Attack on Collaborative Filtering Algorithms from a Low Dimensional Space

By Sandeep Nuckchady

A report submitted as part of the requirement for the
MSc Degree in Information Security at University College London.
It is substantially the result of my own work
except where explicitly indicated in the text.

Supervisor: Dr. Jun Wang

This side is purposely left Blank

Abstract

The main aim of this report is to provide a detailed analysis on a new attack on algorithms used by recommender systems. This attack was first created in low dimensions and then projected back to the original space to reflect the attack profiles in the high dimensions. The results are conclusive for an attack size of at least 10% where a hit ratio of almost ideal was obtained. The detection mechanisms done in this project proved to be ineffective in differentiating between the attack profiles and the authentic ones. The disadvantage of this novel attack is the need for at least part of the hidden user-item matrix. Attempts have been made to solve this by first taking samples of the original matrix to determine the optimum sampling size. Secondly, a method was developed to determine a low approximation of this matrix. Though not much analysis has been done in this area because it constitutes a further work, initial experimental results seem to show that it is possible to achieve this.

This side is purposely left Blank

Acknowledgements

I would like to thank my supervisor, Dr. Jun Wang for his invaluable support, guide and useful suggestions in this MSc project. Besides Tamas Jambor, a Ph.D student at UCL provided tremendous help in modifying the JAVA code for Incremental Singular Value Decomposition. Finally, I am very much in debt to Dr. Bhaskar Mehta from Google for his painstaking help in the Variable Selection algorithm. Though he never had that much time he was very keen and tried his best to help.

This side is purposely left Blank

Contents

1 Introduction **1**

1.1 Background 2

1.2 Structure of this Report 3

2 Related Work **5**

2.1 Collaborative Filtering 5

2.1.1 Predictive Algorithm using Singular Value Decomposition 7

2.1.1.1 Background on Singular Value Decomposition 7

2.1.1.2 SVD and Collaborative Filtering 9

2.1.1.3 Weighted Low Rank Approximation 9

2.2 Attacks on Collaborative Filtering 11

2.2.1 Random Attack 12

2.2.2 Bandwagon Attack 13

2.2.3 Average Attack 13

2.3 Detection Algorithms 14

2.3.1 Detection Methods in the Original space 15

2.3.1.1 Rating Deviation from Mean Agreement 15

2.3.1.2 Weighted Deviation from Mean Agreement 15

2.3.1.3 Degree of Similarity with Top Neighbours 16

2.3.1.4 Rating Missing At Random 16

2.3.1.5 Rated Items Consistency 17

2.3.2 Detection Method in Low Dimensions 18

2.3.2.1 Introduction to Principal Component Analysis 18
2.3.2.2 Idea behind Principal Component Analysis 18
2.3.2.3 Motivation of Variable Selection 20
2.3.2.4 Variable Selection 21

3 Eigen Attack **24**
3.1 Motivation of Eigen Attack 24
3.2 Implementation of Eigen Attack 25
3.3 Limitations of Eigen Attacks 29
3.3.1 Problem 29
3.3.1.1 Deduction of an Approximate Matrix 29
3.3.1.2 Inputs to the Recommender Systems 31
3.3.2 Important Note 33

4 Experiments **35**
4.1 Experimental Data Sets 35
4.1.1 Detection Methods 36
4.1.2 Experimental Evaluations 37
4.1.2.1 Detection of Fake Profiles generated from User Profiles 39
4.2 Results and Discussions: Different Attacks 40
4.2.1 Low Dimension Scatter Plots using Variable Selection 41
4.2.2 Detection Rate using VarSelect 44
4.2.3 Evaluation of VarSelect using F1 measure 46
4.2.4 Receiver Operating Characteristic Curve for evaluating VarSelect 48
4.2.4.1 ROC of RDMA, WDMA, DegSim, RIC and RMAR 49
4.2.5 Evaluations using the Remaining Metrics 52
4.2.6 Limitations of Eigen Attack 61
4.2.6.1 Scatter plots for different Sample Sizes 61
4.2.6.2 F1 scores when VarSelect is used 63
4.2.6.3 Receiver Operating Characteristic for Different Sample Sizes 64
4.2.6.4 All Evaluation Metrics 64

4.3 Determining the low-approximation Matrix 65
4.3.1 Analysis of the Inputs to the Recommender Systems 65

5 Conclusions **70**
5.1 Summary of Eigen Attack 70
5.2 Deduction of the user-item matrix 71
5.3 Further work 71
5.4 Final Notes 72

Bibliography **73**

A Incremental Singular Value Decomposition **77**
A.1 Incremental Singular Value Decomposition 77

List of Figures

2.1 User-item matrix, Υ showing the user and item profiles 6
2.2 User 3 is the attacker (row 3) 12
2.3 Attacking item 3 - After Random Attack 13
2.4 Attacking item 3 - Average Attack 14
2.5 The Scatter plot of the User-Item matrix without any attack profiles . . 23

3.1 Scatter plot of the first two principal components for random attacks . 26
3.2 Crosses showing the fake profiles in low dimensions superimposed on the original data (Eigen Attacks) 26
3.3 Modelling of the Recommender System 30
3.4 Histogram of the overall distribution of authentic ratings with $\mu = 3.53$, $\sigma = 1.13$ and coefficient of variation (cov) = 0.32 32

4.1 ROC curve showing the different important regions 38
4.2 ROC curve showing how the different detection methods perform with "true" user profiles. Attack Size (AS) = 10% 40
4.3 Scatter plot of the first two principal components in the case of Random Attacks; Attack Size(AS) = 10% and Filler Size(FS) = 5% 42
4.4 Scatter plot showing that the points from the first two principal components merging with the authentic data (Bandwagon Attacks) 42
4.5 Scatter plot of the first two principal components in the case of Average Attacks; Attack Size(AS) = 10% and Filler Size(FS) = 3% 43

4.6 Squares showing the attack profiles data superimposed on the original data as expected; Attack Size:10% (Eigen Attacks) 43
4.7 F1 measure showing how well VarSelect can detect the different attacks; Note that the curves for random and average are almost on top of each other; Attack Size (AS) 10% and Filler Size (FS) 3% 47
4.8 ROC showing how well the different detection methods can detect different attacks i.e. Attack Size (AS) 10% and Filler Size (FS) 3% 48
4.9 ROC curve showing how good RDMA, WDMA, DegSim, RIC, RMAR are at detecting random attacks; Attack Size (AS) 10% and Filler Size (FS) 3% 50
4.10 ROC curve showing how well RDMA, WDMA, DegSim, RIC, RMAR are at detecting Bandwagon Attacks; Attack Size(AS): 10% and Filler Size(FS): 3% 50
4.11 ROC curve showing how well RDMA, WDMA, DegSim, RIC, RMAR are at detecting Average Attacks; Attack Size (AS) 10% and Filler Size (FS) 3% 51
4.12 Receiver Operating Characteristic showing how poor RDMA, WDMA, DegSim, RIC and RMAR can detect Eigen attacks; Attack Size (AS) = 10% 51
4.13 Scatter plot after using only 1% of the user-item matrix 61
4.14 Scatter plot after using only 5% of the user-item matrix 62
4.15 Scatter plot after using only 10% of the user-item matrix 62
4.16 The F1 measure for the three different sample sizes: 1%, 5%, 10% . . . 63
4.17 The ROC curve for the three different sample sizes: 1%, 5%, 10% 64
4.18 Histogram showing the distribution of the ratings from the inputs . . . 66
4.19 The F1 score showing that indeed a very low detection rate is obtained 67
4.20 ROC curve showing that inputs are mostly undetected when using VarSelect with an area under the curve of 0.802; 68

List of Tables

2.1 VarSelect algorithm 22

3.1 Comparison of the performance of the different attacks 24
3.2 Algorithm showing an Implementation of Eigen Attack 28
3.3 Different Rating Distributions for Different Range of Ratings 31
3.4 Implementation of Inputs to Recommender Systems 33
3.5 Exploiting the Recommender Systems 34

4.1 Detection rate for the Different attacks using VarSelect 45
4.2 Mean AUC for the Random Attacks 53
4.3 Mean AUC for the Bandwagon Attacks 55
4.4 Mean AUC for the Average Attacks 57
4.5 Mean AUC for the Eigen Attacks 59
4.6 The percentage of Eigen attacks detected when VarSelect has been used 65
4.7 The percentage of inputs which are detected to be dissimilar to the true users 66

This page is purposely left Blank

Chapter 1

Introduction

The purpose of this project is to show that most of the detection methods already available cannot discriminate between newly-designed attack profiles from user profiles. To achieve this overly optimistic goal, the attack (named Eigen Attack) was built from a low dimensional model which is a new approach of creating fake profiles. As far as the author of this report is concerned, all attacks currently present are created in the original space. Comparisons of the results from the developed attack with different attacks people have implemented in the past will be provided to show that Eigen Attack indeed renders the recommender systems more vulnerable. Varselect algorithm did not detect any of the attacks and other detection measures like RDMA and WDMA performed worse when compared to the detection of the other conventional attacks. The main difficulty encountered with this new attack which has to do with the unavailability of the user-item matrix will be discussed. To solve this, it was decided to investigate the minimum sample size of the user-item matrix needed to perform Eigen attack. Finally, an attempt is made at finding an approximate version of part of the hidden user-item ratings. Initial results seem to be very promising.

1.1 Background

Consider the quotation from the famous entrepreneur, Jeff Bezos, the CEO of Amazon:

> WE DON'T MAKE MONEY WHEN WE SELL THINGS; WE MAKE MONEY WHEN WE HELP PEOPLE MAKE PURCHASE DECISIONS.

This is a very interesting statement since it immediately implies the use of some form of recommender systems which will participate in products sales. Jeff Bezos meant that artificially boosting the ratings or reviews of products does not guarantee the products will be well sold in the long term though this could be in the short term. The reason behind this lies predominantly in the lack of credibility of the site on the part of the users in the long term. An item that a user does not love but was yet recommended as one that should be would most likely cause disbelief in the recommender system in the long term. So, the loyalty that is the value added relationship between the customers and the site (in this case Amazon.com) would be damaged in the long term [5]. The aim of the attacker which should promote the sales of specific items at least for a short period of time would in the end cause the customers to distrust the site once they understand that many of the items they do not like are being recommended.

Recommender systems are increasing in importance since they are being seen more and more as a way to provide *accurate* recommendations to users and , thus, relieve information overload. It is not only a business for the producers but it also helps the customers to save time and allows homophilous diffusion [26]. In homophilous diffusion, people of similar tastes will be compared to determine which specific items they will like next. Recommender systems have certainly found widespread importance in the realm of e-commerce. People have tried through many years to develop better algorithms which would recommend the most likely items a user would prefer. The underlying algorithms which help in the prediction of items are termed collaborative filtering algorithms. Greater accuracy in the prediction algorithms, undeniably, implies greater confidence of the users on the list of recom-

mendations which *ergo* means greater likelihood that items will be purchased. Several e-commerce sites use different information on the customers to help predict which items users might prefer. For example, the website might retrieve information based on the past buying history of the customers or based on the demographic location of the buyers or simply the top most popular items are recommended [5]. One could think of recommender systems as a virtual store selling only the items that particular customers would like [5]. As well pointed out in [27], recommender systems are not only used in e-commerce applications but also in the recommendation of movies (e.g. Netflix), music (e.g LastFm) and many others. The mere fact that recommender systems are linked to profitability is very attractive to those attackers who want to affect the sales of specific low-rated and probably unpopular items. Indeed, the inherent direct link between recommender systems and the amount of money spent on products recommended has, unfortunately, attracted malicious producers whose main purpose are to manipulate the system, hence, increasing the probability of their items being found in the list. In this project, the problem is looked on the attackers' side i.e. what are new possible attacks which could increase the chance of some items being found in the recommended list. The position that is undertaken is that of a white hacker. The intention is not to create an attack which would only harm the recommender systems (aim of black hat hackers). Instead the newly developed attack which should be immune to most of the detection mechanisms currently in use would show a vulnerability in the detection methods and so new improved algorithms should be built to detect these attacks.

1.2 Structure of this Report

The rest of this report is structured as follows. Chapter 2 will provide introductory materials on recommender systems. The predictive algorithm will be discussed briefly for completion but much more focus will be placed on different types of attacks attempted on recommender systems. The different ways of detecting these attacks will be given. A newly developed attack called Eigen attack has been developed in Chapter 3. The motivation and its limitations is also discussed. Chapter 4

gives all the experiments done on the different attacks and the common metrics to evaluate the performance of attacks will be explained. Comparisons between Eigen attacks and the other usual attacks are done. The end of Chapter 4 provides experiments on the ways to solve the limitation of Eigen attacks. These, in fact, constitute some initial further work. Finally, Chapter 5 concludes this report.

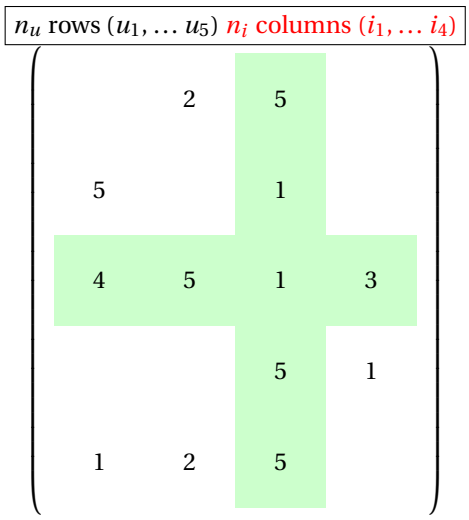
Chapter 2

Related Work

The primary focus of this chapter is to give the reader background materials on collaborative filtering. Since this project is mainly on the attack sides not much will be given on the prediction part but still sufficient information to understand the whole of this report. Related work on different attacks and how people in the past have prevented them from influencing the predictions will be discussed.

2.1 Collaborative Filtering

In 1992, Goldberg et al. coined the term collaborative filtering. Collaborative filtering is used to predict items users might prefer based on his or her previous likings. Those previous interests are cryptographically stored in a database to maintain the privacy of each individual. Intuitively, the database can be thought to be a matrix of n_u rows (total number of users, u) by n_i columns (total number of items, i). This matrix will, henceforth, be denoted by Υ. The sets of n_u users consist of a class of U = $\{u_1, u_2, ...u_{n_u}\}$ and the set of n_i items are represented by : I = $\{i_1, i_2, ...i_{n_i}\}$ where each subscripts refer to the each user, u and item, i respectively. Also, let I_{u_i} denote the list of items that user u_i has rated [23]. Consider Matrix 2.1 showing the difference between a user profile and an item profile.



Matrix 2.1 — User-item matrix, Υ showing the user and item profiles

From Matrix 2.1, the rows represent each user and the columns represent each item. So, there are 5 users and 4 items. The item profile of i_3 (highlighted column 3) is { (1,5), (2,1), (3,1),(4,5),(5,5) } whilst the user profile of u_3 (highlighted row 3) is { (1,4), (2,5), (3,1), (4,3) }. All those entries in Matrix 2.1 are ratings. Those items which were not given any scores are left blank in the matrix. Furthermore, collaborative systems can be based on implicit or explicit ratings. As the name suggests, in explicit rating the users' interests for a particular item might e.g. be rated on a scale of 1-5. In contrast, implicit rating focuses on the behavioural aspect of users. E.g. purchase history could be considered [24]. Nowadays, collaborative filtering is automated and, in the jargon, the software tool used to achieve this is called Automated Collaborative Filtering (ACF). ACF systems gather human opinions in terms of ratings (e.g.) and compare this database of items and ratings with other users which exhibit similar characteristics. The aim of ACF is to predict the rating(s) of specific items. To achieve this, usually, two modes of operations of ACF are identified: Prediction and Recommendation modes. In prediction mode, a numerical value bounded by the scale given by the system is predicted [23]. On the other hand, in recommendation mode a list of items the users might have a predilection to select

is given to the user. In brief, Automated Collaborative Filtering uses statistics together with ratings to find out users' preferences [25]. In the literature there are loads of ways to predict values based on similarities between user profiles or item profiles which can be done by using some similarity measures or some probabilistic methods or some matrix decomposition algorithms. An example of similarity measure is Pearson correlation which will be briefly defined in Section 2.3.1.3. Probabilistic algorithms will not be discussed and matrix decomposition such as singular value decomposition will be discussed since this is the state of the art predictive algorithm and it has been implemented in this project.

2.1.1 Predictive Algorithm using Singular Value Decomposition

Singular Value Decomposition (SVD) is a very popular factorization tool used in many applications ranging from oceanography to image processing to collaborative filtering. The reason lies predominantly in that it aims at achieving the best approximation of any matrix.

2.1.1.1 Background on Singular Value Decomposition

Singular Value Decomposition is a dimension reduction technique similar to principal component analysis (PCA discussed in Section 2.3.2) that aims at factorizing any matrices decomposing it into orthogonal components. Similar to PCA, SVD can be used to analyze multivariate data where the noise in the data are reduced. Consider the user-item matrix $\mathbf{\Upsilon} \in \Re^{n_u \times n_i}$. SVD is defined by equation (2.1)

$$\Upsilon = \mathbf{U\Sigma V}^{\mathrm{T}} \tag{2.1}$$

where $\mathbf{U}$ is an orthogonal or rather orthonormal n_u by n_i matrix and Σ is an n_i by n_i diagonal matrix where the main entries are the positive square root of the eigenvalues of $\mathbf{A}^{\mathrm{T}}\mathbf{A}$ or $\mathbf{AA}^{\mathrm{T}}$ and they are called singular values. An eigenvector, $\mathbf{v}$ exists if there is a real number, λ called the eigenvalue for which $\mathbf{A}v = \lambda\mathbf{v}$ where $\mathbf{A}$ is a symmetric positive semidefinite matrix (i.e. $\mathbf{bAb}^{\mathrm{T}} \geq 0$, $\forall$ row vector $\mathbf{b}$) [1]. $\mathbf{V}^{\mathrm{T}}$ is also an orthonormal matrix with size n_i by n_i. Rows of $\mathbf{V}^{\mathrm{T}}$ are called right singular vectors

whilst columns of **U** are called left singular vectors. The singular values, are usually arranged in descending order i.e. $\sigma_{u,i} \geq \sigma_{u+1,i+1}$ where u and i indexes the row and column of $\boldsymbol{\Upsilon}$.

Theorem 2.1.1 *Eckart-Young Theorem*

Consider a matrix $\boldsymbol{X}$ *of* $\Re^{n\times m}$ *with rank,* $r < min(m,n)$. *From* $\boldsymbol{X} = \boldsymbol{U\Sigma V}^T$ *with the singular values arranged in decreasing sequence then* $\exists$ *an* $m \times n$ *matrix* $\boldsymbol{X}_k$ *such that* $\boldsymbol{X}_k = \boldsymbol{U\Sigma}_k\boldsymbol{V}^T$ *which minimizes the Frobenius norm between the lower rank matrix,* $\boldsymbol{X}$ *and* $\boldsymbol{X}_k$.

According to Theorem 2.1.1 [12] SVD minimizes the Frobenius norm of $\mathbf{D} = \boldsymbol{\Upsilon} - \mathbf{U\Sigma}_k\mathbf{V}^{\mathrm{T}}$ defined by equation (2.2)

$$\| \mathbf{D} \|_{\mathrm{F}}^2 = \sum_{u=1}^{n_u}\sum_{i=1}^{n_i} | d_{u,i} |^2 = trace(\mathbf{D}\cdot\mathbf{D}) \tag{2.2}$$

where **D** has already been defined, $d_{u,i}$ refers to each element in the matrix, **D** found at row u and column i, n_u and n_i are the total number of users and items. The resulting matrix can be denoted by X_k which is called the low rank approximation matrix. The elements of $\boldsymbol{\Sigma}$ in the main diagonal are denoted by $\sigma_1....\sigma_{n_i}$ where $\sigma_b > 0$ for $1 \leq b \leq r$ where r is the rank of the matrix X and $\sigma_{b+1} = 0$ for $r < b+1 \leq n_i$.

Singular value decomposition is similar to eigenvalue decomposition for a square matrix. The relationship between SVD and PCA is clear when the principal components are calculated from the zero-mean covariance matrix of $\boldsymbol{\Upsilon}$. Eigen decomposition of a square matrix causes it to be split into its corresponding matrix of eigenvectors as given by equation (2.3).

$$\mathbf{A} = \mathbf{U\Lambda U}^{\mathrm{T}} \tag{2.3}$$

where $\boldsymbol{\Lambda}$ is a diagonal matrix of eigenvalues, **U** need not be an orthogonal matrix and **A** is a square matrix. Since the matrix $\mathbf{C} = \mathbf{E\Lambda E}^{\mathrm{T}}$ is symmetric positive definite (i.e. $\mathbf{bAb}^{\mathrm{T}} > 0$, $\forall$ row vector **b**) eigen decomposition becomes similar to SVD. In the case of **C** being the covariance matrix implies that **C** is symmetric (and is proportional to $\mathbf{X}\cdot\mathbf{X}^{\mathrm{T}}$). If this is the case, **E** will be orthogonal as shown in [10]. So, by doing an eigen

decomposition of the covariance matrix of $\mathbf{\Upsilon}$ the eigenvectors of $\mathbf{V}^{\mathrm{T}}$ are found. From the definition of Principal Components(PCs) it can be deduced that the PC of the elements of the covariance matrix will be equivalent to the eigenvectors of X [10]. The eigenvalues of $\mathbf{X}^{\mathrm{T}} \cdot \mathbf{X}$ are equivalent to σ_b^2. The eigenvectors can be found if their dot product with the rows of a square symmetric positive definite matrix, **A** and the resulting values are proportional to themselves [11].

2.1.1.2 SVD and Collaborative Filtering

The user-item matrix, $\mathbf{\Upsilon}$ is usually too large to work with in practice. Consider the Netlix[1] movie database which consists of over 480,000 users and about 17,000 movies. If the matrix was completely filled the number of entries will get close to 8.5 billion entries. However, this is not the case because of missing data. Even then, somehow, the algorithm will have to scale with such huge databases and it is computationally time-consuming to go through all these entries one by one. Many people have tried to use variation of SVD e.g. incremental SVD by Brandyn and weighted SVD so that by using a lower rank approximation of the whole matrix accurate predictions can be made. SVD is supposed to find the best features of the matrix e.g. users preferences and all those preferences which account little to the whole matrix are discounted. In the following sections, a brief explanation will be given on weighted SVD which have been considered to be robust to random and average attacks [14].

2.1.1.3 Weighted Low Rank Approximation

This is the original work of Nathan Srebo and Tommi Jaakkola [7]. Consider the rating matrix $\mathbf{\Upsilon}$ and the corresponding weight matrix, W $\in \Re^{n_u \times n_i}$ filled with ones and zeros satisfying the following conditions: $w_{u,i} = 1 \ \forall \ w_{u,i} \in$ R & $w_{u,i} = 0 \ \forall \ w_{u,i} \notin$ R where R $\in$ {1,2,3,4,5}. A lower rank-k approximation, $\Upsilon = \mathrm{UV}^{\mathrm{T}}$ to the original matrix is constructed so that the Frobenius norm error, E(U,V) which is defined by

[1] Netflix prize was a competition by Netflix to increase the accuracy of the predictions in collaborative filtering. The competition was opened from 2006 and has closed in July 2009 for the winner of the prize has already been found.

equation 2.4 is minimized.

$$E(U,V) = \sum_{u,i} W_{u,i}(\Upsilon_{u,i} - (UV^T)_{u,i})^2 \tag{2.4}$$

where W is the weight matrix, U and V are decomposed matrices, and u,i references elements in the corresponding matrices, Υ is the user-item matrix.

If the matrices U and V are orthogonal, the low rank approximation is similar to singular value decomposition. Weighted singular value decomposition does not calculate the eigenvectors of the user-item matix but instead tries to find the global minima. In an attempt to minimize such an error a gradient descent algorithm is applied to the error. So, the partial derivative of U and V are computed. From [7] denote $U_V^* = \arg_U \min E(U,V)$. It is shown that to solve for U separate pseudo-inverse is needed for each row $(U_V^*)_u$ of U_V^* and this is given by equation 2.5 [7]

$$(U_V^*)_u = (V^T \cdot diag(W_u) \cdot V)^{-1} V^T \cdot diag(W_u) \cdot \Upsilon_u \tag{2.5}$$

One of the optimization method [7] makes use of the idea that weighted low rank approximation can be solved by assuming the problem is similar to the maximum-likelihood problem with missing values. The problem can be simplified by assuming the following linear regression equation $\Upsilon = X + G$ where G is a Gaussian noise and X is the low rank matrix. Perhaps the most well-known way to calculate the maximum-likelihood is to use Expectation-Maximization (EM) algorithms. As the name suggests there are two steps: Expectation and maximization. In the expectation step EM is used to maximize the expected log-likelihood of the matrix filled with estimated values for the missing ratings. The authors of [7] show that in the case of missing data, the expectation step fills in values from the current estimate of the approximated matrix, X. In the maximization step a new approximated matrix is derived from the low-rank approximation of the mean of newly created matrix. The EM update is then given by equation 2.6.

$$X^{t+1} = svd(W \otimes \Upsilon + (1 - W) \otimes X^t) \tag{2.6}$$

where t is the current step and $t+1$ is the next step, W is the weight matrix, X is the

approximated matrix, Υ is the user-item matrix, $\otimes$ is the element by element multiplication also called the tensor product. For more involved mathematical details about the algorithm the reader is referred to [7]. Weighted svd was implemented in Matlab and more information on the experimental setup is given in Chapter 4.

Whichever be the methods of predictions malicious people and some producers have always tried to promote the products they want to sell. For years people have tried to increase the accuracy of predictions whilst still trying to immunize the collaborative algorithm against attackers.

2.2 Attacks on Collaborative Filtering

There are considerable efforts being made to prevent users from shilling collaborative filtering algorithms. Since collaborative filtering systems are originally conceived based on the opinions of a large number of people, it could be adversely affected by fake user profiles, the owner of which has only one primary goal: move the poorly rated target items into the top-n recommender list as already pointed out [16]. From [15] each profile can be thought of consiting of ratings to four sets of items:

- A singleton target item i_t,
- A set of selected items determined by the attacker, i_s
- A set of filler items, i_f
- A set of unrated items i_ϕ

Recommender systems are usually based on users first being registered before providing ratings. The rating system will then analyze whether the username is associated with the rating of a particular product. If this is the case, the item cannot be rated several times by the same user having a single username. One of the easiest way to attack such a recommender system is to have the users create multiple accounts and insert their preferred ratings. Such an attack is called *shilling attack* or *profile injection attack*. This can be divided into either push attacks (increase the overall rating of a target item, i_t) or nuke attacks (decrease the overall rating). In this

project only the former is considered. Some of the possible attacks which can be used to achieve this are listed below:

1. Random Attack
2. Bandwagon Attack
3. Average Attack

2.2.1 Random Attack

Random attack (and also Average attacks) was first proposed by Shyong Lam and John Riedl [17]. In a random attack, the attacker provides ratings to items (excluding the target item) around the mean and standard deviation of all the ratings in the database. It is assumed that the noise added to the overall mean rating follows a normal distribution [17]. The target item(s) is/are rated with the extreme values (e.g. on a scale of 1-5 either 1 is injected to nuke the rating or 5 is injected to push the rating). Consider Matrix 2.2 as an example.

Matrix 2.2 — User 3 is the attacker (row 3)

$$\Upsilon = \begin{pmatrix} 2 & 3 & 4 \\ & 2 & 4 \\ - & - & ? \end{pmatrix} \qquad \text{(Matrix reference)}$$

From Matrix 2.2 the attacker willing to push the rating of the target item 3 (denoted by ?) will need to have access to the overall rating, which is given by equation (2.7)

$$\bar{r} = \frac{2+3+4+2+4}{5} = 3 \tag{2.7}$$

The attacker will now fill the last row with the rating of 3 around the standard deviation. If the malicious user would like to decrease the overall rating of item 3, he or she might inject 1 in the item's profile to nuke the overall rating. This will cause

the average rating of item 3 to drop from 4 to 3. The resulting matrix is shown as Matrix 2.3.

Matrix 2.3 — Attacking item 3 - After Random Attack

$$\Upsilon = \begin{pmatrix} 2 & 3 & 4 \\ & 2 & 4 \\ 3 & 3 & \mathbf{1} \end{pmatrix} \qquad \text{(Matrix reference)}$$

2.2.2 Bandwagon Attack

Bandwagon attack does not need the same amount of resource as average attacks which is discussed in section 2.2.3 [16]. So, average ratings of each item is not known. Instead ratings will be selected to be around the mean of the overall mean ratings of the items in the database with subsets of the popular items rated highly. As usual, the target item is given a specific rating depending on the intent of the attack as already described. So, this attack requires knowledge of the overall mean of the ratings and knowing which items are popular.

2.2.3 Average Attack

Average attack is more difficult to implement by the adversary because the rating of each item to be filled is needed [17]. Again, excluding the targeted items, the other remaining items (or subsets) are given the average ratings of each of their correponding item profile added with some gaussian noise. In this way, it will be more difficult to filter out the attacker because the fake profiles will be closer to the other genuine user's profile. As an example, consider Matrix 2.2 again. The average rating of each item is computed. Filling the matrix with the mean of each item and targeting item 3, Matrix 2.4 results. In practice, the exact value for the mean of each item will rarely be filled.

Matrix 2.4 — Attacking item 3 - Average Attack

$$\Upsilon = \begin{pmatrix} 2 & 3 & 4 \\ & 2 & 4 \\ 2 & 2.5 & \mathbf{1} \end{pmatrix} \quad \text{(Matrix reference)}$$

Average attacks have always been considered to be the one which is the most difficult to detect and it also requires greater knowledge on the mean ratings of each item [14]. In this project, experiments have been done on random, bandwagon and average attacks though it should be noted that very few research papers provide experiments on Bandwagon attacks. Note that the attacker does not have access to the matrix but the overall rating or the average ratings should be available which is usually the case. E.g. consider the website IMDB (Internet Movie Database) which gives the average ratings of each movie.

Due to all these different types of attack, through years researchers have tried to find different ways to detect and discard them in predictions. In the next section the different detection methods to achieve this will be described.

2.3 Detection Algorithms

The purpose of this section is to give some details on the different detection methods documented in the literature. These are split into two categories:

1. **Detection Methods in the Original space**
    - Rating Deviation from Mean Agreement (RDMA)
    - Weighted Deviation from Mean Agreement (WDMA)
    - Degree of Similarity with Top Neighbours (DegSim)
    - Rating Missing At Random (RMAR)
    - Rated Items Consistency (RIC)

2. **Detection Method in Low Dimensions**

    - Variable Selection

2.3.1 Detection Methods in the Original space

In this section some of the detection methods used in the original space will be explained.

2.3.1.1 Rating Deviation from Mean Agreement

Rating Deviation from Mean Agreement (RDMA) was developed by Chirita et al. [20]. The idea is based on the fact that attackers have a tendency to rate away from the average. Equation (2.8) shows how RDMA is computed.

$$\text{RDMA} = \frac{1}{n_{\text{I}_u}} \cdot \sum_{i=0}^{n_{\text{I}u}} \frac{\mid r_{i,u} - \bar{r}_i \mid}{n_{ip}} \tag{2.8}$$

where n_{I_u} is the number of items users, u have rated, $\bar{r}_i$ is the average rating of item i, n_{ip} is the total number of items in a specific item profile, $r_{i,u}$ is the rating given by user, u to item, i. Equation (2.8) shows that RDMA calculates the deviation of the users' ratings from the average rating of an item which is weighted by the total number of items in a given item profile. Attackers who will try to push ratings of disliked items will have to insert high values of ratings for those targeted items. This means that the numerator of equation (2.8) will be high. Conversely, for normal users the RDMA value should be expected to be lower. High values of RDMA indicate attacker's profiles while low values indicate true user profiles since users have a tendency to remain similar to the mean. Obviously, those users who rate away from the mean will be identified as being attackers. The high probability of obtaining these false positives is what makes people less inclined to use RDMA on its own.

2.3.1.2 Weighted Deviation from Mean Agreement

Weighted Deviation from Mean Agreement (WDMA) is very similar to RDMA but instead of providing a linear weight on the deviation of the ratings from the average

the weight is quadratic as shown by equation 2.9 [21]. This causes the the summation part to be larger for unpopular items than for popular items.

$$\text{WDMA} = \frac{1}{n_{\text{I}_u}} \cdot \sum_{i=0}^{n_{\text{I}_u}} \frac{\mid r_{i,u} - \bar{r}_i \mid}{n_{ip}^2} \tag{2.9}$$

where n_{I_u} is the number of items users, u have rated, $\bar{r}_i$ is the average rating of item i, n_{ip} is the total number of items in a specific item profile, $r_{i,u}$ is the rating given by user, u to item, i.

2.3.1.3 Degree of Similarity with Top Neighbours

Degree of Similarty with top neighbours (DegSim) [21] was motivated from the fact that attack profiles tend to be close to each other since they are usually generated from the same process.

$$\text{D}egSim = \frac{1}{n_k} \cdot \sum_{u=1}^{n_k} \omega_{pearson}(u,s) \tag{2.10}$$

where n_k is the total number of neighbours and u and s are users. Equation (2.10) shows that the average similarity is calculated between the user u and its top-n_k neighbours. $\omega_{pearson}(u,s)$ is a measure of similarity between two users u and s. It is defined by equation (2.11).

$$\omega_{pearson}(u,s) = \frac{cov(r_u, r_s)}{n_{tot} \cdot \sigma_u \cdot \sigma_s} \tag{2.11}$$

where n_{tot} is the total number of items being rated, $cov(r_u, r_s)$ is the covariance of the ratings of two users r_u and r_s and σ_u and σ_s are the standard deviation of the ratings given by user u and user s.

2.3.1.4 Rating Missing At Random

Rating Missing At Random (RMAR) was developed in [22]. The idea is based on the assumption that attack profiles are missing at random. This means normal users have a tendency to rate items they like or dislike i.e. most ratings are both on the extremity of the rating scale. The equation to compute RMAR is given by 2.12

$$\text{RMAR} = -\frac{1}{\binom{n_i}{2}} \cdot \sum_{j=1}^{n_i} \sum_{j+1}^{n_i} \omega_s(i_j, i_{j+1}) \tag{2.12}$$

where $\omega_s(i_j, i_{j+1})$ is similar to Pearson correlation between two different items, i_j and i_{j+1} and is defined by equation (2.13)

$$\omega_s(i_j, i_{j+1}) = \frac{\sum_{u=1}^{\text{U}} (r_{u,i_j} - \bar{r}_u)(r_{u,i_{j+1}} - \bar{r}_u)}{\sqrt{\sum_{u}^{\text{U}} (r_{u,i} - \bar{r}_u)^2} \sqrt{\sum_{u}^{\text{U}} (r_{u,i_{j+1}} - \bar{r}_u)^2}} \tag{2.13}$$

where r_{u,i_j} and $r_{u,i_{j+1}}$ are the ratings given by user u on items i_j and i_{j+1}, $\bar{r}_u$ is the average rating of an item profile. Authentic users will have positive RMAR values whilst attackers will be identified with negative ones.

2.3.1.5 Rated Items Consistency

Rated Items Consistency (RIC) is again taken from [22]. It is similar to RMAR except that the similarity measure, ω_s is weighted and there is no negation. With RMAR if two items are sufficiently different RMAR will not be able to distinguish that from an attacker. In RIC the similarity measure is weighted to decrease the effect of the latter. RIC is defined as equation (2.14).

$$\text{RIC} = \frac{1}{\binom{n_i}{2}} \cdot \sum_{j=1}^{n_i} \sum_{j+1}^{n_i} \omega_s(i_j, i_{j+1}) \cdot \frac{max(r) - | r_j - r_{j+1} |}{max(r)} \tag{2.14}$$

where r_j and r_{j+1} are the ratings of two different items, n_i is the total number of items, ω_s is the similarity measure already defined above. Authentic users will have a higher positive RIC value compared to the attacker.

A few years ago a detection mechanism based on principal component analysis has been implemented. So, it's important to show its strength at detecting attacks.

2.3.2 Detection Method in Low Dimensions

It is also possible to detect attacks in the low dimension. This has been done by authors of [2] where their algorithm has been based on Variable Selection which in fact uses Principal Component Analysis (PCA). A brief overview on PCA will now follow.

2.3.2.1 Introduction to Principal Component Analysis

In Principal Component Analysis (PCA) a series of independent (orthonormal) random variables are derived from the linear combination of dependent (correlated) random variables. The variables which are related to one another (yet uncorrelated to the others) are combined into one single component (in the jargon it is called principal component). In brief, PCA helps to reduce the dimensions of the original data set whilst helping to find out any hidden patterns between the random variables. Before explaining how PCA is used in detecting attack profiles, some background material related to this dimension reduction technique will be given.

2.3.2.2 Idea behind Principal Component Analysis

Consider the user-item matrix, $\boldsymbol{\Upsilon}$ again. The elements of the matrix denote the ratings of each item given by some specific users. If the items were to be plotted, they will form a cloud of points in n_u space where n_u is the total number of users. Each item can be represented by a column vector and each point will be in n_u dimensions. The idea behind PCA is to project the n_i points from n_u dimensions to a lower dimension where the main characteristics of the original data are preserved. This is represented by equation (2.15) which shows that $\boldsymbol{\beta}$ has tranformed $\mathbf{x}$ to another basis, **PC**.

$$\mathbf{PC} = \boldsymbol{\beta}^{\mathrm{T}} \boldsymbol{x} \tag{2.15}$$

So, equation (2.15) can be rewritten as equation (2.16).

$$\mathrm{PC}_i = \beta_{i_1}(x_i) + \beta_{i_2}(x_2) + \ldots + \beta_{i_k}(x_k) \tag{2.16}$$

where PC_i is the subject's score on *principal component* i (the i^{th} component extracted), β_i is the principal coefficient for each observed variable from i^{th} principal component, x_i is the actual value on the observed variable.

Each PC_i accounts for the maximum variance in the data set which has not been taken into account by the previous principle components. Ideally, each principle component is correlated with at least a non-overlapping part of the data set. The differences between PC_i and PC_{i-1} are the following. The latter will take into account the variances which have not been taken into consideration by the former and, additionally, both of them should not be related to each other. The principal components are calculated by following these few steps [3]:

- Calculate the covariance matrix, $\mathbf{C}_{\bar{\mathrm{I}}}$, of the zero-centered (subtract the mean) matrix $\boldsymbol{\Upsilon}$, which is defined by equation (2.17)

$$\mathbf{C}_{\bar{\mathrm{I}}} = \frac{\mathbf{I} \cdot \mathbf{I}^{\mathrm{T}}}{n_u - 1} \tag{2.17}$$

  where each element of $\mathbf{C}_{\bar{\mathrm{I}}}$, c_{ij} represents the covariance between $\mathbf{I}_i$ and $\mathbf{I}_j$ dimensions. The covariance of two random variables is zero only if the two variables are uncorrelated and it the same as the variance if the two random variables are equal. In calculating a principal component the amount of redundancy in the data should be decreased. This immediately implies that the covariance matrix of each principal component would be diagonalized. Consequently, only the variances will remain (the effects of the other variables will be zeroed).

- Find the eigenvectors of the covariance matrix, $\mathbf{C}_{\bar{\mathrm{I}}}$.

- Select the preferred eigenvectors based on the eigenvalues to form the feature vectors. Usually, the eigenvectors corresponding to the highest eigenvalues are selected because these contain the most amount information in terms of variance.

Technically, PCA can be computed by doing an eigen-decomposition of the covariance matrix [4]. Eigen-decomposition as explained in Section 2.1.1.1 factorizes a

square matrix into its corresponding eigenvectors and eigenvalues found in the entries of the diagonal matrix. Equation (2.18) is the standard equation for computing eigen-decomposition of a square matrix.

$$\mathbf{A} = \mathbf{Q}\mathbf{\Sigma}\mathbf{Q}^{\mathrm{T}} \tag{2.18}$$

where $\mathbf{A}$ is any square matrix, $\mathbf{\Sigma}$ contains the eigenvalues and $\mathbf{Q}$ is a square matrix. Equation (2.19) shows that if $\mathbf{Q}$ is chosen to be the eigenvectors, $\boldsymbol{\beta}$ then C_{PC} is diagonalized [8].

$$\begin{aligned} C_{PC} &= \frac{\mathbf{PC}\cdot\mathbf{PC}^{\mathrm{T}}}{n-1} && (2.19)\\ &= \frac{\boldsymbol{\beta}\cdot\boldsymbol{x}\boldsymbol{x}^{\mathrm{T}}\cdot\boldsymbol{\beta}^{\mathrm{T}}}{n-1} && (2.20)\\ &= \frac{\boldsymbol{\beta}\cdot(\boldsymbol{\beta}^{\mathrm{T}}\mathbf{\Sigma}\boldsymbol{\beta})\cdot\boldsymbol{\beta}^{\mathrm{T}}}{n-1} && (2.21)\\ &= \frac{\mathbf{\Sigma}}{n-1} && (2.22) \end{aligned}$$

Mehta et al. [2] have used the idea that eigenvectors (in PCA) point in the direction of the most variance to detect the fake profiles. The algorithm they have built is related to the conventional "Variable Selection" (VarSelect) algorithm used in machine learning.

2.3.2.3 Motivation of Variable Selection

Attackers usually work in groups. The size of the groups is referred to as attack size which is a fraction of the total number of users in the database. The effects of individual users might not be easily detected because some authentic users might appear to be similar to the attackers due to their atypical choices. Even if this was the case, it is important to note that the effect of one user on a database consisting of thousands of users (i.e. an attack size << 1%) will most probably not be sufficient to push or nuke the rating of even a not so popular target item. It is expected that one highly correlated cluster representing the attack profiles will be distinguished from

the normal users [2]. VarSelect is capable of not only identifying this cluster but also one single attacker although this will depend on the first few principal components.

2.3.2.4 Variable Selection

Variable Selection (VarSelect) has especially been applied to datasets containing over thousands of variables. Through VarSelect, the features extracted should be representative of the whole original datasets. One of the means of achieving this is to use PCA [3] to decrease the dimensions while still retaining the main hidden characteristics. Since the attackers usually contribute little to the whole of the matrix because of the low covariance of the data (compared to the normal users) inserted in the matrix, it is very likely that PCA will discard most of the fake profiles [2]. Using the notation from [2], users are interpreted as variables and items represent observations. The principal components in PCA will point more towards the authentic users. To detect the fake users it is just necessary to select the users with the least covariance. Finally, only a subset of the total number of variables (variable selection) is selected [9].

Use of VarSelect in the detection is mainly based on the idea that attack profiles tend to be highly correlated which is indeed the case if only one of the different types of attacks is done at a given time. E.g. consider the random attack where the new attack profiles are filled with items rated around the mean of the whole ratings in the database. This implies that such profiles will tend to be very similar to one another. This is also the case with average attacks where the variance of the attackers' profiles will be small (since each attacker will rate around the mean of each item). The implication is that attack profiles as a whole tend to add very little information to the user-item matrix, Υ. This means when VarSelect is applied to Υ, a definite cluster representing the attacker will be visible compared to the data without the fake profiles. Based on this, the algorithm to calculate VarSelect is given as Algorithm 2.1

Algorithm 2.1 — VarSelect algorithm

for u=1 to n_u

$\Upsilon_{zscore} \leftarrow \frac{\Upsilon_u - \mu_u}{\sigma_u}$

where u indexes the row of Υ and μ_u and σ_u are the

mean and standard deviation of each user

end

transpose Υ_{zscore} so that users are variables and items are observations

Calculate the covariance matrix of Υ_{zscore}

Calculate the eigenvectors

Calculate the principal components

Select the first three principal components i.e. PC_1, PC_2 and PC_3

$PC_{sum} \leftarrow$ Compute $|PC_1|^2 + |PC_2|^2 + |PC_3|^2$

Sort the variables/users in ascending order i.e. sort all the PC_{sum}

From this sorted list select top-m users which will be the attackers

The attack profiles are selected by understanding that the $Var(PC_i) = \lambda_i$ where $Var(PC_i)$ denotes the variance of the i^{th} principal component [3] and λ_i is the i^{th} eigenvalue. This means that each row of $\boldsymbol{\beta}$ will account for the variance of that row (refer to equation (2.15)). As the fake profiles have low covariance [2], each will contribute little to the principal component. So, the sum of the row of $\boldsymbol{\beta}$ for the attack profiles will be small, thus enabling their detections. If no a priori information is known about the size of the attack profiles, the top-m variables can be selected by looking for the peak in the F1 score as will be shown in Chapter 4. When this algorithm was applied to the user-item matrix without any fake profiles and the first two components were plotted, Figure 2.5 was obtained.

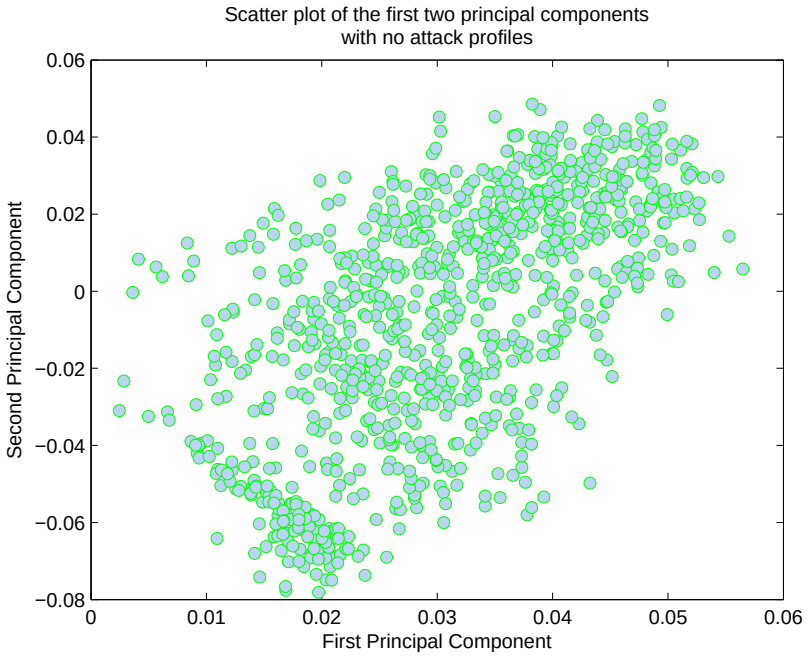


Figure 2.5 — The Scatter plot of the User-Item matrix without any attack profiles

The purpose of Figure 2.5 is to enable the reader to compare it with the other related figures illustrated Chapter 4. Mehta et al. [9] has demonstrated that VarSelect can detect random and average attacks. Authors of [9] have not provided any experimental results of the performance of VarSel on Bandwagon attacks. This is, however, done in this project. In the next chapter a new idea to develop a novel attack will be demonstrated.

Chapter 3

Eigen Attack

This chapter will develop a new attack called Eigen Attack which is designed from a low dimensional space which has not been done before. This will be the main contribution to this MSc project. The main motivation behind this new attack will be mentioned and its limitation will be explained. Different ways to solve these difficulties will be investigated.

3.1 Motivation of Eigen Attack

Random, Bandwagon and Average attacks have all been created from the original space. Consider Table 3.1 which shows how well VarSelect can detect random, Bandwagon and average attacks (in the low dimensions).

Table 3.1 — Comparison of the performance of the different attacks

		Evaluation Methods	
		VarSel detection rate in %	Hit Ratio in %
Attacks	Random Attacks	100	10
	Bandwagon Attacks	0-100 (depends)	0
	Average Attacks	99.0	30

Table 3.1 is just a general overview of the performance of random, Bandwagon and

average attacks after using VarSelect algorithm. This algorithm easily detects random and average attacks. Though only part of Bandwagon attacks are detected they don't affect the predictive algorithms based on weighted singular value decomposition and so are ineffective. Now, it will be shown that it is possible to design another attack which will be able to remain undetected but still influence the ratings of items to such an extent that the hit ratio is close to ideal. This attack called Eigen attack is different from all the other attacks because it is not created in the original space but rather in the low dimensions and those data are, finally, projected to the original space. If this attack is successful it will be important for people in the community to develop new detection mechanisms to prevent the attack from affecting the ratings of recommender systems.

3.2 Implementation of Eigen Attack

The current scenario for random attacks is shown in Figure 3.1 which is a representation of the attack profiles in low dimensions. Similar plots result for average and Bandwagon attacks. Figure 3.2 shows the result that the new attack should be able to achieve i.e. there should not be any clustering of the attack profiles in the low dimensions.

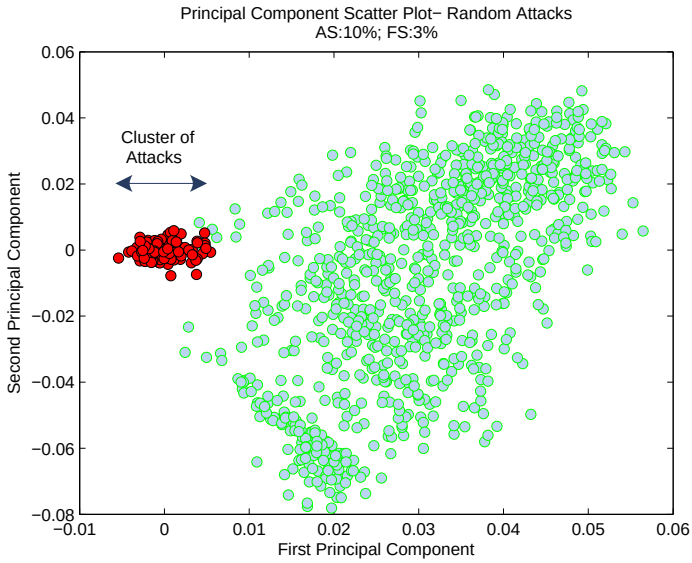


Figure 3.1 — Scatter plot of the first two principal components for random attacks

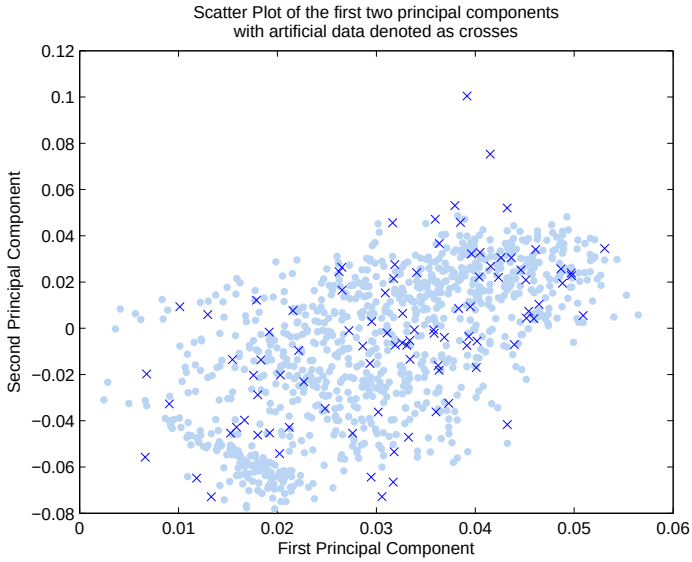


Figure 3.2 — Crosses showing the fake profiles in low dimensions superimposed on the original data (Eigen Attacks)

Figure 3.1 identifies two main regions: cluster of attack profiles (**A**) in red and the user profiles scattered mostly away from zero denoted as green circles (**B**). In fact, **A** is the main region where the attacks are usually found in the low dimensional model. The magnitude of the principal components are usually low which is why VarSelect is successful at detecting attacks which cluster points at **A**. In contrast, **B** (green circles) is the region where the true users are found. The idea is to design a new attack in such a way that the first few magnitudes of the principal components are comparable to those derived from the user-item matrix, Υ. Hence, in the ideal scenario the points should follow almost the same distrbution as the true users as shown by the crosses in Figure 3.2. Instead of solving the problem by generating fake profiles in the original space which should in the end have the coordinates of the principal components clustered with the true data, the problem has been approached in the other direction. The fake data will be generated from the low dimension instead of generating the data in the high dimensions directly. So in a way, the attack exploits the direction in which the data are scattered in the low dimensional model. Basically, the main idea is based on making sure the fake data all lie almost in the direction of the main eigenvectors. In the end the fake profiles should all be spread over the user profiles in low dimensions as shown in Figure 3.2. The algorithm to achieve Eigen attack is given as Algorithm 3.2.

Algorithm 3.2 — Algorithm showing an Implementation of Eigen Attack

Input: User-Item Matrix, Υ

Output: Attack Profiles

for u=1 to n_u

$\boldsymbol{\Upsilon}_{zscore} \leftarrow \frac{\Upsilon_u - \mu_u}{\sigma_u}$

where u indexes the row of Υ and μ_u and σ_u are the

mean and standard deviation of each user

end

transpose $\boldsymbol{\Upsilon}_{zscore}$ so that users are variables and items are observations

$cov(\Upsilon_{zeromean}) \leftarrow$ find the covariance matrix of Υ_{zscore}

calculate the eigenvalue and eigenvector of $cov(\Upsilon_{zscore})$

select the top-n most significant principal component

calculate the mean and standard deviation of each principal component

use a Gaussian model to generate the data close to the first n principal components

convert the data to the high dimention by multiplying by the score

multiply by σ_u and add μ_u (reverse steps for calculating zscore)

target item's rating $\leftarrow$ high ratings to push

if elements of profile is not within range of 1-5

$0 \leftarrow r \leq 0.5$

$1 \leftarrow 0.5 < r < 1$

$5 \leftarrow r > 5$

end

Algorithm 3.2 which exploits the low-dimensional space has been implemented to ensure the first few points of the principal components are all within a certain range. Having carefully selected these points, the artificial data are projected back to the upper $m \times n$ dimensions. The results are shown in Figure 3.2 which shows the combination of the data from the fake profiles (crosses) and the data from the true user (filled circles).

Since the fake data was generated from a Gaussian distribution some of the crosses

(fake data) are not in the main user space. This can be solved by using a lower standard deviation. These points, however, did not cause any significant decrease in accuracy when all the different metrics described in Chapter 4 were used for testing the power of Eigen Attacks. Eigen attacks have a main disadvantage though as will be explained in the next section.

3.3 Limitations of Eigen Attacks

In this section the limitation of Eigen attacks is explained. Two propositions are made to resolve this issue.

3.3.1 Problem

Eigen attacks can only be implemented if the whole user-item matrix, Υ is known. Since it is very likely that this matrix will not be available in practice, Eigen attacks will be difficult to implement. To resolve this issue two solutions are provided:

1. Take samples of the user-item matrix and investigate whether Eigen attack is still feasible for small sample sizes and whether it causes significant increase in the ratings of targeted items. Some experiments have been done for sample sizes of 1%, 5% and 10%.

2. Deduce an approximation of the user item-matrix, Υ. To achieve this care must be taken to select the appropriate inputs to be fed into the recommendation systems. These combined with the recommendation lists from the output of the collaborative filtering algorithm proved to be a good attempt at finding out some pattern in the original matrix.

3.3.1.1 Deduction of an Approximate Matrix

This final section will focus on finding a deterministic model which will be able to approximate part of the matrix in the user-item database, Υ. Figure 3.3 forms the basis of the idea where the recommender system is modelled as a black box problem.

This means no information on the algorithm being used is given. The only information given are the average rating of each item and the popularity of the items. In realistic situations, the users will feed some inputs (items with ratings) to the recommender system (shown as IN in Figure 3.3) after which the recommender system would provide a list recommendations which would be similar to some other users' in the matrix. This is denoted by OUT in Figure 3.3.

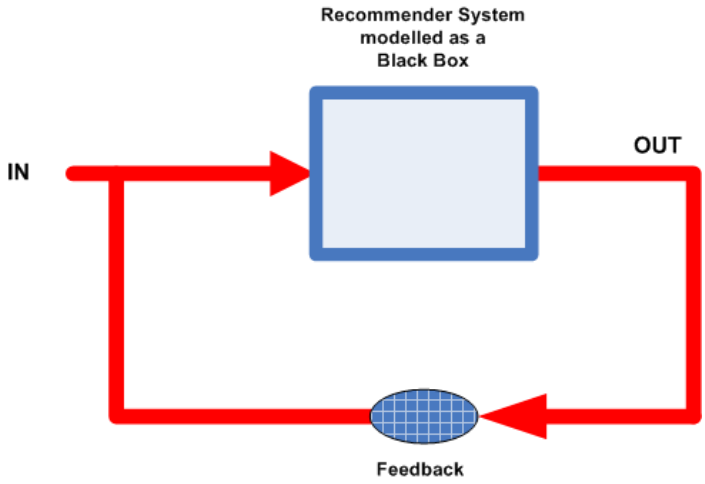


Figure 3.3 — Modelling of the Recommender System

Further to this interpretation, the outputs can be modified and fed back to the inputs. It is hoped that through an iterative process the recommendations will become closer and closer to a particular user in the database. Now, the first step is to identify what will be the inputs to the recommender systems. This is a very important issue which has to be addressed. The initial inputs should be selected in such a way that the recommender systems provide accurately some recommended items. For example, consider this situation. An input consisting of only one rating for the target item. Because of lack of knowledge on this input, the collaborative algorithm will not be able to sufficiently narrow down the recommended items, thus decreasing the accuracy. For this reason it was decided to create a specific algorithm which should already be quite close to some user profiles in the matrix.

3.3.1.2 Inputs to the Recommender Systems

The most popular items in the 943 users by 1682 items needs to be known. From this list, the items were ordered into at least four groups. The subsets were selected as follows: 1-100, 101-400, 401-1000, 1001-1682. From the first and second groups, a large percentage of the attack size is selected. This should, in principle, increase the similarity between users and input profiles since items that most people like are also included in the inputs. Once these have been selected, the average rating of each item needs to be known. Based on this average, different distributions of ratings and items were created. These are shown in Table 3.3 where r is equal to the non-zero values of $\Upsilon_{u,i}$ and $r \subset \mathrm{R}$ where $\mathrm{R} \in \{1,2,3,4,5\}$.

Table 3.3 — Different Rating Distributions for Different Range of Ratings

Range of Ratings	Distributions of Ratings
$r < r* < r+0.3$	positively skewed distribution
$r+0.3 < r* < r+0.7$	Gaussian distribution with standard deviation 0.6
$r+0.7 < r* < r+1$	negatively skewed distribution

An example will help to illustrate the intuition behind this idea. Consider an average rating of 2.8 to a given item. It is likely that most users have given the ratings of 3 and to a lesser extent ratings of 2. Since, there would also be some atypical users the distributions in Table 3.3 (in this case the third row) will also take this into account by assigning extreme ratings of 1, 4 and 5 with a lower probability. Analysis of the fake input profiles created from the above idea showed that the overall distribution of the ratings approached a negatively skewed distribution. In fact, this is to be expected because the mean of the ratings of the whole user-item matrix, $\boldsymbol{\Upsilon}$ (without the attack profiles) is 3.5 which implies that the highest probability of occurrence of the ratings should be expected to be centered around 3.5. Figure 3.4 shows the histogram of all the ratings from the authentic users.

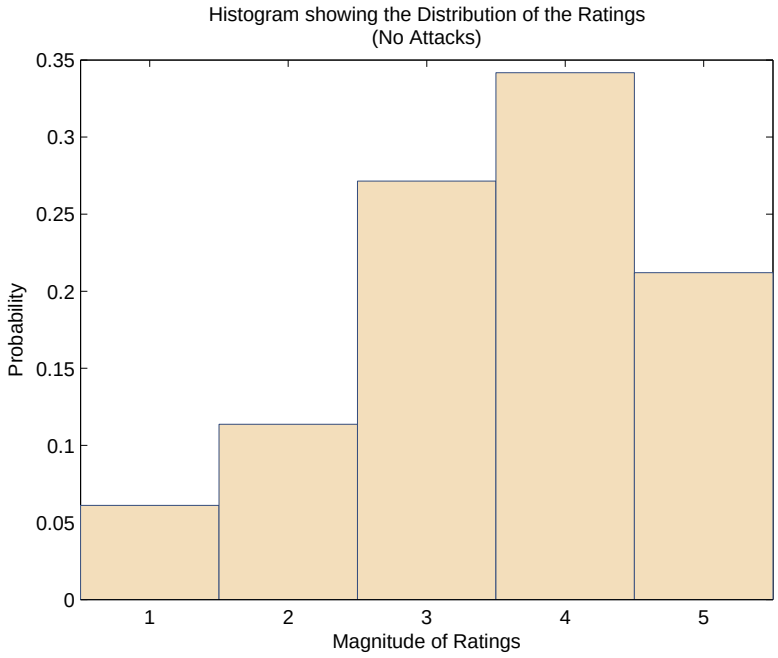


Figure 3.4 — Histogram of the overall distribution of authentic ratings with $\mu = 3.53$, $\sigma = 1.13$ and coefficient of variation (cov) $= 0.32$

Figure 3.4 shows the distribution of the ratings of items. The distribution is mostly negatively skewed with most people providing ratings of 4 and 3. The coefficient of variation (cov) is the ratio of the standard deviation to the mean. Algorithm 3.4 explains how the new new inputs can be created.

Algorithm 3.4 — Implementation of Inputs to Recommender Systems

Input: User-Item Matrix, Υ

Output: Inputs to Recommender Systems (AP)

for i = 1 to AP

Sort the most popular items in ascending order (most popular at the top)

From the top-m(100) items choose randomly $t1$ items i.e. $\binom{100}{t1}$

From the next top-m (300) items choose randomly $t2$ items i.e. $\binom{300}{t2}$

For the next 800 items choose randomly $t3$ items

For the remaining items choose randomly $t4$ items

Find the average ratings, μ_i of the chosen items

AP_i = For each range of μ_i choose distribution from Table 3.3

end

3.3.2 Important Note

It might seem the inputs bear some similarity to attack profiles from Bandwagon attack which also uses popular items. However, there are many differences and these are listed as follows. Bandwagon Attacks give popular items high ratings. Here, this might not be the case because it uses the average of each item. A poorly rated item might still be very popular. Also, bandwagon is based on the overall mean of the items in the database. Bandwagon attacks use one distribution which is usually selected to be a gaussian distribution. On the other hand, profiles injected into IN use three distributions which are the two skewed distributions and a gaussian distribution. Care has been taken to ensure that items in the fake profiles going into the recommender systems have been categorised into one of the four ranges such that even the not so popular (but still quite popular items) are rated. Other variations are still possible to account for the fact that some users deviate from the usual negatively skewed distribution. Since, the inputs have already been created the recommender system can now be used to determine the user-item matrix. Once an approximate version has been found, Eigen attack can be implemented.

Algorithm 3.5 — Exploiting the Recommender Systems

Input: Select fake profiles, n_{fp} from Algorithm 3.4
Output: Fake profiles
for 1 to n_{fp}
  Target item, i_t in profiles ← ratings of 1 or 2
  Rate the recommended items highly
  Copy all items recommended from OUT into IN
end
apply Eigen attack Algorithm

Algorithm 3.5 is purposely made simple to enhance understanding. So, once inputs have been created, the values of the target items are set to 1 or 2 (not 4 or 5). If this is done then the recommender system will have a better chance of finding all those users which are closer to the attacker i.e. all those users who have rated the target items poorly. Once these outputs have been created, they are given high ratings and fed back to the inputs. This process is repeated until no more than five different items are being recommended. It is hoped that the final outputs will be similar at least to some of the users who did not like the target item. From these outputs, which most probably is representing part of the matrix, the algorithm of Eigen Attack would be used. Initial results will be given in Chapter 4.

Chapter 4

Experiments

In this chapter all the experimental results done on random, Bandwagon, average and Eigen attacks will be provided. This analysis will show that Eigen attacks are not detected as easily as all the others. Comparisons will also be done with random, Bandwagon and average attacks. Further experiments have been done on approximating the user-item matrix.

4.1 Experimental Data Sets

For testing purposes, the free movieLens data sets have been used. This data was collected from a public trial conducted in a seven week period in 1996 by GroupLens. Two hundred and fifty users had provided a total of 47,569 ratings and a trimmed down version has been provided for free. This version consists of 943 users and 1682 movies with 100,000 ratings. Half of this was used for training purposes and the remaining was used for testing purposes. The mean of the overall ratings (in the movieLens data) were found to be 3.53 with standard deviation of 1.05. These were used to assign ratings to different items to best simulate the overall distribution of the user-item matrix. The number of items filling a user profile determines the **filler size** (FS) or profile size and the fraction of attack profiles injected is a proportion of the total number of users in the database. This fraction is related to **attack size** (AS).

It's good to point out that the JAVA codes for RDMA, WDMA, DegSim, RIC, RMAR were implemented from the last year MSc student and these were used. However, all the experimental evaluations in this project were done in Matlab by the author. The prediction part of the algorithm was based on weighted singular value decomposition algorithm and was implemented in Matlab by the author. The authors of [7] explain that initializing the low rank approximation matrix, X to zero proved to be a good choice in the optimization algorithm. Thus, it was decided to use this in the initialization step. The rank was set to 7 and the number of iterations before accepting complete convergence to the minima was set to 20. For the movieLens data of 943 × 1682 doing such an intensive computations which include the singular value decomposition cost quite some time to perform. This method can converge to a local minimum instead of a global minimum which is a problem of this algorithm. The algorithm was first run without any attacks and the prediction obtained for a target item, p_{old} was noted. Next the different attacks were inserted each separately and weighted svd was again applied and the prediction of the target item, p_{new} was noted. Based on these the prediction shift and hit ratio were computed (both of which are defined in Section 4.1.2)

4.1.1 Detection Methods

The different detection algorithms explained in Chapter 3 will be used for each attack. These are summarized as follows:

- RDMA
- WDMA
- DegSim
- RIC
- RMAR
- VarSelect

4.1.2 Experimental Evaluations

Each attack will be evaluated using the following metrics:

- **F1 measure** or F1 Score which is defined by equation (4.1) .

$$F1 = 2\frac{P \cdot R}{P + R} \tag{4.1}$$

$$= \frac{|\,TP\,|}{|\,TP\,| + |\,FP\,| + |\,FN\,|} \tag{4.2}$$

  where | TP | is the number of true positives (attacks have been correctly identified), FP is the number of false positives (true users detected as attackers), FN is the number of false negatives (true attackers were not found). P is the precision which is defined by P = $\frac{|TP|}{|TP|+|FP|}$ and R is recall defined by R = $\frac{|TP|}{|TP|+|FN|}$. A high value of F1 measure means a large proportion of true positives has been obtained.

- **Receiver Operating Characteristics** (ROC). ROC is a plot of recall (or sensitivity) versus 1-specificity. Specificity is defined as $specificity = \frac{|TN|}{|FP|+|TN|}$. This graph which is plotted while a threshold is varied is suited for assessing whether attackers have been accurately classified. All lines should pass through at least two coordinates: (0,0) and (1,1). Refer to Figure 4.1 for a visual interpretation. At (0,0) (**C**) no attackers were detected but all non-attackers were correctly classified. At (1,1) (**B**) all the attackers were correctly identified whilst incorrect discrimination occurs for all genuine users. The point (0,1) (**A**) in ROC plot represents perfect classification. Any points falling on the diagonal line (**D**) from (0,0) to (1,1) implies that the number of attackers found have been guessed at random. For instance, if 85% of the attackers were correctly, though randomly, classified as being in the positive class the false positive rate will rise to 85% which cause the point (0.85,0.85) to fall on the diagonal line. The area under the ROC curve (AUC) is equivalent to the probability of distinguishing a random attacker from a group compared to a randomly chosen non-attacker. So, the larger the area the better will be the classification.

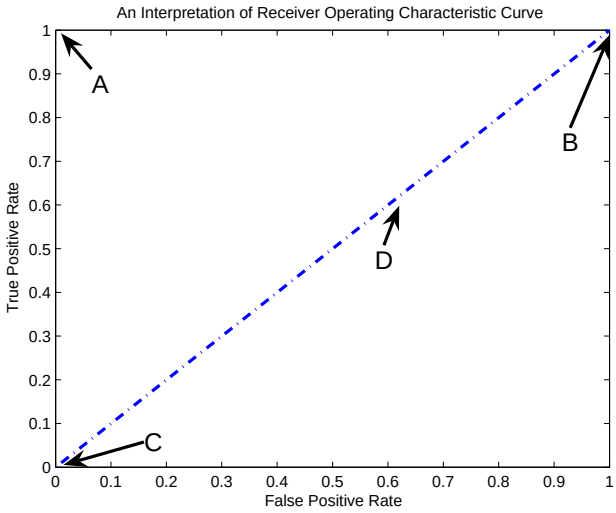


Figure 4.1 — ROC curve showing the different important regions

- **Prediction Shift** (*PredShift*) is the average difference between the ratings given by n users before and after an attack. *PredShift* has a tendency to give a false impression on the items to be recommended. This is because an attack profile affects not only the target item but also all the other remaining items. For example, an item which had a rating of 1 before the attack might be pushed to 3 after the attack. But it might be that all the other items' ratings have also increased. Consequently, in the worst scenario all the other items will have ratings above 3 which implies that the disliked target item will not be in the recommended list. For this reason, some people prefer hit ratio.

- Hit Ratio (*hitratio*) determines whether an attack has been able to affect the list of recommendations given to the user. A hit is said to occur if an item is recommended to a user. This can be represented by equation (4.3)

$$hitratio = \frac{1}{n} \sum_{u \in \mathrm{U}} h_u \tag{4.3}$$

where a hit is denoted by $h_u = 1$ and a miss by $h_u = 0$ and n is the total number of users being considered in the superset U.

To set an experimental framework showing that some of the detection measures detect too many false positives, 10% of the authentic users have been selected from the movieLens data set. High ratings have been assigned to a specific target item (push attack). In practice, this is what attackers try to achieve i.e. attackers try to get attack profiles to be very similar to the other users in the database. Having reached this goal, it is expected that none of these would be detected because only a small percentage of the total number of filler items have been altered. It will now be shown that RDMA and WDMA cannot, unfortunately, be relied upon because too many false positives are detected.

4.1.2.1 Detection of Fake Profiles generated from User Profiles

The experiment was setup by choosing 10% of the true user-item in the matrix, Υ and inserting high ratings (4 and 5) for one particular target item. Since only one item out of the mean of the filler items was changed the attack should be successful.

Figure (4.2) shows the ROC plot when RDMA,WDMA, DegSim, RIC and RMAR were used to detect attacks generated from the true user profiles.

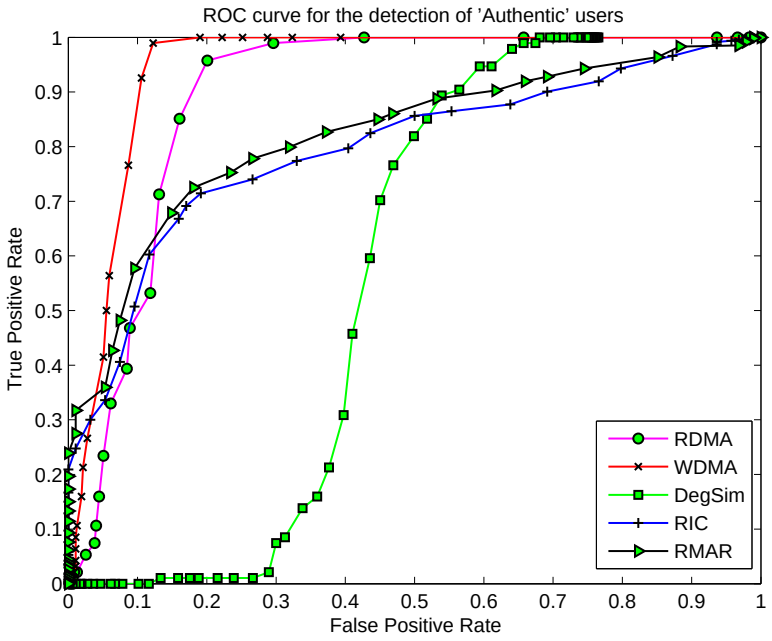


Figure 4.2 — ROC curve showing how the different detection methods perform with "true" user profiles. Attack Size (AS) = 10%

The ideal scenario should be a straight line from (0,0) to (1,1) where the area under the curve would be 0.5. However, in this case the areas under the curve are 0.895, 0.943, 0.571, 0.796, and 0.821 for RDMA, WDMA, DegSim, RIC and RMAR respectively. Unfortunately, all these algorithms are detecting true users as being fake. As already pointed out but also mentioned in [21] RDMA detects lots of false positives which has just be shown (refer to Figure 4.2). Here, DegSim has the closest value to 0.500 and so could now be seen as causing less misclassification. So, in interpreting these plots and corresponding AUCs care is necessary.

4.2 Results and Discussions: Different Attacks

This section will provide all the experimental results obtained when all the different evaluation metrics defined in Section 4.1.2 have been used to evaluate random,

Bandwagon, average and Eigen attacks. For comparison purposes, each attack's performance will be investigated in the following order:

1. The scatter plots in low dimensions for all the attacks will be given
2. The F1 measure and Receiver Operating Characteristic curve (ROC) when VarSelect is used as the detection method will be given for all the attacks
3. The F1 measure and ROC will be plotted for RDMA, WDMA, DegSim, RIC and RMAR.
4. The detection rate for all the different detection methods will be tabulated including prediction shift and hit ratio.

4.2.1 Low Dimension Scatter Plots using Variable Selection

Figure 4.3, Figure 4.4, Figure 4.5 and Figure 4.6 show the scatter plots of the first two principal components in the case of random, Bandwagon, average and Eigen attacks respectively. All these plots are for an Attack Size (AS) of 10% and a Filler Size (FS) of 3%.

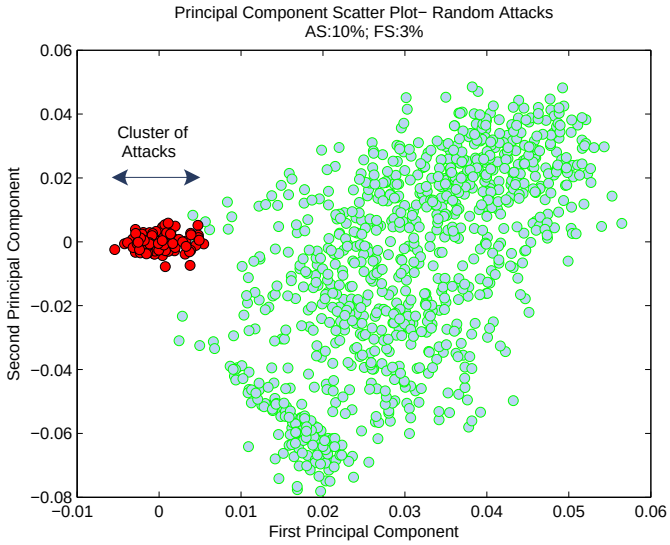


Figure 4.3 — Scatter plot of the first two principal components in the case of Random Attacks; Attack Size(AS) = 10% and Filler Size(FS) = 5%

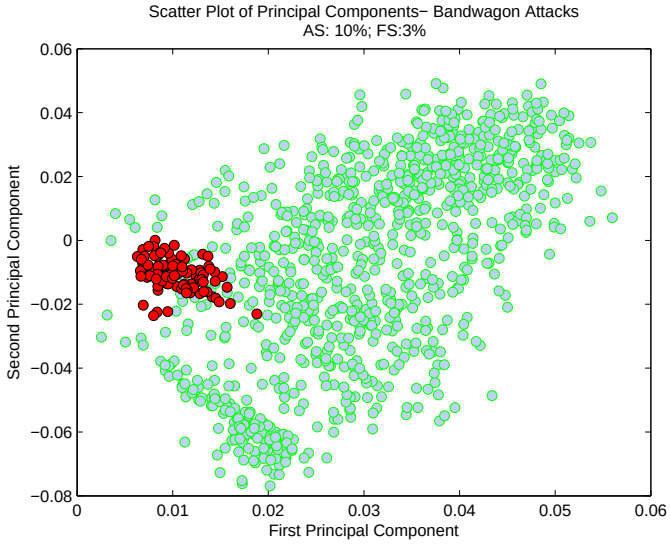


Figure 4.4 — Scatter plot showing that the points from the first two principal components merging with the authentic data (Bandwagon Attacks)

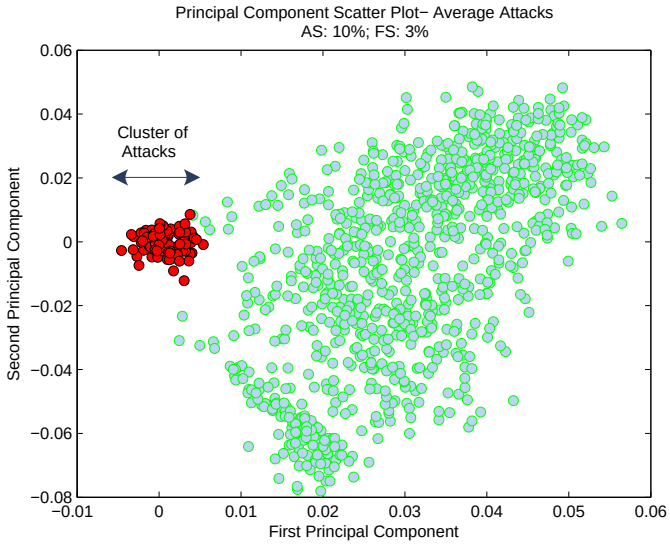


Figure 4.5 — Scatter plot of the first two principal components in the case of Average Attacks; Attack Size(AS) = 10% and Filler Size(FS) = 3%

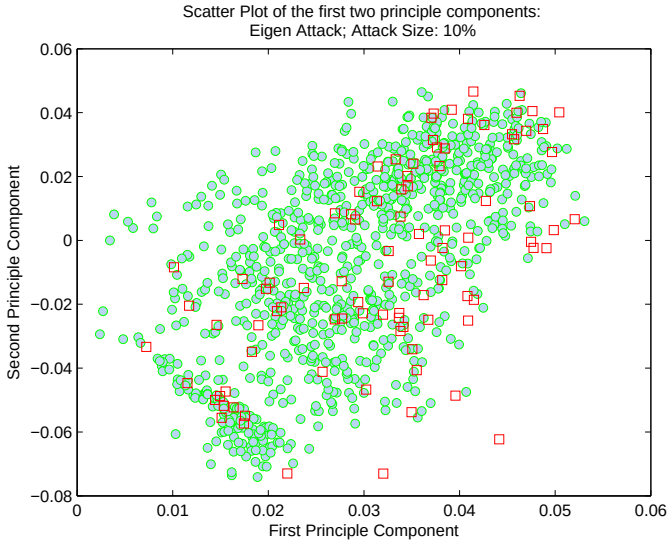


Figure 4.6 — Squares showing the attack profiles data superimposed on the original data as expected; Attack Size:10% (Eigen Attacks)

Clearly, VarSelect algorithm can be used to detect random and average attacks since a clear cluster of random attack profiles is located around zero mean (i.e. where the magnitudes of the first two principal components are small). Figures 4.3 and 4.5 show the visual representation of the unique cluster formed. The detection process has been done similar to how Mehta et. al. did in his paper [2] i.e. take the top few variables sorted in ascending order as explained in Algorithm 2.1 in Chapter 2 and the discussion following it. Bandwagon attacks which have not been investigated by the authors of [9] have the magnitudes of the first principal component comparable to at least some user profiles. This can easily be seen from Figure 4.4 where the attack profiles have merged with the user profiles in the low dimension. If the third principal component is also comparable with the other users, VarSelect will have a worse performance when compared to the random and average attacks.

Finally, in terms of similarity between the true and fake users in the low dimensions, Eigen attack is easily the best since from Figure 4.6 the attack profiles are all spread over the user profiles. It should be noted that in the case of Eigen attacks there is no specific filler size because items are naturally selected from Algorithm 3.2 in Chapter 3 when converting from low to high dimensions. Both of the principal components are not close to zero and, hence, it is expected that VarSelect will not be able to detect Eigen attacks.

4.2.2 Detection Rate using VarSelect

Table 4.1 shows the proportion of different attacks detected. It should be noted that slight variation of the percentage could be obtained because of the random noise inherent in all the attacks.

Table 4.1 — Detection rate for the Different attacks using VarSelect

		Filler Size in %		
		1	3	10
Random Attacks	AS: 1	100	100	100
	AS: 5	100	100	100
	AS: 10	98.9	98.9	100
Bandwagon Attacks	AS: 1	0.00	0.200	0.800
	AS: 5	10.6	57.4	95.7
	AS: 10	0.00	81.9	96.8
Average Attacks	AS: 1	100	100	90.0
	AS: 5	97.8	97.8	97.8
	AS: 10	97.9	97.9	96.8
Eigen Attacks		2.32-14.1	1.31-33.7	1.25-27.2
	AS: 1	0.00	-	-
	AS: 5	-	0.00	-
	AS: 10	-	-	0.00

Table 4.1 shows that the detection rates in the case of random and average attacks are very high when VarSelect is used as the detection method. Whether the attack size or filler size is small, the percentage of fake profiles detected is very high (close to 100%). Most of the average attacks have been detected supporting the fact that VarSelect is good at detecting profiles from attackers. The performance is slightly worse when compared to the situation when random attacks are inserted. Still, it is possible to prevent the fake profiles from affecting the prediction. On the other hand, Bandwagon attacks are not properly detected except for large filler size. As the number of filler items increase VarSelect is more successful at determining whether an attack profile is fake or not. This is only true because the percentage of popular filler items selected were as follows: 27.8%, 19.6% and 5.85% for filler sizes of 1%, 3% and 10% respectively. This was purposely done to show the reader that an increase in the number of popular items affects the detection rate. Hence, someone might think

that increasing the number of popular items will essentially provide worst detection rate since the attack profiles will be very similar to many user profiles in the database. This idea though partially true will have a severe impact on algorithms using singular value decomposition as will be described in Section 4.2.5. The worse performance was with Eigen attacks where none of them were detected (0.00%) when using VarSelect providing concrete reasons that selecting just the first few components are not enough to detect Eigen attacks. This is an expected result since from Figure 4.6 none of the fake profiles were close to zero mean (in regard to both principal components).

4.2.3 Evaluation of VarSelect using F1 measure

F1 measure will illustrate the reliability of VarSelect. If the number of true positives is equal to the total number of fake profiles, f_T and no false positives or false negatives were recorded F1 score will have an ideal maximum value of 1 occurring at coordinates (f_T,1). On the other hand, if the detection rate is still the maximum i.e. f_T but the number of FN and FP increases the F1 score will decrease by an amount inversely proportional to the number of FP and FN. The x-coordinate will remain the same but the y-coordinate (F1 score) will go down. Figure 4.7 shows the plot of F1 score for random, Bandwagon, average and Eigen attacks.

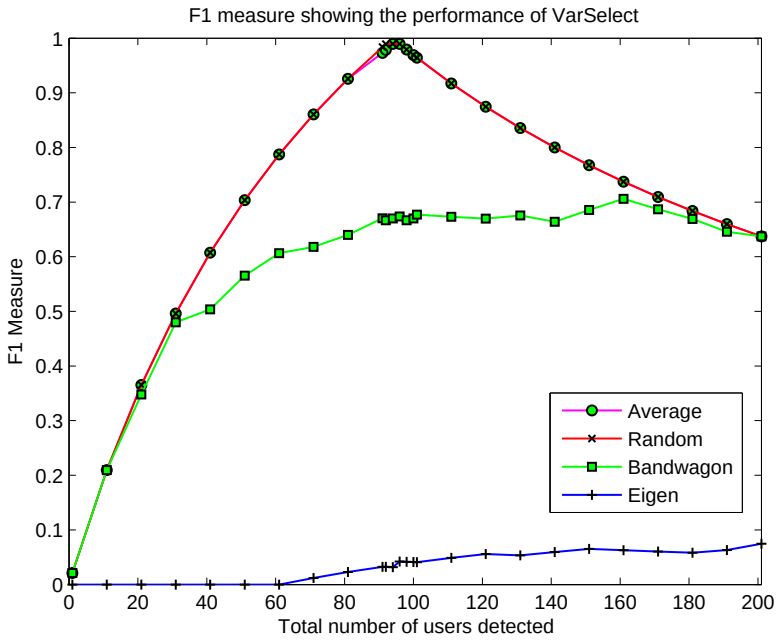


Figure 4.7 — F1 measure showing how well VarSelect can detect the different attacks; Note that the curves for random and average are almost on top of each other; Attack Size (AS) 10% and Filler Size (FS) 3%

From Figure 4.7, an ideal F1 score of one is obtained in the case of random attacks at coordinates (94, 1). This means exactly 94 true Positives attacks were detected which is exactly equal to the number of fake profiles injected. Undoubtedly, VarSelect is very good at detecting random attacks. F1 score does not detect all the average attacks but still the F1 score is very close to one at 93 at coordinates (93, 0.997). This means very few false positives and false negatives have been detected. VarSelect is thus a very good algorithm to detect random and average attacks as already shown in [2]. This is not the case with Bandwagon attacks as there is no distinct peak for the F1 score between the x-coordinates: 90 and 165 where there is an almost flat line with a score of approximately 0.70. The statement "VarSelect struggles to find the correct attackers if attacks are of Bandwagon type" is a fair one due to the lesser number of true positives being detected and the rise in the num-

ber of false positives and false negatives. Finally, eigen attacks are easily the best at not being detected. The actual value of F1 score is too small (0.096) implying that the number of false positives and false negatives far outweighed the number of true positives and so this value is insignificant. Eigen attack is at least 10 (1/0.096) times better than random and average attacks. Also, hardly any significant peaks are present.

4.2.4 Receiver Operating Characteristic Curve for evaluating VarSelect

Another visual interpretation of the detection rate is shown through the ROC curves illustrated as Figure 4.8.

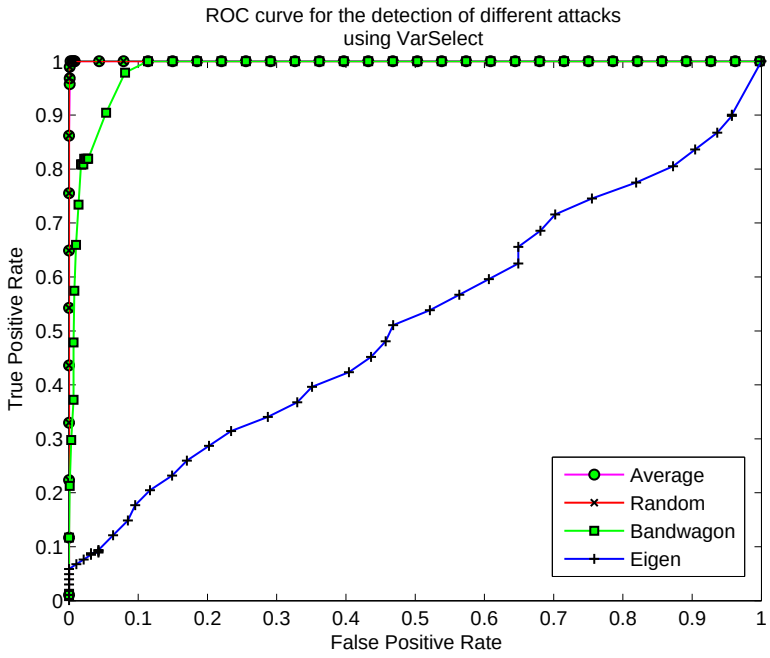


Figure 4.8 — ROC showing how well the different detection methods can detect different attacks i.e. Attack Size (AS) 10% and Filler Size (FS) 3%

From Figure 4.8 it can easily be seen that VarSelect achieves the best performance

for random and average attacks where the area under the curve is 1.00 and 0.999 respectively. Bandwagon attacks are detected to a lesser extent as already explained above. The area under the curve (AUC) is approximately 0.975 which is not perfect. Eigen attacks again showed to have the worst detection rate with an AUC of 0.517 which is very close to the random guess marked by an area of 0.500. It has been shown that VarSelect has the worst detection rate for Eigen attack when compared to random, Bandwagon and average attacks. Now, it remains to be proved that the performace is worst for the other detection methods.

4.2.4.1 ROC of RDMA, WDMA, DegSim, RIC and RMAR

Figure 4.9, Figure 4.10, Figure 4.11 and Figure 4.12 show the receiver operating characteristics of random, Bandwagon, average and Eigen attacks when RDMA, WDMA, DegSim, RIC and RMAR are used in the detection. The steeper the gradient of the curve the better is the detection.

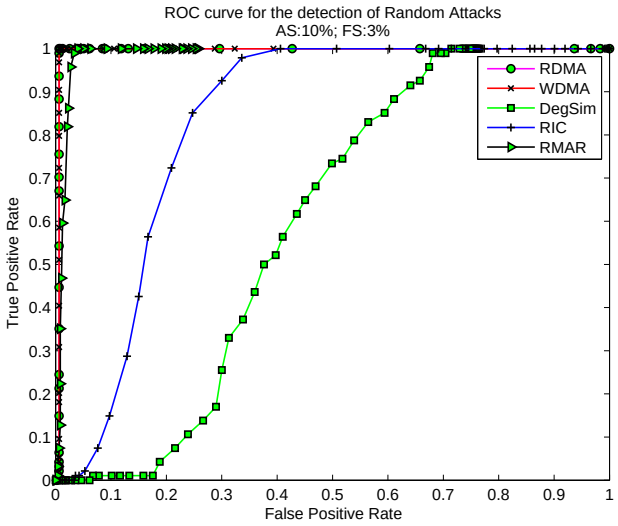


Figure 4.9 — ROC curve showing how good RDMA, WDMA, DegSim, RIC, RMAR are at detecting random attacks; Attack Size (AS) 10% and Filler Size (FS) 3%

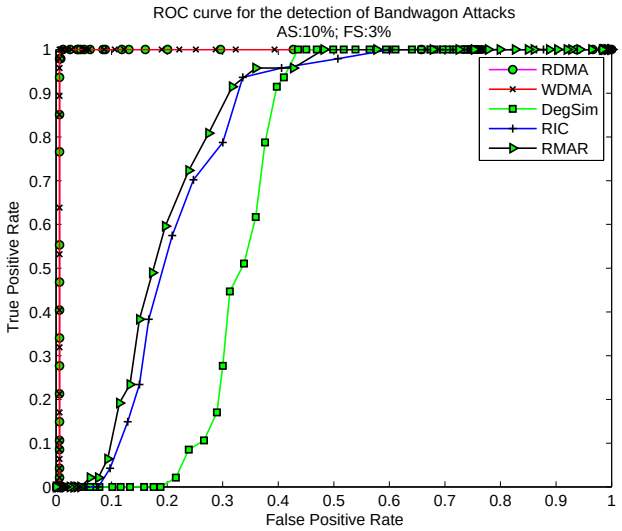


Figure 4.10 — ROC curve showing how well RDMA, WDMA, DegSim, RIC, RMAR are at detecting Bandwagon Attacks; Attack Size(AS): 10% and Filler Size(FS): 3%

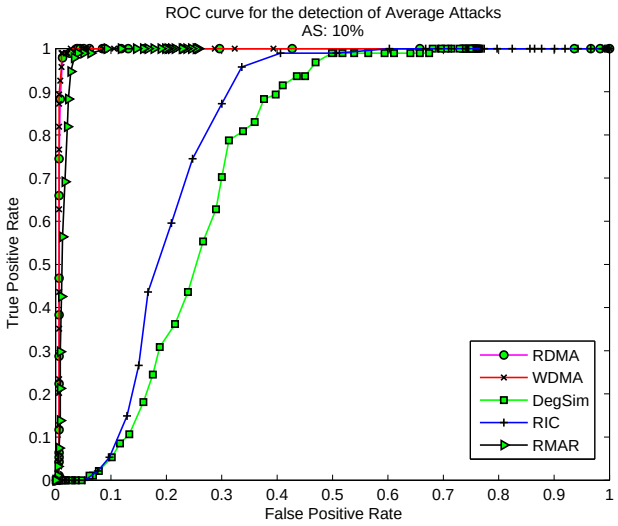


Figure 4.11 — ROC curve showing how well RDMA, WDMA, DegSim, RIC, RMAR are at detecting Average Attacks; Attack Size (AS) 10% and Filler Size (FS) 3%

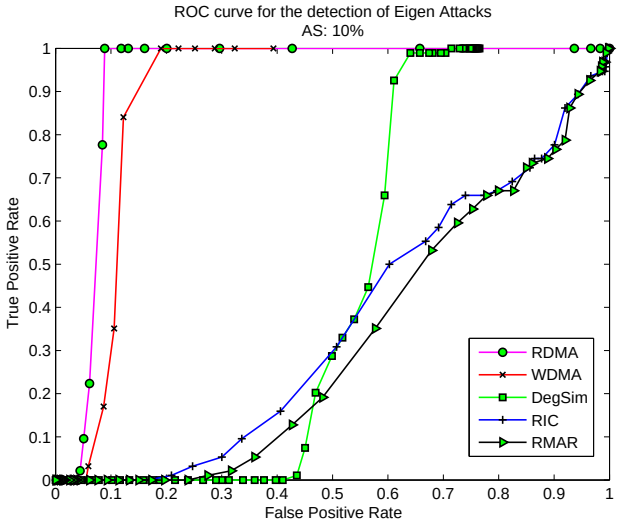


Figure 4.12 — Receiver Operating Characteristic showing how poor RDMA, WDMA, DegSim, RIC and RMAR can detect Eigen attacks; Attack Size (AS) = 10%

Figure 4.9 shows that DegSim has the worst performance in terms of detecting random attacks. Visually, RDMA, WDMA and RMAR are the best though the curve is not perfect. Remember Figure 4.2 which has shown that RDMA and WDMA have a tendency to detect too many false positives. From Figure 4.10 RDMA and WDMA are again good at discriminating between authentic profiles and fake profiles from Bandwagon attacks. The performance in the case of Bandwagon attacks is slightly worse when compared to Random Attacks. Average attacks are detected to a lesser extent when compared to the two attacks. The ROC plot shown in Figure 4.11 gives a comparison between average attacks and the other two attacks discussed above. RIC and DegSim give the worst performance whilst RDMA, RMAR and WDMA give the best but still not ideal especially for small filler sizes. Since the above ROCs plots only gave a visual interpretation, the areas under the curves have been tabulated and are shown in the next section. Again the ROC curve for Eigen attack (refer to Figure 4.12) is considerably worse than random, Bandwagon and average attacks. DegSim, RIC and RMAR performance is much worse when compared to RDMA and WDMA. Even RDMA and WDMA performs worse than the previous attacks confirming that Eigen attack is indeed the strongest attack.

4.2.5 Evaluations using the Remaining Metrics

In this section the area under the receiver operating characteristic curves, the hit ratio and the prediction shift will be tabulated.

I **Random Attacks**

Table 4.2 shows the area under the curves for the different filler and attack sizes. The hit ratio and prediction shift are also given (random attacks).

Table 4.2 — Mean AUC for the Random Attacks

		Filler Size in %		
		1%	3%	10%
Attack Size of 1%	RDMA	0.993	0.994	0.994
	WDMA	0.993	0.994	0.994
	DegSim	0.748	0.624	0.561
	RIC	0.739	0.803	0.832
	RMAR	0.753	0.827	0.861
	VarSelect in %	100	100	100
	PredShift	0.621	1.26	0.973
	Hit Ratio in %	0.312	0.164	0.142
Attack Size of 5%	RDMA	0.994	0.994	0.994
	WDMA	0.993	0.994	0.994
	DegSim	0.506	0.588	0.550
	RIC	0.771	0.836	0.843
	RMAR	0.994	0.857	0.994
	VarSelect in %	100	100	100
	PredShift	2.86	2.48	0.521
	Hit Ratio in %	6.90	5.71	0.231
Attack Size of 10%	RDMA	0.994	0.994	0.994
	WDMA	0.994	0.993	0.994
	DegSim	0.729	0.862	0.811
	RIC	0.765	0.828	0.836
	RMAR	0.994	0.986	0.994
	VarSelect in %	98.9	98.9	100
	PredShift	4.14	2.52	0.438
	Hit Ratio in %	69.3	42.3	0.385

Table 4.2 shows that as the number of filler items increase most of the detection methods easily classify the attackers where RDMA as a whole gives a bet-

ter performance. As usual, RDMA and WDMA showed the best detection with DegSim being the worst. For low filler size (i.e. 1%) and high attack size (i.e. 10%) random attack causes quite a significant prediction shift. But as explained before prediction shift should not be taken on its own to provide proofs of whether the items have been recommended or not. Hit ratio is tabulated to confirm the above (refer to Table 4.2). Indeed a very poor hit ratio for filler sizes of 3% and 10% were found empirically except where the attack size was 10%. Low filler size and high attack size cause the target items to be recommended as confirmed by the hit ratio (in %) of 69.3. This means that WSVD is resistant to random attacks for moderate filler and attack sizes.

II **Bandwagon Attacks**

Table 4.3 shows the area under the ROC graphs, hit ratio and prediction shift when the fake attack profiles from Bandwagon attacks are injected.

Table 4.3 — Mean AUC for the Bandwagon Attacks

		Filler Size in %		
		1%	3%	10%
Attack Size of 1%	RDMA	0.993	0.994	0.994
	WDMA	0.992	0.994	0.994
	DegSim	0.812	0.730	0.531
	RIC	0.696	0.781	0.827
	RMAR	0.713	0.804	0.846
	VarSelect in %	0.00	0.200	0.800
	PredShift	-0.525	0.251	1.10
	Hit Ratio in %	0.00	0.00	0.211
Attack Size of 5%	RDMA	0.993	0.994	0.994
	WDMA	0.994	0.994	0.994
	DegSim	0.747	0.678	0.520
	RIC	0.733	0.762	0.799
	RMAR	0.744	0.776	0.822
	VarSelect in %	10.6	57.4	95.7
	PredShift	-1.19	0.940	0.807
	Hit Ratio in %	0.00	2.54	0.302
Attack Size of 10%	RDMA	0.991	0.994	0.994
	WDMA	0.992	0.994	0.994
	DegSim	0.848	0.669	0.668
	RIC	0.693	0.784	0.814
	RMAR	0.683	0.804	0.835
	VarSelect in %	0.00	81.9	96.8
	PredShift	-0.525	0.513	0.330
	Hit Ratio in %	0.00	0.910	0.173

Table 4.3 shows that the detection rate of DegSim, RIC and RMAR have considerably worsened confirming that Bandwagon attacks are stronger than Ran-

dom attacks. Indeed, giving some popular items high ratings increases the similarity between attackers and fake users. Still, though RDMA and WDMA succeeds in classifying the attackers. The prediction shift is the worst one in all the attacks. In fact, the prediction shift was in the other direction for filler sizes of 1% when a greater proportion of the items were popular. This definitely, means that all those attack profiles which were injected did not contribute at all to the pushing of the ratings of the targeted items. By randomly selecting top-n popular items they become similar to some users but in the end when the low matrix approximation is calculated, SVD will not take into account the attack profiles which are just redundant (most people have rated the popular items). When the number of popular items selected is decreased the effects of the fake profiles are increased as shown by the increase in hit ratio in Table 4.3. Remember that Bandwagon attacks could bypass VarSelect detection mechanism but the prediction shift and the hit ratios are not good at all. The hit ratio was very negligible in most scenario as shown in Table 4.3.

III **Average Attacks**

The tabulations of all the remaining evaluation metrics (for average attacks) is given by Table 4.4.

Table 4.4 — Mean AUC for the Average Attacks

		Filler Size in %		
		1	**3**	**10**
Attack Size of 1%	RDMA	0.993	0.993	0.994
	WDMA	0.992	0.994	0.994
	DegSim	0.652	0.724	0.667
	RIC	0.666	0.807	0.760
	RMAR	0.731	0.864	0.825
	VarSelect in %	100	100	90.0
	PredShift	0.595	1.16	1.24
	Hit Ratio in %	0.132	0.846	0.673
Attack Size of 5%	RDMA	0.992	0.993	0.994
	WDMA	0.992	0.994	0.994
	DegSim	0.991	0.747	0.698
	RIC	0.757	0.811	0.834
	RMAR	0.802	0.861	0.858
	VarSelect in %	97.8	97.8	97.8
	PredShift	2.72	3.14	0.994
	Hit Ratio in %	7.05	10.4	0.543
Attack Size of 10%	RDMA	0.991	0.993	0.994
	WDMA	0.991	0.976	0.994
	DegSim	0.800	0.508	0.673
	RIC	0.716	0.798	0.804
	RMAR	0.761	0.987	0.855
	VarSelect in %	97.9	97.9	96.8
	PredShift	4.08	3.01	0.865
	Hit Ratio in %	70.6	5.58	1.09

Table 4.4 indeed confirms the popular myth: average attacks are the most difficult to detect though this is not the case if VarSelect is used. The best perfor-

mance can again be attributed to RDMA. Now, it can be seen that average attacks are more difficult to detect when compared to random and Bandwagon attacks even for moderately filled profile and attack sizes (except if VarSelect is used). The prediction shift for filler size of 3% and attack sizes of 5% and 10% are greater than in the case of random attacks. Table 4.4 shows that the hit ratio is approximately 1.5 - 5 times better than the hit ratio shown in Table 4.2.

IV **Eigen Attacks**

Eigen attacks are the final attacks and Table 4.5 gives the area under the curve and tabulates all the results from the evaluation metrics.

Table 4.5 — Mean AUC for the Eigen Attacks

		Filler Size in %		
		2.32-14.1%	1.31 - 33.7%	1.25-27.2%
Attack Size of 1%	RDMA	0.922	-	-
	WDMA	0.896	-	-
	DegSim	0.555	-	-
	RIC	0.667	-	-
	RMAR	0.685	-	-
	VarSelect in %	0.00	-	-
	PredShift	0.953	-	-
	Hit Ratio in %	4.01	-	-
Attack Size of 5%	RDMA	-	0.929	-
	WDMA	-	0.893	-
	DegSim	-	0.537	-
	RIC	-	0.658	-
	RMAR	-	0.688	-
	VarSelect in %	-	0.00	-
	PredShift	-	2.15	-
	Hit Ratio in %	-	10.2	-
Attack Size of 10%	RDMA	-	-	0.928
	WDMA	-	-	0.888
	DegSim	-	-	0.549
	RIC	-	-	0.643
	RMAR	-	-	0.678
	VarSelect in %	-	-	0.00
	PredShift	-	-	3.19
	Hit Ratio in %	-	-	95.5

When compared to random and average attacks all the corresponding values in Table 4.5 are considerably worse. As already pointed out RDMA seems to be

the best at detecting Eigen Attacks but RDMA on its own detects loads of false positives as described before. The ultimate aim of this project was to develop an attack which could also cause ratings of the target items to be increased when Weighted Singular Value Decomposition is used as the predictive algorithm for collaborative filtering. Table 4.5 shows by how much the predictions have been affected. The prediction shift is similar to the prediction shift in average attack with high attack size and low filler size (e.g. attack size:10% and filler size:1%). But the most important difference comes after analysing the hit ratio shown in Table 4.5 where a hit ratio of almost perfect is achieved with an attack size of 10%. An attack size of 5% is still better than either random or average attacks (except for an attack size of 10% and filler size of 1% and 3%). The only way for any attacks to influence the rating of a particular user is to make sure that at least some subsets of the attack profiles tend to be close to the user profiles to which recommendations will be listed. If the attack profiles are instead similar to some other genuine users who are in some way different from the incoming user then the attack will be ineffective against that particular user. This is why an attack size of 10% gave a greater influence because there is now a greater probability of the attack profiles being similar to many incoming users.

On the next page the limitations of Eigen attack will be discussed with some initial work provided.

4.2.6 Limitations of Eigen Attack

Eigen Attack is infeasible in practice because the whole matrix, Υ is not available causing the calculation of the principal components by the attacker to be very difficult. If part of the principal components have been leaked or perhaps an approximation of the matrix has been found it is likely that Eigen Attack will be an excellent attack. To prove the last point the above analyses have been repeated with a 1%, 5% and 10% samples of the matrix.

4.2.6.1 Scatter plots for different Sample Sizes

Figure 4.13, Figure 4.14, Figure 4.15 show the scatter plots of the first two principal components for the three different sample sizes of the user-item matrix: 1%, 5%, 10% respectively.

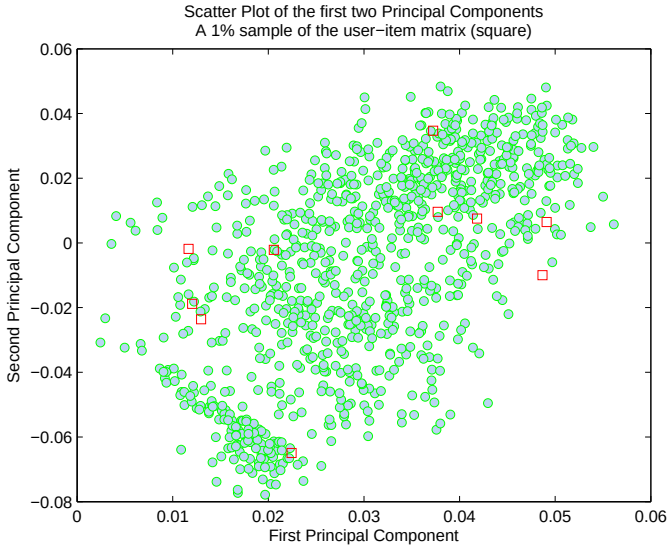


Figure 4.13 — Scatter plot after using only 1% of the user-item matrix

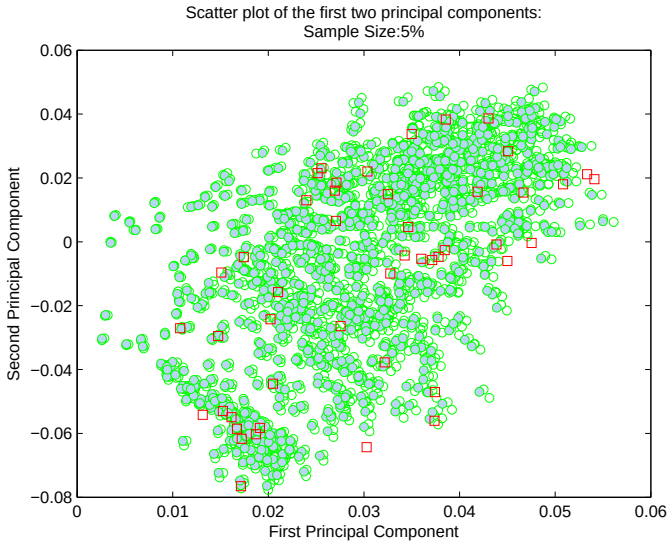


Figure 4.14 — Scatter plot after using only 5% of the user-item matrix

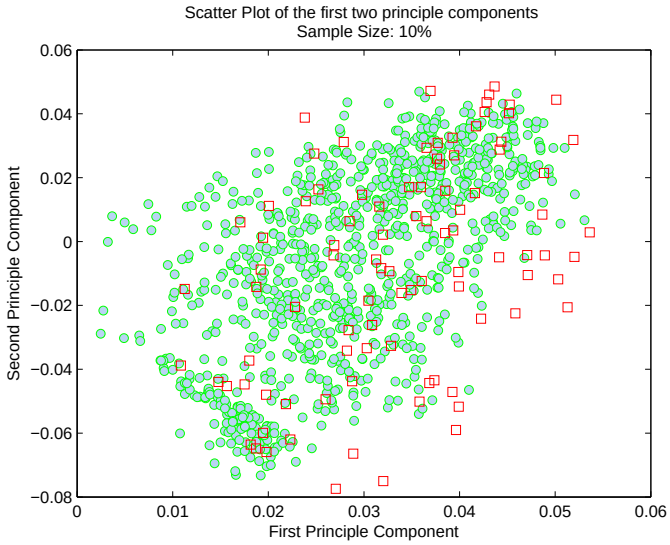


Figure 4.15 — Scatter plot after using only 10% of the user-item matrix

All the figures (4.13, 4.14, 4.15) show that the attack profiles have followed the main

distribution of the authentic users. The squares represent the attack profiles projected to the lower dimensions. It is expected that the different sample sizes will affect especially the hit ratio and the prediction shift. The higher the sampling size the higher the hit ratio should be. Some of the fake profiles deviate away from the main distribution. Nevertheless, it was found that these did not affect the detection rate which was again found to be zero for all the sampling sizes. To confirm this, the F1 score and the ROC curve were plotted.

4.2.6.2 F1 scores when VarSelect is used

The F1 measure for the three sample sizes have been plotted (refer to Figure 4.16).

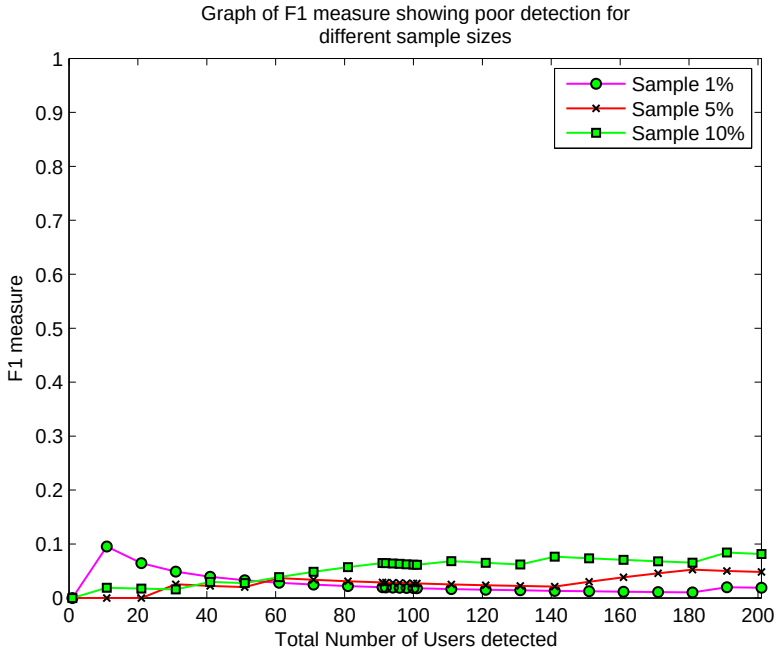


Figure 4.16 — The F1 measure for the three different sample sizes: 1%, 5%, 10%

Figure 4.16 shows that the F1 score for a sample size of 1% is two times higher than the others even though there is one peak which has a too small magnitude to be considered. The F1 score seems to tend towards a value which is lower than 0.100

for all the sample sizes. In brief, the F1 measure value is too small to be considered relevant in the detection of attackers (true positives).

4.2.6.3 Receiver Operating Characteristic for Different Sample Sizes

The ROC graph is given as Figure 4.17.

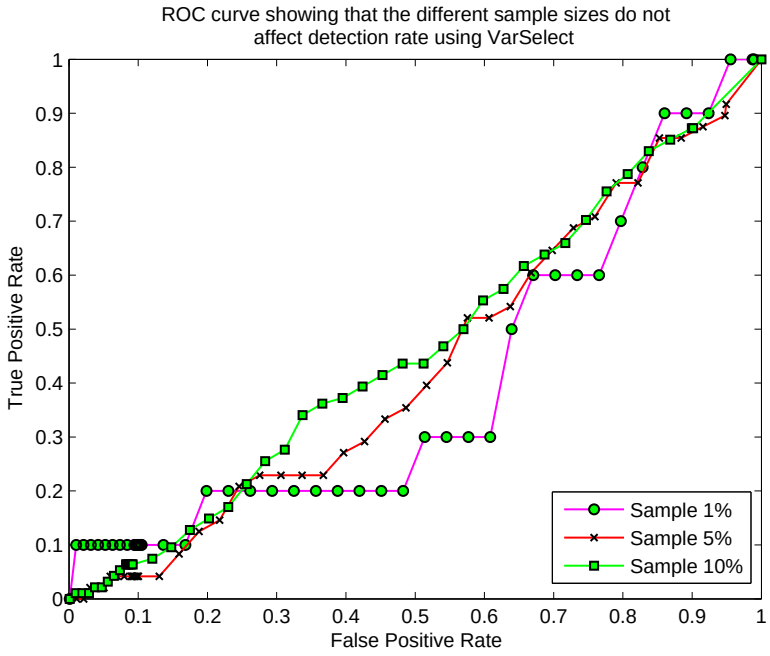


Figure 4.17 — The ROC curve for the three different sample sizes: 1%, 5%, 10%

The area under the curve for the 1%, 5% and 10% are 0.561, 0.690 and 0.510 respectively. Those values are all close to the threshold of 0.500 which quantifies the random nature in the detection process. All these values will be tabulated in Table 4.6 for quick reference.

4.2.6.4 All Evaluation Metrics

For clarity Table 4.6 lists the prediction shift, hit ratios, AUC and detection rate using VarSelect for the three sample sizes.

Table 4.6 — The percentage of Eigen attacks detected when VarSelect has been used

		Sample Size in %		
		1	5	10
Metrics	Prediction Shift	0.248	1.80	3.30
	Hit Ratio in %	0.00	6.20	83.4
	VarSelect in %	0.00	0.00	0.00
	AUC	0.561	0.690	0.510

The values of prediction shift and hit ratios clearly show that a 1% sample size is not enough to make a significant attack. The performance is worse in comparison to the performance of random and average attacks. The hit ratio of 6.20 for a sample size of 5% is much better when compared to the sample size of 1% though not ideal. A sampling size of 10% resulted in a prediction shift and a hit ratio which is much better than the previous two sampling sizes. Currently, it can be said that a sampling size of less than 10% but greater than 5% would be sufficient to perform Eigen attack.

4.3 Determining the low-approximation Matrix

It is now possible to use Eigen Attack to cause most detection mechanisms to fail from correctly distinguishing fake profiles from genuine profiles. Having achieved this main aim (of this project) it was decided to do some more work related to the low matrix approximation of Υ. Extensive analysis has not been done but initial results seem to be very promising.

4.3.1 Analysis of the Inputs to the Recommender Systems

Algorithm 3.4 in Chapter 3 was implemented in Matlab. It was decided to do some similar tests as the above on the inputs to the recommnender systems to determine whether the inputs are sufficiently similar to the genuine users. The overall distribution of the ratings generated for the inputs is shown in Figure 4.18.

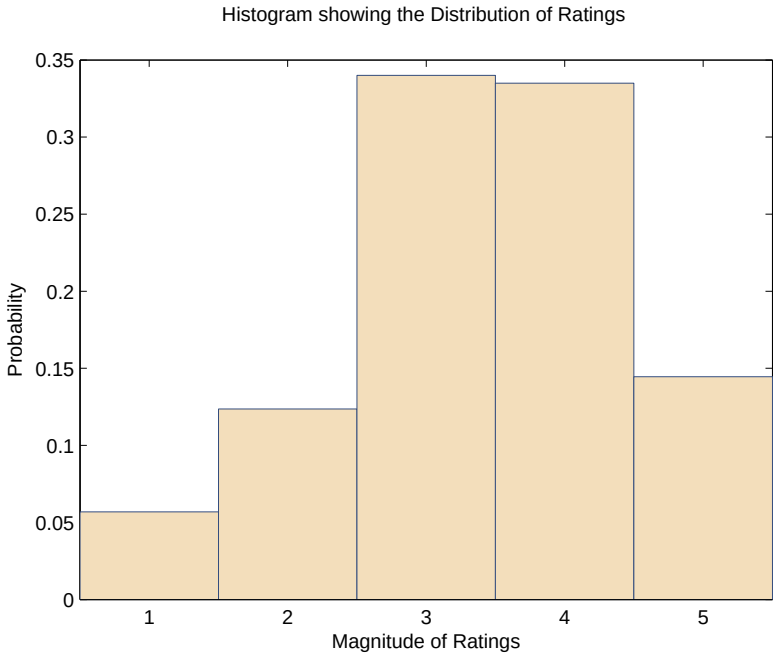


Figure 4.18 — Histogram showing the distribution of the ratings from the inputs

Figure 4.18 shows that the mean and standard deviation are very close to the distributions of the ratings of the true users who also provide most of the ratings around 4. Application of VarSelect to measure the detection rate shows a low rate of detection as given in Table 4.7. The lower the detection the greater is the likelihood the inputs will be closer to the authentic users.

Table 4.7 — The percentage of inputs which are detected to be dissimilar to the true users

		Filler Size in %		
		1	3	10
Size of the Inputs %	1	27.3	9.09	0.00
	5	61.7	48.9	4.3
	10	75.5	70.2	21.3

Table 4.7 shows that it is very likely that the newly created inputs are quite similar to

the users because the detection rate is not perfect.

The F1-measure was computed and the result is shown in Figure 4.19 .

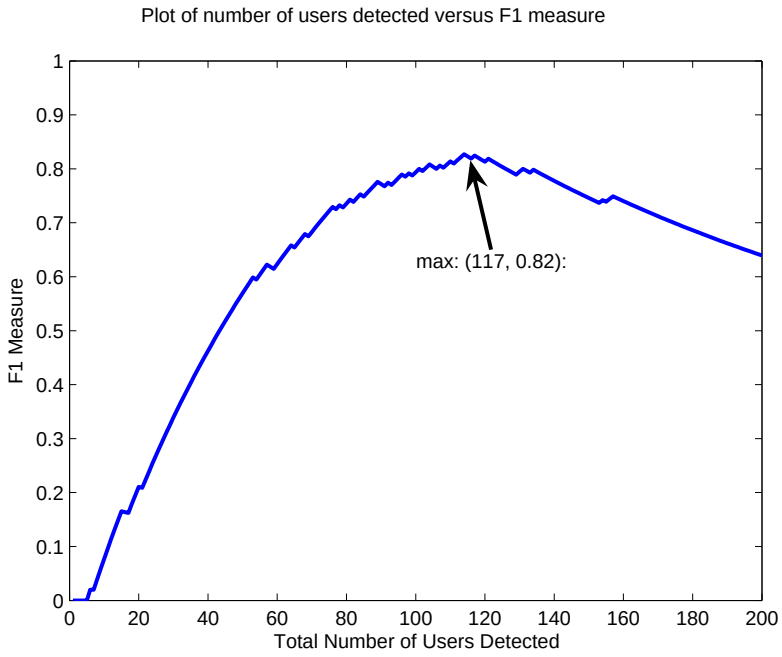


Figure 4.19 — The F1 score showing that indeed a very low detection rate is obtained

Figure 4.19 shows the maximum F1 score occurs at (117, 0.82). This means that a large proportion of attacks were detected to be false positives and false negatives. The number of true positives is also much smaller than 94 because the F1 score hardly reached the ideal value of 1.

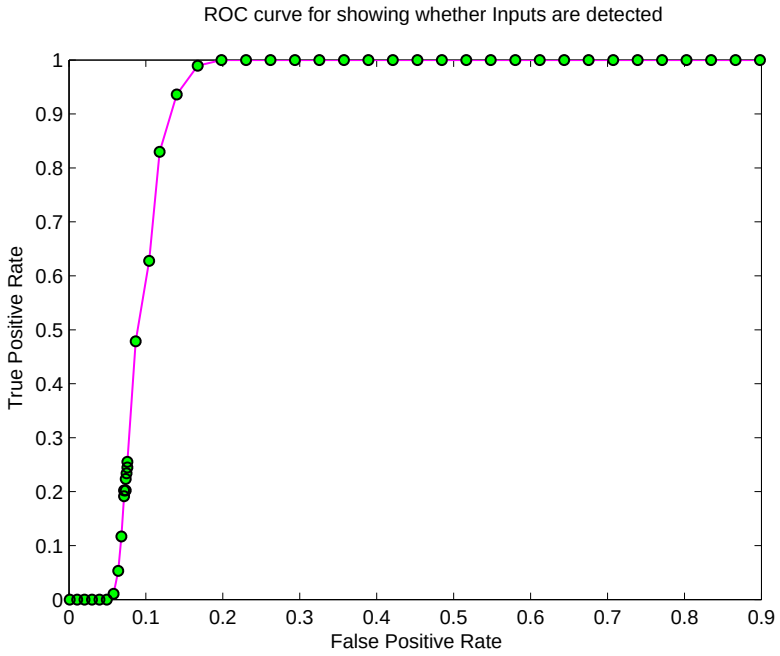


Figure 4.20 — ROC curve showing that inputs are mostly undetected when using VarSelect with an area under the curve of 0.802;

Figure 4.20 again confirms that a low proportion of the inputs is determined to be different from the true users. The area under the curve is only 0.802. To really confirm whether the inputs are close to at least some user profiles it was decided to test whether the predictive algorithm is affected. If this is the case then it is very likely that weighted SVD will take them into account when predicting which items should be in the recommender list. Interestingly, it was found that the prediction shift was 2.10 and the hit ratio (in %) was 73.0%. Based on these inputs, a deduction of the secret matrix is attempted by *abusing* the predictions from the recommender system. The decision was made to not only use WSVD but also Incremental SVD which has been programmed by Time Development and modified partly by Tamas, a Ph.D student at UCL. More information on Incremental SVD is given in Appendix A. The reason for using incremental singular value decomposition is due to this algorithm tying 3^{rd} place in terms of accurate predictions in the Netflix prize com-

petition in 2006.

Initial analysis brought about a very strong argument:

Statement 4.3.1

The greater the accuracy of the predictions from the recommender system, the easier it is to find users which are similar to the matrix.

Statement 4.3.1 gives a preliminary conclusion that the better the algorithm in terms of accuracy the easier it is to attack the system. This still needs to be proved empirically through more extensive analysis. What was done was to increase the accuracy of weighted SVD by increasing the convergence rate two-fold i.e. from 20 to 40.

Slight variations of Algorithm 3.4 were implemented which performs better. An example is as follows, in the first iteration not all the ratings were given high values because the recommender system is usually not that accurate. So, some of the items were randomly chosen and low values of ratings were assigned. These were sent to the input and then as the number of iterations increase less and less items were rated poorly. It was found that approximately $1/3^{rd}$ of the items from one attack profile were found in at least one of the user profiles. E.g. a user rated 170 items and the algorithm recovered 50-65 items for a filler size of 150 and attack size of one. Even though all these items have been recovered it is still necessary for the ratings given to approximate the ratings of those users the attack profile is closer to. This should be the aim in future work.

This concludes the main work achieved in the MSc project. It can convincingly be stipulated that not only the main goal of this project has been achieved but further work has also been done.

Chapter 5

Conclusions

This report has shown several contributions on how to attack collaborative filtering. The problem which was to develop an attack which would show the vulnerability of the detection methods used in ACF has been achieved through the creation of a novel attack called the Eigen attack. This was carefully designed in the low dimensions and successfully implemented to circumvent most of the detection mechanisms. Eigen attack is different from random, Bandwagon and average attacks because these are designed in the original space whilst Eigen attack is not.

5.1 Summary of Eigen Attack

Eigen attack could potentially be a very powerful attack for an attack size of less than 10% but this depends on the proportion of the matrix available. As most attacks, if the fake profiles are only similar to some particular users then only they would be affected by the attack supporting the inclination of using large attack size. This is why one particular attack profile can push the ratings of target items whilst the same attack profile will fail on a completely different user profile. The overall performance of Eigen attack is extremely favourable since it was built from a low dimensional model which as far as the author of this report is concerned has not been done before. The hit ratio measurements showed that it's possible to recommend with high cer-

tainty undesirable items for an attack size of 10%. The VarSelect detection algorithm also failed at detecting any of these attacks. Unfortunately, Eigen attacks also come with a bane: they rely heavily on knowing at least part of the matrix which is not usually available in practice rendering the attack difficult to implement. However, if this is not the case or part of the top principal components are leaked it will then be possible to perform Eigen attack. As a quick remedy to the problem related to Eigen attack, new ideas were developed to create an approximation of the user-item matrix by modelling the recommender system as a black box problem. The inputs to the black box were specifically designed to be at least similar to some users and to have the overall rating distribution mapping the original data matrix. It was also decided to investigate the appropriate sample size of the user-item matrix needed to achieve high hit ratio. It was found that the whole of the user-item matrix is not necessary. In fact, an upper bound of 10% seems to be sufficient for the sample size.

5.2 Deduction of the user-item matrix

As a means to solve the low approximation matrix problem an algorithm was created to have the inputs to the recommender systems already similar to at least some users in the database. The inputs were created from popular items and all the ratings were discrete. Outputs obtained from the recommender system were then modified and fed back. Using this iterative process similarities between authentic users and outputs from the recommender system were increased. It is likely that this might be at least one of the ways to determine the hidden user-item matrix.

5.3 Further work

Although the sample size of 10% was found to provide an adequate attack, more analyses need to be done to know what is the exact optimum sampling size. Secondly, the inputs being fed into the recommender systems for deducing the secret matrix could also be improved so that, initially, the profiles are already very similar to some user-item matrix. Finally, the feedback mechanism could be modified to

ensure a greater fraction of user-item matrix, Υ is retrieved. This could be done by e.g. performing intersections between the different outputs obtained from different modified inputs.

5.4 Final Notes

The combination of Eigen attack and the algorithm used for deducing the user-item matrix suggests that the recommender systems might be affected by malicious users though more resources in terms of time, money and effort are needed. It is hoped that other detection mechanisms should be built to prevent such attacks from taking place.

Bibliography

[1] Ronald A. Thisted, *Elements of Statistical Computing: Numerical Computation*, Chapman and Hall, 1988

[2] B. Mehta, T. Hofmann, P. Fankhauser, *Lies and Propaganda: Detecting Spam Users in Collaborative Filtering*, 2007

[3] I. Jolliffe, *Principal Component Analysis*, Springer, 2^{nd} Edition, 2002

[4] W. Press, A. Teukolsky,W. Vetterling, B. Flannery, *Numerical Recipes, The art of Scientific Computing*, Cambridge university Press, 3^{rd}, 2007

[5] J. B. Schafer, J. Konstan, J. Riedl, *Recommender Systems in E-Commerce*, Proceedings of 1st ACM conference on Electronic commerce, 1999

[6] D. Perrar, W. Dubitzky, M. Granzow, *A practical Approach to Microarray Data Anaylysis*, Springer, 2002

[7] N. Srebo, T. Jaakkola, *Weighted Low-Rank Approximation*, International Conference on Machine Learning (ICML), 2003

[8] Jon Shlens, *A Tutorial on Principal Component Analysis*, 2003

[9] B. Mehta, W. Nejdl, *Attack Resistant Collaborative Filtering*, ACM, 2008

[10] T. Speed, *Statistical analysis of gene expression microarray data*, Chapman & Hall, 2003

[11] Carl Wunsch, *The Ocean Circulation Inverse Problem*, Cambridge University Press, 1996

[12] K. T. Poole, *Spatial Models of Parliamentary Voting*, Cambridge University Press, 2005

[13] http://sifter.org/ simon/journal/20061211.html, last verified on 7^{th} August 2009

[14] S. Zhang, Y. Ouyang, J. Ford, F. Makedon, *Analysis of a Low-dimensional Linear Model under Recommendation Attacks*, SIGIR, 2006

[15] Chad A. Williams, Bamshad Mobasher, *Defending recommender systems:detection of profiled injection attacks*, Service Oriented Computing and Applications 1(3): 157-170, 2007

[16] Bhaskar Mehta, Thomas Hoftman, *A survey of Attack-Resistant Collaborative Filtering Algorithms*, IEEE Computer Society, 2008

[17] Shyong K. Lam, John Riedl, *Shilling Recommender Systems for Fun and Profit*, GroupLens Research, 2004

[18] Bamshad Mobasher, Robin Burke, Runa Bhaumik, J.J Sandvig, *Attacks and Remedies in Collaborative Recommendation*, IEEE Computer Security, 2007

[19] Bamshad Mobasher, *Secure Personalization: Building Trustworthy Recommender Systems*, 2007

[20] P-A. Chirita, W. Nejdl, C. Zamfir, *Preventing shilling attacks in online recommender Systems*, ACM, 2005

[21] C. A. Williams, B. Mobasher, R. Burke, R. Bhaumik, *Detecting Profile Injection Attacks in Collaborative Filtering: A Classification-Based Approach*, Springer Verlag, 2007

[22] Anonymous, *Defence against Profile Injection Attacks on Collaborative Filtering*, MSc thesis UCL, 2008

[23] Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl, *Item-based Collaborative Filtering Recommendation Algorithms*, 2001

[24] John S. Breese, David Heckermen, Carl Kadie, *Empirical Analysis of Predictive Algorithms for Collaborative Filtering*, Microsoft Research, 1998

[25] Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl, *Application of Dimensionality Reduction in Recommender System– a Case Study*, GroupLens Research Group

[26] John Canny, *Collaborative Filtering with Privacy*, Proceedings of the 2002 IEEE Symposium on Security and Privacy, Page 45, 2002

[27] P-A Chirita, W. Nejdl, C. Zamfir, *Preventing Shilling Attacks in Online Recommender Systems*, Workshop on Web Information And Data Management, Proceedings of the 7th ACM international workshop on web information and data management, 2005

[28] Hyung Jun Ahn, *A new similarity measure for collaborative filtering to alleviate the new user cold-starting problem*, Information Sciences, Elsevier, 2007

[29] Robin Burker, Bamshad Mobasher, Chad Williams, *Runa Bhaumik, Classification Features for Attack Detection in Collaborative Recommender Systems*, Philadelphia Pennsylvannia, USA, ACM 2006

[30] G. Gorrell, *Generalized Hebbian Algorithm for Incremented Singular Value Decomposition in Natural Language Processing*, EACL, 2006

[31] K. F. Riley, M. P. Hobson,S. J. Bence, *Mathematical Methods of Physics and Engineering*, Cambridge University Press, 2nd Edition, 2002

[32] H. S. Baird, D. P. Lopresti, *Human Interactive Proofs:* Second International Workshop, HIP 2005, Bethlehem, PA, USA, May 19-20, 2005, Proceedings (Lecture Notes in Computer Science / Security and Cryptology)

[33] T. Hoftmann, *Latent Semantic Models for Collaborative Filtering*, ACM, Trans. Inf. Syst. 22, 2004, no.1 89-115

[34] M. P. O'Mahony, N. J. Hurley, G. C. M. Silvestre, *An evaluation of neighbourhood formation on the performance of collaborative filtering*, Artificial Intelligence Review, Special Issue 21, 2004, 215-228

[35] Xiaoyuan Su, Taghi M. Khoshgoftaar, *Collaborative Filtering for Multi-class Data Using Belief Nets Algortithms*, Proceedings of the 18th IEEE International Conference on Tools with Artificial Intelligence, 2006

[36] Haifeng Chen, *Principal Component Analysis With Missing Data and Outliers*, Electrical and Computer Engineering Department, at www.caip.rutgers.edu/riul/research/tutorials/tutorialrpca.pdf (last accessed 31st August 2009)

Appendix A

Incremental Singular Value Decomposition

A.1 Incremental Singular Value Decomposition

A different way of computing SVD without the need to calculate eigenvectors motivated the incremental Singular Value Decomposition developed by Brandyn Webb (aka. Simon Funk) in 2006. The user-item matrix, $\mathbf{\Upsilon}$ is decomposed into a lower rank-k U and V^T singular vectors as shown in equation (A.1). The singular values are absorbed into $\mathbf{U}$ and $\mathbf{V}^T$ which explains why it is not shown here.

$$\Upsilon = \mathbf{U}\mathbf{V}^T \tag{A.1}$$

The prediction of the rating is then given by equation (A.2).

$$pred_{u,i} = \sum_{k}^{F} U_k \cdot V_k^T \tag{A.2}$$

The error between the actual rating and the predicted rating is given by equation (A.3)

$$error_{u,i} = \Upsilon_{u,i} - pred_{u,i} \tag{A.3}$$

Doing a partial derivative of the squared of $error_{u,i}$ with respect to U_k and V_k gives equation (A.4) and (A.5) respectively.

$$\frac{\partial(pred_{u,i})^2}{\partial \mathrm{U}_k} = -2 \cdot \mathrm{V}_k \left(\Upsilon_{u,i} - pred_{u,i}\right) \tag{A.4}$$

Similarly,

$$\frac{\partial(pred_{u,i})^2}{\partial \mathrm{V}_k} = -2 \cdot \mathrm{U}_k \left(\Upsilon_{u,i} - pred_{u,i}\right) \tag{A.5}$$

Applying gradient descent with a learning rate of λ equation (A.6) is derived.

$$\mathrm{U}_k^{new} = \mathrm{U}_k^{old} - \lambda \cdot (\frac{\partial(pred_{u,i})^2}{\partial \mathrm{V}_k}) = \left(\lambda \cdot (\Upsilon_{u,i} - pred_{u,i})\mathrm{U}_k^{old} + \mathrm{U}_k^{old} + \mathrm{V}_k^{old}\right) \tag{A.6}$$

The same applies for V_k^{new}.

By applying the above equation one feature will be trained and the remaining features that will most reduce the error term, [13]. This process is repeated for the remaining features. Brandyn Webb (aka. Simon Funk) then did some refinements to increase the accuracy of the prediction The Incremental SVD code was implemented by Time Development and modified by Tamas a PhD student at UCL.

This side is purposely left Blank