{i,j} = 0 \\
\text{end if} \\
\text{end for} \\
\text{end for} \\
\text{Return } \rho
\]

Algorithm 7 Hamming Top-N

\[
\text{for } j=1...m \text{ do} \\
\quad \text{for } i=1...m \text{ do} \\
\quad \quad \text{if } i \neq j \text{ then } \rho_{i,j} = |R(\ast, i) - R(\ast, j)| \\
\quad \quad \text{else Let } \rho_{i,j} = 0 \\
\quad \text{end if} \\
\text{end for} \\
\text{for } i=1...m \text{ do} \\
\quad \text{if } \rho_{i,j} \neq \text{ the k smallest values in } \rho_{\ast,j} \text{ then } \rho_{i,j} = \infty \\
\text{end if} \\
\text{end for} \\
\text{end for} \\
\text{Return } \rho
\]

In all of the above algorithms, we note that the only change within each is the method used to compute the distance.
5.2.2 Time vs. Storage

Throughout the execution of the experiment, we will record the system time of each algorithm in its execution [each algorithm is designed as a function call for benchmarking].

5.2.3 Experimental Design: Responses

Since we do not know and cannot find the true Top-N items in a given dataset, we will benchmark our method based on its ‘agreement’ with classically accepted metrics on synthetic datasets. So, to determine the closeness of the solutions between two competing algorithms, we use the intersection of the Top-N items. Given a matrix $m$ the closeness of the solutions of Algorithm $A$ and Algorithm $B$ is determined by:

$$closeness(A, B) = closeness(B, A) = \frac{TopN_A(m) \cap TopN_B(m)}{N}$$  \hspace{1cm} (5.6)

To compare the storage, if we let $f(t)$ represent a function of the total RAM consumed at a given time $t$.

$$Storage_A = f(t_{final}) - f(t_{initial})$$  \hspace{1cm} (5.7)

Whereas runtime is measured (as an average) over $k$ replications.

$$Runtime_A = \frac{\sum_{i=1}^{k} t_{final} - t_{initial}}{k}$$  \hspace{1cm} (5.8)
5.3 Bernoulli Experimentation

In this section, we will be conducting three experiments to analyze the effectiveness of the H-K measure against other well-established methods, in each we will measure the aforementioned responses respectively.

<table>
<thead>
<tr>
<th>Factor Levels for Experiments 1-3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Factor</td>
</tr>
<tr>
<td>Top-N number</td>
</tr>
<tr>
<td>No. of Items</td>
</tr>
<tr>
<td>Customers</td>
</tr>
<tr>
<td>P</td>
</tr>
</tbody>
</table>

The Factor Levels proposed for the Experiment with 1000 replications at each point for a total of $5^4(10)$ or 6,250 replications (10 replications at each point).

5.3.1 Accuracy Comparison

In order to explore the accuracy against other algorithms, we run the aforementioned experiment and regress the cosine/spectral agreement against the cosine/dotproduct agreement assuming that the cosine and dotproduct themselves demonstrate “closeness” as it is defined in eqn(5.6). We find that there is significant disagreement among the Cosine and Dot-Product Metrics as seen in Figure 5.1. Also, from Figure 5.1 we can note that our method seems to be much more consistent with the Cosine and Hamming
similarity measures. The algorithms themselves are not exactly equivalent to one another, but we acknowledge that the Cosine and Dotproduct measures are well accepted in the Top-N Algorithm itself for continuous data, and that the Hamming measure for Binary Data is a reasonable and well-tested choice to measure distance. With these things in mind, we define agreement equivalently to our “closeness” parameter. Since the parameter itself is a quantity of how many items are shared in the outputs of each algorithm.

![BoxPlots demonstrating the agreement by each method](image)

**Fig. 5.1:** BoxPlots demonstrating the agreement by each method

We analyze the competing similarity measures in *Figure 5.1*; the results are counterintuitive. On the left, we can see that the Dotproduct and the Cosine similarity measures agreed strongly and often, however we can easily construct pathological cases where the Dotproduct does not easily explain differences in binary data. For instance,
suppose we have the following three series observed:

\[
\begin{align*}
11101010010101001 & \\
1111111011101111 & \\
11101011010101101 & \\
\end{align*}
\]

\[
(11101010010101001) \cdot (1111111011101111) = 9
\]

\[
(11101010010101001) \cdot (11101011010101101) = 9
\]

We can clearly see that there is a difference in the latter two series. The Dot product is also the numerator in the Cosine Measure, and again, we can easily construct pathological cases where normalizing the length does not protect against bit-shift. However, this normalization brings Cosine slightly closer to our measure, though chiefly, we should benchmark ourselves based on the Hamming distance because it is a gold-standard for binary vector comparison. In the binary case, we can see that the Hamming distance is the same as the squared Euclidean measure, which is the same as the Manhattan, and City-Block similarity measures.

The measure (given \( \vec{x} \) and \( \vec{y} \)) is as follows:

\[
\sum_i D_i \quad \text{where:}
\]

\[
D_i = 1 \text{ if } x_i \neq y_i
\]
\[ D_i = 0 \text{ o.w.} \]

Which, we can easily see in the real space is the same as:

\[
\sum_i |x_i - y_i| = \sqrt{x_i^2 - y_i^2} = ||x - y||_1
\]

The conclusion from this is a long lauded one, which is that no two distance measures are created equal. Further, the selection of a measure is dependent on the application. There has been prior work demonstrating that the Dotproduct and Cosine often agree on binary matrices, though they wildly disagree from the Hamming Distance. (Choi et al. 2010) So then, we benchmark ourselves to the Hamming measure but we keep in mind the computational differences of the Dotproduct and Cosine measures. This is because when applied to continuous vectors, our computation time should not suffer much if at all, and our similarity measure should actually improve in the continuous case. Fitting spectra to binary series is a much more difficult process than a continuous series and due to its construction, will have a harder time summarizing the data. This is especially true if the continuous demand series observed contain an autoregressive component.
In Figure 5.2, we can see that as the number of customers increased, our accuracy increased slightly relative to the Hamming Distance. This is an artifact which is beneficial in view of a Big-Data paradigm. Further, as the matrix length changed, we did not vary much, and so we can conclude that our measure is consistent given the samples we generated. Since we are generating the distance matrix with respect to the items, this is likely an artifact of asymptotic properties when the series lengthen. Both that our spectral measure should perform better in summarizing longer series, and that statistically, all similarity measures should be more accurate when the comparisons are of larger length.
In Figure 5.3 we see a similar story in that the H-K accuracy did not respond reliably in the number of items. From this graph, it is difficult to say if the accuracy worsened on as a result of our method: because we view a similar decrease in accuracy relative to the cosine measure as well. Since we could easily take the transpose of the matrix (i.e. switch customers and items) this plot would suggest that even up to 500 customers our values were consistent, if slightly less in agreement than the dot-product and cosine measures. However this plot and the previous point to a sparsity ailment. Further, with a given Top-N items which our methods are set to agree upon, any small differences in the methods would become magnified as the number of series increased, and also we would be less likely to detect the same five in five-hundred versus 50.
Figure 5.4 suggests that when the number of Top-N items selected is small, we perform well, though the H-K measure’s accuracy drops off as the number is increased. Again, we see precision but a drop in accuracy as the matrix becomes larger. It is also important to note that our accuracy did not suffer as drastically with respect to the Hamming distance as it did the Cosine, this is encouraging since the Hamming distance is probably a better indicator of our performance on binary data.
In Figure 5.5 we see conclusive evidence that the noisy spectra are to blame for our drop in accuracy. With respect to the hamming distance, the Hutchison-Kumara measure performed equally well or better than with respect to Cosine throughout the range of probabilities. When the series were sparse or the probability of generating a 1 became very high, we performed very well with respect to the Hamming measure. In the sparse case, the Cosine did not fare well, and this is likely due to pathological cases. For our purposes, this is very encouraging in that the majority of the applied domain will be focused on very sparse matrices.

Fig. 5.5: Density of Agreement by Probability of Binomial of size one
5.3.2 Run-Time Comparison

![Runtime in Microseconds](image)

It is important to note at this juncture that the Hamming measure used was composed in Fortran, and the Spectral measure was composed in R. On a single core, the dot-product and Cosine were about five orders of magnitude greater in computation time, and the Hutchison-Kumara measure was slightly faster than the Hamming measure in the experiment conducted as described by Figure 5.6. This is very encouraging because in an extension to continuous series where our method should perform better we will see the same computational gains with these simple implementations, and it may be possible to treat continuous data as quickly as it is binary data with this method.
In Figure 5.7 we see that as the number of customers increased [i.e. the computational workload per unit distance calculation is increased] we begin to see the same summary wherein the Hamming and Spectral measures overtake the Cosine and Dot-product measures by a large margin. Also, we can see that in general, the Spectral measure was still faster to compute than the Hamming measure.
In Figure 5.8 we see that as the number of items increased (i.e., the computational workload per entire distance calculation is increased by a factor squared) we see that the Hamming and Spectral measures overtake the Cosine and Dotproduct measures by a large margin and the variance of the runtimes are not increased as dramatically. The same is true in that the spectral measure was faster to compute on average. The scales of the plots here tell a story, and it is a story which we investigate in greater detail later in Section 6.3. In general, as the number of items increases, the computational workload increases significantly, and it is this problem which we attempt to ameliorate.
Since computing the Top-N items basically is a sort-routine within the algorithm we expect not to see much change in Figure 5.9, and fortunately there is not much to observe. The computational time is roughly constant throughout the range of Top-N.
In Figure 5.10 we would expect to see that the Hamming computation would demonstrate some variance throughout the range, and contrary to common sense, this is difficult to observe from these plots.
5.3.3 RAM Consumption Comparison

![Box plots showing RAM Consumption in KB by Method](image)

Fig. 5.11: RAM Consumption in KB by Method

In Figure 5.11 we can see that the RAM consumption of the Hamming Distance in the worst case was approximately half of the worst case of the Cosine. In general, the Dotproduct and the Hamming Distance were about equal in terms of RAM consumption, with Hamming gaining a very slight edge. In observing the quantiles of the plots, the Spectral method outperformed the Hamming on average, on variance, and in the worst case. This is very encouraging, and in line with the theoretical predictions.
Fig. 5.12: RAM Consumption in KB by Number of Customers

In Figure 5.12 an interesting insight emerges about the generalization we made about the Hamming and Dotproduct measures. It would seem that the Dotproduct tends to be consistent in its RAM consumption increase across the number of customers, though the Hamming tends to steadily improve as the Customer number increases: making them appear roughly equal from an aggregate standpoint. Again, the Cosine performs worst and the Spectral performs best in terms of the RAM consumption. We see the axes change because in each distance calculation the series length to be compared lengthened, making it linearly more difficult. We should see a much more dramatic difference in view of the number of items.
In Figure 5.13, generalizations are not easily made about the RAM consumption because the distribution of RAM consumption appears similar across the items. The axes change very dramatically though, and here we note in particular the dramatic worsening of the distance matrix calculation.
Fig. 5.14: RAM Consumption in KB by Top-N

In *Figure* 5.14 we don’t expect to see change, and we do not see it. This would suggest that the experiments themselves are consistent.
Fig. 5.15: RAM Consumption in KB by Probability

In Figure 5.15 we observe that the probability also did not affect the RAM consumption very much.

5.3.4 Summary Tables (Means)

In this section, we conduct experiments on the single processor architecture and vary the number of customers and items, effectively summarizing the effect which each has on the runtime/memory consumption by method. In the following tables, the columns ($m$) are indicative of the number of customers in the trial and the rows ($n$) are the number of items in a given trial. The values in the table are given in either microseconds (runtime) or kilobytes (RAM consumption) and this distinction is given in the header of each table.
5.3.4.1 Runtime

Table 5.2: Average Cosine Runtime in Microseconds In Five Separate Trials

<table>
<thead>
<tr>
<th>m</th>
<th>100</th>
<th>250</th>
<th>500</th>
<th>750</th>
<th>1000</th>
</tr>
</thead>
<tbody>
<tr>
<td>50</td>
<td>35089.87</td>
<td>4861.24</td>
<td>233783.1</td>
<td>371671.6</td>
<td>235614.4</td>
</tr>
<tr>
<td>100</td>
<td>149692.70</td>
<td>208833.93</td>
<td>1310500.2</td>
<td>1406128.1</td>
<td>1160731.9</td>
</tr>
<tr>
<td>n</td>
<td>150</td>
<td>454530.29</td>
<td>3279647.8</td>
<td>3406141.9</td>
<td>2983207.4</td>
</tr>
<tr>
<td>200</td>
<td>613812.38</td>
<td>834253.42</td>
<td>6103574.9</td>
<td>6401152.9</td>
<td>5738191.3</td>
</tr>
<tr>
<td>500</td>
<td>3884971.30</td>
<td>5440976.13</td>
<td>37073026.9</td>
<td>3406141.9</td>
<td>37584010.0</td>
</tr>
</tbody>
</table>

Table 5.3: Dotproduct Runtime in Microseconds Across Five Separate Trials

<table>
<thead>
<tr>
<th>m</th>
<th>100</th>
<th>250</th>
<th>500</th>
<th>750</th>
<th>1000</th>
</tr>
</thead>
<tbody>
<tr>
<td>50</td>
<td>20174.17</td>
<td>29589.78</td>
<td>221402.6</td>
<td>161665.9</td>
<td>173612.4</td>
</tr>
<tr>
<td>100</td>
<td>84789.95</td>
<td>120791.91</td>
<td>1152070.6</td>
<td>1327173.5</td>
<td>1344679.7</td>
</tr>
<tr>
<td>n</td>
<td>150</td>
<td>275751.31</td>
<td>3181481.9</td>
<td>3192895.2</td>
<td>3004318.8</td>
</tr>
<tr>
<td>200</td>
<td>348662.25</td>
<td>488986.66</td>
<td>5247581.9</td>
<td>5955606.3</td>
<td>5230606.1</td>
</tr>
<tr>
<td>500</td>
<td>2181249.76</td>
<td>3279740.94</td>
<td>34792526.5</td>
<td>35457103.3</td>
<td>32996182.2</td>
</tr>
</tbody>
</table>
Table 5.4: Hamming Runtime in Microseconds Across Five Separate Trials

<table>
<thead>
<tr>
<th>m</th>
<th>100</th>
<th>250</th>
<th>500</th>
<th>750</th>
<th>1000</th>
</tr>
</thead>
<tbody>
<tr>
<td>50</td>
<td>27962.35</td>
<td>37447.76</td>
<td>47768.11</td>
<td>66012.04</td>
<td>73040.45</td>
</tr>
<tr>
<td>100</td>
<td>120605.69</td>
<td>153095.99</td>
<td>204867.86</td>
<td>242751.65</td>
<td>300124.53</td>
</tr>
<tr>
<td>150</td>
<td>276171.82</td>
<td>339529.20</td>
<td>455190.97</td>
<td>572532.50</td>
<td>707637.98</td>
</tr>
<tr>
<td>200</td>
<td>490105.50</td>
<td>606511.28</td>
<td>827692.72</td>
<td>1026592.50</td>
<td>1245167.37</td>
</tr>
<tr>
<td>500</td>
<td>3185242.50</td>
<td>4032359.30</td>
<td>5521703.25</td>
<td>6983981.44</td>
<td>8608572.63</td>
</tr>
</tbody>
</table>

Table 5.5: Spectral Runtime in Microseconds Across Five Separate Trials

<table>
<thead>
<tr>
<th>m</th>
<th>100</th>
<th>250</th>
<th>500</th>
<th>750</th>
<th>1000</th>
</tr>
</thead>
<tbody>
<tr>
<td>50</td>
<td>16909.19</td>
<td>24466.91</td>
<td>33939.46</td>
<td>51379.72</td>
<td>55030.22</td>
</tr>
<tr>
<td>100</td>
<td>73302.23</td>
<td>100021.87</td>
<td>146646.69</td>
<td>186347.49</td>
<td>237793.15</td>
</tr>
<tr>
<td>150</td>
<td>167198.10</td>
<td>224018.16</td>
<td>343182.45</td>
<td>467555.43</td>
<td>587717.04</td>
</tr>
<tr>
<td>200</td>
<td>299889.37</td>
<td>415152.65</td>
<td>603427.14</td>
<td>824817.30</td>
<td>1019130.68</td>
</tr>
<tr>
<td>500</td>
<td>1939329.97</td>
<td>2811636.51</td>
<td>4322823.44</td>
<td>6488239.24</td>
<td>8176309.62</td>
</tr>
</tbody>
</table>
### 5.3.4.2 RAM Consumption

Table 5.6: Cosine Memory Consumption in KB Across Five Separate Trials

<table>
<thead>
<tr>
<th>m</th>
<th>100</th>
<th>250</th>
<th>500</th>
<th>750</th>
<th>1000</th>
</tr>
</thead>
<tbody>
<tr>
<td>50</td>
<td>113.22</td>
<td>170.90</td>
<td>397.16</td>
<td>580.26</td>
<td>691.78</td>
</tr>
<tr>
<td>100</td>
<td>489.10</td>
<td>800.78</td>
<td>1665.42</td>
<td>2295.52</td>
<td>2861.36</td>
</tr>
<tr>
<td>150</td>
<td>1129.16</td>
<td>1861.28</td>
<td>3752.24</td>
<td>5210.16</td>
<td>6382.48</td>
</tr>
<tr>
<td>200</td>
<td>2025.46</td>
<td>3289.02</td>
<td>6717.36</td>
<td>9367.40</td>
<td>11382.46</td>
</tr>
<tr>
<td>500</td>
<td>12878.34</td>
<td>21029.86</td>
<td>42289.58</td>
<td>58199.60</td>
<td>73313.42</td>
</tr>
</tbody>
</table>

Table 5.7: DotProduct Memory Consumption in KB Across Five Separate Trials

<table>
<thead>
<tr>
<th>m</th>
<th>100</th>
<th>250</th>
<th>500</th>
<th>750</th>
<th>1000</th>
</tr>
</thead>
<tbody>
<tr>
<td>50</td>
<td>37.20</td>
<td>89.98</td>
<td>233.82</td>
<td>310.72</td>
<td>395.56</td>
</tr>
<tr>
<td>100</td>
<td>242.82</td>
<td>413.72</td>
<td>948.50</td>
<td>1343.44</td>
<td>1679.26</td>
</tr>
<tr>
<td>150</td>
<td>580.24</td>
<td>983.72</td>
<td>2161.14</td>
<td>3014.16</td>
<td>3754.00</td>
</tr>
<tr>
<td>200</td>
<td>1040.54</td>
<td>1759.64</td>
<td>3808.98</td>
<td>5340.22</td>
<td>6658.02</td>
</tr>
<tr>
<td>500</td>
<td>6276.10</td>
<td>10934.96</td>
<td>24057.46</td>
<td>33949.14</td>
<td>42594.18</td>
</tr>
</tbody>
</table>
Table 5.8: Hamming Memory Consumption in KB Across Five Separate Trials

<table>
<thead>
<tr>
<th>m</th>
<th>100</th>
<th>250</th>
<th>500</th>
<th>750</th>
<th>1000</th>
</tr>
</thead>
<tbody>
<tr>
<td>50</td>
<td>77.98</td>
<td>118.94</td>
<td>198.54</td>
<td>279.48</td>
<td>358.70</td>
</tr>
<tr>
<td>100</td>
<td>408.02</td>
<td>624.00</td>
<td>934.66</td>
<td>1259.46</td>
<td>1564.36</td>
</tr>
<tr>
<td>n</td>
<td>150</td>
<td>949.72</td>
<td>1400.48</td>
<td>2184.36</td>
<td>2865.06</td>
</tr>
<tr>
<td>200</td>
<td>1709.78</td>
<td>2540.00</td>
<td>3818.02</td>
<td>4990.54</td>
<td>6067.64</td>
</tr>
<tr>
<td>500</td>
<td>10738.40</td>
<td>15597.16</td>
<td>23402.70</td>
<td>31342.64</td>
<td>39710.58</td>
</tr>
</tbody>
</table>

Table 5.9: Spectral Memory Consumption in KB Across Five Separate Trials

<table>
<thead>
<tr>
<th>m</th>
<th>100</th>
<th>250</th>
<th>500</th>
<th>750</th>
<th>1000</th>
</tr>
</thead>
<tbody>
<tr>
<td>50</td>
<td>40.42</td>
<td>61.68</td>
<td>139.24</td>
<td>192.72</td>
<td>232.02</td>
</tr>
<tr>
<td>100</td>
<td>238.32</td>
<td>412.06</td>
<td>644.82</td>
<td>869.94</td>
<td>1100.10</td>
</tr>
<tr>
<td>n</td>
<td>150</td>
<td>583.52</td>
<td>943.12</td>
<td>1497.88</td>
<td>1989.10</td>
</tr>
<tr>
<td>200</td>
<td>1049.12</td>
<td>1654.76</td>
<td>2642.10</td>
<td>3449.84</td>
<td>4224.14</td>
</tr>
<tr>
<td>500</td>
<td>6541.60</td>
<td>10390.60</td>
<td>16804.22</td>
<td>24440.16</td>
<td>31568.86</td>
</tr>
</tbody>
</table>
Fig. 5.16: Average Runtime in Microseconds by Method and by Items, Customers

Fig. 5.17: Average RAM in KB by Method and by Items, Customers
We can conclude from the tables that the runtimes increase substantially as $m$ increases, though not necessarily linearly as we would expect. It would appear that code optimizations ensure slightly better performance within each language. Though we note, and emphasize, that the increase in $n$ or the number of items has very drastic effects on both the memory consumption and the runtime. In general, the RAM consumption differential is much more dramatic, likely because the distance matrix must be stored in memory and increases exponentially in storage size with respect to $n$. 
Chapter 6

Caveat Exploration: Filters, Sparsity, and Architecture

The purpose of this chapter is to explore possible avenues which may be of interest to the practitioner. First, we attempt to filter the spectra themselves. This is because a filter may improve the results by removing some of the noise in the spectra. Alternatively, if a filter does not harm the results meaningfully, we may be able to reduce the series even more compactly than by a factor of two. In order to do this, we conduct a factorial experiment and analyze the results in terms of accuracy. Second, we analyze whether or not more exotic noise processes for the binary data affect the performance of the method meaningfully. In the prior experiment, one could argue that in many cases, the Bernoulli model of the demand series is insufficient, and that real data may contain sparsity of varying types. That is to say, reducing the overall proportion of zeros may not be enough, but rather long runs of zeros or ones may have an effect on the spectral method (and certainly the Dotproduct and Cosine measures in view of the pathological case we constructed in the prior chapter). Lastly, we analyze how the method scales on different architectures within the paradigm of the $R$ programming language. First, we provide a benchmark experiment, and use a benchmarking software to determine information about the processors used in the experiment. We then use explicit parallel programming to scale the method through a range of matrix sizes and number of cores to determine if the method is scalable relative to the competitors.
6.1 Adding the Danielli Filter

In this section, we conduct an experiment wherein we vary the window of the Danielli filter used to smooth the spectra. We choose the Danielli filter because it is computationally quick to compute (since it is simply the average of a window size of points) and also because if successful, it can summarize the spectra more compactly (assuming we use the point-wise averages as opposed to the smoothing window). We use the Top-N algorithm as provided in Algorithm 8 wherein:

\[ \rho \text{ is the Nonparametric Spectral Density} \]

\[ \beta \text{ is the } t - \beta \text{ and } t + \beta \text{ window for the Kernel} \]

Algorithm 8 Spectral Top-N with Filter

\[
\text{for } j = 1 \ldots m \text{ do} \\
\quad \text{for } i = 1 \ldots m \text{ do} \\
\quad \quad \text{if } i \neq j \text{ then} \\
\quad \quad \quad \rho_{i,j} = \sum \text{[Danielli Kernel((R(* , j), R(* , i)), \beta)]} \\
\quad \quad \quad \text{else Let } \rho_{i,j} = 0 \\
\quad \quad \text{end if} \\
\quad \text{end for} \\
\quad \text{for } i = 1 \ldots m \text{ do} \\
\quad \quad \text{if } \rho_{i,j} \neq \text{the k largest values in } \rho_{* , j} \text{ then } \rho_{i,j} = 0 \\
\quad \text{end if} \\
\text{end for} \\
\text{end for} \\
\text{Return } \rho
\]
Table 6.1: Factor Levels for Filtering Experiment

<table>
<thead>
<tr>
<th>Factor</th>
<th>L1</th>
<th>L2</th>
<th>L3</th>
<th>L4</th>
<th>L5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Top-N number</td>
<td>3</td>
<td>5</td>
<td>10</td>
<td>15</td>
<td>20</td>
</tr>
<tr>
<td>No. of Items</td>
<td>50</td>
<td>100</td>
<td>150</td>
<td>200</td>
<td>500</td>
</tr>
<tr>
<td>Customers</td>
<td>100</td>
<td>250</td>
<td>500</td>
<td>750</td>
<td>1000</td>
</tr>
<tr>
<td>P</td>
<td>.1</td>
<td>.2</td>
<td>.3</td>
<td>.5</td>
<td>.9</td>
</tr>
<tr>
<td>β</td>
<td>None</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>6</td>
</tr>
</tbody>
</table>

The Factor Levels Used to Analyze the Effect of Filtering the Spectra

Table 6.1 delineates the parameter levels which are used in the spectra filtering experiment. We vary $n$ and $m$, which is the number of customers and items respectively. Also, we vary the probability of the Bernoulli distribution of size one [$P$]. We use these parameters because Top-N, $n$, and $m$ have been shown to affect accuracy, and intuitively, $P$ may affect the filter performance. $\beta$ is the size of the filter window used, which we vary from no filter at all to a window size of six. The purpose of this experiment is to determine if we can leverage noise out of the spectra or if we can represent the series more compactly.
Fig. 6.1: The Accuracy of the Methods by Filter Level

In Figure 6.1 we can visually determine that the accuracy is negatively affected by increasing the span of the filter window. Even a slight increase [a window size of two] had a negative effect on the accuracy. That said, this is important because it opens a new area of investigation and opportunity. If we can get away with only a small decrease in accuracy by increasing the filter size to two, that would mean that we can summarize the series by another, more compact series $\frac{1}{4}$ the original length. Though we should note that in the profiling routine which we used in R, the original spectra are computed within the function and the memory footprint should not see significant savings because of this. Though if the code was optimized, we should see RAM savings to a much greater degree in terms of maintaining the demand matrix in memory [also if we did summarize with a Danielli filter with a windowsize of two, the distance matrix would be decreased by a factor of 4 as well: i.e. $\frac{nm^2}{8}$.]
In Figure 6.2, we see that the run time does not suffer very much with the addition of the Danielli filter calculation. This is in line with what we would expect because the computation of the filter is not very difficult computationally. Another corollary from the plot is that the Spectral method simply outperforms the Cosine measure in terms of runtime, though that is a foregone conclusion.
Fig. 6.3: The Memory Consumed of the Cosine and Spectral Method by Filter Level (in log scale)

*Figure 6.3* is possibly misrepresentative in that when R computes the Danielli filter, it maintains the original information in memory and simply adds to it. More explicitly, it alters the original information in place, and thus we do not see any benefit in the RAM consumption by the filter window. However, it could be expressed in such a way that we would be able to more compactly represent the series if we were to use the window-wise averages as the new series implicitly. This is an area which deserves further research to see if we can increase the RAM benefits and decreased computational burden with this method while not further reducing the accuracy significantly.
In Figure 6.4 we see that the runtime did not worsen with the filter window nor improve, simply because R was doing the pointwise averages and modifying the series in place. We investigate this further below.
In *Figure* 6.5, we analyze if parametric statistics are possible in order to assess formally whether the filter window affected the accuracy. Since in *Figure* 6.5, it is non-normal and likely cannot be transformed to normal, we revert to the Wilcoxon Rank Sum Test with Continuity Correction.

Table 6.2: Wilcoxon Rank Sum Test with Continuity Correction: Filter Window against Hamming-Spectral Agreement

<table>
<thead>
<tr>
<th>Null Hypothesis</th>
<th>True location shift is zero</th>
</tr>
</thead>
<tbody>
<tr>
<td>Formula</td>
<td>Spectral-Hamming Agreement by Window Size</td>
</tr>
<tr>
<td>W</td>
<td>1953125</td>
</tr>
<tr>
<td>P-value</td>
<td>$2.2 \times 10^{-16}$</td>
</tr>
</tbody>
</table>

Using *Table* 6.2 we can conclusively say that the addition of the filter window most definitely affected the accuracy of the Hamming Spectral agreement, and if we are to believe the beanplots above, the accuracy was affected negatively.

In *Tables* 6.3 and 6.4 we take a more in depth look at the effect of the window size against the run-times and memory consumption. Since the window size is only
Table 6.3: Mean Memory Consumed in KB by Filter Span

<table>
<thead>
<tr>
<th>Spans</th>
<th>0</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>TopN-Cos</td>
<td>10743.091</td>
<td>10714.835</td>
<td>10738.356</td>
<td>10736.667</td>
<td>10739.716</td>
</tr>
<tr>
<td>TopN-DP</td>
<td>6114.475</td>
<td>6100.576</td>
<td>6118.794</td>
<td>6120.183</td>
<td>6113.044</td>
</tr>
<tr>
<td>TopN-Hamming</td>
<td>6257.929</td>
<td>6195.681</td>
<td>6245.150</td>
<td>6249.237</td>
<td>6252.421</td>
</tr>
<tr>
<td>TopN-Spectral</td>
<td>4522.853</td>
<td>4457.663</td>
<td>4512.668</td>
<td>4516.063</td>
<td>4524.426</td>
</tr>
</tbody>
</table>

Table 6.4: Mean Runtime in Milliseconds by Filter Span

<table>
<thead>
<tr>
<th>Spans</th>
<th>0</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>TopN-Cos</td>
<td>6310749.20</td>
<td>6317858.10</td>
<td>6346698.19</td>
<td>6268261.56</td>
<td>6432629.74</td>
</tr>
<tr>
<td>TopN-DP</td>
<td>5593673.72</td>
<td>5661147.82</td>
<td>5771620.03</td>
<td>5695793.57</td>
<td>5688109.58</td>
</tr>
<tr>
<td>TopN-Hamming</td>
<td>1443612.32</td>
<td>1436165.36</td>
<td>1439235.18</td>
<td>1439572.54</td>
<td>1438410.93</td>
</tr>
<tr>
<td>TopN-Spectral</td>
<td>1182928.89</td>
<td>1178362.80</td>
<td>1183934.26</td>
<td>1178901.29</td>
<td>1178516.35</td>
</tr>
</tbody>
</table>

relevant to the Spectral method, we would expect to see that the values of the spectral method would vary more than the other methods if they were affected. These tables tell us that, at least in R, this is not the case. In order to institute an algorithm where this is true, we would have to slide the window over the series and only maintain a summary of the series as the pointwise averages themselves per each four observations, as opposed to sliding the window over the entire series and maintaining a series equivalent to the original length.

6.2 Evaluating Sparsity and its Effects

The motivation for this section is that sparsity can be expressed in many different flavors, and a simple probabilistic model may not be enough. Here, we investigate a more exotic noise generation process in order to produce runs of 0’s and 1’s in the synthetic data [as would be likely in real-world data] and analyze whether or not this has any real
effect on the performance of the method with respect to our “closness”, run-time, or RAM consumption.

6.2.1 Methodology

In order to generate the series and analyze the results, we perform a similar experiment as in Chapter 5, however we vary the noise generation process so that we can obtain insight about the effects of sparsity. In this experiment we propose four new parameters:

- $P_{00}$: the probability of transitioning from state zero to state zero
- $P_{10}$: the probability of transitioning from state one to state zero
- $P_{01}$: the probability of transitioning from state zero to state one
- $P_{11}$: the probability of transitioning from state one to state one

Fig. 6.6: A Binary Markov Model: The Beginnings of a Noise Generation Process
In order to conduct a factorial experiment, we use the fact that $P_{10} + P_{00} = 1$ and $P_{01} + P_{11} = 1$. As a result, we vary two parameters $P_{00}$ and $P_{10}$ (which are representative of sparsity jointly) and the resulting parameter set is easily found. In this way, we can determine the effect of sparsity by examining the relative change in accuracy as $P_{00}$ is changed, because this parameter is responsible both for the general level of sparsity and the tendency to have runs of zeros.

Table 6.5: Factor Levels for Sparsity Experiment

<table>
<thead>
<tr>
<th>Factor</th>
<th>L1</th>
<th>L2</th>
<th>L3</th>
<th>L4</th>
<th>L5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Top-N number</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>No. of Items</td>
<td>50</td>
<td>100</td>
<td>500</td>
<td>1000</td>
<td>2500</td>
</tr>
<tr>
<td>Customers</td>
<td>100</td>
<td>500</td>
<td>1000</td>
<td>2000</td>
<td>5000</td>
</tr>
<tr>
<td>$P_{00}$</td>
<td>.5</td>
<td>.6</td>
<td>.7</td>
<td>.8</td>
<td>.9</td>
</tr>
<tr>
<td>$P_{10}$</td>
<td>.5</td>
<td>.6</td>
<td>.7</td>
<td>.8</td>
<td>.9</td>
</tr>
</tbody>
</table>

The Factor Levels proposed for the Sparsity Experiment

We vary the number of items and customers over a large range in order to make the new noise generation process meaningful. Also, we fix the Top-N number in order to make computation tenable and also to allow for a change in accuracy to be attributable to the transition probabilities if there exists one. Lastly, we make these probabilities higher in terms of their factor levels in order to make sure we generate sparse data.
In Figure 6.7 we demonstrate how the proportion of zero’s in the series are affected when the respective parameters are changed in the experiment shown in Table 6.5. The results are intuitive, if the probability of transitioning from zero to zero and from one to zero are increased largely, then the majority of the series is composed of zeros. We are interested though in the $P_{00}$ primarily because it will be reflective of the number of runs of zeros.
In Figure 6.8, we see again the artifacts which make Cosine and Hamming fundamentally different, as the proportion of zeros and the tendency to have runs of zeros was increased, we note that the Cosine measure did not perform well or consistently, whereas the Hamming-Spectral agreement improved as the sparsity and runs of zeros were increased. This is encouraging because we hope to match the Hamming measure on binary data and also because real data in this application area will be very sparse.

Table 6.6: Kruskal-Wallis Rank Sum Test: Hamming Spectral Agreement against $P_{00}$

<table>
<thead>
<tr>
<th>Null Hypothesis Formula</th>
<th>$P_{00}$ has no effect</th>
<th>$P_{00}$ has no effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kruskal-Wallis $\chi^2$</td>
<td>Spectral-Hamming Agreement by $P_{00}$</td>
<td>14.7665</td>
</tr>
</tbody>
</table>
In Table 6.6 we can see that the $P_{00}$ does have an effect on the agreement, insofar that when it is increased, so too does our accuracy.

![Runtime and Memory Consumed: A Comparison of Hamming and Spectral](image)

Fig. 6.9: Runtime and Memory Consumed: A Comparison of Hamming and Spectral

In Figure 6.9 we can see that the Spectral Method with the new series generation process still handily outperformed the Hamming measure in terms of runtime and memory consumed. In Table 6.7 we can see that the sparsity did not have an effect on the Hamming or Spectral runtimes or memory consumption for any reasonable $\alpha$. 
Table 6.7: Kruskal-Wallis Rank Sum Tests $P_{00}$ against Hamming and Spectral Runtimes and Memory Consumption

<table>
<thead>
<tr>
<th>Null Hypothesis</th>
<th>$P_{00}$ has no effect on Spectral Runtime</th>
</tr>
</thead>
<tbody>
<tr>
<td>Formula</td>
<td>Spectral Runtime against $P_{00}$</td>
</tr>
<tr>
<td>Kruskal-Wallis $\chi^2$</td>
<td>.0874</td>
</tr>
<tr>
<td>Degrees of Freedom</td>
<td>4</td>
</tr>
<tr>
<td>P-value</td>
<td>.9991</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Null Hypothesis</th>
<th>$P_{00}$ has no effect on Spectral Memory Used</th>
</tr>
</thead>
<tbody>
<tr>
<td>Formula</td>
<td>Spectral Memory Used against $P_{00}$</td>
</tr>
<tr>
<td>Kruskal-Wallis $\chi^2$</td>
<td>.014</td>
</tr>
<tr>
<td>Degrees of Freedom</td>
<td>4</td>
</tr>
<tr>
<td>P-value</td>
<td>.9999</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Null Hypothesis</th>
<th>$P_{00}$ has no effect on Hamming Runtime</th>
</tr>
</thead>
<tbody>
<tr>
<td>Formula</td>
<td>Hamming Runtime against $P_{00}$</td>
</tr>
<tr>
<td>Kruskal-Wallis $\chi^2$</td>
<td>.1263</td>
</tr>
<tr>
<td>Degrees of Freedom</td>
<td>4</td>
</tr>
<tr>
<td>P-value</td>
<td>.9981</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Null Hypothesis</th>
<th>$P_{00}$ has no effect on Hamming Memory Used</th>
</tr>
</thead>
<tbody>
<tr>
<td>Formula</td>
<td>Hamming Memory Used against $P_{00}$</td>
</tr>
<tr>
<td>Kruskal-Wallis $\chi^2$</td>
<td>.0099</td>
</tr>
<tr>
<td>Degrees of Freedom</td>
<td>4</td>
</tr>
<tr>
<td>P-value</td>
<td>.9999</td>
</tr>
</tbody>
</table>
6.3 Performance on Different Computational Architectures

In this section, we attempt to assess the scalability of the Top-N algorithm with Hutchison-Kumara similarity. In order to do this, we use explicit parallelization and rewrite the existing codebase to function on multiple core systems. We first use the Unix benchmarking tool *Likwid* to characterize the processors and RAM on our local machine (which is responsible for the numbers of cores 1-8). We then utilize Amazon’s Elastic Cloud Computing service to run an instance with 36 cores, and tabulate the results.

6.3.1 Characterization of the Architectures

The CPU used in the benchmarks for cores 1-8 as well as all other experiments was an Intel(R) Core(TM) i7-2670QM CPU with 2.20GHz clock and 8 GB of RAM. The following tables simply characterize this hardware as well as any nuances in its particular manufacture. The data in Tables 6.8 to 6.19 are simply reference data for our particular bench. After characterizing the architecture, we move on to methodology.
Table 6.8: Hardware Thread Topology of this SandyBridge Processor [CPU Stepping: 7]

<table>
<thead>
<tr>
<th>Sockets</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cores per Socket</td>
<td>4</td>
</tr>
<tr>
<td>Threads per Core</td>
<td>2</td>
</tr>
<tr>
<td>Socket 0</td>
<td>(0,1,2,3,4,5,6,7)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>HWThread</th>
<th>Thread</th>
<th>Core</th>
<th>Socket</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>3</td>
<td>0</td>
</tr>
</tbody>
</table>
Table 6.9: Cache Topology of this SandyBridge Processor

<table>
<thead>
<tr>
<th>Level</th>
<th>Size</th>
<th>Cache Groups</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>32 kilobytes</td>
<td>( 0 1 ) ( 2 3 ) ( 4 5 ) ( 6 7 )</td>
</tr>
<tr>
<td>2</td>
<td>256 kilobytes</td>
<td>( 0 1 ) ( 2 3 ) ( 4 5 ) ( 6 7 )</td>
</tr>
<tr>
<td>3</td>
<td>6 Megabytes</td>
<td>( 0 1 2 3 4 5 6 7 )</td>
</tr>
</tbody>
</table>

Table 6.10: NUMA Topology of this SandyBridge Computer

<table>
<thead>
<tr>
<th>NUMA domains</th>
<th>Processors</th>
<th>Relative distance to nodes</th>
<th>Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0 1 2 3 4 5 6 7</td>
<td>10</td>
<td>8139.8 Megabytes</td>
</tr>
</tbody>
</table>
Table 6.11: Clock and Power Consumption of this SandyBridge Processor

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Base clock</strong></td>
<td>2933.26 MHz</td>
</tr>
<tr>
<td><strong>Minimal clock</strong></td>
<td>1066.64 MHz</td>
</tr>
<tr>
<td><strong>Turbo Boost Steps</strong></td>
<td></td>
</tr>
<tr>
<td><strong>C0</strong></td>
<td>4133.23 MHz</td>
</tr>
<tr>
<td><strong>C1</strong></td>
<td>3999.90 MHz</td>
</tr>
<tr>
<td><strong>C2</strong></td>
<td>3733.24 MHz</td>
</tr>
<tr>
<td><strong>C3</strong></td>
<td>3733.24 MHz</td>
</tr>
<tr>
<td><strong>Thermal Spec Power</strong></td>
<td>45 Watts</td>
</tr>
<tr>
<td><strong>Minimum Power</strong></td>
<td>36 Watts</td>
</tr>
<tr>
<td><strong>Maximum Power</strong></td>
<td>45 Watts</td>
</tr>
<tr>
<td><strong>Maximum Time Window</strong></td>
<td>0.03125 micro sec</td>
</tr>
</tbody>
</table>

Table 6.12: FLOPS Performance of this Processor by Event on Likwid Benchmark

<table>
<thead>
<tr>
<th>Event</th>
<th>Counter</th>
<th>Sum</th>
<th>Max</th>
<th>Min</th>
<th>Avg</th>
</tr>
</thead>
<tbody>
<tr>
<td>INSTR_RETIRED_ANY STAT</td>
<td>FIXC0</td>
<td>362474</td>
<td>351174</td>
<td>0</td>
<td>45309.25</td>
</tr>
<tr>
<td>CPU_CLK_UNHALTED_CORE STAT</td>
<td>FIXC1</td>
<td>686432</td>
<td>616967</td>
<td>0</td>
<td>85804</td>
</tr>
<tr>
<td>CPU_CLK_UNHALTED_REF STAT</td>
<td>FIXC2</td>
<td>1241372</td>
<td>1050346</td>
<td>0</td>
<td>155171.5</td>
</tr>
<tr>
<td>FP_COMP_OPS_EXE_SSE_FP_PACKED_DOUBLE STAT</td>
<td>PMC0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>FP_COMP_OPS_EXE_SSE_FP_SCALAR_DOUBLE STAT</td>
<td>PMC1</td>
<td>783</td>
<td>774</td>
<td>0</td>
<td>97.875</td>
</tr>
<tr>
<td>FP_256_PACKED_DOUBLE STAT</td>
<td>PMC2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Table 6.13: FLOPS Performance of this Processor by Metric on *Likwid* Benchmark

<table>
<thead>
<tr>
<th>Metric</th>
<th>Sum</th>
<th>Max</th>
<th>Min</th>
<th>Avg</th>
</tr>
</thead>
<tbody>
<tr>
<td>Runtime (RDTSC) [s] STAT</td>
<td>1.907225e-08</td>
<td>2.384031e-09</td>
<td>2.384031e-09</td>
<td>2.384031e-09</td>
</tr>
<tr>
<td>Runtime unhalted [s] STAT</td>
<td>3.127370e-04</td>
<td>0.0002810889</td>
<td>0</td>
<td>3.909213e-05</td>
</tr>
<tr>
<td>Clock [MHz] STAT</td>
<td>2087.4455</td>
<td>1289.282</td>
<td>0</td>
<td>260.9306875</td>
</tr>
<tr>
<td>CPI STAT</td>
<td>7.904215</td>
<td>6.147345</td>
<td>0</td>
<td>0.988026875</td>
</tr>
<tr>
<td>MFlops/s STAT</td>
<td>328435.319</td>
<td>324660.2</td>
<td>0</td>
<td>41054.414875</td>
</tr>
<tr>
<td>32b AVX MFlops/s STAT</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Packed MUOPS/s STAT</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Scalar MUOPS/s STAT</td>
<td>328435.319</td>
<td>324660.2</td>
<td>0</td>
<td>41054.414875</td>
</tr>
</tbody>
</table>

Table 6.14: L2 Performance by Event on *Likwid* Benchmark

<table>
<thead>
<tr>
<th>Event</th>
<th>Counter</th>
<th>Sum</th>
<th>Max</th>
<th>Min</th>
<th>Avg</th>
</tr>
</thead>
<tbody>
<tr>
<td>INSTR RETIRED ANY STAT</td>
<td>FIXC0</td>
<td>662713</td>
<td>307040</td>
<td>0</td>
<td>82839.125</td>
</tr>
<tr>
<td>CPU CLK UNHALTED CORE STAT</td>
<td>FIXC1</td>
<td>2060679</td>
<td>631327</td>
<td>0</td>
<td>257584.875</td>
</tr>
<tr>
<td>CPU CLK UNHALTED REF STAT</td>
<td>FIXC2</td>
<td>3865994</td>
<td>1163558</td>
<td>0</td>
<td>483249.25</td>
</tr>
<tr>
<td>L1D REPLACEMENT STAT</td>
<td>PMC0</td>
<td>19678</td>
<td>5672</td>
<td>0</td>
<td>2459.75</td>
</tr>
<tr>
<td>L1D M EVICT STAT</td>
<td>PMC1</td>
<td>4061</td>
<td>1127</td>
<td>0</td>
<td>507.625</td>
</tr>
</tbody>
</table>
Table 6.15: L2 Performance by Metric on *Likwid* Benchmark

<table>
<thead>
<tr>
<th>Metric</th>
<th>Sum</th>
<th>Max</th>
<th>Min</th>
<th>Avg</th>
</tr>
</thead>
<tbody>
<tr>
<td>Runtime (RDTSC) [s] STAT</td>
<td>1.885376e-08</td>
<td>2.35672e-09</td>
<td>2.35672e-09</td>
<td>2.35672e-09</td>
</tr>
<tr>
<td>Runtime unhalted [s] STAT</td>
<td>0.0009387753</td>
<td>0.0002876111</td>
<td>0</td>
<td>1.173469e-04</td>
</tr>
<tr>
<td>Clock [MHz] STAT</td>
<td>4667.377</td>
<td>1317.04</td>
<td>0</td>
<td>583.422125</td>
</tr>
<tr>
<td>CPI STAT</td>
<td>14.438539</td>
<td>5.398634</td>
<td>0</td>
<td>1.804817375</td>
</tr>
<tr>
<td>L2 Load [MBytes/s] STAT</td>
<td>534383400</td>
<td>154031000</td>
<td>0</td>
<td>66797925</td>
</tr>
<tr>
<td>L2 Evict [MBytes/s] STAT</td>
<td>110282090</td>
<td>30605250</td>
<td>0</td>
<td>13785261.25</td>
</tr>
<tr>
<td>L2 bandwidth [MBytes/s] STAT</td>
<td>644665600</td>
<td>180970200</td>
<td>0</td>
<td>80583200</td>
</tr>
<tr>
<td>L2 data volume [GBytes] STAT</td>
<td>0.001519296</td>
<td>0.000426496</td>
<td>0</td>
<td>0.000189912</td>
</tr>
</tbody>
</table>

Table 6.16: L2 Cache Performance by Event on *Likwid* Benchmark

<table>
<thead>
<tr>
<th>Event</th>
<th>Counter</th>
<th>Sum</th>
<th>Max</th>
<th>Min</th>
<th>Avg</th>
</tr>
</thead>
<tbody>
<tr>
<td>INSTR_RETIRED_ANY STAT</td>
<td>FIXC0</td>
<td>298313</td>
<td>298313</td>
<td>0</td>
<td>37289.125</td>
</tr>
<tr>
<td>CPU_CLK_UNHALTED_CORE STAT</td>
<td>FIXC1</td>
<td>546469</td>
<td>546469</td>
<td>0</td>
<td>68308.625</td>
</tr>
<tr>
<td>CPU_CLK_UNHALTED_REF STAT</td>
<td>FIXC2</td>
<td>1502688</td>
<td>1502688</td>
<td>0</td>
<td>187836</td>
</tr>
<tr>
<td>L2_TRANS_ALL_REQUESTS STAT</td>
<td>PMC0</td>
<td>24198</td>
<td>24198</td>
<td>0</td>
<td>3024.75</td>
</tr>
<tr>
<td>L2_RQSTS_MISS STAT</td>
<td>PMC1</td>
<td>4504</td>
<td>4504</td>
<td>0</td>
<td>563</td>
</tr>
</tbody>
</table>
Table 6.17: L2 Cache Performance by Metric on *Likwid* Benchmark

<table>
<thead>
<tr>
<th>Metric</th>
<th>Sum</th>
<th>Max</th>
<th>Min</th>
<th>Avg</th>
</tr>
</thead>
<tbody>
<tr>
<td>Runtime (RDTSC) [s] STAT</td>
<td>2.445362e-08</td>
<td>3.056702e-09</td>
<td>3.056702e-09</td>
<td>3.056702e-09</td>
</tr>
<tr>
<td>Runtime unhalted [s] STAT</td>
<td>0.0002489515</td>
<td>0.0002489515</td>
<td>0</td>
<td>3.111894e-05</td>
</tr>
<tr>
<td>Clock [MHz] STAT</td>
<td>798.2657</td>
<td>798.2657</td>
<td>0</td>
<td>99.7832125</td>
</tr>
<tr>
<td>CPI STAT</td>
<td>1.831865</td>
<td>1.831865</td>
<td>0</td>
<td>0.228983125</td>
</tr>
<tr>
<td>L2 request rate STAT</td>
<td>0.08111614</td>
<td>0.08111614</td>
<td>0</td>
<td>0.0101395175</td>
</tr>
<tr>
<td>L2 miss rate STAT</td>
<td>0.01509824</td>
<td>0.01509824</td>
<td>0</td>
<td>0.00188728</td>
</tr>
<tr>
<td>L2 miss ratio STAT</td>
<td>0.1861311</td>
<td>0.1861311</td>
<td>0</td>
<td>0.0232663875</td>
</tr>
</tbody>
</table>

Table 6.18: Branch Performance by Event on *Likwid* Benchmark

<table>
<thead>
<tr>
<th>Event</th>
<th>Counter</th>
<th>Sum</th>
<th>Max</th>
<th>Min</th>
<th>Avg</th>
</tr>
</thead>
<tbody>
<tr>
<td>INSTR RETIRED ANY STAT</td>
<td>FIXC0</td>
<td>293806</td>
<td>293806</td>
<td>0</td>
<td>36725.75</td>
</tr>
<tr>
<td>CPU CLK UNHALTED CORE STAT</td>
<td>FIXC1</td>
<td>520379</td>
<td>520379</td>
<td>0</td>
<td>65047.375</td>
</tr>
<tr>
<td>CPU CLK UNHALTED REF STAT</td>
<td>FIXC2</td>
<td>1430902</td>
<td>1430902</td>
<td>0</td>
<td>178862.75</td>
</tr>
<tr>
<td>BR INST RETIRED ALL BRANCHES</td>
<td>PMC0</td>
<td>61288</td>
<td>61288</td>
<td>0</td>
<td>7661</td>
</tr>
<tr>
<td>BR MISP RETIRED ALL BRANCHES</td>
<td>PMC1</td>
<td>2834</td>
<td>2834</td>
<td>0</td>
<td>354.25</td>
</tr>
</tbody>
</table>
Table 6.19: Branch Performance by Metric on *Likwid* Benchmark

<table>
<thead>
<tr>
<th>Metric</th>
<th>Sum</th>
<th>Max</th>
<th>Min</th>
<th>Avg</th>
</tr>
</thead>
<tbody>
<tr>
<td>Runtime (RDTSC) [s] STAT</td>
<td>2.651015e-08</td>
<td>3.313769e-09</td>
<td>3.313769e-09</td>
<td>3.313769e-09</td>
</tr>
<tr>
<td>Runtime unhalted [s] STAT</td>
<td>0.0002370659</td>
<td>0.0002370659</td>
<td>0</td>
<td>2.963324e-05</td>
</tr>
<tr>
<td>Clock [MHz] STAT</td>
<td>798.2897</td>
<td>798.2897</td>
<td>0</td>
<td>99.7862125</td>
</tr>
<tr>
<td>CPI STAT</td>
<td>1.771165</td>
<td>1.771165</td>
<td>0</td>
<td>0.221395625</td>
</tr>
<tr>
<td>Branch rate STAT</td>
<td>0.2086002</td>
<td>0.2086002</td>
<td>0</td>
<td>0.026075025</td>
</tr>
<tr>
<td>Branch misprediction rate STAT</td>
<td>0.009645821</td>
<td>0.009645821</td>
<td>0</td>
<td>1.205728e-03</td>
</tr>
<tr>
<td>Branch misprediction ratio STAT</td>
<td>0.0462407</td>
<td>0.0462407</td>
<td>0</td>
<td>0.0057800875</td>
</tr>
<tr>
<td>Instructions per branch STAT</td>
<td>4.793859</td>
<td>4.793859</td>
<td>0</td>
<td>0.599232375</td>
</tr>
</tbody>
</table>

### 6.3.2 Methodology

Table 6.20: Benchmark 1 for Computational Architectures

<table>
<thead>
<tr>
<th>Factor</th>
<th>L1</th>
<th>L2</th>
<th>L3</th>
<th>L4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Top-N number</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$P$</td>
<td>.5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>No. of Items</td>
<td>100</td>
<td>300</td>
<td>500</td>
<td>1000</td>
</tr>
<tr>
<td>Customers</td>
<td>100</td>
<td>300</td>
<td>500</td>
<td>1000</td>
</tr>
</tbody>
</table>

In Table 6.20 we define the benchmark which is used for each computational architecture, which we use to assess the relative performance of the methods when scaled.
We fix $P$ and Top-N to make computation tenable and also to determine the memory and runtime performance simply relative to the size of the matrix which we generate.

Table 6.21: Mean Runtime in Seconds by Number of Cores

<table>
<thead>
<tr>
<th>Processor</th>
<th>TopN DP</th>
<th>TopN Cos</th>
<th>TopN Hamming</th>
<th>TopN Spec</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>615.2926</td>
<td>1001.7759</td>
<td>682.3792</td>
<td>457.1635</td>
</tr>
<tr>
<td>2</td>
<td>348.28225</td>
<td>351.5065</td>
<td>292.7074</td>
<td>275.3073</td>
</tr>
<tr>
<td>4</td>
<td>285.1801</td>
<td>283.1096</td>
<td>272.6860</td>
<td>269.5439</td>
</tr>
<tr>
<td>6</td>
<td>286.6155</td>
<td>284.2028</td>
<td>274.0947</td>
<td>271.5858</td>
</tr>
<tr>
<td>8</td>
<td>291.0284</td>
<td>288.1535</td>
<td>280.2938</td>
<td>277.8396</td>
</tr>
<tr>
<td>36</td>
<td>287.2965</td>
<td>273.8893</td>
<td>267.0014</td>
<td>263.7453</td>
</tr>
</tbody>
</table>

In Table 6.21 we characterize the results by the mean runtimes by the number of cores. This can be thought of as a summary, though graphical exploration tells us more of the story. We can see from the summary that as the number of cores increases, the computation time of each method reliably decreases, though the overhead of reassembling the results makes for a nonlinear decrease. This is a well understood phenomenon in multicore computing.
In Figure 6.10 we see exactly what we expect to see in the simplest case, which is that as \( n \) and \( m \) increase, so too do the runtimes, and also we see that Cosine performs worst, the Dotproduct performs second worst, followed closely by Hamming, and that the Spectral method is the fastest.
Table 6.22: Quadratic Models Describing Runtime by Method

<table>
<thead>
<tr>
<th>Formula</th>
<th>Top-N Cos Runtime as predicted by Items*Items</th>
</tr>
</thead>
<tbody>
<tr>
<td>Coefficients</td>
<td>Estimate</td>
</tr>
<tr>
<td>Intercept</td>
<td>-635.7062</td>
</tr>
<tr>
<td>Items</td>
<td>2.1277</td>
</tr>
<tr>
<td>Residual Standard Error</td>
<td>774.2</td>
</tr>
<tr>
<td>Multiple R-Squared</td>
<td>.7171</td>
</tr>
<tr>
<td>F-Statistic</td>
<td>35.48</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Formula</th>
<th>Top-N DP Runtime as predicted by Items*Items</th>
</tr>
</thead>
<tbody>
<tr>
<td>Coefficients</td>
<td>Estimate</td>
</tr>
<tr>
<td>Intercept</td>
<td>-395.3576</td>
</tr>
<tr>
<td>Items</td>
<td>2.1277</td>
</tr>
<tr>
<td>Residual Standard Error</td>
<td>592.5</td>
</tr>
<tr>
<td>Multiple R-Squared</td>
<td>.6225</td>
</tr>
<tr>
<td>F-Statistic</td>
<td>23.09</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Formula</th>
<th>Top-N Hamming Runtime as predicted by Items*Items</th>
</tr>
</thead>
<tbody>
<tr>
<td>Coefficients</td>
<td>Estimate</td>
</tr>
<tr>
<td>Intercept</td>
<td>446.1390</td>
</tr>
<tr>
<td>Items</td>
<td>2.3758</td>
</tr>
<tr>
<td>Residual Standard Error</td>
<td>485.1</td>
</tr>
<tr>
<td>Multiple R-Squared</td>
<td>0.7541</td>
</tr>
<tr>
<td>F-Statistic</td>
<td>42.94</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Formula</th>
<th>Top-N Spectral Runtime as predicted by Items*Items</th>
</tr>
</thead>
<tbody>
<tr>
<td>Coefficients</td>
<td>Estimate</td>
</tr>
<tr>
<td>Intercept</td>
<td>-282.853</td>
</tr>
<tr>
<td>Items</td>
<td>1.558</td>
</tr>
<tr>
<td>Residual Standard Error</td>
<td>346.6</td>
</tr>
<tr>
<td>Multiple R-Squared</td>
<td>0.7209</td>
</tr>
<tr>
<td>F-Statistic</td>
<td>36.17</td>
</tr>
</tbody>
</table>
In the above table, we can see that the quadratic model has varying levels of success in attempting to predict the runtime based upon the items. This is likely because there is some noise relative to the variance in customers. Also, we know that there seems to be other factors which would affect the model, such as the probability, or the language which the algorithm is coded in, or even artifacts within the language itself. Nonetheless, we get prediction rates between 69-74% when attempting to predict the runtime solely from the number of items themselves. One would expect that if the number of customers were held constant, this value would improve.

Fig. 6.11: Runtime During Benchmark 1: 2 Cores

In Figure 6.11 as we begin the parallelization on two cores, we begin to see a piecewise pattern emerging, at this juncture this pattern is hard to characterize, but we
can see that the number of customers increases the runtime slightly (as we expect it to linearly) and the number of items has a significant effect. Early in the experiment, the methods were approximately equal, jockeying for position, though at the conclusion, Cosine performed slightly worse than the Dotproduct, while the Hamming and Spectral Methods handily outperformed each, with the Spectral method gaining an advantage. We should also note that on two cores, the worst runtime was approximately 1400 seconds, whereas on 1 core the worst runtime was approximately 5000 seconds.
Fig. 6.12: Runtime During Benchmark 1: 4 Cores

Fig. 6.13: Runtime During Benchmark 1: 6 Cores
In Figures 6.12 and 6.13 we can see the worst runtime drops off to 800 seconds, and more interesting artifacts emerge. In these instances, the performance was much more consistent across the board, with Hamming and Spectral outperforming Cosine and Dotproduct by a very slight margin. There is a cost to cluster computing, and that cost is coordination and reassembly. Because of this cost, we do not see linear benefits in computation time by number of cores. In order to see significant improvements with more cores, it would be necessary to have many more items so that the division of tasks is more significant than the coordination and reassembly operations. This can be viewed in that six cores performs slightly worse than four in the worst cases (because the data is not large enough to justify this parallelization.)

![Runtime in Seconds: 8 Cores](image)

Fig. 6.14: Runtime During Benchmark 1: 8 Cores
In Figure 6.14 we see the computation times level out relative to an increase in the number of items, and Cosine only just performs worse than the Dotproduct as recently as the last iteration. Hamming and Spectral retain faster runtimes with Spectral being slightly faster.

![Runtime in Seconds: 36 Cores](image)

**Fig. 6.15: Runtime During Benchmark 1: 36 Cores**

In Figure 6.15 we see that the Dotproduct performs the worst, likely because it is very parallelizable and allotted to more processors, thus making the reassembly task more difficult. Also we see that the Spectral method still retains an advantage though not by much, and even though the computational difficulty increases in the final stretch, the runtime actually decreases in some iterations. This would suggest that for the benchmark given, 36 cores was far too many, and the added cost of parellization was not necessarily
worth the benefits, as we saw similar results on four cores without the added noise from the re-aggregation of the data.
Chapter 7

Framing the H-K Measure as Apriori: SPEEDUP

The Apriori Algorithm is widely used in mining co-purchased items. Though several variations have appeared, the proposed enhancements do not escape the computational complexity of counting frequent itemsets. In this dissertation proposal, we explored an algorithm with a paradigm shift into the frequency domain, where fast Fourier transforms can be used instead of considering every separate itemset as a sovereign entity. Processing transaction data as signals, we can decrease the computation time compared to benchmark similarity methods, and also decrease the storage size requirements by summarizing transaction data at the Fourier Frequencies only.

The apriori algorithm (Agrawal et al. 1994) is among one of the best methods to mine association rules from a given database. Though the majority of the implementations suffer from the Frequent Set Counting problem, which is the most computationally demanding process within the algorithm. The majority of the revisions focus on effective pruning and frequent itemset representation. (Perego et al. 2001),(Lazcorreta et al. 2008) It cannot be rightly said that we are implementing the Apriori algorithm, because the frequent itemset representation is integral to its definition. We can say that given a number of demand series, we can use the periodicity information to make statements which are equivalent to the Apriori algorithm in terms of detecting co-purchased items.
The *SPEEDUP* (SPectral Estimation to Extract and Decompose Unknown Periodicities) Algorithm uses the Fast Fourier Transform to compactly represent a given time series at its Fourier Frequencies. The result is a series representing the sample spectra. The method then infers what is a relatively large sample spectra for the item series and what is a relatively large jointly occurring sample spectra relative to the rest of the dataset in a given period.

![Example Series and Spectral Density](image)

**Fig. 7.1:** An example of a sine wave with Gaussian Noise and its decomposition

*Figure 7.1* demonstrates the decomposition from a series into its corresponding spectrum. We can see that as the period increases (especially around .4) there are a large number of strong frequencies suggesting that the cycles in this particular (messy) example tend to be longer. In fact, we can transform directly to periods in that the frequency (in this case .4) 1/.4 yields the cycle in periods. So, for our example, 2.5 periods is the dominant cycle approximately. Though this is true, using the FFT to compute the
periodogram is suboptimal; since the periodogram itself is an inconsistent estimator of the spectrum. We can rectify this by using any of the various window functions available; though they will inflate the bounds on our confidence intervals unfortunately, and so this is a question that requires investigation (though for our purposes, the selection should seek to minimize the degrees of freedom of the one-sided Chi-square distribution).

7.1 Methodology

An intuition of the SPEEDUP algorithm is as follows: if we observe a large number of series jointly occurring in a given time window, and we can decompose the relative strength of the sine-cosine pairs at the Fourier Frequencies, items which have relatively strong co-occurring frequencies are then said to be in the same itemset. That is to say, if the item demand series demonstrates strong co-occurring frequencies with another item, the transactions are occurring simultaneously. The co-occurrence can be treated either nonparametrically or parametrically. Previously, we have investigated the parametric case, which is computationally efficient in the case where the series to be analyzed are large and the analyst is comfortable relying on asymptotic properties. In the event this is not true, we develop and test a nonparametric methodology employing empirical methods.

7.1.1 Naive Parametric Case

Suppose we examine the following argument: here, we employ a naive model which ignores the inconsistent estimation of the spectra by the periodogram itself.

First, we can express the series in terms of the finite Fourier representation:
\[ x_t = a_o + \sum_{p=1}^{(N/2)-1} \left[ a_p \cos \left( \frac{2\pi pt}{N} \right) + b_p \sin \left( \frac{2\pi pt}{N} \right) \right] + a_{N/2} \cos (\pi t) \]  

(7.1)

where \( a_o \) is the mean of \( x_t \), \( a_{N/2} = (-1)^t x_t/N \),

\[ a_p = 2(\sum x_t \cos (2\pi pt/N)/N) \] and

\[ b_p = 2x_t \sin (2pt/N)/N \] for \( p = 1..(N/2 - 1) \)

and that the height of the periodogram at frequency \( w_p \) can be expressed as:

\[ I(w_p) = \frac{1}{\pi N} \left| \sum_{t=1}^{N} x_t e^{itw_p} \right|^2 = \]

\[ \left[ \sum x_t \cos (2\pi pt/N)^2 + \sum x_t \sin (2\pi pt/N)^2 \right]/N\pi \]  

(7.2)

However, if we let: \( c_k = \sum_{t=1}^{N-k} (x_t - \bar{x})(x_{t+k} - \bar{x})/N \) then we see that:

\[ I(w_p) = |c_o + 2 \sum_{k=1}^{N-1} c_k \cos (w_p k)/\pi \]  

(7.3)

\[ E[I(w)] = \gamma_o + 2 \sum_{k=1}^{i} n \gamma_k \cos (wk)/\pi \]  

(7.4)
Assuming the spectrum is uniform (i.e. observations are independent normal): $a_p$ and $b_p$ are sums of normal iid variables above and will themselves be normal as well.

(more specifically $a_p, b_p$ are $N(0, \frac{2\sigma^2}{N})$)

\[
\text{Cov}(a_p, b_p) = \frac{4\text{Cov}[\sum x_t \cos(w_p t), \sum x_t \sin(w_p t)]}{N^2} = 4\sigma^2(\cos(w_p t)\sin(w_p t))
\]

\[
\frac{N^2}{(7.5)}
\]

\[
\frac{4\sigma^2(\cos(w_p t)\sin(w_p t))}{N^2}
\]

The above stems from the fact that observations are independent (since they are orthogonal and the correlation is zero, zero correlation for bivariate normals is a sufficient condition for independence). Finally, since we can standardize $a_p$ and $b_p$ by dividing them by $\sqrt{\frac{2\sigma}{\sqrt{N}}}$ and this implies we have the sum of two independent $N(0,1)$ variables, which is a Chi-Square distribution with two degrees of freedom. (Chatfield 2013)

So then for our purposes we will assume uniform spectra across items and periods. For each item $n$ and period $t$ we define:

\[
c_n(t) = \chi^2_\nu(1 - \alpha/2)
\]

where $\nu = 2$ if employing the above logic (though different for other window estimators)

Let the spectral signature for item $n$ in period $t$ be $S_{nt}$. We define a strong cooccurrence for $n$ items in period $t$ as satisfying one condition:
1. \( S_{nt} > c_{nt} \) for item \( n \) and period \( t \)

All \( S_{nt} \) which satisfy this condition in a given period are a measure of how frequently items are bought between users. And we apply this procedure iteratively through all periods produced in the FFT (which is total number of items/4 in the best case).

### 7.1.2 Nonparametric Case

For the nonparametric method, for each item in \( n \) we define:

\[
c_n = (1 - \psi)x_j + \psi x_{j+1}
\] (7.8)

where \((n + 1) \ast (1 - \alpha) = j + \psi\)

and \(j = \text{floor}((n + 1)(1 - \alpha))\)

and \(x\) is the ranked spectra for item \( n \)

For each period \( t \) we define:

\[
c_t = (1 - \psi)x_j + \psi x_{j+1}
\] (7.9)

where \((n + 1) \ast (1 - \alpha) = j + \psi\)

and \(j = \text{floor}((n + 1)(1 - \alpha))\)

and \(x\) is the ranked spectra for period \( t \)
Let the spectral signature for item \( n \) in period \( t \) be \( S_{nt} \). We define a strong co-occurrence for items \( n \) by customer \( t \) as satisfying two conditions:

1. \( S_{nt} > c_n \) for item \( n \)

2. \( S_{nt} > c_t \) for user \( t \)

All \( S_{nt} \) which satisfy this condition in a given period are assumed to be co-occurring items, which may further refine the clustering apriori. We would apply this procedure iteratively through all periods produced in the FFT (which is total number of user observations per item.)

### 7.1.3 Algorithm and Complexity

To understand the intuition of the algorithm, there are two main ideas which are paramount. When generating the periodograms for each item and comparing across items we must first ask ourselves whether or not the observed spectral signature in a given period is high for that period. If the answer to this question is yes, then we have an item (or itemset) which moves at a pace different than the other items. To determine this, we aggregate all spectra from all items across the period and check whether the observed spectra is greater than the \((\alpha)\)th quantile (i.e. \( C_t \)). Stating this another way, we find all items in a given period \( t \) which have a spectral strength in the top quantile for this period. This condition is strong, but not strong enough to act alone. Suppose we have a period in which we observe virtually no activity across periods. If this is true, there still will exist a top 5th quantile of the periods aggregate spectral distribution;
however this particular quantile will see relationships which are not present since the
vision of the statistic is limited only to the current period (i.e. a very inactive one.)

Algorithm 9 Nonparametric Implementation

for item=1...N Items do
    Compute Periodogram with FFT as Estimator
    Let $S_{nt}$ be observed spectra for item $n$ in period $t$
    Compute $c_n$ of periodogram for item $n$ such that:
    $P(X < c_n) = 1 - \alpha$ where $X$ is the item spectra
    for period=1...T do
        Compute $c_t$ of periodogram for period $t$ such that:
        $P(X < c_t) = 1 - \alpha$ where $X$ is the period spectra
        $\forall$ items
        $FrequentItemset_t = S_{nt} \geq c_n$ for item $n$ and $c_t$
        for current period $t$
    end for
end for

To counteract this phenomenon, we define a second statistic to assess whether or not the current periods signature is high for the item itself. In other words, if we observe spectra passing the first condition, but they do not fall within the $(\alpha)$th quantile (i.e. $C_n$) for the item, we simply disregard it and do not add it to the itemset. In this way, an inactive period will detect points which are significant in that period, but since these signatures are not high relative to the item spectra from whence they come, we simply disregard them. A more explicit definition of the above statistic can be viewed in Algorithm 9.

For a dataset of $n$ items over $t$ periods:

1. The FFT is $O(n \log(n))$ in complexity, and so full evaluation will be $n[O(n \log(n))]$

2. The sort on each column to compute the quantile will be $n/4[O(n \log(n/4))]$
3. The sort on each row to compute the quantile will be $t/4[O(n\log(t/4))]$

4. The iteration will be $t/4$ operations

(Alternatively, if using the nonparametric algorithm, we could use Multidimensional FFT on the array over both sets of indices (column and row); since algorithms exist which keep multidimensional FFT on $N$ indices to be $N\log(N)$ in complexity)

Note: none of the above complexities are exponential in $n$ or $t$.

These two particular statistics acting in tandem should prevent erroneous classification of additional items and help us correctly detect the items of interest [i.e. “moving together”] (provided alpha is set correctly). In terms of computational efficiency, the algorithm is efficient, since we do not see exponential increases in the parameters, and also because we decrease the computational load relative to the Cosine, Dotproduct, and Hamming [as seen in chapters 3, 4, 5, and 6]. This is because when decomposing the spectra, the length of the series decreases by a factor of two to four, which gives us less to iterate over. We should note that it is also possible to further hasten algorithm execution by subsetting the columns and rows which meet the conditions necessary. In the vertical direction, we perform a FFT only once per item, and iterate over total-periods/4 to classify the items and we do so simultaneously. We do not suffer the combinatorial drawback of finding frequent itemsets in all possible combinations.
Fig. 7.2: red, on left for each item starting at top) and the corresponding spectra (blue, on right) used in the proof of concept example
7.1.4 Proof of Concept

In the exploratory stage, three itemsets were generated by convoluting sine-cosine pairs and adding white noise to the result. Three distinct known sets were generated to explore the merit of further exploration. Each sine-cosine pair was assigned a specific frequency for each itemset. In this simple example, items 1-4 were an itemset, items 5-8 were an itemset, and items 9-10 were an itemset. In the preliminary stages, it is evident that tuning alpha is a very important step for algorithm accuracy. In this case, since we were only viewing ten items, high values of alpha would truncate the distribution too greatly to glean meaningful results.

The method correctly classified the itemsets in this case, though this test is not rigorous. One could see that there are many factors which may affect the performance of the algorithm and therefore we require a more intimate exploration of the method. So, we move toward an experimental design.

7.1.5 Pilot Experimental Design

In order to give the Apriori method a fairly rigorous testing, we should test SPEEDUP through a range of values in order to understand what the general performance/behavior of the method is like. Further, we may receive insight on how to tune the parameters or limitations of the method computationally. First, we consider the factors which are likely to harm any method meant to detect co-occurring phenomena. First, we assume that the total number of items will matter, and also the series length will matter. Also, we can assume the group size, the underlying frequency, and our
chosen alpha parameter likely matter. We will measure accuracy as the number misclassified in each period (though we know this has two components, those items which go unrecognized, and those items which are erroneously recognized).

Intuition tells us that we will likely experience an increase in accuracy as $N$ and $L$ increase, since we will be taking advantage of the asymptotic nature of the empirical distribution. Also, a smaller group size will likely increase our accuracy (less chance to erroneously drop/add randomly). How to tune alpha is a sensitive question. If $N$ is small (around 100 items) and the itemset is larger than 10, we cannot detect it without setting our alpha lower than .9. This is an additional argument for employing caution when $n$ is small.

Table 7.1: DOE Factor Levels

<table>
<thead>
<tr>
<th>Factor</th>
<th>Gentle</th>
<th>Moderate</th>
<th>Strict</th>
</tr>
</thead>
<tbody>
<tr>
<td>No. of Items([N])</td>
<td>100</td>
<td>500</td>
<td>1000</td>
</tr>
<tr>
<td>Series Length([L])</td>
<td>100</td>
<td>250</td>
<td>500</td>
</tr>
<tr>
<td>Group Size</td>
<td>2</td>
<td>5</td>
<td>10</td>
</tr>
<tr>
<td>$\alpha$</td>
<td>.85</td>
<td>.9</td>
<td>.95</td>
</tr>
<tr>
<td>Frequency (strength)</td>
<td>.2</td>
<td>.3</td>
<td>.5</td>
</tr>
</tbody>
</table>

The Factor Levels proposed for the Experiment with 1000 replications at each point for a total of $3^6(1000)$ or 729,000 replications
To perform the initial experiment, we generate a group (of group level items) at a given factor level of frequency. The rest of the demand series will be composed of frequencies generated from the uniform distribution (up to the frequency factor level minus some epsilon). We then proceed with the SPEEDUP algorithm and record when items are correctly classified [as an explicit group] in each period. There is an argument to add additional sine-cosine convolutions, but Wold’s Theorem should free us from this. We argue that the algorithm under a tame experimental design before stress-testing it with eclectic noise processes. This is because preliminary evidence suggests tuning of $\alpha$ and the level of Gaussian noise can affect the results significantly.
7.1.6 Pilot Experimental Results

We first define the number of periods in which the correct items were detected in an explicit and sovereign set; this is our response. For instance, if the correct items (and only the correct items) were detected in fifteen out of fifty periods in a given replication, then the response would be fifteen.

Fig. 7.4: The Experimental Response and its resulting distributional properties.
We can see from the Quartile-quartile Plot that the response is non-normal, and cannot be transformed to normal [most likely] since the Box-Cox Ladder of Powers did not successfully normalize the distribution. N-Way nonparametric ANOVA does not exist at this time and so we will first continue by a simple exploratory analysis with boxplots.

Fig. 7.5: Boxplots of Number Correctly Detected vs. Factor Levels
The number of items did not affect the mean of the response much (Figure 7.6, this is encouraging for future work: since we hope the method scales well. Also, on average, items were correctly classified more often when the series length increased Figure 7.5. We hope to see improvement as \( n \) increases for any statistic. Group size did not affect the response much at all Figure 7.5), implying the algorithm can deduce the group sizes[the number of groups where items are moving in separate patterns].

As we set our alpha parameter to be more strict, we classify correctly more on average Figure 7.5). Though, alpha should be approached with caution because high values of alpha and a low number of items causes erroneous classification with probability unity.

Frequency is an interesting artifact here, albeit an artificial one: bearing further inspection. Infrequent item movement in our preliminary experiment correlated to better
detection. One can only speculate why this might be; probably because frequent item movement results in more noise. The experiment was applied both with the single item quantile rule, and the item/period quantile rule pair. Intuitively, the two pass rule should be better, but we find it is less resistant to noise. Inference from the boxplots is clear as \( n \) is extremely large for this experiment \([729,000]\). There is much to do before the method matures, but the experiment is encouraging.

### 7.2 Summary

In summary, we can say the methodology has theoretical backing; and also that the preliminary results demonstrate promise in the accuracy of the spectral similarity metric as a proxy for the Dotproduct. Further, it would seem there is the possibility that we can pick up simultaneous purchases with a few modifications using SPEEDUP and partially ameliorate the frequent set counting problem in the Apriori Algorithm.
BigData: Case Study

In order to fully make sense of the method in a practical scenario, we will perform case studies using the spectral similarity metric.

8.1 Case Study Top-N

In order to determine the scalability of our Top-N and also its eligibility for use on continuous data, we generate a case study using continuous data from an autoregressive model, and allow the Spectral Top-N to compete with the Cosine Top-N.

8.1.1 Methodology

In order to generate the synthetic data, we first define parameters that we believe would be a sufficient stress test. In this case, we generate a data set with ten thousand items and one million customers. We generate the data artificially so that we know which items are most similar apriori. We do this by generating demand series from an autoregressive model as given by:

$$X_t = K + \phi X_{t-1} + \epsilon_t$$  \hspace{1cm} (8.1)

Where $X_t$ is the item demand by customer $t$, epsilon is a random normal variable, and K is chosen arbitrarily because the series are demeaned before processing.
For the first ten items, we generate the demand series from the model with \( \phi = 0.3 \); in items 11:10000, we use \( \phi = 0.7 \). We emphasize that by doing this, the customer buying patterns of items 1:10 and 11:10,000 have slightly different functional forms. We arbitrarily designate item \( N \) as item 1. Therefore, we know apriori that with respect to the first item [the one which we are calculating the Top-N with respect to] the corresponding most similar ten items would be the items numbered 1-10 by definition of our noise generation process. A corollary, is that the most similar item to the first item is the item itself. Since the demand series across the customers is identical inter-item. This experiment is to assess whether or not our method picks up the correct items and/or also outperforms the Cosine measure meaningfully in terms of speed and memory consumption. An important feature to note is that it would not matter whether or not we added a mean \( K \) to these series, because the spectral method will demean prior to calculation.

In order to obtain these results, an instance of R was run on each core of an 8 core machine, and the cumulative runtime and RAM consumption is demonstrated in Table 8.2. Note that Spectral Item No. and Cosine Item No. in the table refer to the item index (or row) which each method determines as the Top-N rank given in the leftmost column.
Table 8.1: Top-N Identified by Method

<table>
<thead>
<tr>
<th>Top-N Rank</th>
<th>Spectral Item No.</th>
<th>Cosine Item No.</th>
</tr>
</thead>
<tbody>
<tr>
<td>1st</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2nd</td>
<td>10</td>
<td>9</td>
</tr>
<tr>
<td>3rd</td>
<td>5</td>
<td>16</td>
</tr>
<tr>
<td>4th</td>
<td>2</td>
<td>2647</td>
</tr>
<tr>
<td>5th</td>
<td>8</td>
<td>1892</td>
</tr>
<tr>
<td>6th</td>
<td>9</td>
<td>796</td>
</tr>
<tr>
<td>7th</td>
<td>6</td>
<td>5646</td>
</tr>
<tr>
<td>8th</td>
<td>3</td>
<td>2995</td>
</tr>
<tr>
<td>9th</td>
<td>7</td>
<td>9783</td>
</tr>
<tr>
<td>10th</td>
<td>4</td>
<td>4</td>
</tr>
</tbody>
</table>

Table 8.2: Run-Time and RAM Consumed

<table>
<thead>
<tr>
<th>Spectral Runtine(seconds)</th>
<th>RAM(GB)</th>
<th>Cosine Runtine(seconds)</th>
<th>RAM(GB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>9620962</td>
<td>84.804</td>
<td>18386838</td>
<td>162.002</td>
</tr>
</tbody>
</table>
Chapter 9

Conclusions, Contributions, and Future Work

In this dissertation, we have proposed an alternative similarity measure [the H-K similarity measure] based on spectral analysis and reviewed the past and recent literature germane to it. We have demonstrated that this measure satisfies the properties of a similarity measure. In numerous experiments on varying sets of data (even those meant to be difficult and nuanced) we have demonstrated that the method has computational and storage benefits compared to other common similarity measures. We have done this primarily using the vehicle of Top-N.

The spectral technique holds promise for reducing the data complexity because there are very fast and well vetted methods for computing Fast Fourier Transform which significantly reduces the complexity in distance matrix calculation. If practitioners are willing to accept a slight loss in accuracy for a large increase in speed and a large decrease in memory footprint, our measure is a good candidate for binary data. The loss in accuracy will be characterized by how well-compressible the data is when undergoing the DFT. In the event the data is continuous, autoregressive, or periodic, the loss in accuracy is expected to be very minimal. If the data resembles purely a noise process, the loss in resolution can be expected to be up to fifty percent. This is firmly conclusive. We can say also that the method is scalable because it has been explored in a multicore capacity: at least as scalable as other similarity measures. The experiments suggest
the computational benefits are retained when scaled. We can say that small amounts of filtering and thus further compression of the series is not necessarily ill advised, but application specific caution should be employed due to a further [but quantifiable] loss in fidelity. The measure performs better when binary data is sparse.

With our brief foray into continuous series, we can say that Spectral analysis is a very effective tool for summarizing continuous series, and this fact aligns with theoretical work on autoregressive spectra as well. If the demand series demonstrate an autoregressive component [as in the case study discussed in Chapter 8], the spectral method retains excellent resolution with an equal amount of compression. In the analysis of continuous data, a method which is as fast or faster than the gold standard for binary data would be an excellent tool in the analyst’s arsenal, we admit further exploration is necessary to validate the claim completely (though the case study is compelling). We extend the measure to the Apriori Algorithm and this extension assists in ameliorating the frequent itemset problem. Notably, it requires the original transaction data to be of very fine resolution. For the practitioner, this is an important consideration.

9.1 Contributions

In a user-item system the complexity to employ Top-N is $O(users^2 \ast items)$ since we have to perform $users(users - 1)$ calculations of possibly $items$ length. The focus of this dissertation at a very high level is merely a method by which to speed up the calculation of the distance matrix. We do this in a fashion which has not been done before: using the frequency domain to compress the existing data and reduce the number of operations required to compute the distance matrix. We reduce the number of
calculations by a factor of two or greater; and this is of practical benefit for computation
time and RAM consumption. Further, we extend this method to the Apriori algorithm
using nonparametric methods in a way that has not been done to quickly detect frequent
item-sets simultaneously provided the data is collected at a resolution that we can qualify
each demand series as a time series.

9.2 Future Work

In future work, more extensive experimentation should be done to determine
whether or not the method is well suited to continuous data. This is no small feat,
since continuous data can be generated in a variety of fashions, but a good start would
be the generation of autoregressive integrated moving average models and testing the
performance there. This is not out of line with what common sense would suggest many
demand series fall into. Also, there are a number of debated methods by which to cal-
culate similarity for continuous data, and so this undertaking again is no small feat. In
addition, the algorithm should be coded into a language which is more effectively par-
allelized and faster. For instance, Fortran would provide tremendous speed advantages
over R, and for commercial applications, this translation would be necessary. Also, it
would be of benefit to see what kind of loss of accuracy would be present if the series were
simply represented by smaller smoothed spectra [or at a smaller number of frequencies] to
further quicken the calculations. The methods by which one could accomplish this task
are also many given the variety of filters and window functions available, and this too
is no small feat. The binary as well as continuous experiments could likely benefit with
the addition of new and more exotic noise generation processes. The method should be
employed in conjunction with singular value decomposition in order to see even greater improvements in the RAM consumption at runtime. This method should be extended to user and item clustering individually. This method should be compared to change-point analysis. This method, it would seem from the literature, may also be applied in some medical fields, such as detecting differences in electrocardiogram or electroencephalogram readings over time, or comparing different sets of these types of sinusoidal data.
Bibliography


Sørensen, T. (1948) {A method of establishing groups of equal amplitude in plant sociology based on similarity of species and its application to analyses of the vegetation on Danish commons}. Biol. skr. 5:1–34.


Vita
Kenneth Hutchison

Education

Dual Ph.D. in Industrial & Manufacturing Engineering &
Operations Research, expected in August 2015
Area of Specialization: Big Data, Statistics, Internet of Things

B.S. in Industrial and Management Engineering
M.S. in Industrial and Systems Engineering

Awards and Honors

Best Undergraduate Paper Award: ISERC 2015 2015
2nd Place PSU Major League Hackathon 2014
Informs Chapter President 2014–2015
Informs Chapter Vice-President 2013–2014
Industrial Engineering Graduate Association Treasurer 2012–2013

Research Experience

Doctoral Research The Pennsylvania State University 2012–2015
Thesis Advisor: Prof. Soundar Kumara
Application of the FFT/Spectral Density in Similarity Calculation

Graduate Research The Pennsylvania State University 2012–2015
Research Advisor: Prof. Soundar Kumara
Social Media scrapin and analytics techniques

Independent Research The Pennsylvania State University 2012–2015
The Doubly Stochastic Poisson Process and ARIMA
Developed a software framework for the Internet of Things
Developed an embedded device framework for the Internet of Things
Developed an Ontological model to reduce shelter euthanization in Baltimore, MD
Developed a Permutation Test for Political Analysis in Jacksonville, Florida

Undergraduate Research Rensselaer 2008–2012
Research Advisor: Dr. Thomas Reed Willemain
Estimating the Intensity function of the Doubly Stochastic Poisson Process
Development of Hip-Replacement Therapies (CAPSTONE)

Teaching Experience

Seminar Lecturer The Pennsylvania State University 2013–2015
Provided a series of lectures and workshops in the R Programming Language at PSU

Research Assistant The Pennsylvania State University 2012–2015
Worked closely with DARPA, MANTECH, and various private companies on various automated assembly problems. Principal Investigator: Dr. Finke