Original Research

Presentation of Algorithm Using SVD Technique to Predict Diseases

Shabnam Zarghami[1], Gholam Hassan Shirdel[2], Mojtaba Ghanbari[3], Mohammad Reza Eskandari[4]

1. Ph.D. Student, Department of Mathematics, University of Qom, Qom, Iran. **Orcid:** 0000-0001-9409-9512
2. Associate Professor, Department of Mathematics, University of Qom, Qom, Iran. **Orcid:** 0000-0003-2759-4606
3. Assistant professor, Department of Mathematics, Farahan branch, Islamic Azad University, Farahan, Iran. **Orcid:** 0001-0001-5874-4182
4. Neurology and Psychiatry Subspecialist, Harvard University, USA. **Orcid:** 0000-0003-3935-073x.

***Corresponding Author: Gholam Hassan Shirdel.** Associate Professor, Department of Mathematics, University of Qom, Qom, Iran. **Email:** shirdel81math@gmail.com

Abstract

Background: Data mining, it is considered as knowledge discovery in data science, is the technique for patterns discovery and other valuable data from huge sets. Due to the evolution of data storage technology and the growth of big data, the use of data mining techniques has increased dramatically in the last two decades. The purpose of data mining is to transform the raw data of organizations into useful knowledge. They express the final data set and predicting the outcomes utilizing machine learning techniques. These approaches are utilized to supply data like the fraud detection and user performance, bottlenecks and even security problems.

Methods: In the current study, after preparing data, disease prediction is done utilizing large matrix and data mining approaches. By investigating the new vector, it can be find out which diseases of matrix is near to this one with new signs employing the matrix rows to classify it. The study is descriptive-analytical approach which can be applicable in medical and engineering.

Results: In this research, we implemented data mining techniques using Python software to predict brain and nerve diseases.

Conclusion: The technique used by Python software, the doctor enters the symptoms of the patient and the output of the program indicates 3 diseases close to the input signs for each meter, and ultimately all the meters are evaluated and the meter that has a weaker outcome is considred each time it is run. The priority of each of these meters are expressed in the article and resenting the algorithm employing the svd approach to predict diseases that decrease the disease duration.

Keywords: Prediction of Neurological Diseases Treatment, Treatment Methods, Diseases, Data Mining, Using svd Technique

Submitted: 22 Sep 2024 Revised: 27 Oct 2024 Accepted: 16 Nov 2024

Introduction

Data mining analyzes databases and large datasets to discover and extract valuable information. Such studies and explorations can be considered as an extension and continuity of ancient knowledge and intertwined statistics (1-3). The major difference lies in the scale, scope, and diversity of fields and applications, as well as the dimensions and sizes of today's data, where machine learning methods related to learning, modeling, and training are used in computer science to discover patterns among data, usually raw and often meaningless data enters the system and after necessary processing, results are extracted from the data, which are called information. General applications of data mining in computer science include:

Discovering patterns among data

Approximate prediction of results

Obtaining practical information focusing on big data

Data mining refers to a set of applicable methods on large and complex databases to discover hidden and interesting patterns among data. Data mining methods are almost always computationally expensive. The interdisciplinary science of data mining revolves around tools, methodologies, and theories used to disclose existing patterns in data and is considered a fundamental step towards discovering knowledge. There are various reasons why data mining has become such an important area of study. Some of these reasons are outlined below.[4]

1. Explosive growth of data in a wide range of industries and universities supported by:

Storage devices becoming cheaper and unlimited in capacity, such as cloud storage spaces

Faster communications with higher connection speeds

Better database management systems and software support

2. Rapidly increasing computational processing power

With such a high volume and variety of available data, data mining methods help extract information from data. In this regard, Jiawei Han, a data scientist and author of the book "Data Mining: Concepts and Techniques," says:

"As a result, the data collected in databases have been transformed into data tombs... The widening gap between data and information necessitates the systematic development of data mining tools that can turn data tombs into gold nuggets."

Data mining methods come in various types, ranging from regression to complex pattern detection methods with high computational costs rooted in computer science. The main goal of learning methods (data mining) is to make predictions. However, this is not the only goal of data mining. Data mining methods are used in the long process of research and product development. Therefore, the evolution of data mining began when business data started to be stored on computers. Data mining allows users to navigate through data in real time. Data mining is used in the business community because it utilizes three mature technologies. These technologies include:

Mass Data Collection

With powerful multi-processor computers, the growth and increasing attention to data mining algorithms have always raised the question "Why data mining?". In response to this question, it must be stated that data mining has many applications. Thus, it is considered a young and promising field for the current generation. This field has managed to attract a lot of attention to information industries and societies. Despite the wide range of data available, there is an absolute need to convert such data into information and knowledge.

Therefore, humans use information and knowledge for a wide range of applications, from market analysis to disease diagnosis, fraud detection, and stock price prediction. In summary, it can be stated that the English proverb "Necessity is the mother of invention" applies to data mining, which is used for automating processes and making predictions in large databases. Questions that require extensive analysis can now be answered using data analysis. Targeted marketing is a prime example of predictive marketing. Additionally, data mining is used for targeted and optimized advertising emails. In fact, data mining is used to maximize returns on investment in sending advertising emails. Another predictive issue is bankruptcy prediction. Identifying segments of society that may show similar reactions to an

event is another capability of data mining. Data mining tools are used to examine databases. It is also useful for identifying patterns of previously unknown data. A very good example of pattern exploration is the analysis of retail sales data. This task is performed to identify unrelated products that are usually purchased together. Moreover, there are other pattern mining issues, such as identifying fraudulent transactions in credit cards. In such cases, unknown and new data patterns can indicate the occurrence of credit card information theft and other types of fraud.

Scientific Data

Across the globe, various communities are collecting massive amounts of scientific data. This scientific data needs analysis. This is while there is always a need for more rapid registration of new data. Data mining in various scientific fields helps analyze data and discover knowledge from them.

Personal and Medical Data

Data, from personal to public and from individual to governmental, can be collected for various purposes and analyzed. These data are needed for different individuals and groups, and when collected, extracting information from them can unveil important issues. Among personal data, one can refer to individuals' banking transaction information or their medical records. Data mining plays a significant role in prevention, discovery, and even treatment of diseases in medical data.

-Surveillance Images and Videos

With the decrease in the price of cameras and the existence of cameras in smartphones, a large volume of multimedia data is generated every moment. At the same time, a large volume of images and videos is also collected by surveillance cameras. These data can be used for various data analysis purposes.

-Sports Competitions

There is a vast amount of data and statistics surrounding sports competitions that can be collected and analyzed. Among these, one can mention game information and player statistics.

-Digital Media

There are many reasons for the explosion of digital data repositories. These include affordable scanners, desktop video cameras, and digital cameras. At the same time, large companies such as NHL and NBA have begun the process of converting their collections into digital data, highlighting the need for analyzing massive amounts of data.

Virtual Worlds

There are numerous computer-aided design systems for architects. These systems are used to generate massive amounts of data. Additionally, software engineering data can be used as a source of data along with abundant codes for various purposes.

Virtual Universes

Today, many applications use three-dimensional virtual spaces. Moreover, these spaces and the objects within them need to be described with specific languages, such as Virtual Reality Modeling Language.

Reports and Text Documents

Communications in many companies are based on reports and documents with textual formats. These documents are kept for future analysis. On the other hand, a vast amount of data available on the web for data mining is in the form of unstructured text data, which grows in volume every day.

Data mining, also known as "knowledge discovery from data," is the process of extracting information and knowledge from data in databases or data warehouses.

"Data Cleansing"; "Data Integration"; "Data Selection"; "Data Transformation"; "Data Mining"; "Pattern Evaluation"; "Knowledge Presentation".

We hope that by reading this article, you will gain useful and effective information for future research. The article is prepared as follows: Section 2 explains the methods and techniques for discovering knowledge in databases and the concepts of data mining. It outlines the research strategy used in these studies. Section 3 is a part of the data related to symptoms and diseases extracted from the Aminoff book and specialized consultations during numerous sessions with a neurologist. Section 4 is about the disease detection and prediction algorithm. Section 5 is the implementation of the code implemented by Python software for disease prediction. Section 6 is the conclusion.

After constructing this matrix using various data mining methods, we focused on the following: if a disease with specific symptoms is identified, it is entered into the software as input. The algorithm implemented using

Python software generates three outputs, which are the diseases closest to the input symptoms. In other words, using large matrix methods and data mining techniques after matrix preparation, if a disease with certain symptoms is known, by examining a new vector, we can identify if this new disease with new symptoms will be closer to which diseases in the matrix rows, respectively.

The next step is to compare different data mining methods used for this matrix and observe which one provides the optimal answer or has less error. The important result of this research is to select the best method that has the least time complexity to obtain results. It is noteworthy that this method can be generalized to many other situations.

The increasing advancements in healthcare science have led to longer life expectancy, reduced mortality rates, and an increase in the elderly population.

Materials and Methods

This is a descriptive-analytical and applied research, with one of the most effective data mining methods used in it. Multiple data mining techniques have been employed for disease prediction and early diagnosis.(2,3)

System Identification

Identifying the domain where data mining is to be conducted and possessing the relevant knowledge for this research are crucial. Therefore, in the initial phase, consultation with a neurologist, thorough study of the "Clinical Neurology" book by Aminoff, as well as research on neurological diseases to identify influential factors in infection, treatment, and diagnostic methods, along with preventive measures, have been undertaken to ensure a proper understanding of the study domain.

Data Preparation (Diseases and Symptoms)

The data used in this study is sourced from the "Clinical Neurology" book by Aminoff, consultations with a neurologist, and clinical data. After consulting with the relevant physician and utilizing clinical data from archives, a matrix consisting of approximately 150 rows and 500 columns has been formed. The elements of this matrix represent the j-th sign for the i-th disease. Textual studies and consultations with a specialized physician have been incorporated into the design.

The compilation of diseases and data has been defined in a tabular format in Excel, with Table (2), (1) serving as an example where its rows denote diseases and its columns denote symptoms, following the data collection methods as described. Table (3) serves as an example representing symptom codes, and Table (4) serves as an example representing disease codes. Disease diagnosis codes have been modeled using Python programming and data mining techniques such as Manhattan distance, k-nearest neighbor, Pearson distance, Minkowski distance, and cosine similarity. Data representing diseases have been obtained for various symptoms (matrix columns) from 1 to 150 (matrix rows).

Table 1: Symptoms (Headache) (1) The compilation of diseases and data has been defined in a matrix format in Excel. , where its rows represent diseases, and the numbers against the rows essentially represent the columns indicating symptoms. Intensity of symptoms from zero to ten has been assigned based on specialized studies.

Visualizations aid us in understanding data more effectively. By creating visual representations, we strive to transform numerical data into a format that humans can comprehend because numerical data alone may not be helpful. It is through modeling and analyzing the structure of this data that we can gain a proper understanding of the reality behind these numbers. One of the most important visualizations is heatmaps. The goal of a heatmap is, in fact, to create an initial clustering and display numerical information using colors. In the heatmap below, you can see numerical values represented by colors in the column and row sections. Each cell of this visualization represents a spectrum that corresponds to a numerical value. In the figure, the spectrum is displayed with different colors, with values below zero shown in red and those above zero in blue. Zero values are displayed in black. By viewing this heatmap, the magnitude of each section can be observed.

The clustering section present in the heatmap aims to cluster genes or samples. Clustering implies that these genes or samples contain similar information and are grouped into a cluster. Various algorithms have been introduced for clustering, and hierarchical

clustering is used in heatmaps. This type of clustering also utilizes different algorithms, which vary depending on the distance metric used. In this heatmap, the Euclidean distance is employed as the distance metric.

Standard deviation (symbolized as σ) is one of the measures of dispersion that indicates how much the data points deviate from the mean on average (6-8). One of the main features of the median is that the sum of the absolute differences between various variable values and the median is minimized.

There are several species of means in mathematics, particularly in statistics. In the study of the distribution of a statistical population, the representative value around which the measurements are distributed is called the central value, and any numerical measure that represents the center of a dataset is called a measure of central tendency. Mean and median are among the most common measures of central tendency (5).

Modeling

Various data mining methods exist for modeling. Therefore, in this study, modeling was carried out in Python software using data mining techniques, focusing on the development of predictive models.

Jaccard Distance

The coefficient de communaté, originally devised by Paul Jaccard, provides a measure of distance to indicate how closely two sets are related. It is formally written as follows under the name Jaccard Index or Jaccard Similarity Coefficient for finding the similarity between two items [9]:

$$similarity_{Jaccard}(i,j) = \frac{\#users\ that\ bought\ both\ items}{\#users\ who\ bought\ either\ i\ or\ j}$$

Where, i represents item 1 and j represents item 2.

Measuring Distance with Lp Norms

A general method for measuring distances is through Lp norms. Therefore, in this section, we will explore two different metrics: L1 norm, L2 norm, and Lp norms.

L1 Norm (Manhattan Distance)

The simplest distance metric is the Manhattan distance, also known as the taxicab distance, which excels in speed. The Manhattan distance is calculated by summing the absolute differences between the x's and the y's:

(1)

$$|x_1 - x_2| + |y_1 - y_2|$$

L2 Norm:

The L2 norm, also known as the Euclidean norm:

(2)

$$= \left\| r_{sara} - r_{pietro} \right\|_2 = \sqrt{\sum_{i=1}^{n} \left| r_{sara,i} - r_{pietro,i} \right|^2}$$

1) Distance (Sara, Pietro)

Cosine Similarity

Cosine similarity is highly prevalent in text processing and is utilized in collaborative filtering. It disregards 1-1 metrics and is introduced through equation (4) as follows:

(4)

$$\cos(x, y) = \frac{x, y}{\|x\| \times \|y\|} \quad (1$$

$$sim\ (i,\ j) \quad (2$$

$$= \frac{ri.rj}{\left| r_1 \right\|_2 \left\| r_j \right|_2} = \frac{\Sigma u^r i, u^r j, u}{\sqrt{\Sigma u^{r^2} i, u} \sqrt{\Sigma u^{r^2} j, u}} \quad (3$$

Discovering Hidden Genres (Categories) with Matrix Factorization

Our discussion revolves around latent factors in content data. Now, the hidden factors related to collaborative filtering will be addressed, which refers to behavioral data.

While many names have been discarded, I have considered this: hidden genres are essentially latent factors, particularly when discussing films. It is said that these factors are hidden because they are defined by something calculated by an algorithm, not by humans. They are biased towards data representing or explaining user preferences. These biases or factors are also hidden, as even if the data seems data-wise and logical, it is not easy to determine what these factors mean. As we proceed, I will explain this. Additionally, we will focus on something called a rating matrix.

Before moving forward, I would like to set a stage. We will begin with numerous discussions about Singular Value Decomposition (SVD) (10). It is a well-known linear algebra method, and there are many tools available to assist you in calculating existing matrix factorizations. I will show you a tool with Scikit-learn, a machine learning library for Python.

With a real SVD, you can easily add new users. However, calculating an SVD is quite slow, and if you have a large dataset, it will be time-consuming. More importantly, there are strict requirements regarding what should be done about empty cells in the rating matrix. To address this issue, we will move towards Funk SVD, which is becoming the most common choice for usage. Adding new users is not a simple task but it is feasible.

Finding hidden factors is a task that can be approached in various ways. In the realm of collaborative filtering, finding hidden factors has primarily been done through matrix factorization based on the rating matrix.

Data reduction can be beneficial in some cases. The reason for reducing dimensions could be to extract a signal from the data. For example, the top pattern represents a scatter plot of noisy data (disturbances), while the bottom pattern represents the true signal - the information present in the data. Simplifying data can sometimes make it easier to understand hidden information within them.

In essence, you can have the same information for points on a line, as shown in the figure, only points that also have noise. This same principle applies when performing dimensionality reduction, where you have high-dimensional data.

Consider the data in Figure 4 as a cloud of points that you want to project onto a lower-dimensional space, where the distance between objects remains the same. Points that were farther apart before reduction remain farther apart afterward, and close cases become closer after reduction.

Matrix Factorization

= Creating a Factorization using SVD

One of the most common methods utilized for matrix factorization is a technique named Singular Value Decomposition (SVD), to obtain elements for recommending to users, and to do this using factors extracted from the rating matrix.

We want to create two matrices from the rating matrix M so that we can use them: one representing customer preferences and the other containing item profiles. Using SVD, we create three matrices: U, Σ, and Vt. Since we want to end up with two matrices, we multiply the square root of Σ into one of the other two matrices, leaving two matrices. But before doing this, we want to use an intermediate matrix that provides us with information about the amount of reduction needed. Figure 5 shows SVD.

Figure 5 represents a matrix that can be decomposed into three matrices:

M: The matrix you want to decompose; in your case, it is the rating matrix.

U: The user composition matrix.

Σ: The diagonal weights matrix.

VT: The item composition matrix.

Diagonal Matrix

A diagonal matrix is one that has only zero values.

The central diagonal matrix Σ contains components sorted from largest to smallest. These components are called singular values, and they represent the amount of information generated by this combination for the dataset. A combination here refers to a column in the user matrix U and a row in the item matrix VT (both). Now, you can choose r combinations and consider the rest of the diagonal as zeros. When you consider the values outside the central box as zeros, what remains from the matrices is removing all the rightmost columns in the user matrix U and all the bottom rows from V*, while keeping only the top r rows.

How much should we reduce (shrink) the matrices?

We can reduce the dimensions using two cases, and still create a plot similar to the one shown in Figure 6. Another good reason for reducing the matrix to two dimensions is that by observing the weights in the sigma matrix (Σ), we can obtain more information using just two combinations.

Dealing with zeros in the rating matrix by using imputation

However, often you will encounter situations where only 1% of the cells in the rating matrix have values. Something needs to be done. To

achieve this goal, we have two common methods:
*We can calculate the average of each element (or user) and fill each row (or column) of this matrix, which contains zeros, with this average.
* We normalize each row in such a way that all components are centered around zero, so the zeros will become the mean.
Both approaches are considered as imputation. This solution shows you part of the way, but we can have better performance with something called baseline predictors, which we will discuss soon. In the next step, we will fill the zero cells with averages obtained (product obtained) from the ratings.
Normalizing the ratings
Calculate the average of the movies r_average=M[M>0.0]. mean () ←
Set zero for all inputs for NaN (not a number) M[M==0] =np.NaN←
Fill all NaNs with averages M.fillna (r_average,inplace=True)←

Adding a New User with Insertion (Folding in New Entries)
An interesting point about the SVD method is that we can fold in new users and items to the system.
Expressed as a vector, it will be as follows:

$$r_{kim} = (4.0, 5.0, 0.0, 3.0, 3.0, 0.0)$$

You can compute the new row using the formula below: (Figure 7)

$$u_{kim} = r_k V^t \sum^{-1}$$

Where, u_{kim} is the user vector in the reduced space representing the new user. *r_k is the vector for rating the new user. $\sum^{(-1)}$ is the inverse of the sigma matrix. V^T is the item matrix.
To use this in Python, we have executed the following code in a sample script:

Folding in new users

$$from\ numpy.linalg\ import\ anv$$

$$r_{kim} = np.array([4.0,5.0,0.0,3.0,3.0,0.0])$$

$$u_{kim} = r_{kim}\ ^{*} vt\_reduced.T^{*}\ inv(Sigma\_reduced)$$

Now, we can also predict ratings for the user "kim". Similarly, we can fold in a new item using the following formula:

$$\hat{\imath}_{new} = r^{T}_{new\ item} U \sum^{-1}$$

* i_{new} is a vector in the reduced space representing the new item.
* r new item is the rating vector of the new item.
*$\sum^{-1}$ is the inverse of the sigma matrix.
*-U is the user matrix.
Remember, this reduction is done to extract topics from the data. When you add a new user or item, these topics do not update; they are compared with the discussions that were previously available.
Updating SVD is often important as much as possible. Depending on the number of new users and items, you should perform this task once a day or once a week. An interesting point about folding in a new user is that if the new user only has one rating, whether it is high or low, it does not matter. The recommendation list will remain exactly the same.

Performing Recommendations with SVD
There are two methods for providing recommendations: calculating all predicted ratings and considering the highest-rating items that the user has not encountered before, or iterating through each item and obtaining similar dot products in the reduced space. The third method can involve utilizing your new matrices to compute collaborative filtering. The reason for considering this as a good idea is that matrices contain all non-zero inputs (at least if normalized). In this compressed space, you have a much better chance of finding similar items or users.
I could continue writing about SVD and its capabilities, but I would like to explore another type of reduction method, similar to SVD but much more efficient for computation. The SVD method you have seen so far has several drawbacks: first, dealing with unfilled cells in the rating matrix is necessary, and computing large matrices is slow. On the positive side, adding new users when they enter is possible. However, keep in mind that the SVD model is static and should ideally be updated frequently. The next matrix decomposition algorithm is interesting, but as always, I will take a moment to focus on something called baseline predictors, which make filling in the gaps in the matrix easier. Although they can be used as a recommendation system, here they are used as a method for better matrix decomposition.

Baseline Predictors

Apart from the types of items and user preferences, there are other aspects of items and users that can be considered. If a movie is generally considered good, its average rating is likely slightly higher than the global average of all movies, and conversely, if a movie is considered bad, its average rating is likely lower than the global average. If you have such information, you can add a slightly higher default rating to an item. However, some users may be more important or positive compared to others. An item that is above or below the average can be said to be biased. The same applies to users; you can say that users have biases compared to the global average.

If you are able to extract these biases for items and users, then you are in a position to provide baseline predictors, which are much better than using averages, as you did earlier when filling in empty cells of the rating matrix. Using these biases, you can create baseline predictors. A baseline predictor is the sum of the global average, plus the item bias, plus the user bias. You will use the following equation:

$$b_{ui} = \mu + b_u + b_i$$

Where,

*b_u is the baseline prediction for item i for user u.

*b_u is the user bias.

*b_i is the item bias.

*μ is the mean of all ratings.

Estimation of Biases by Least Squares

You want to obtain biases that represent baseline predictions close to known ratings. If you consider the same ratings used previously, you will ask what values should be determined for the biases to minimize the following relationship as much as possible.

$$\min (r_{(sara,civil\ war)} - b_{(sara,civil\ war)})^2$$

$$\min (r_{(sara,civil\ war)} - \mu - b_{sara} - b_{civil\ war})^2$$

To ensure that no one is left behind, I will quickly address this task. This equation signifies your effort to find bs that minimizes or least squares the equation. For multiple ratings, it can be written as follows:

$$\min_b \sum_{(u,i)\in K} (r_{(u,i)} - \mu - b_u - b_i)^2$$

Where,

(u, i) ϵ K represents all the ratings you have had so far.

A simpler method to find these biases is to utilize the equations described in this section. Initially, compute the bias for each user (b_u) by considering the sum of differences between user ratings and the mean, then divide it by the number of ratings, meaning the result is the average difference between the mean and user ratings.

$$b_u = \frac{1}{|I_u|} \sum_{i\in I_u} (r_{u,i} - \mu)$$

After calculating all user biases, compute the item bias (b_i) using the same method.

$$b_i = \frac{1}{|U_i|} \sum_{u\in U_i} (r_{u,i} - b_u - \mu)$$

The biases calculated can be used to fill in empty spaces in the rating matrix, instead of SVD, or in fact, most matrix factorization algorithms, may perform better. I have calculated the biases for test data.

While we talk about bias as static, a user can range from a happy individual to a grumpy elder, and biases should adjust to reflect that. This applies to item bias adjustment over time as well since items enter and exit fashions. Predictions of ratings can also vary over time, so you can consider your rating prediction function as a function of time. In such cases, you need to modify the previous equation to the following time-dependent equation:

$$b_{ui}(t) = \mu + b_u(t) + b_i(t)$$

Consider this especially if you have long-term data with numerous ratings. If you want to improve the accuracy of your recommender, keep this in mind.

If your data spans a long period and has many ratings, you should pay attention to the temporal aspects. Otherwise, start with a simpler approach and then upgrade. You can delve into research describing how to approach this. A good starting point is collaborative filtering with temporal dynamics by [10-12].

Decomposition Using Funk SVD

The SVD method puts significant weight on the rating matrix, but this is a sparse matrix (quiet, scattered, low density), and one should not heavily rely on the concept that the likelihood of finding a crowded cell with a rating can be less than 1%. Instead of using the entire matrix,

Simon Funk proposed a method where only the necessary things are used.

You start this method by looking at RMSE, which is used to provide a measure of how close you are to the known rating. By looking at your toolbox, you will notice something called gradient descent, which uses RMSE to improve the solution. When you have that, you will pay attention to using baseline predictors. Earlier, I referred to them as a method for better prediction compared to average rating information. By learning all these, you will explore the Funk SVD algorithm.

Adding Biases

In the previous section, we discussed biases. Even though this equation may be slightly complex, adding them is valuable.

The approach I am considering is one where the user prefers a specific type of movie, encoded in the user factors, while a negative (or positive) bias is encoded in the item factors. Now, a predicted rating is the sum of these four components, as shown in Figure 8.

When adding them to the equation, the new function you want to minimize is as follows:

$$min_{b,p,q} \sum_{(u,i)\in K} (r_{iu} - \mu - b_u - b_i - q_i p_u)^2$$

You perform this according to the stochastic gradient descent approach and by considering the derivative of the mean squared error, you obtain these equations.

If your rating matrix is sparse, you may encounter issues (problems) of overfitting. Overfitting occurs as matrices U and V can precisely calculate appropriate values for existing ratings, but when it comes to predicting new cases, they completely fail. One way to address this is by introducing something called a regularization term, which minimizes the following relationship:

$$min_{uv} \sum_{(u,i)\in known} (r_{ui} - u_u v_i) + \lambda(||u||^2 + ||v||^2)$$

Brute Force Recommendation (Theory, Recommendation) Calculation

The brute force recommendation is straightforward: determine a predicted rate for each user and item, then sort all predictions and return the top N. While doing this, you can also save all predictions for use when users visit later.

This is a non-negligent (nonsense) method of doing unnecessary work. Keep in mind that doing this may require a lot of time and force your system to perform many computations that will never be used. You can optimize this to some extent, but a better approach is to save factors and biases and use them to calculate recommendations.

Instead of using the original rating data, you can use the factors you have calculated yourself. This means you calculate similarities where items are closer and in smaller dimensions, making the task easier.

If you have already observed the factor space, you can create user-based or item-based recommendations. Either way, you will benefit from the vectors created representing users and items.

Results

Python Code Implemented on Data

After collecting the data, we implemented the algorithm shown in the figure using Python. The metrics show the closest similarities to our input symptoms. When a patient visits a doctor with specific symptoms, the doctor enters the symptoms into the software we have implemented, and ultimately the output will show the three diseases closest to the symptoms for predicting the disease to the doctor. Further comparisons are made between the metrics, and the worst metric, which has weaker results compared to the others, is determined. In the Appendix A, the program execution is included, and the results are presented for further information.

Running the Code and Disease Prediction

After preparing the data and implementing the algorithm in Table 5, the execution of the code for disease prediction is presented in Figure 1 in this section. Data mining is capable of discovering and extracting new knowledge from past data. The preprocessing of data and the selection of variables also have a significant impact on knowledge discovery. Various data mining techniques exist for disease prediction. In this article, five data mining algorithms were used, which will be explained further. The empirical results demonstrate the effectiveness and reliability of all three methods compared

based on sensitivity, specificity, and accuracy. Pruning and boosting techniques were employed to find the desired structure and improve the accuracy and validity of the results. The examined database in this article focused on data mining prediction approaches in neurological diseases and their diagnosis. Therefore, based on the notes mentioned regarding research gaps and the use of data mining prediction approaches in early detection of various diseases (medical), new research can be initiated in this field (5, 6).

Example 1.5. As an example, we examined a sample and entered the disease symptoms in order, including decreased level of consciousness, confusion (B=9), neck stiffness (D=8), and bilateral extensor plantar response (G=6) into the software implemented with Python. The software outputs diseases close to this patient's condition, and the output is shown according to various metrics. The Jaccard metric output indicates three diseases, namely migraine, spinocerebellar degeneration due to Phenytoin, and vasovagal syncope. The Minkowski metric output indicates muscular dystrophy diseases, Duchenne muscular dystrophy, and facioscapulohumeral muscular dystrophy. All output diseases are close to the input symptoms in the domain of motor disorders, sensory impairments, motor deficits, and visual impairments. Our software identified the weakest metric, which indicates weaker results compared to other metrics, as the Pearson metric. Additionally, using the SVD technique, the program executes in less time. The results were reviewed with a neurology specialist, and the examination outcome showed that the results are entirely acceptable and accurate.

This graph represents the evaluation of the target algorithm. Cross-validation technique has been employed, with K set as 40 as depicted in Figures 11 and 12. Cross-validation has been performed on 40 configurations, and accuracy has been calculated. The results are illustrated below. Upon examining the results of matrix factorization, I believe achieving a service coverage of up to 95% for diseases is excellent.

One way to look at the data is to compare the training error with the testing error, as shown in the figure below. It is crucial for the lines to have an angle less than or equal to 45 degrees, indicating that the test error is proportional to the training error. It is advisable to run the experiment for a certain number of iterations, such as 50 or 100 times, while some articles suggest looking at RMSE for each iteration and stopping when the change in RMSE is less than a certain threshold. Plotting MSE as shown below, it is a good idea to look for intersections in the graph. The intersection is often where the algorithm avoids overfitting to its known data and starts to overfit excessively. A line in the figure below represents training with 75 factors. As you can see, the testing MSE has a small intersection at around 400 iterations. Here we should only use 20 factors because the test line with 20 factors has a small intersection at around 275 iterations, so it might be a good place to stop.

The return values of this function increase as the items become more similar. The better method depends on your domain and data. In general, the relationship between similarity and distance is as follows:

As the distance increases, similarity tends towards zero.

As the distance tends towards zero, similarity tends towards one.

In this section, we measure similarity using different algorithms and compare their accuracies with each other.

Conclusion

The aim of this research was to design an efficient model for discovering knowledge in predicting neurological diseases based on the latest dataset of indicators in this field related to public health and to provide an accurate analysis of data mining techniques for predicting neurological diseases. In other words, research efforts have been made to employ data mining techniques based on the use of disease and symptom datasets through business intelligence programs to provide important results in accurate decision-making and timely presentations (7,8). For comprehensive explanations and overall conclusions, various metrics were utilized and the algorithm was implemented using Python software for predicting neurological diseases. The advantages of each of these metrics have been elaborated upon, and the SVD technique

has also been employed to reduce program execution time for disease prediction.

Acknowledgments

I extend my sincere gratitude to the esteemed and knowledgeable professor, Dr. Mohammad Reza Eskandari, for his assistance in data analysis and medical sections.

Funding

None

Authors Contributions

The author contributed to the data analysis. Drafting, revising and approving the article, responsible for all aspects of this work.

Ethical Consideration

None

References

1. Nilashi M, Abumalloh RA, Alyami S, Alghamdi A, Alrizq M. A Combined Method for Diabetes Mellitus Diagnosis Using Deep Learning, Singular Value Decomposition, and Self-Organizing Map Approaches. Diagnostics. 2023 May 22;13(10):1821.
2. Peng L, Huang L, Su Q, Tian G, Chen M, Han G. LDA-VGHB: identifying potential lncRNA–disease associations with singular value decomposition, variational graph auto-encoder and heterogeneous Newton boosting machine. Briefings in Bioinformatics. 2024 Jan 1;25(1):bbad466.
3. Sheng N, Huang L, Lu Y, Wang H, Yang L, Gao L, Xie X, Fu Y, Wang Y. Data resources and computational methods for lncRNA-disease association prediction. Computers in Biology and Medicine. 2023 Feb 1;153:106527.
4. Michael J, Aminoff Md DSc FRCP, S. 2021.Andrew Josephson S.Aminoff's Neurology and General Medicine. 6th ed. Academic Press :1230.
5. zacharski R. 2021.A prorammers Gui to Data Mining: the ancient Art of the Numeriati. 395.
6. Falk K. 2019.Practical Recommender Systems.1st ed.USA, 432.
7. Yang HF, Phoebe Chen YF. 2015.Data mining in lung cancer pathologic staging diagnosis: Correlation between clinical and pathology information. 42(15-16): 6168-6176.
8. Yi Yeh J, His Wu T, Wei Tsao CH. 2011.Using data mining techniques to predict hospitalization of hemodialysis patients .50(2):448-439.
9. Sornalakshmi, M., Devakanth, J. J. M. A., Rajalakshmi, R., & Velmurugadass, P. (2023). An energy-aware heart disease prediction system using ESMO and optimal deep learning model for healthcare monitoring in IoT. Journal of Biomolecular Structure and Dynamics, 1-15.
10. Junior SB, Guido RC, Aguiar GJ, Santana EJ, Junior ML, Patil HA. Multiple voice disorders in the same individual: investigating handcrafted features, multi-label classification algorithms, and base-learners. Speech Communication. 2023 Jul 1;152:102952.
11. Zhang Y, Xiang J, Tang L, Yang J, Li J. PGAGP: Predicting pathogenic genes based on adaptive network embedding algorithm. Frontiers in Genetics. 2023 Jan 20;13:1087784.

Tables & Figures

Table 1: Intensity of symptoms

Disease	Symptoms (Headache)
Subarachnoid hemorrhage	10
Meningitis or encephalitis	9
Intracranial hypertension (encephalopathy)	9
Giant cell arteritis	9
Intracranial mass	8
Pseudotumor cerebri (idiopathic intracranial hypertension)	9
Trigeminal neuralgia	0
Glossopharyngeal neuralgia	0
Postherpetic neuralgia	0
Hypertension	9

Table 2: Symptoms B (5) (Decreased level of consciousness, confusion)

Diseases	Symptoms (Decreased level of consciousness, confusion)
Subarachnoid hemorrhage	9
Meningitis or encephalitis	9
Intracranial hypertension (encephalopathy)	0
Giant cell arteritis	0
Intracranial mass	7
Pseudotumor cerebri (idiopathic intracranial hypertension)	0
Trigeminal neuralgia	0
Glossopharyngeal neuralgia	0
Postherpetic neuralgia	0
Hypertension	0

Table 3: Some symptoms extracted from Aminoff's book and consultation with a specialist physician (4).

Code	Symptoms	Code	Symptoms
A	Headache	G	Plantar reflex (bilateral extensor or Babinski reflex)
B	Decreased level of consciousness (confusion)	H	Hemiparesis (paralysis of one limb or one side of the body)
C	Vomiting	I	Aphasia (speech disturbance)
D	Stiff neck	J	Visual field defects or visual changes
E	High blood pressure	K	Herniation
F	Fever	L	Progressive drowsiness

Table 4: Some of the diseases extracted from Aminoff's book and consultation with a specialist physician (4).

1) Subarachnoid hemorrhage
2) Meningitis or encephalitis
3) Hypertensive encephalopathy
4) Giant cell arteritis
5) Intracranial mass
6) Pseudotumor cerebri (Idiopathic intracranial hypertension)
7) Trigeminal neuralgia
8) Glossopharyngeal neuralgia
9) Postherpetic neuralgia
10) Hypertension
11) Atypical facial pain
12) Migraine
13) Cluster headache
14) Tension-type headache
15) Ice pick headache

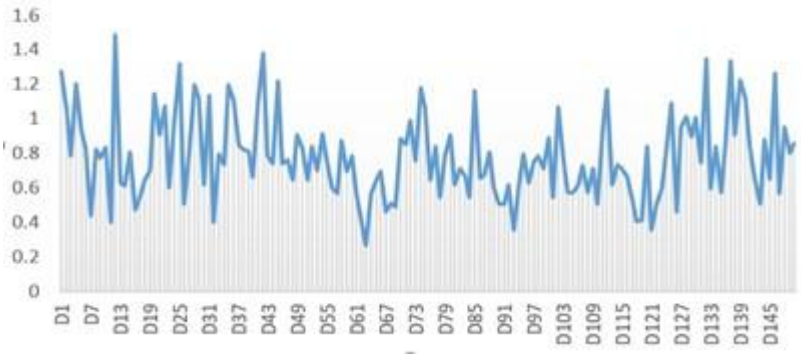


Figure 1. Standard deviation data

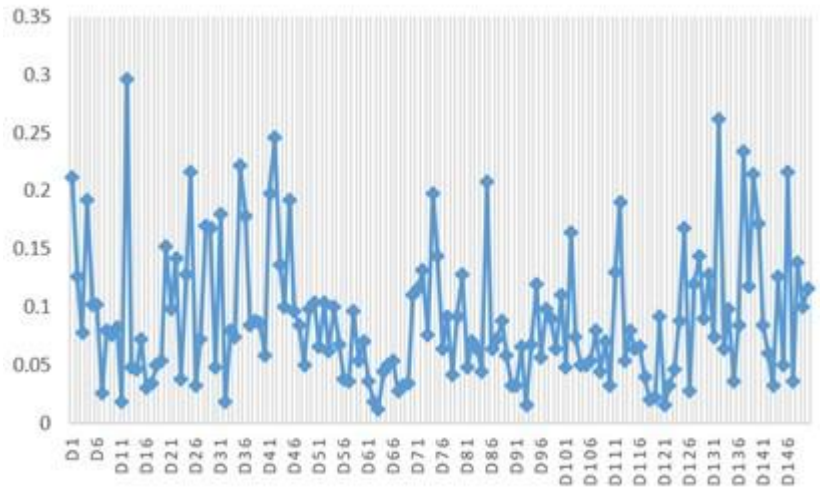


Figure 2. Mean data

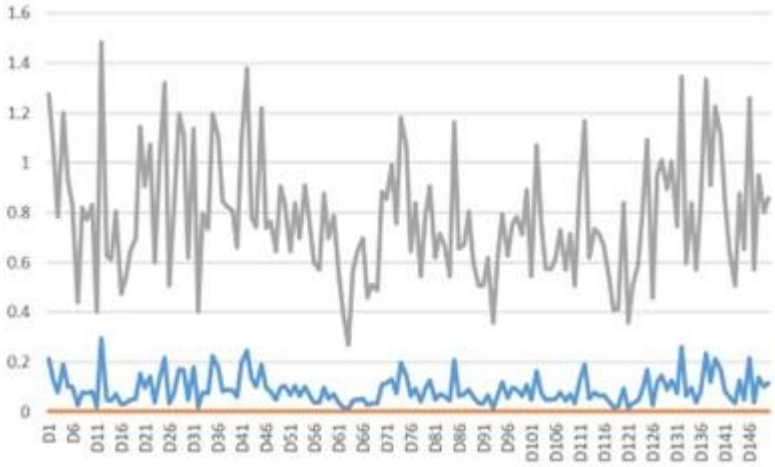


Figure 3. Check the median data and the standard deviation data and the mean data

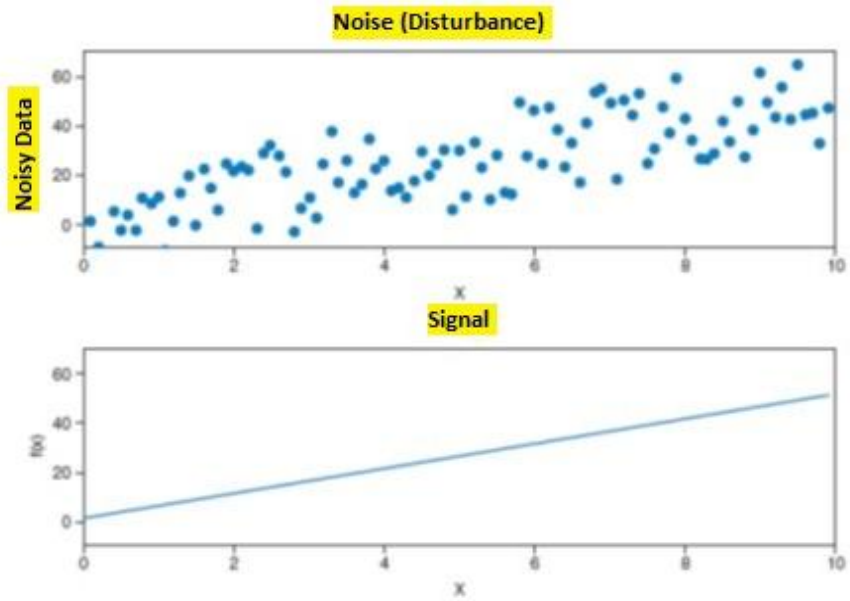


Figure 4 .depicts a scatter plot of noisy data points (upper) and signals that reveal the information present in the data (lower).

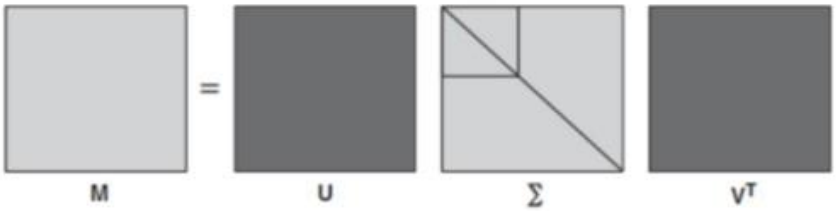


Figure 5: Singular Value Decomposition procedure

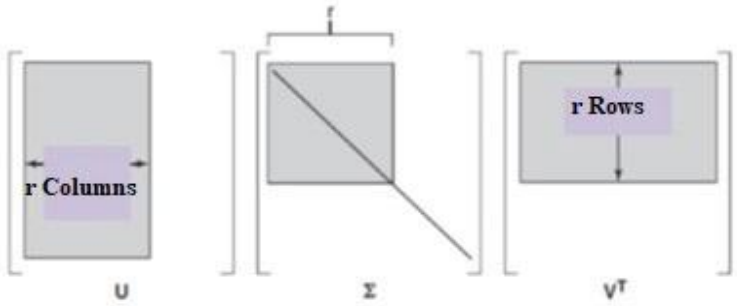


Figure 6. SVD reduction to zero by setting small values in Σ.

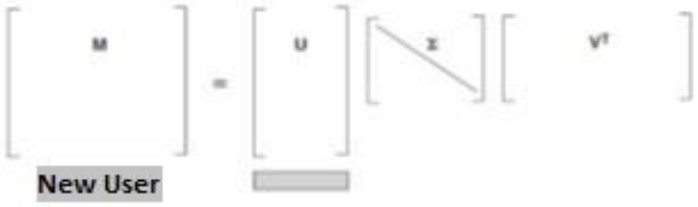


Figure 7. Graphs illustrating the technique of folding in (folding in new items) using SVD

Detailed Program

Obtaining the global mean, which can be done initially by obtaining the mean of each column and then finding the mean of those means.

Subtracting the global mean from all non-zero ratings.

$$global\_mean = M[M > 0].mean(\square).mean(\square)$$

$$M\_minus\_mean = M[M > 0] - global\_mean$$

The mean of each row is equal to the user bias.

$$user\_bias = M\_minus\_mean.T.mean(\square)$$

$$item\_bias = M\_minus\_mean.apply(lambda\ r: r - user\_bias).mean(\square)$$

Subtracting the user bias from each row, then considering the mean of each column gives you the item bias.

Temporal Dynamics

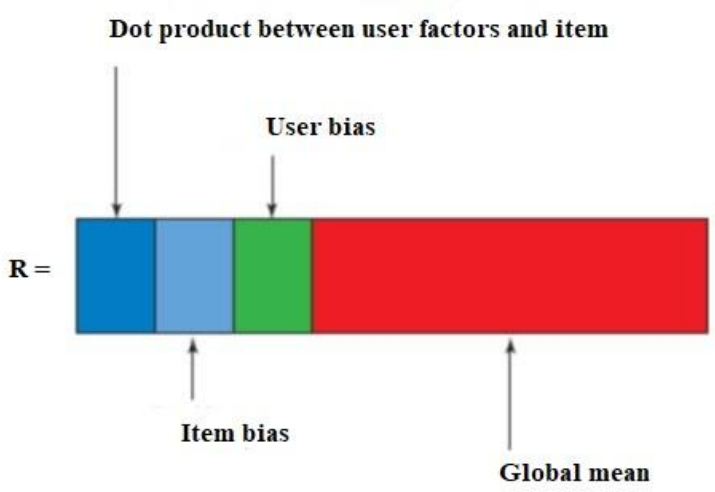
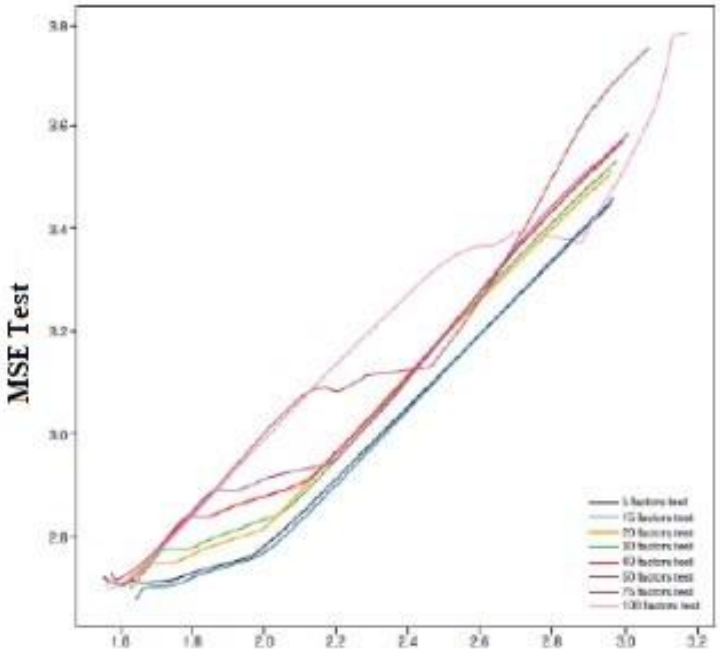


Figure 8. A predicted rating is a combination of these four elements.



MSE Test

Figure 9. Compares the MSE test with the training MSE to indicate whether over fitting occurs or not.

```
    df.iloc[disease_name][1:len(df)] = 0
    for i in range(how_many_number):
        index, df.iloc[disease_name][index] = [int(x) for x in input().split()]
    recommend(df, int(disease_name))

if show_type == '2':
    disease_name = input('Enter the Number of the disease: ')
    recommend(df, int(disease_name))

if show_type == '3':
    print("Enter %d Feature Values" % (len(df.columns)))
    for i in range(len(df.columns)):
        df.iloc[disease_name][i] = input()
    recommend(df, int(disease_name))
```

```
Choose: 1: Input NoN-Zero Features From User, 2: Input Features From File, 3: Input all Features From user 1
How many non-zero Features do you have? 3
Enter two values for 3 times: First is the Feature Index (from 1 to 500) and Second is the Feature Value
2 9
4 8
7 6
Distance Type ( 1: Jaccard similarity, 2: Cosine similarity l1-norm, 3: Cosine similarity l2-norm): 1
Do you want to compare distances? (y (yes), n (no)): y
Jaccard similarity (top 3):  [ 12  37 146] [0.125, 0.125, 0.14285714285714285]
Jaccard similarity (top 3):  [ 12  37 146] [0.125, 0.125, 0.14285714285714285]
Cosine similarity l1-norm (top 3):  [  0 104 103] [0.0, 0.0, 0.0]
Cosine similarity l2-norm (top 3):  [  0 104 103] [0.0, 0.0, 0.0]

Worst distance calculation is Jaccard similarity
```

Figure 10: Execution of Python Code for Disease Prediction and Interpretation of Results in Example 1.5

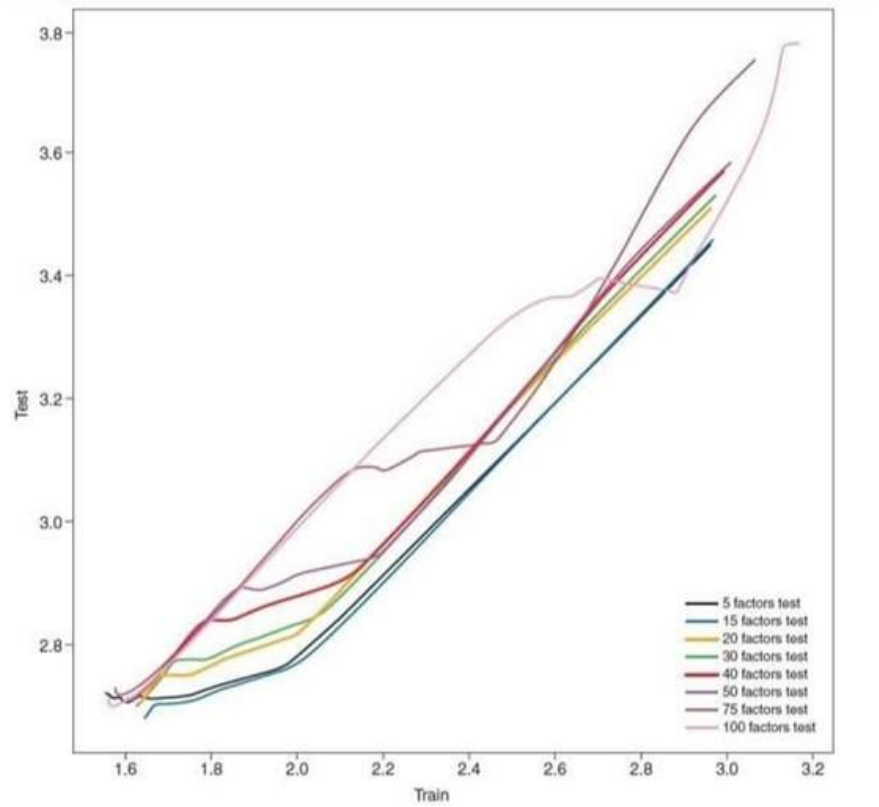


Figure 11. Similarity check diagram using svd

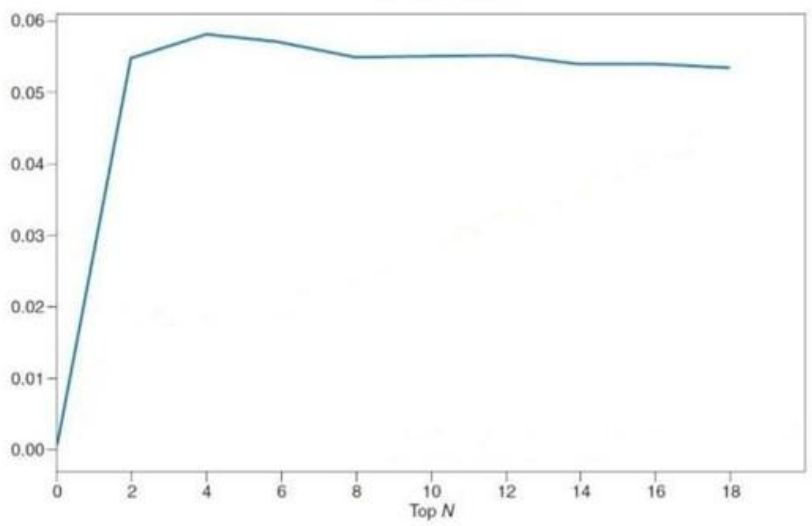


Figure 12. Similarity check diagram using data mining techniques

Appendix A: Code

```
from math import sqrt
import pandas as pd
from numpy import dot
from collections import Counter
from itertools import chain
from numpy import linalg
import numpy as np
from scipy.linalg import svd
def jaccard_similarity(arr1, arr2):
intersection = len(list(set(arr1).intersection(arr2)))
 union = (len(set(arr1)) + len(set(arr2))) – intersection
return float(intersection) / union
def l1_norm(arr):
return sum([abs(i) for i in arr])
def l2_norm(arr):
return sqrt(sum([pow(i, 2) for i in arr]))
def cosine_similarity_l1(arr1, arr2):
    return dot(arr1, arr2) / (l1_norm(arr1) * l1_norm(arr2))
def cosine_similarity_l2(arr1, arr2):
    return dot(arr1, arr2) / (l2_norm(arr1) * l2_norm(arr2))
def recommend(df, disease_name):
    distance_type = input         'Distance Type ( 1: Jaccard similarity, 2: Cosine similarity l1-norm, 3:
Cosine similarity l2-norm): ')
do_compare = input('Do you want to compare distances? (y (yes), n (no)): ')
   distance = []
    distances = []
  for i in range(df.shape[0]):
      if i != disease_name:              distance.append(jaccard_similarity(df.iloc[disease_name].values,
df.iloc[i].values))
   distances.append(distance)
    distance = []
```

```
    for i in range(df.shape[0]):
        if i != disease_name:
            distance.append(cosine_similarity_l1(df.iloc[disease_name].values, df.iloc[i].values))
    distances.append(distance)
    distance = []
    for i in range(df.shape[0]):
        if i != disease_name:
            distance.append(cosine_similarity_l2(df.iloc[disease_name].values, df.iloc[i].values))
    distances.append(distance)
    if distance_type == '1':
        print('Jaccard similarity (top 3): ', numpy.argsort(distances[0])[0:3], sorted(distances[0])[0:3])
    if distance_type == '2':
        print('Cosine similarity l1-norm (top 3): ', numpy.argsort(distances[1])[0:3],
sorted(distances[1])[0:3])
    if distance_type == '3':
        print('Cosine similarity l2-norm (top 3): ', numpy.argsort(distances[2])[0:3],
sorted(distances[2])[0:3])
    if do_compare == 'y':
        print('Jaccard similarity (top 3): ', numpy.argsort(distances[0])[0:3], sorted(distances[0])[0:3])
        print('Cosine similarity l1-norm (top 3): ', numpy.argsort(distances[1])[0:3],
sorted(distances[1])[0:3])
        print('Cosine similarity l2-norm (top 3): ', numpy.argsort(distances[2])[0:3],
sorted(distances[2])[0:3])
        counts = Counter(chain(*map(set,
                                    [sorted(distances[0])[0:3], sorted(distances[1])[0:3],
sorted(distances[2])[0:3]])))
common_remove = [[i for i in sublist if counts[i] == 1] for sublist in [sorted(distances[0])[0:3],
sorted(distances[1])[0:3], sorted(distances[2])[0:3]]]
        list_size = []
        for i in range(len(common_remove)):
            list_size.append(len(common_remove[i]))
        max_size_list = list_size.index(max(list_size))
            if max_size_list == 0:
            print('\n' + 'Worst distance calculation is Jaccard similarity')
        if max_size_list == 1:
            print('\n' + 'Worst distance calculation is Cosine similarity l1-norm')
        if max_size_list == 2:
            print('\n' + 'Worst distance calculation is Cosine similarity l2-norm')
disease_name = 100
dfs = pd.read_excel('Future_Signs_final(1).xlsx', header=None)
first_row = dfs.iloc[0, 1:]
first_column = dfs.iloc[1:, 0]
first_row.to_excel("features.xlsx", sheet_name='Sheet1')
first_column.to_excel("disease_names.xlsx", sheet_name='Sheet1')
df = dfs.iloc[1:, 1:]
X= dfs.iloc[1:,1:].values
Y = X.astype('float64')
# SVD
U, Sigma, Vt = svd(Y)
#print(U)
#print(Sigma)
#print(Vt)
```

```
# reducing the matrix
def rank_k(k):
    U_reduced= np.mat(U[:,:k])
    Vt_reduced= np.mat(Vt[:k,:])
    Sigma_reduced= np.eye(k)*Sigma[:k]
    return U_reduced, Sigma_reduced, Vt_reduced,
U_reduced, Sigma_reduced,  Vt_reduced= rank_k(4)
Y_hat = U_reduced * Sigma_reduced * Vt_reduced
#Predict a rating
Y_hat_matrix = pd.DataFrame(Y_hat).round(2)
#Reducing the matrix
def rank_k2(k):
    U_reduced= np.mat(U[:,:k])
    Vt_reduced = np.mat(Vt[:k,:])
    Sigma_reduced = Sigma_reduced = np.eye(k)*Sigma[:k]
    Sigma_sqrt = np.sqrt(Sigma_reduced)
    return U_reduced*Sigma_sqrt, Sigma_sqrt*Vt_reduced
U_reduced, Vt_reduced = rank_k2(4)
Y_hat2 = U_reduced * Vt_reduced
def meta_parameter_train(self, ratings_df):
  for k in [5, 10, 15, 20, 30, 40, 50, 75, 100]:
        self.initialize_factors(ratings_df, k)
        test_data, train_data = self.split_data(10, ratings_df)
        columns = df.columns
        ratings = train_data[columns].as_matrix()
        test = test_data[columns].as_matrix()
        self.MAX_ITERATIONS = 100
        iterations = 0
        index_randomized = random.sample(range(0, len(ratings)),(len(ratings) - 1))
        for factor in range(k):
            factor_iteration = 0
            last_err = 0
            iteration_err = sys.maxsize
            finished = False
            while not finished:
                train_mse = self.stocastic_gradient_descent(factor,index_randomized,ratings)
                iterations += 1
                finished = self.finished(factor_iteration,last_err,iteration_err)
                last_err = iteration_err
                factor_iteration += 1
                test_mse = self.calculate_mse(test, factor)
def initialize_factors(self, ratings, k=25):
    self.disease_name = set(ratings['disease_name'].values)
    self.features = set(ratings['features'].values)
    self.u_inx = {r: i for i, r in enumerate(self.disease_name)}
    self.i_inx = {r: i for i, r in enumerate(self.features)}
```

```
    self.distance_factors = np.full((len(self.i_inx), k), 0.1)
    self.disease_name_factors = np.full((len(self.u_inx), k), 0.1)
    self.all_features_mean = self.calculate_all_features_mean(ratings)
    self.disease_name_bias = defaultdict(lambda: 0)
    self.distance_bias = defaultdict(lambda: 0)
def predict(self, disease_name, distance):
    avg = self.all_features_mean
    pq = np.dot(self.distance_factors[distance],self.disease_name_factors[disease_name].T)
    b_ui = avg + self.disease_name_bias[disease_name] + self.distance_bias[distance]
    prediction = b_ui + pq
    if prediction > 10:
        prediction = 10
    elif prediction < 1:
        prediction = 1
    return prediction
def train(self, ratings_df, k=20):
    self.initialize_factors(ratings_df, k)
    ratings = ratings_df[['disease_name_id', 'features_id', 'rate']].as_matrix()
    index_randomized = random.sample(range(0, len(ratings)),(len(ratings) - 1))
    for factor in range(k):
        iterations = 0
        last_err = 0
        iteration_err = sys.maxsize
        finished = False
        while not finished:
            start_time = datetime.now()
            iteration_err = self.stocastic_gradient_descent(factor,index_randomized,ratings)
            iterations += 1
            finished = self.finished(iterations,last_err,iteration_err)
            last_err = iteration_err
        self.save(factor, finished)
def finished(self, iterations, last_err, current_err):
    if iterations >= 100 or last_err < current_err:
        print('Finish w iterations: {}, last_err: {}, current_err {}'.format(iterations, last_err, current_err))
        return True
    else:
        self.iterations +=1
        return False
    def save(self):
    print("saving factors")
    with open('disease_name_factors.json', 'w') as outfile:
        json.dump(self.disease_name_factors, outfile)
    with open('distance_factors.json', 'w') as outfile:
        json.dump(self.distance_factors, outfile)
    with open('disease_name_bias.json', 'w') as outfile:
        json.dump(self.disease_name_bias, outfile)
    with open('distance_bias.json', 'w') as outfile:
        json.dump(self.distance_bias, outfile)
def recommend_distance_by_ratings(self, disease_name_id, active_disease_name_distance, num=6):
    rated_features = set(active_disease_name_distance.values('disease_name_id'))
    disease_name = self.disease_name_factors.loc[disease_name_id]
    scores = self.distance_factors.dot(disease_name)
```

```
    scores.sort_values(inplace=True, ascending=False)
    result = scores[:num + len(rated_disease_name)]
    recs = r[1] + self.disease_name_bias[disease_name_id] + self.distance_bias[r[0]]
    sorted_distance = sorted(recs.distance(),key=lambda distance:-
float(distance[1]['prediction']))[:num]
    return sorted_distance
show_type = input(
    'Choose: 1: Input NoN-Zero Features From User, 2: Input Features From File, 3: Input all Features
From user ')
if show_type == '1':
    how_many_number = int(input("How many non-zero Features do you have? "))
    print("Enter two values for %d times: First is the Feature Index (from 1 to %d) and Second is the
Feature Value" % (
        how_many_number, (len(df.columns))))
    df.iloc[disease_name][1:len(df)] = 0
    for i in range(how_many_number):
        index, df.iloc[disease_name][index] = [int(x) for x in input().split()]
    recommend(df, int(disease_name))
    if show_type == '2':
    disease_name = input('Enter the Number of the disease: ')
    recommend(df, int(disease_name))
    #u, sigma, vt = linalg.svd(df)
if show_type == '3':
    print("Enter %d Feature Values" % (len(df.columns)))
    for i in range(len(df.columns)):
        df.iloc[disease_name][i] = input()
    recommend(df, int(disease_name))
```