Using Transitivity with Nearest Neighbor to Reduce Error in Sample-Based Pearson Correlation Coefficients

Taylor Phillips
University of Denver

Follow this and additional works at: https://digitalcommons.du.edu/etd

Recommended Citation
https://digitalcommons.du.edu/etd/515

This Thesis is brought to you for free and open access by the Graduate Studies at Digital Commons @ DU. It has been accepted for inclusion in Electronic Theses and Dissertations by an authorized administrator of Digital Commons @ DU. For more information, please contact jennifer.cox@du.edu,dig-commons@du.edu.
Using Transitivity with Nearest Neighbor to Reduce Error in Sample-Based Pearson Product-Moment Correlation Coefficients

A Thesis
Presented to
the Faculty of Engineering and Computer Science
University of Denver

In Partial Fulfillment
of the Requirements for the Degree
Master of Science

by
Taylor Phillips
June 2010
Advisor: Chris GauthierDickey
Abstract

Pearson product-moment correlation coefficients are a well-practiced quantification of linear dependence seen across many fields. When calculating a sample-based correlation coefficient, the accuracy of the estimation is dependent on the quality and quantity of the sample. Like all statistical models, these correlation coefficients can suffer from overfitting, which results in the representation of random error instead of an underlying trend.

In this paper, we discuss how Pearson’s product-moment correlation coefficients can utilize information outside of the two items for which the correlation is being computed. By introducing a relationship with one or more additional items that meet specified criterion, our Transitive Pearson product-moment correlation coefficient can significantly reduce the error, up to over 50%, of certain sparse, sample-based estimations.
## Contents

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

2 Background ................................................. 3

3 Motivation .................................................. 6
   3.1 Pearson Product-Moment Correlation Coefficient ........ 6
   3.2 Overfitting ........................................... 7

4 Algorithms .................................................. 9
   4.1 Heuristics .............................................. 9
   4.2 Transitive PMCC ....................................... 10

5 Experimental Methodology ......................... 12

6 Results ..................................................... 14
   6.1 Heuristics .............................................. 14
   6.2 Transitive PMCC ....................................... 15
   6.3 Error Distributions of PMCC Estimations ............ 18
   6.4 Data Density .......................................... 21

7 Conclusion ................................................ 25

8 Future Work ............................................... 26

9 Appendix A ............................................... 28
1 Introduction

Statistical models are used in the day-to-day lives of modern humans. Alleviating traffic congestion, predicting weather patterns, or investing in the stock market are all common examples of such models. When insufficient quantities of data are used by these models, they exhibit a phenomenon known as overfitting. This overfitting causes the models to display random error instead of an underlying trend, which in turn makes it difficult to utilize the results in a sensible fashion.

In this paper, we propose an algorithm that reduces the effects of overfitting by using information in the data set other than that which the statistical model was built to utilize. For the purpose of discussion, we focus on a particular, ubiquitous example of a statistical model that is susceptible to overfitting known as the Pearson product-moment correlation coefficient (PMCC). This PMCC measures the correlation, or linear dependence, between two vectors, \( i \) and \( j \), and relies solely on the intersection of those two vectors. Our proposed algorithm works by finding transitive neighbors, \( k \), such that the \( k \) are the vectors in the data set most similar to \( i \). These \( k \) are then used to form estimates for \( i \)’s relationship with \( j \), allowing our algorithm to incorporate auxiliary information that is normally disregarded.

The existing approaches to alleviating the effects of overfitting don’t address this issue directly. Instead, such techniques to utilize models that do suffer from overfitting in a specific application. Our work is different because it presents an idea that could be used to
improve estimations of many statistical models using sparse data.

To quantify the performance of the algorithms presented in this paper, the Netflix Prize data set was used. This readily available data consists of approximately 100 million user-movie pairs. The results demonstrate that our Transitive Pearson product-moment correlation coefficient algorithm can reduce the error by up to 50% of PMCC certain approximations in low-density vectors.

The notion of utilizing transitivity in statistical models to reduce the effects of sparse data is both abstract and powerful. The algorithm proposed in this paper is important because it is the first to demonstrate a significant reduction in error of sparse, sample-based PMCC estimations. PMCCs find uses in education, psychology, physics, mathematics, economics, and finance, all of which can suffer from overfitting and can subsequently benefit from the ideas presented in this paper. Further, this notion of neighbor transitivity used by our algorithm could be extended to reduce the error of other statistical models operating on sparse data.

The following section describes the process behind arriving at the ideas presented in this paper. Section 3 provides a description of overfitting and PMCCs while the proposed algorithms are discussed in Section 4. The experimental methodology is outlined in Section 5 and the subsequent results in Section 6. The paper is concluded in Section 7 with future work in Section 8. Lastly, the source code for the Transitive Product-moment correlation coefficient is given in Appendix A.
2 Background

The intention of the work that went into this thesis was to explore and contribute to the growing area known as collaborative filtering. Collaborative filtering (CF) is the process of filtering raw data for information or patterns using techniques involving collaboration among multiple agents, viewpoints, data sources, etc [16]. The first opportunities for collaborative filtering came through the internet where users were able to provide real time feedback on a product or service. Tivo and Amazon were the first to take advantage of it and in 2003 a paper covering an analysis of collaborative filtering in Amazon was published which stated a 20% increase in sales attributed to personalization through collaborative filtering [9]. For a survey of existing collaborative filtering techniques see [2].

The Netflix Prize has drawn a particular amount of buzz to collaborative filtering with articles appearing in the NY Times [15] [11], Wired Magazine [8], The Washington Post [12] and the like. The Netflix Prize [1] contest began in 2006 when the Netflix online, movie rental delivery company offered $1,000,000 to anyone who could improve their proprietary movie recommendation algorithm by some quantified metric. This algorithm was designed to recommend movies to customers based on how a specific customer rated his previously viewed movies. This same task of recommending movies could also be looked as the task of predicting a rating for an unseen movie, and then recommend movies with the highest predicted rating. This means the goal of the Netflix Prize is to improve the
accuracy of the predictions of unseen movies.

The work on this thesis began in 2008 when the Netflix Prize was already two years underway. There was an abundance of discussion happening on the Netflix Prize forums and numerous publications had been made by successful contestants. Robert Bell, Koren Yehuda, and Chris Volinsky from AT&T Labs stood out as leaders and had held the top spot on the leaderboard for the vast majority of the time and boasted several publications which provided a strong framework from which to jumpstart this thesis [4, 13, 10, 6, 5].

These publications discussed various recommendation models. Paper [4] focused on the nearest neighbor model while [10, 6] discuss latent factor models and combining models together. Lastly, [13, 5] provides a synopsis of their work on the Netflix Prize. Although related to our own work, we are more interested in estimating the measures of similarity employed by these various algorithms.

At the time, the AT&T competitors had the highest ranked submission and reported that it used a combination of 107 different sets of predictions from six different models to achieve their best result [13]. In September of 2009, one of Bell, Yehuda, and Volinsky’s teams was awarded the prize money more than three years after the contest began. Based on the huge international success of the Netflix Prize, Netflix has already announced that there will be a second contest in the future.

Our efforts were first focused on improving the quality of predictions. After limited success, our direction changed and we began to search for aspects of CF that could be improved besides the predictions themselves. We looked at reducing the time complexity of the nearest neighbor algorithm by limiting neighbor selections to a well-chosen set of global critics or users who could accurately represent the views of the entire population. Limiting the search for neighbors to only this set could significantly reduce the asymptoptic complexity of the algorithm in general. Several methods of choosing critics were tested, including a highly specific Pagerank algorithm [7] implementation, all of which ultimately yielded unimpressive results.
Our direction changed again when we modified the previous idea to reducing the time complexity of the computation of correlation coefficients. The most widely used correlation coefficient in the Netflix Prize was the Pearson product-moment correlation coefficient (PMCC) \[4, 10, 6, 3, 14\]. Each of these papers either use PMCCs directly as a measure of similarity \[4, 3, 14\] or recommend them as an alternative \[10, 6\]. PMCCs are used as a measure of similarity in numerous applications of many different models, including matrix factorization and the nearested neighbor models. Through several techniques, we attempted to estimate these correlation coefficients and ultimately, this too was not successful.

Our efforts were redirected a final time to improving the accuracy of the estimated correlation coefficients. Like any statistical model, the usefulness of the correlation coefficients is contingent on having sufficient data. Unfortunately for Netflix Prize competitors, having data may seem like a reasonable demand, but in actuality it is the largest impediment in collaborative filtering. In fact, if the complete set of data was made available there would not even be any predictions left to make because all of the true results would already be known. Thus, collaborative filtering could be viewed as the task of correctly predicting the missing data points.

With this in mind, we were able to make significant improvements in estimating sample-based Pearson correlation coefficients when there is very small amounts of data. The Pearson correlation coefficient measures the correlation between two user’s ratings. Thus, to compute the correlation coefficient for two users, an overlap in movies seen is needed to draw any conclusion about the relation between those two users. Our algorithm does not have this requirement and is able to make estimations of correlation coefficients when there is absolutely no data of this kind. This in turn allows models that make use of Pearson correlation coefficients to improve estimations of sample-based data and even make predictions that simply were not possible without this technique.
3 Motivation

Quantifying a relationship between users or items is an important component of collaborative filtering. One relationship that is a common focus of numerous publications is similarity \[4, 10, 6, 3, 14\]. Each of these publications take a different approach to quantifying similarity and using it to weight different opinions. The general trend is the more similarity, the greater the weight. Determining the distribution of weight from similarity is a specific focus in \[10, 6\] which demonstrate how crucial similarity and weighting are when forming approximations from sparse data. There are multiple interpretations of similarity, but one commonly accepted method is correlation (linear or otherwise), which is used or recommended in \[4, 3, 14\].

3.1 Pearson Product-Moment Correlation Coefficient

The Pearson product-moment correlation coefficient (PMCC) is a measure of linear dependence between two vectors, \(i\) and \(j\), in the range of \([-1, 1]\). A PMCC of 1 indicates an exact positive correlation, -1 indicates an exact negative correlation, while 0 indicates there is no linear relationship. The formula for the PMCC of \(i\) and \(j\) can take many forms, one of which is shown in Eq. 1. The PMCC of \(i\) and \(j\), dubbed \(r_{ij}\), is based only on \(i \cap j\) which are the points of data common between both variables. We will refer to \(|i \cap j|\) or \(|r_{ij}|\) as the direct sample size of \(i\) and \(j\).
\[ r_{ij} = \frac{\sum_{x}(i_x - \bar{i})(j_x - \bar{j})}{\sqrt{\sum_{x}(i_x - \bar{i})^2} \sqrt{\sum_{x}(j_x - \bar{j})^2}} \]  

PMCCs are a standard for measuring linear dependence and thus, play a role in many fields ranging from math and statistics to social sciences and psychology. These correlations are, however, limited by the classic phrase that correlation does not equal causation. This means that, for example, although temperature and humidity are negatively correlated, it does not imply that the increase in temperature caused the reduction in humidity. Correlations can still provide insight because they demonstrate that historical data indicates that there simply is a negative correlation, regardless of who caused what.

Other measures of similarity used in collaborative filtering include Euclidean distance and the Cosine similarity. Euclidean distance, defined as \( d(i, j) = \sqrt{\sum_{x=1}^{n} (i_x - j_x)^2} \), finds a natural usage when dealing with spatial proximities. The Cosine similarity finds the angle between vectors just as PMCCs find the slope between vectors and takes the form \( \cos(\theta) = \frac{i^j}{||i|| ||j||} \).

3.2 Overfitting

This versatile PMCC can be limited in practice due to its susceptibility to overfitting. Overfitting, also referred to as inductive bias, is a symptom exhibited by statistical models that causes them to display random error instead of an underlying relationship. This means that a statistical model can indicate a relationship that is not true as a result of insufficient data. PMCCs are often used in sample-based scenarios which can have a small set of data that is not guaranteed to be representative of the theoretical, complete set of data. For example, if a statistical model relies on a single point of data, that point could be an outlier causing the model to predict incorrectly.

An alternative way of understanding overfitting is rooted in the law of large numbers. The law of large numbers states that the more data points that exist for a random variable, the more likely that data is to be representative of the expected value of that random variable. Rolling a single dice has six possible outcomes or values, all of which are equally
likely. Because each outcome is equally likely, the expected value can be computed as the average of all outcomes, which for this example is \((1 + 2 + 3 + 4 + 5 + 6)/6 = 3.5\). If the dice is rolled only once yielding a one, there will be significant error if that single roll is assumed to be indicative of all possible rolls. Furthermore, statisticians are able to provide a confidence interval using the law of large numbers. That is, a sample size and interval can be specified such that the sample mean will fall within the specified interval the desired percentage of the time. The desired percentage can be made small or large enough to hold up in court today, such as cryptography used in electronic banking transactions.

This idea can be applied analogously to the Netflix Prize where a given user’s rating is the random variable in question. If we have only one rating, that rating is not necessarily representative of the long term opinions of that user. The law of large numbers states that the more ratings we have, the more likely that data will be representative of the long term. Thus, when dealing with an incomplete set of data, as in collaborative filtering, it is important to understand and account for this overfitting.

Various heuristics exist to curb overfitting and arrive at a more conservative estimate. This can be beneficial, or even necessary in some situations, but the ability of these techniques to significantly improve the estimations is limited. Such heuristics can be as simple as skewing the original, overfitted value towards the mean of all values. In this paper, we propose an algorithm that minimizes the effects of overfitting of sample-based PMCCs by using information other than \(i \cap j\). The myriad of applications of PMCCs can also suffer from overfitting and can subsequently benefit from the ideas presented in this paper.
4 Algorithms

In this section we describe two heuristics and our Transitive PMCC algorithm. For the following sections we assume there is a universe of vectors, for which a PMCC could be computed between any two vectors using Eq. 1. The algorithms will operate on some original PMCC, $r_{ij}$, and yield a new PMCC, $r'_{ij}$, that is intended to replace the $r_{ij}$ for all subsequent applications.

4.1 Heuristics

The two heuristics presented in this section dampen the effects of overfitting by reducing the reliance on the data specific to $r_{ij}$. This is done by using a linear combination of the original $r_{ij}$ and some given constant. The first heuristic algorithm, HeuristicA, takes two constants $\alpha$ and $C$ where $\alpha$ is the linear combination weight given to $r_{ij}$ and $1 - \alpha$ is given to $C$. For our purposes we choose $C = 0$, indicating that the more weight $C$ gets, the more it would transform $r_{ij}$ to zero which, for PMCCs, means that there is no linearly dependent relationship between $i$ and $j$. This choice of $C$ curbs overfitting by skewing the actual $r_{ij}$ towards this conservative value. HeuristicA is described formally in Eq. 2.

$$HeuristicA_{ij} = \alpha r_{ij} + (1 - \alpha)C = \alpha r_{ij} \quad \text{for } C = 0$$

Note that $\alpha$ is not dependent on anything and thus, the linear combination weight given to $r_{ij}$ is fixed. The problem with HeuristicA is that regardless of the direct sample size of $r_{ij}$,
the linear combination weight remains fixed. Thus, an \( r_{ij} \) with a very large direct sample size would receive exactly \( \alpha \) weight just as an \( r_{ij} \) with a very small direct sample size. An improvement can be made by having the linear combination weight of \( r_{ij} \) be a function of the direct sample size of \( r_{ij} \). This means that when there is a smaller direct sample size \( r_{ij} \) will get less weight, but as direct sample size increases \( r_{ij} \) gets more weight. This is useful because as direct sample size increases, the effects of overfitting ought to decrease and thus, the original value can be weighted more heavily.

HeuristicB uses \(|r_{ij}|\) to arrive at a weight for \( r_{ij} \) that is more appropriate for the specific pair of \( i \) and \( j \). It is described in Eq. [3] where \( \beta \) is the linear combination weight of \( r_{ij} \) and \( C \) is a chosen constant. As with HeuristicA, \( C = 0 \) was chosen so that the linear combination would be skewed towards 0, the equivalent of no relationship. For \( \beta = 5 \) with a direct sample size of 100, HeuristicB gives \( r_{ij} \) 95% of the linear combination weight and only 5% to no relationship. If the direct sample size were only 5, \( r_{ij} \) would receive only 50% of the weight while no relationship would also get 50%. By determining a linear combination weight for \( r_{ij} \) from the direct sample size of \( r_{ij} \), HeuristicB incorporates the conservative estimate with small amounts of data, but has little impact when there is larger amounts.

\[
HeuristicB_{ij} = r_{ij} \frac{|r_{ij}|}{|r_{ij}| + \beta} + (1 - \frac{|r_{ij}|}{|r_{ij}| + \beta})C = r_{ij} \frac{|r_{ij}|}{|r_{ij}| + \beta} \text{ for } C = 0 \tag{3}
\]

4.2 Transitive PMCC

Our proposed Transitive PMCC algorithm (TPMCC) works to find information beyond \( i \cap j \) to develop a stronger estimate for \( r'_{ij} \). This extra information is rooted in the neighbors that are chosen to represent \( i \)'s relationship with \( j \). That is, the TPMCC algorithm takes the items most similar to \( i \) and uses their relationships with \( j \) to estimate \( i \)'s relationship with \( j \). To determine an ordering of neighbors for \( i \) by similarity, we use \( \text{abs}(r_{ik}) \), such that
neighbors most similar will have a strong correlation. This strong linear dependence can be either positive or negative denoted by the absolute value.

The process of selecting a set of neighbors for a given pair, \(i\) and \(j\), begins by examining all possible neighbor candidates, \(k\). The candidates are then narrowed down, keeping only those whose \(\text{abs}(r_{ik}) > \delta\) for which \(\delta\) is some chosen constant. Additionally, we want to require some sufficient direct sample size on \(r_{ik}\) and \(r_{kj}\) so that we can have a degree of certainty that the neighbors themselves aren’t suffering from overfitting. These constraints take the form of \(|r_{ik}| \geq \gamma_{ik}\) and \(|r_{kj}| \geq \gamma_{kj}\). We then take our final neighbor set \(N_{ij}\), as the set of all \(k\)s that meet the previously stated criterion with \(\delta\), \(\gamma_{ik}\), and \(\gamma_{kj}\). The number of neighbors in \(N_{ij}\) will be referred to as the transitive sample size. The Transitive PMCC algorithm is then described in Eq. 4 where \(w(i, j)\) is the weight of the actual \(r_{ij}\) and \(w(i, j, k)\) is the weight of neighbor \(k\).

\[
TPMCC_{ij} = \frac{r_{ij}w(i, j) + \sum_{k \in N_{ij}} r_{kj}w(i, j, k)}{w(i, j) + \sum_{k \in N_{ij}} w(i, j, k)}
\]

In order to examine all possible neighbor candidates, every unique PMCC must be computed. This step alone has an asymptotic complexity of \(O(n^2)\) in running time where \(n\) is the number of vectors in the universe. The TPMCC algorithm then examines all \(n - 2\) neighbor candidates for each of the \(O(n^2)\) unique PMCCs, making the asymptotic complexity of the Transitive PMCC algorithm \(O(n^3)\). This is somewhat alleviated by being trivially parallelizable, but the cubic complexity must be considered. Because PMCCs and similar types of statistical analysis are used in education, psychology, physics, mathematics, economics, and finance, which can also suffer from overfitting and can subsequently benefit from the ideas presented in this paper.
5 Experimental Methodology

The Netflix Prize data set was used to experimentally measure the performance of the heuristics and TPMCC algorithm presented in the previous section. The Netflix Prize data set contains rating history for 480,189 users, 17,770 movies, and 100,480,507 ratings. The movies data consists of a title, release year, and a unique identifier while the users consist of a unique identifier only. Lastly, the ratings data consists of a unique user identifier, a unique movie identifier, date of the rating, and a value of the rating ranging from one to five.

One hundred million ratings may appear substantial, but it only represents 1% of the total possible ratings. That is, if every user rated every movie then every possible rating would already be known, while in actuality 99% of those ratings are missing. This missing data complicates the use of PMCCs as they are based on only a subset of the possible data. Thus, the goal is to compute the PMCCs of the complete set of data using only a subset of the data. These sets could be thought of as a grading set and training set respectively. Below, we will first thoroughly examine the effects of the TPMCC training on 50%. Subsequently, we will discuss how different amounts of training data influence the results.

For our purposes, the only points of data used were the rating’s unique user identifier, the unique movie identifier, and the rating value. Using only one random half of the ratings data the PMCCs were computed for all pairs of movies. With 17,770 movies this results in
157,877,565 unique pairs of movies, each with their own PMCC. The PMCC for movies $i$ and $j$ in this set will be denoted $Original_{ij}$. Another set of PMCCs, $Final$, was computed using the entire set of ratings data and is used to grade the accuracy of the $Original$ PMCCs and the PMCCs created by the algorithms.

The $Original$ PMCCs will be used by the heuristics and TPMCC algorithm as input to provide new estimates for the $Final$ PMCCs. To quantify the error between any two sets of PMCCs, we use the root mean-squared error (RMSE). The formula for RMSE is shown in Eq. 5 where $a$ and $b$ are sets of PMCCs of size $n$.

$$RMSE = \sqrt{\frac{1}{n} \sum_{x=1}^{n} (a_x - b_x)^2}$$ (5)

The resulting RMSE between $Original$ and $Final$ is 0.468. Theoretically, the worst possible RMSE could be 2.0. This would happen if, for example, the $Final$ PMCCs were all 1 and all of the $Original$ PMCCs were -1. However, given a distribution of data and predicting the mean yields much lower measures of error in practice. For example, the RMSE of absolutely no data, which is predicting 0 for every PMCC, yields an RMSE of 0.542. This means that using the $Original$ PMCCs computed using half of the data only reduced the error of predicting 0 for all PMCCs by 13.7%.
6 Results

6.1 Heuristics

Each heuristic was run using the Original PMCCs as input yielding two new sets of PMCCS, HeuristicA and HeuristicB. A plot of the RMSE of HeuristicA is shown in Figure 1 for different values of $\alpha$. In this plot it is visible that the RMSE of HeuristicA is minimized for $\alpha = 0.6$, which reduced the RMSE to $0.425$ - a $9.1\%$ reduction of the RMSE of Original. The value of $\alpha$ that achieved the lowest RMSE is between 0 and 1, indicating that Original does suffer from overfitting and benefits from the HeuristicA algorithm. If $\alpha = 0$ or $\alpha = 1$ yielded the least RMSE, it would mean that predicting 0 for all PMCCs was best or using the unmodified HeuristicA was best, respectively. When $\alpha = 0.6$, HeuristicA is going to scale the Original PMCCs down to $60\%$ of the linear combination weight and give $40\%$ to 0. The contrast with the effects of HeuristicA on the PMCCs produced by the TPMCC algorithm as also shown in Figure 1 will be discussed in the following section.

The RMSE of the PMCCs of the HeuristicB algorithm were computed for various values of $\beta$ and displayed in Figure 2. Note that choosing $\beta = 0$ results in no change to Original and $\beta = \infty$ would result in a prediction of 0 for all PMCCs. The RMSE for HeuristicB was minimized using $\beta = 2$, which achieved a total reduction of RMSE of nearly $9.8\%$ over Original. Like HeuristicA, this demonstrates that HeuristicB does reduce the RMSE of the
PMCCs indicating that Original suffers from overfitting. This value of $\beta$ means that PMCCs with a direct sample size of 2 were reduced to 50% of their value, while PMCCs with a direct sample size of 20 were reduced to only 90.1% of their original value. The contrast with the effects of HeuristicB on the PMCCs produced by the TPMCC algorithm as also shown in [2] will be discussed in the following section.

6.2 Transitive PMCC

Multiple sets of PMCCs were computed with the proposed TPMCC algorithm using Original as input. The different sets were computed with different constraints on the neighbor sets. We chose a fixed $\delta = 0.9$ and $\gamma_{kj} = 1$, but used multiple values of $\gamma_{ik}$ ranging from 3, 6, 12, and 24. This means that the neighbors for the Original$_{ij}$ were limited to $ks$ such that $\text{abs}(\text{Original}_{ik}) \geq 0.9$, the direct sample size of Original$_{kj}$ is greater than zero and the direct sample size of Original$_{ik}$ ranged from greater than or equal to 3, 6, 12, and 24. Our implementation was in Java and the computation was performed in parallel.
Figure 2. RMSE of HeuristicB with Original and Transitive vs. $\beta$. Note that HeuristicB exhibits the lowest RMSE with Original when $\beta$ is 2 yielding a 9.8% improvement of the RMSE over Original.

on four machines. The machines had 4GB of RAM, 2.13GHz Intel Core 2 CPU and were running Debian GNU/linux 2.6.18. Depending on the value for $\gamma_{ik}$ (which determined the size of the neighbor sets), the entire operation would take five to eight hours. In contrast, a standard pearson calculation could be done on a single machine in less than an hour.

The RMSEs for the different values of $\gamma_{ik}$ are shown in Figure 3. The second axis of the figure displays the average number of transitive and direct neighbors of $i$ and $k$ for each $\gamma_{ik}$. Note that with $\gamma_{ik} = 24$ it was difficult to even find a large number of direct neighbors and thus, didn’t have a significant impact on the data. Both the improvements from $\gamma_{ik} = 24$ to $\gamma_{ik} = 12$ and $\gamma_{ik} = 12$ to $\gamma_{ik} = 6$ were substantial, while the change form $\gamma_{ik} = 6$ to $\gamma_{ik} = 3$ had little impact. This shows that direct sample sizes like 6 and 12 held a strong balance between attainability and usefulness. Neighbors that only have a very small direct sample size are less reliable because such a small direct sample size could easily misrepresent the complete set of data, however, they were still able to make a
positive contribution to reducing the overall RMSE. The TPMCC algorithm is minimized for $\gamma_{ik} = 3$ with nearly 1300 transitive neighbors, which reduces the RMSE of Original to 0.28, a 40.1% reduction in RMSE. This set of PMCCs, denoted Transitive, will be used in subsequent comparisons to other sets of PMCCs.

Looking back at Figures 1 and 2, both plots also display the results of each heuristic algorithm on the PMCCs produced by TPMCC. In these figures Transitive is minimized by the heuristic algorithms when they don’t effect them at all - namely $\alpha = 1$ and $\beta = 0$ for HeuristicA and HeuristicB respectively. As dicussed in the above subsection, these values of $\alpha$ and $\beta$ have no effect on Transitive, and further, the RMSE gets progressively worse as the heuristics make a larger impact. This is directly indicative that Transitive, unlike Original, already accounts for overfitting and is only made worse by the heuristics.
6.3 Error Distributions of PMCC Estimations

The distribution of error from each algorithm’s PMCCs, including \textit{Original}, are shown in Figure 4. The plot was built by computing the absolute value of the error and counting the frequency of errors falling into each bucket. The buckets have a lower and upper threshold, all of which were chosen to have width 0.1 and range from 0 to 2. A particular error value falls into the first bucket for which the error is less than that bucket’s upper threshold. We will refer to the first bucket, containing values ranging from 0 to 0.1, as the “0.1 bucket” and all subsequent buckets will be denoted by their upper threshold.

In Figure 4, \textit{Original} has the largest error of any other set of PMCCs. The \textit{Transitive} PMCCs contain the most values in the 0.1 bucket with over 30% of all PMCCs falling into this category. Both heuristic PMCC sets are close behind, while \textit{Original} has only 25%. In the next two buckets, \textit{Original} and both heuristics differ slightly, but TPMCC has about 5% more. In addition, \textit{Transitive} is the only set of PMCCs to have any significant effect on
Figure 5. Distribution of RMSE by direct sample sizes of PMCCs where the X axis represents upper threshold (inclusive). Note that Transitive exhibits significantly lower RMSEs for direct sample sizes less than or equal to 8. After this point Transitive has higher RMSEs demonstrating that the transitive data becomes less valuable as direct sample size increases.

...buckets 0.7 to 1, which each contain roughly 5% of all other sets of PMCCs. Transitive has much less, emphasizing the fact that it has much fewer high-error PMCCs. These buckets are likely populated by PMCCs that have a very small direct sample size which results in overfitting and high error. TPMCC’s performance in this situation is indicative that the it is doing more than curing the symptoms of overfitting, but actually using the extra information to improve estimations.

To further examine the PMCCs and understand the implications of Figure 4, a second distribution was made to show the RMSE for different direct sample sizes of $i$ and $j$ in Figure 5. Like Figure 4 this distribution was sampled using thresholds and each bucket shall be denoted by its upper threshold, where the first bucket contains only those PMCCs who had a direct sample size of zero. The remaining buckets have exponential widths ranging from the previous buckets upper threshold (exclusive) to its own upper threshold (inclusive).
The PMCCs for all algorithms, excluding *Transitive*, are identical for the first two buckets, 0 and 1, as they had no data on which to operate and predicted 0. The TPMCC algorithm was able to produce PMCCs that reduced this error in bucket 0 by over 31.2%. For bucket 1, *Transitive* further improves and demonstrates its ability to operate with little direct data and reduces the error of all other algorithms by 41.4%. *Original* doesn’t improve much in bucket 2, but both heuristics show a drastic change and reduce the RMSE of *Original* by 17%. The *Transitive* continues to improve and reduces the RMSE of *Original* for the bucket 2 by 50.4%. These contrasting results demonstrate the ability of the TPMCC algorithm to extract indirect information from transitive neighbors and improve the accuracy of predictions with limited amounts of data.

The *Transitive* continues to outperform all other algorithms by a similarly significant margins up to bucket 8. For buckets larger than 8, a new trend develops and *Transitive* begins to have a larger RMSE than the other algorithms. This interesting behavior implies that there exists a direct sample size at which point enough direct information renders the transitive neighbor information detrimental. This is somewhat intuitive as the larger the direct sample size that is available, the more trust that can be placed on the subsequent results. Thus, when the results are sufficiently trusted, the *Transitive* uses less accurate and indirect information from transitive neighbors that actually increases the RMSE of the PMCC estimations.

To gain insight as to how significantly Figure 5 will impact the overall RMSE, a third distribution was made. This distribution is shown in Figure 6 and displays the percent of all PMCCs to fall in each of the buckets used in Figure 5. It shows that roughly 7% of all PMCCs have a direct sample size of 0. The thresholds with direct sample sizes from 0 to 8 account for 67.9% of all data and *Transitive* was able to reduce the RMSE by 42.7%. In addition, the thresholds where *Transitive* is detrimental, direct sample sizes of with thresholds 64 and greater, all combine to make up only 12.9% of all unique pairs.
Figure 6. Distribution of all PMCCs by direct sample sizes where X axis represents upper threshold. Note that samples sizes 0 through 8, where TPMCC performs well, account for 67.9% of all PMCCs. The TPMCC has a negative impact on buckets with direct sample sizes of 64 and greater, which combine to only 12.9%.

6.4 Data Density

As discussed in Section 5, the Netflix Prize data set has approximately 1% density. This means for the results presented above, the algorithms operated on only 0.5% data density. The following analysis addresses how the different algorithms perform as the amount of data is reduced.

The plot in Figure 7 shows a RMSE of different sets of PMCCs as the amount of data is varied. \textit{OrigHeuristicA} and \textit{OrigHeuristicB} are the PMCCs resulting from using \textit{Original} with each heuristic. \textit{Transitive} is the results from TPMCC algorithm while \textit{TransHeuristicA} and \textit{TransHeuristicB} are from the heuristics operating on \textit{Transitive}. With 0% of the data, all algorithms produce the same set of PMCCs which amounts to predicting 0 for all PMCCs. The 2% sample points, or 0.02% data density, show little improvement because the data is still too sparse to support sufficient intersection between movies. This is confirmed in Figure 8 that displays the mean number of transitive neighbors found by TPMCC.
and the number of direct neighbors for normal PMCCs. For the 2% predictions, it is clear that neither transitive nor direct neighbors exist in usable quantities.

The next point with 0.10% data density shows an interesting trend. With nearly 100 transitive neighbors, Transitive reduces the RMSE of Original, by 12.5%. The heuristics for Transitive fail to make additional improvement. Original and its heuristics actually perform slightly worse than predicting 0 for all PMCCs. This demonstrates that such sparse data causes overfitting, and in this case, can actually be improved by not utilizing the data at all.

At 25%, or 0.25% data density, Original is still being out performed by predicting 0 for all PMCCs. Its heuristics do slightly better, but Transitive is able to achieve a 34.8% reduction in RMSE over Original. Back to 50% of the Netflix Prize data, Original makes a drastic improvement with a mean intersection size of 51. Transitive continues to improve reaching a 40.1% reduction over Original. Note that the heuristics operating on Transitive are always only made worse, demonstrating that the TPMCC algorithm has already reduced overfitting beyond the aid of the heuristics.

It is interesting to note that the Transitive PMCC algorithm is able to begin reducing RMSE with only 100 transitive neighbors as shown in the 0.1% data density point in the plots above. However, with 0.5% data density, it further benefits from over 1000 neighbors. This means that the Transitive algorithm benefits from being in a wide data set, or a data set that has lots of users and movies. If there were only 10 movies, it would be very difficult for the Transitive algorithm to find a sufficient number of neighbors. With the Netflix Prize data, there are almost 20,000 movies along with half a million users which gives plenty of opportunities to find different neighbors, transitive or not.
Figure 7. RMSE vs. percent of Netflix Prize data used, *Transitive* and both heuristics for each. Note that *Original* actually increases RMSE as amount of data increases until 50% data while *Transitive* is able to make increasingly notable improvements starting with 10% data.
Figure 8. Mean direct sample size and mean transitive sample size vs. percent of Netflix Prize data used. Note that neither reach usable quantities for 2% data density, but at 10% the mean number of transitive neighbors reaches 50, while mean number of direct neighbors only reaches just over 2. At 50% data there are nearly 1300 and 50 mean transitive and direct neighbors respectively.
7 Conclusion

The proposed nearest neighbor PMCC algorithm increases the accuracy of PMCCs estimations when dealing with sparse, sample-based data. In such sample-based data, statistical models can suffer from the lack of data and represent random error instead of underlying trends in a phenomenon known as overfitting. The results of the experiments with the Netflix Prize data demonstrate that the proposed heuristics and TPMCC algorithm are able to reduce the error in such PMCC estimations.

The PMCCs computed from the random test set reduced the error of predicting 0 for all PMCCs by only 13.7%. The heuristics reduced the error of the test set PMCCs by up to 9.8%, while the TPMCC algorithm was able to achieve a 40.1% reduction. For pearson estimates with direct sample sizes of two, which account for 13.6% of the population, the TPMCC reduced the error by over 50%. Lastly, the TPMCC algorithm is able to provide comparable improvements with reduced amounts of data, however, there will be This reduction in error of PMCCs will strengthen the variety of applications in which they are applied and allow statistical models to be utilized in situations where they otherwise could not. Furthermore, the abstract notion of gathering information from transitive neighbors is likely to have a positive effect in new applications.
8 Future Work

The results with the Netflix Prize data establish our TPMCC algorithm as a proof-of-concept. Compared to those topping the charts of the Netflix Prize [4, 10, 6, 8, 14], the nearest neighbor portion of the TPMCC algorithm is relatively barebones. It could be improved by tweaking the restrictions placed on the input parameters $\delta$, $\gamma_{ik}$, and $\gamma_{kj}$. For example, if the data were more dense it would contain more neighbor candidates which could permit for a more restrictive neighbor selection. The functions used to weight the neighbor estimates, $w(i, j)$ and $w(i, j, k)$, can also be improved. Such weighting methods are discussed in detail in [10, 6, 14].

The use of the TPMCC algorithm on PMCCs can be extended to improve other aspects of a collaborative filtering system. First, measures of similarity other than a PMCC could be examined like the Jaccard index, Euclidean distance, and Spearman rank coefficient. Also, the use of transitivity in the TPMCC algorithm could extend to other applications that rely on sample based data.

The TPMCC’s temporal computational complexity is cubic, requiring an $O(n)$ operation for each of the $O(n^2)$ unique PMCCs. This running time can be reduced to $O(kn^2)$ by selecting some well-chosen subset of size $k$ to represent all possible neighbor candidates. Furthermore, if the TPMCC algorithm was computed only for a subset of $c$ PMCCs, those that are likely to benefit the most (e.g. those with a very small sample size), it could be
reduced to $O(ckn)$ which could make it much more pragmatic in real life situations.
9 Appendix A

This appendix contains the Java 5.0 source code of the Transitive Pearson product-moment correlation coefficient algorithm used in this paper. The algorithm assumes a class, PearsonsData, that provides an interface to query for the PMCC or intersection size of any unique $i, j$ pair. The algorithm also requires three input parameters for neighbor selection: minPearson, minIntersectIK, and minIntersectKJ. minPearson is $\delta$, the required strength of the unsigned correlation. The $\gamma_{ik}$ and $\gamma_{kj}$ are represented by minIntersectIK and minIntersectKJ, which places intersection size restriction on neighbor candidates. The configuration that achieved the lowest RMSE discussed in this paper is provided as the default values in the class.

```java
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import cf.MyTimer;

public class TransitivePearson {

    // Constants
    final int NUM_MOVIES = 17770;
    final int NUM_UNIQUE_PEARSONS = NUM_MOVIES * (NUM_MOVIES - 1) / 2;

    // Members
    float minPearson = 0.9 // pearson between I and K (delta)
    int minIntersectIK = 3; // sample size of I and K (gamma_ik)
    int minIntersectKJ = 1; // sample size of J and K (gamma_kj)

```
```java
public TransitivePearson(float minPearson, int minIntersectIK, int minIntersectKJ) {
    this.minPearson = minPearson;
    this.minIntersectIK = minIntersectIK;
    this.minIntersectKJ = minIntersectKJ;
}

/**
 * @param filename - path to store output
 * @param numGroups - number of groups (used for parallelization)
 * @param group - current group (used for parallelization)
 */
public void compute(String filename, int numGroups, int group) {
    System.out.println("Computing transitive pearson "+ filename + ", group " + group + " of " + numGroups);

    MyTimer t = new MyTimer();
    File file = new File(filename + ".part" + group);
    FileWriter fos = null;
    BufferedWriter bos = null;

    // determine starting and ending group
    int start = 1;
    int end = 1;
    int currentGroup = 0;
    int numPearsonsInGroup = 0;
    for(int i=1; i <= NUM_MOVIES; i++) {
        numPearsonsInGroup += (end * (end - 1) / 2) - ((end - 1) * (end - 2) / 2);
        if(numPearsonsInGroup >= NUM_UNIQUE_PEARSONS / numGroups || i == NUM_MOVIES) {
            if(currentGroup == group)
                break;
            else {
                currentGroup++;
                numPearsonsInGroup = 0;
                start = end + 1;
            }
        }
        end++;
    }

    try {
        fos = new FileWriter(file);
        bos = new BufferedWriter(fos);
        double avgNeighbors = 0;
        double count = 0;
        // loop through each unique pearson
        for(int i=start; i <= end; i++) {
            for(int j=1; j < i; j++) {
                double newPearson = PearsonsData.getPearson(i, j) * PearsonsData.getIntersection(i, j);
                double totalWeight = PearsonsData.getIntersection(i, j);

                // loop through all possible neighbor candidates
            }
        }
    }
```
for(int k=1; k <= NUM_MOVIES; k++) {

    // filter neighbor candidates
    if(k == i || k == j) continue;
    if(Math.abs(PearsonsData.getPearson(i, k)) < minPearson) continue;
    if(PearsonsData.getIntersection(i, k) < minIntersectIK) continue;
    if(PearsonsData.getIntersection(j, k) < minIntersectKJ) continue;

    // incorporate valid neighbor
    float trans = PearsonsData.getPearson(j, k);
    float weight = Math.min(PearsonsData.getIntersection(i, k), PearsonsData.getIntersection(j, k));
    newPearson += trans * weight;
    totalWeight += weight;
    avgNeighbors++;

    // k
    count++;
    if(totalWeight == 0)
        newPearson = 0;
    else
        newPearson /= totalWeight;

    bos.write(i + "," + j + "," + newPearson + "," + (int)totalWeight + "\n");

} // j
} // i

avgNeighbors /= count;
System.out.println("Finished transpearson " + filename + " - " + t.getElapsedTime());
System.out.println("Avg. neighbors " + avgNeighbors + " for " + numPearsonsInGroup + " unique pearsons");
bos.close();
} catch (IOException e) {
    e.printStackTrace();
}
References


