TRANSACTIONS ON DATA PRIVACY 4 (2011) 129–146

Efficient Protocols for Principal Eigenvector Computation over Private Data

Manas A. Pathak, Bhiksha Raj

Carnegie Mellon University, Pittsburgh, PA 15213, USA.

E-mail: manasp@cs.cmu.edu, bhiksha@cs.cmu.edu

Abstract. In this paper we present a protocol for computing the principal eigenvector of a collection of data matrices belonging to multiple semi-honest parties with privacy constraints. Our proposed protocol is based on secure multi-party computation with a semi-honest arbitrator who deals with data encrypted by the other parties using an additive homomorphic cryptosystem. We augment the protocol with randomization and oblivious transfer to make it difficult for any party to estimate properties of the data belonging to other parties from the intermediate steps. The previous approaches towards this problem were based on expensive QR decomposition of correlation matrices, we present an efficient algorithm using the power iteration method. We present an analysis of the correctness, security, and efficiency of protocol along with experimental results using a prototype implementation over simulated data and USPS handwritten digits dataset.

1 Introduction

Eigenvector computation is one of the most basic tools of data analysis. In any multivariate dataset, the eigenvectors provide information about key trends in the data, as well as the relative importance of the different variables. These find use in a diverse set of applications, including principal component analysis [2], collaborative filtering [3] and PageRank [4]. Not all eigenvectors of the data are equally important; only those corresponding to the highest eigenvalues are used as representations of trends in the data. The most important eigenvector is the *principal* eigenvector corresponding to the maximum eigenvalue.

In many scenarios, the entity that actually computes the eigenvectors is different from the entities that possess the data. For instance, a data mining agency may desire to compute the eigenvectors of a distributed set of records, or an enterprise providing recommendations may want to compute eigenvectors from the personal ratings of subscribers to facilitate making recommendations to new customers. We will refer to such entities as *arbitrators*. Computation of eigenvectors requires the knowledge of either the data from the individual parties or the correlation matrix derived from it. The parties that hold the data may however consider them private and be unwilling to expose any aspect of their individual data to either the arbitrator or to other parties, while being agreeable, in principle, to contribute to the computation of a global trend. As a result, we require a privacy preserving algorithm that can compute the eigenvectors of the aggregate data while maintaining the necessary privacy of the individual data providers.

*A preliminary version of this article appeared as [1].

The common approach to this type of problem is to obfuscate individual data through controlled randomization [5]. However, since we desire our estimates to be exact, simple randomization methods that merely ensure accuracy in the mean cannot be employed. Han *et al.* [6] address the problem by computing the complete QR decomposition [7] of privately shared data using cryptographic primitives. This enables all parties to collaboratively compute the complete set of global eigenvectors but does not truly hide the data from individual sources. Given the complete set of eigenvectors and eigenvalues provided by the QR decomposition, any party can reverse engineer the correlation matrix for the data from the remaining parties and compute trends among them. Canny [8] present a different distributed approach that does employ an arbitrator, in their case a *blackboard*, however although individual data instances are hidden, both the arbitrator and individual parties have access to all aggregated individual stages of the computation and the final result is public, which is much less stringent than our privacy constraints.

In this paper, we propose a new privacy-preserving protocol for shared computation of the principal eigenvector of a distributed collection of privately held data. The algorithm is designed such that the individual parties, whom we will refer to as "Alice" and "Bob" learn nothing about each others' data, and only learn the degree to which their own data follow the global trend indicated by the principal eigenvector. The arbitrator, who we call "Trent", coordinates the computation but learns nothing about the data of the individual parties besides the principal eigenvector which he receives at the end of the computation. In our presentation, for simplicity, we initially consider two parties having a share of data. Later we show that the protocol can be naturally generalized to N parties. The data may be split in two possible ways: along data instances or features. In this work, we principally consider the instance-split case. However, as we show, our algorithm is easily applied to feature-split data as well.

We primarily use the power iteration method [7] to compute the principal eigenvector. We will use a combination of randomization, homomorphic encryption [9] and oblivious transfer (OT) [10] to enforce privacy on the computation. The algorithm assumes the parties to be *semi-honest*. While they are assumed to follow the protocol correctly and refrain from using falsified data as input, they may record and analyze the intermediate results obtained while following the protocol in order to gain as much information as possible. It is also assumed that no party collude with Trent as this will give Trent access to information.

The computational requirements of the algorithm are the same as that of the power iteration method. In addition, each iteration requires the encryption and decryption of two k dimensional vectors, where k is the dimensionality of the data, as well as transmission of the encrypted vectors to and from Trent. Nevertheless, the encryption and transmission overhead, which is linear in k, may be expected to be significantly lower than the calculating the QR decomposition or similar methods which require repeated transmission of entire matrices. In general, the computational cost of the protocol is dependent on the degree of security we desire as required by the application.

2 Preliminaries

2.1 Power Iteration Method

The power iteration method [7] is an algorithm to find the principal eigenvector and its associated eigenvalue for square matrices. To simplify explanation, we assume that the matrix is diagonalizable with real eigenvalues, although the algorithm is applicable to gen-

eral square matrices as well [11]. Let A be a size $N \times N$ matrix whose eigenvalues are $\lambda_1, \ldots, \lambda_N$.

The power iteration method computes the principal eigenvector of A through the iteration

$$x_{n+1} = \frac{Ax_n}{\|Ax_n\|_2},$$

where x_n is a N dimensional vector. If the principal eigenvalue is unique, the series $\omega_n = A^n x_0$ is guaranteed to converge to a scaling of the principal eigenvector. In the standard algorithm, ℓ_2 normalization is used to prevent the magnitude of the vector from overflow and underflow. Other normalization factors can also be used if they do not change the limit of the series.

We assume wlog that $|\lambda_1| \geq \cdots \geq |\lambda_N| \geq 0$. Let v_i be the normalized eigenvector corresponding to λ_i. Since A is assumed to be diagonalizable, the eigenvectors $\{v_1, \ldots, v_N\}$ create a basis for $\mathbb{R}^N$. For unique values of $c_i \in \mathbb{R}^N$, any vector $x_0 \in \mathbb{R}^N$ can be written as $x_0 = \sum_{i=1}^{N} c_i v_i$. It can be shown that $\frac{1}{|\lambda_1|^n} A^n x_0$ is asymptotically equal to $c_1 v_1$ which forms the basis of the power iteration method and the convergence rate of the algorithm is $\left|\frac{\lambda_2}{\lambda_1}\right|$. The algorithm converges quickly when there is no eigenvalue close in magnitude to the principal eigenvalue.

2.2 Homomorphic Encryption

A homomorphic encryption algorithm allows for operations to be perform on the encrypted data without requiring to know the unencrypted values. If $\cdot$ and $+$ are two operators and x and y are two plaintext elements, a homomorphic encryption function E satisfies

$$E[x] \cdot E[y] = E[x + y].$$

In this work, we primarily use the asymmetric key Paillier cryptosystem [9] which provides additive homomorphism as well as semantic security. We briefly describe the formulation of Paillier cryptosystem below.

1. Key Generation: We choose two large prime numbers p and q, and let $N = pq$. Let $\mathbb{Z}^*_{N^2} \subset \mathbb{Z}_{N^2} = \{0, 1, ..., N^2 - 1\}$ denote the set of non-negative integers that have multiplicative inverses modulo N^2. Select $g \in \mathbb{Z}^*_{N^2}$ such that $\gcd(L(g^\lambda \mod N^2), N) = 1$, where $\lambda = \text{lcm}(p-1, q-1)$, and $L(x) = \frac{x-1}{N}$. Let (N, g) be the public key, and (p, q) be the private key.

2. Encryption: For a plaintext message $m \in \mathbb{Z}_N$, the ciphertext is given by

$$E[m] = g^m r^N \mod N^2, \tag{1}$$

   where $r \in \mathbb{Z}^*_N$ is a number chosen at random.

3. Decryption. For a ciphertext $c \in \mathbb{Z}_{N^2}$, the corresponding plaintext is given by

$$E^{-1}[c] = \frac{L(c^\lambda \mod N^2)}{L(g^\lambda \mod N^2)}. \tag{2}$$

Note that the decryption works irrespective of the value of r used during encryption. Since r can be chosen at random for every encryption, the Paillier cryptosystem is probabilistic, and therefore semantically secure. It can be easily verified that the following homomorphic properties hold for the mapping given by Equation (1) from the plaintext group $(\mathbb{Z}_N, +)$ to the ciphertext group $(\mathbb{Z}^*_{N^2}, \cdot)$.

3 Privacy Preserving Protocol

3.1 Data Setup and Privacy Conditions

We formally define the problem, in which multiple parties, try to compute the principal eigenvector over their collectively held datasets without disclosing any information to each other. For simplicity, we describe the problem with two parties, Alice and Bob; and later show that the algorithm is easily extended to multiple parties.

The parties Alice and Bob are assumed to be *semi-honest* which means that the parties will follow the steps of the protocol correctly and will not try to cheat by passing falsified data aimed at extracting information about other parties. The parties are assumed to be curious in the sense that they may record the outcomes of all intermediate steps of the protocol to extract any possible information. The protocol is coordinated by the semi-honest arbitrator Trent. Alice and Bob communicate directly with Trent rather than each other. Trent performs all the intermediate computations and transfers the results to each party. Although Trent is trusted not to collude with other parties, it is important to note that the parties do not trust Trent with their data and intend to prevent him from being able to see it. Alice and Bob hide information by using a shared key cryptosystem to send only encrypted data to Trent.

We assume that both the datasets can be represented as matrices in which columns and rows correspond to the data samples and the features, respectively. For instance, the individual email collections of Alice and Bob are represented as matrices A and B respectively, in which the columns correspond to the emails, and the rows correspond to the words. The entries of these matrices represent the frequency of occurrence of a given word in a given email. The combined dataset may be split between Alice and Bob in two possible ways. In a *instance*-split, both Alice and Bob have a disjoint set of data instances with the same set of features. The aggregate dataset is obtained by concatenating columns given by the data matrix $M = \begin{bmatrix} A & B \end{bmatrix}$ and correlation matrix $M^T M$. In a *feature*-split, Alice and Bob have different features of the same data. The aggregate data matrix M is obtained by concatenating rows given by the data matrix $M = \begin{bmatrix} A \\ B \end{bmatrix}$ and correlation matrix MM^T. If v is an eigenvector of $M^T M$ with a non-zero eigenvalue λ, we have

$$M^T M v = \lambda v \;\Rightarrow\; MM^T M v = \lambda M v.$$

Therefore, $Mv \neq 0$ is the eigenvector of MM^T with eigenvalue λ. Similarly, any eigenvector of feature-split data MM^T associated with a non-zero eigenvalue is an eigenvector of instance-split data $M^T M$ corresponding to the same eigenvalue. Hence, we mainly deal with calculating the principal eigenvector of the instance-split data. In practice the correlation matrix that has the smaller size should be used to reduce the computational cost of eigen-decomposition algorithms.

For the instance-split case, if Alice's data A is of size $k \times m$ and Bob's data B is of size $k \times n$, the combined data matrix will be $M_{k\times(m+n)}$. The correlation matrix of size $(m+n)\times(m+n)$ is given by

$$M^T M = \begin{bmatrix} A^T A & A^T B \\ B^T A & B^T B \end{bmatrix}.$$

3.2 The Basic Protocol

The power iteration algorithm computes the principal eigenvector of $M^T M$ by updating and normalizing the vector x_t until convergence. Starting with a random vector x_0, we calculate

$$x_{i+1} = \frac{M^T M\, x_i}{\|M^T M\, x_i\|}.$$

For privacy, we split the vector x_i into two parts, α_i and β_i. α_i corresponds to the first m components of x_i and β_i corresponds to the remaining n components. In each iteration, we need to securely compute

$$M^T M x_i = \begin{bmatrix} A^T A & A^T B \\ B^T A & B^T B \end{bmatrix} \begin{bmatrix} \alpha_i \\ \beta_i \end{bmatrix} = \begin{bmatrix} A^T(A\alpha_i + B\beta_i) \\ B^T(A\alpha_i + B\beta_i) \end{bmatrix} = \begin{bmatrix} A^T u_i \\ B^T u_i \end{bmatrix} \tag{3}$$

where $u_i = A\alpha_i + B\beta_i$. After convergence, α_i and β_i will represent shares held by Alice and Bob of the principal eigenvector of $M^T M$.

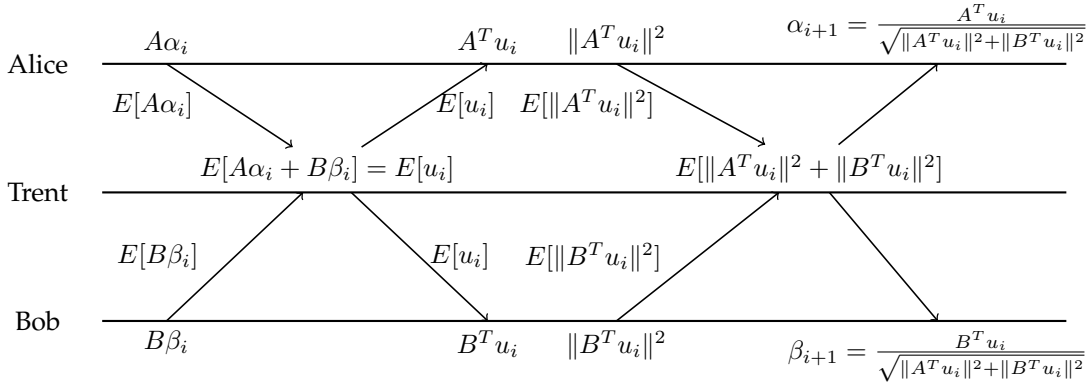

Figure 1: Visual description of the protocol.

This now lays the groundwork for us to define a distributed protocol in which Alice and Bob work only on their portions of the data, while computing the principal eigenvector of the combined data in collaboration with a third party Trent. An iteration of the algorithm proceeds as illustrated in Figure 1. At the outset Alice and Bob randomly generate component vectors α_0 and β_0 respectively. At the beginning of the i^{th} iteration, Alice and Bob possess component vectors α_i and β_i respectively. They compute the product of their data and their corresponding component vectors as $A\alpha_i$ and $B\beta_i$. To compute u_i, Alice and Bob individually transfer these products to Trent. Trent adds the contributions from Alice and Bob by computing

$$u_i = A\alpha_i + B\beta_i.$$

He then transfers u_i back to Alice and Bob, who then individually compute $A^T u_i$ and $B^T u_i$, without requiring data from one other. For normalization, Alice and Bob also need to securely compute the term

$$\|M^T M\, x_i\| = \sqrt{\|A^T u_i\|^2 + \|B^T u_i\|^2}. \tag{4}$$

Again, Alice and Bob compute the individual terms $\|A^T u_i\|^2$ and $\|B^T u_i\|^2$ respectively and transfer it to Trent. As earlier, Trent computes the sum

$$\|A^T u_i\|^2 + \|B^T u_i\|^2$$

and transfers it back to Alice and Bob. Finally, Alice and Bob respectively update α and β vectors as

$$\begin{aligned} u_i &= A\alpha_i + B\beta_i, \\ \alpha_{i+1} &= \frac{A^T u_i}{\sqrt{\|A^T u_i\|^2 + \|B^T u_i\|^2}}, \\ \beta_{i+1} &= \frac{B^T u_i}{\sqrt{\|A^T u_i\|^2 + \|B^T u_i\|^2}}. \end{aligned} \tag{5}$$

The algorithm terminates when the α and β vectors converge.

3.3 Making the Protocol More Secure

The basic protocol described above is provably correct. After convergence, Alice and Bob end up with the principal eigenvector of the row space of the combined data, as well as concatenative shares of the column space which Trent can gather to compute the principal eigenvector. However the protocol is not completely secure; Alice and Bob obtain sufficient information about properties of each others' data matrices, such as their column spaces, null spaces, and correlation matrices. We present a series of modifications to the basic protocol so that such information is not revealed.

3.3.1 Homomorphic Encryption: Securing the data from Trent.

The central objective of the protocol is to prevent Trent from learning anything about either the individual data sets or the combined data other than the principal eigenvector of the combined data. Trent receives a series of partial results of the form $AA^T u$, $BB^T u$ and $MM^T u$. By analyzing these results, he can potentially determine the entire column spaces of Alice and Bob as well as the combined data. To prevent this, we employ an additive homomorphic cryptosystem introduced in Section 2.2.

At the beginning of the protocol, Alice and Bob obtain a shared public key/private key pair for an additive homomorphic cryptosystem from an authenticating authority. The public key is also known to Trent who, however, does not know the private key; While he can encrypt data, he cannot decrypt it. Alice and Bob encrypt all transmissions to Trent, at the first transmission step of each iteration Trent receives the encrypted inputs $E[A\alpha_i]$ and $E[B\beta_i]$. He multiplies the two element by element to compute $E[A\alpha_i] \cdot E[B\beta_i] = E[A\alpha_i + B\beta_i] = E[u_i]$. He returns $E[u_i]$ to both Alice and Bob who decrypt it with their private key to obtain u_i. In the second transmission step of each iteration, Alice and Bob send $E[\|A^T u_i\|^2]$ and $E[\|B^T u_i\|^2]$ respectively to Trent, who computes the encrypted sum

$$E\left[\|A^T u_i\|^2\right] \cdot E\left[\|B^T u_i\|^2\right] = E\left[\|A^T u_i\|^2 + \|B^T u_i\|^2\right]$$

and transfers it back to Alice and Bob, who then decrypt it to obtain $\|A^T u_i\|^2 + \|B^T u_i\|^2$, which is required for normalization.

This modification does not change the actual computation of the power iterations in any manner. Thus the procedure remains as correct as before, except that Trent now no longer

has any access to any of the intermediate computations. At the termination of the algorithm he can now receive the converged values of α and β from Alice and Bob, who will send it in clear text.

3.3.2 Random Scaling: Securing the Column Spaces.

After Alice and Bob receive $u_i = A\alpha_i + B\beta_i$ from Trent, Alice can calculate $u_i - A\alpha_i = B\beta_i$ and Bob can calculate $u_i - B\beta_i = A\alpha_i$. After a sufficient number of iterations, particularly in the early stages of the computation (when u_i has not yet converged) Alice can find the column space of B and Bob can find the column space of A. Similarly, by subtracting their share from the normalization term returned by Trent, Alice and Bob are able to find $\|B^T u_i\|^2$ and $\|A^T u_i\|^2$ respectively.

In order to prevent this, Trent multiplies u_i with a randomly generated scaling term r_i that he does not share with anyone. Trent computes

$$(E[A\alpha_i] \cdot E[B\beta_i])^{r_i} = E[r_i(A\alpha_i + B\beta_i)] = E[r_i u_i]$$

by performing element-wise exponentiation of the encrypted vector by r_i and transfers $r_i u_i$ to Alice and Bob. By using a different value of r_i at each iteration, Trent ensures that Alice and Bob are not able to calculate $B\beta_i$ and $A\alpha_i$ respectively. In the second step, Trent scales the normalization constant by r_i^2,

$$\left(E\left[\|A^T u_i\|^2\right] \cdot E\left[\|B^T u_i\|^2\right]\right)^{r_i^2} = E\left[r_i^2\left(\|A_i^T u\|^2 + \|B_i^T u\|^2\right)\right].$$

Normalization causes the r_i factor to cancel out and the update rules remain unchanged.

$$\begin{aligned} u_i &= A\alpha_i + B\beta_i, \\ \alpha_{i+1} &= \frac{r_i A^T u_i}{\sqrt{r_i^2\left(\|A^T u_i\|^2 + \|B^T u_i\|^2\right)}} = \frac{A^T u_i}{\sqrt{\|A^T u_i\|^2 + \|B^T u_i\|^2}}, \\ \beta_{i+1} &= \frac{r_i B^T u_i}{\sqrt{r_i^2\left(\|A^T u_i\|^2 + \|B^T u_i\|^2\right)}} = \frac{B^T u_i}{\sqrt{\|A^T u_i\|^2 + \|B^T u_i\|^2}}. \end{aligned} \tag{6}$$

The random scaling does not affect the final outcome of the computation, and the algorithm remains correct as before.

3.3.3 Data Padding: Securing null spaces.

In each iteration, Alice observes one vector $r_i u_i = r_i(A\alpha_i + B\beta_i)$ in the column space of $M = [A\ B]$. Alice can calculate the *null space* $H(A)$ of A, given by

$$H(A) = \{x \in \mathbb{R}^m | Ax = 0\}$$

and pre-multiply a non-zero vector $x \in H(A)$ with $r_i u_i$ to calculate

$$x r_i u_i = r_i x(A\alpha_i + B\beta_i) = r_i x B\beta_i.$$

This is a projection of $B\beta_i$, a vector in the column space of B into the null space $H(A)$. Similarly, Bob can find projections of $A\alpha_i$ in the null space $H(B)$. While considering the projected vectors separately will not give away much information, after several iterations

Alice will have a projection of the column space of B on the null space of A, thereby learning about the component's of Bob's data that lie in her null space. Bob can similarly learn about the component's of Alice's data that lie in his null space.

In order to prevent this, Alice *pads* her data matrix A by concatenating it with a random matrix $P_a = r_a I_{k\times k}$, to obtain $\begin{bmatrix} A & P_a \end{bmatrix}$ where r_a is a positive scalar chosen by Alice. Similarly, Bob pads his data matrix B with $P_b = r_b I_{k\times k}$ to obtain $\begin{bmatrix} B & P_b \end{bmatrix}$ where r_b is a different positive scalar chosen by Bob. This has the effect of hiding the null spaces in both their data sets. In the following lemma, we prove that the eigenvectors of the combined data do not change after padding, while every eigenvalue λ of MM^T is now modified to $\lambda + r_a + r_b$. Please refer to appendix for the proof.

Lemma 1. *Let $\bar{M} = \begin{bmatrix} M & P \end{bmatrix}$ where M is a $s \times t$ matrix, and P is a $s \times s$ orthogonal matrix. If $\bar{v} = \begin{bmatrix} v_{t\times 1} \\ v'_{s\times 1} \end{bmatrix}$ is an eigenvector of $\bar{M}^T\bar{M}$ corresponding to an eigenvalue λ, then v is an eigenvector of M^TM.*

While the random factors r_a and r_b prevent Alice and Bob from estimating the eigenvalues of the data, the computation of principal eigenvector remains correct as before.

3.3.4 Oblivious Transfer: Securing Krylov spaces.

For a constant c, we can show that the vector $u_i = A\alpha_i + B\beta_i$ is equal to cMM^Tu_{i-1}. The sequence of vectors $U = \{u_1, u_2, u_3, \ldots\}$ form the Krylov subspace $(MM^T)^n u_1$ of the matrix MM^T. Knowledge of this series of vectors can reveal all eigenvectors of MM^T. Consider $u_0 = c_1v_1 + c_2v_2 + \cdots$, where v_i is the i^{th} eigenvector. If λ_j is the j^{th} eigenvalue, we have $u_i \propto c_1\lambda_1 v_1 + c_2\lambda_2 v_2 + \cdots$. We assume wlog that the eigenvalues λ are in a descending order, *i.e.*, $\lambda_j \geq \lambda_k$ for $j < k$. Let u_{conv} be the normalized converged value of u_i which is equal to the normalized principal eigenvector v_1.

Let $w_i = u_i - (u_i^T u_{conv})u_i$ which can be shown to be equal to $c_2\lambda_2 v_2 + c_3\lambda_3 v_3 + \cdots$, *i.e.*, a vector with no component along v_1. If we perform power iterations with initial vector w_1, the converged vector w_{conv} will be equal to the eigenvector corresponding to the second largest eigenvalue. Hence, once Alice has the converged value, u_{conv}, she can subtract it out of all the stored u_i values and determine the second principal eigenvector of MM^T. She can repeat the process iteratively to obtain all eigenvectors of MM^T, although in practice the estimates become noisy very quickly. As we will show in Section 4, the following modification prevents Alice and Bob from identifying the Krylov space with any certainty and they are thereby unable to compute the additional eigenvectors of the combined data.

We introduce a form of oblivious transfer (OT) [10] in the protocol. We assume that Trent stores the encrypted results of intermediate steps at every iteration. After computing $E[r_iu_i]$, Trent either sends this quantity to Alice and Bob with a probability p or sends a random vector $E[u'_i]$ of the same size ($k \times 1$) with probability $1-p$. As the encryption key of the cryptosystem is publicly known, Trent can encrypt the vector u'_i. Alice and Bob do not know whether they are receiving r_iu_i or u'_i. If a random vector is sent, Trent continues with the protocol, but ignores the terms Alice and Bob return in the next iteration, $E[A\alpha_{i+1}]$ and $E[B\beta_{i+1}]$. Instead, he sends the result of the last non-random iteration j, $E[r_ju_j]$, thereby restarting that iteration.

This sequence of data sent by Trent is an example of a Bernoulli Process [12]. An illustrative example of the protocol is shown in Figure 2. In the first two iterations, Trent sends valid vectors r_1u_1 and r_2u_2 back to Alice and Bob. In the beginning of the third iteration, Trent receives and computes $E[r_3u_3]$ but sends a random vector u'_3. He ignores what Alice

and Bob send him in the fourth iteration and sends back $E[r_3u_3]$ instead. Trent then stores the vector $E[r_4u_4]$ sent by Alice and Bob in the fifth iteration and sends a random vector u'_2. Similarly, he ignores the computed vector of the sixth iteration and sends u'_3. Finally, he ignores the computed vector of the seventh iteration and sends $E[r_4u_4]$.

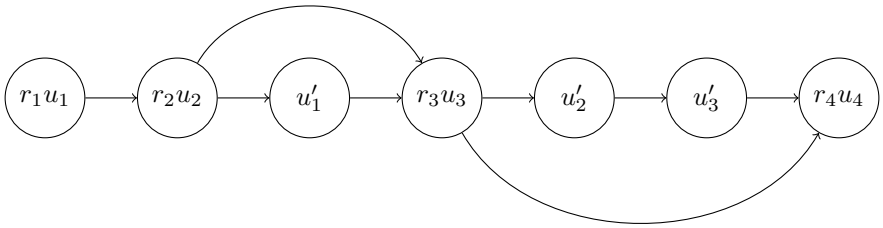

Figure 2: An example of the protocol execution with oblivious transfer.

This modification has two effects – firstly it prevents Alice and Bob from identifying the Krylov space with certainty. As a result, they are now unable to obtain additional Eigenvectors from the data. Secondly, oblivious transfer essentially obfuscates the projection of the column space of B on to the null space of A for Alice, and analogously for Bob by introducing random vectors. As Alice and Bob do not know which vectors are random, they cannot completely calculate the true projection of each others data on the null spaces. This is rendered less important if Alice and Bob pad their data as suggested in the previous subsection.

Alice and Bob can store the vectors they receive from Trent in each iteration. By analyzing the distribution of the normalized vectors, Alice and Bob can identify the random vectors using a simple outlier detection technique. To prevent this, one possible solution is for Trent to pick a previously computed value of r_ju_j and add zero mean noise e_i, for instance, sampled from the Gaussian distribution.

$$u'_i = r_ju_j + e_i, \;\; e_i \sim \mathcal{N}(0, \sigma^2).$$

Instead of transmitting a perturbation of a previous vector, Trent can also use perturbed mean of a few previous r_ju_j with noise. Doing this will create a random vector with the same distributional properties as the real vectors. The noise variance parameter σ controls the error in identifying the random vector from the valid vectors and how much error do we want to introduce in the projected column space.

Oblivious transfer has the effect of increasing the total computation as every iteration in which Trent sends a random vector is wasted. In any secure multi-party computation, there is an inherent trade-off between computation time and the degree of security. The parameter p which is the probability of Trent sending a non-random vector allows us to control this at a fine level based on the application requirements. As before, introducing oblivious transfer does not affect the correctness of the computation – it does not modify the values of the non-random vectors u_i.

3.4 Extension to Multiple Parties

As we mentioned before, the protocol can be naturally extended to multiple parties. Let us consider the case of N parties: $P_1, \ldots, P_N$ each having data $A_1, \ldots, A_N$ of sizes $k \times$

$n_1, \ldots, k \times n_N$ respectively. The parties are interested in computing the principal eigenvector of the combined data without disclosing anything about their data. We make the same assumption about the parties and the arbitrator Trent being *semi-honest*. All the parties except Trent share the decryption key to the additive homomorphic encryption scheme and the encryption key is public.

In case of a data split, for the combined data matrix $M = \begin{bmatrix} A_1 & A_2 & \cdots & A_N \end{bmatrix}$, the correlation matrix is

$$M^T M = \begin{bmatrix} A_1^T A_1 & \cdots & A_1^T A_N \\ \vdots & \ddots & \vdots \\ A_N^T A_1 & \cdots & A_N^T A_N \end{bmatrix}.$$

We split the eigenvector into N parts, $\alpha_1, \ldots, \alpha_N$ of size $n_1, \ldots, n_N$ respectively, each corresponding to one party. For simplicity, we describe the basic protocol with homomorphic encryption; randomization and oblivious transfer can be easily added by making the same modifications as we saw in Sections 3.3. One iteration of the protocol starts with the i^{th} party computing $A_i\alpha_i$ and transferring to Trent the encrypted vector $E[A_i\alpha_i]$. Trent receives this from each party and computes

$$\prod_i E\left[A_i\alpha_i\right] = E\left[\sum_i A_i\alpha_i\right] = E[u]$$

where $u = \sum_i A_i\alpha_i$, and product is an element-wise operation. Trent sends the encrypted vector $E[u]$ back to $P_1, \ldots, P_N$ who decrypt it and individually compute $A_i^T u$. The parties individually compute $\|A_i^T u\|^2$ and send its encrypted value to Trent. Trent receives N encrypted scalars $E\left[\|A_i^T u\|^2\right]$ and calculates the normalization term

$$\prod_i E\left[\|A_i^T u\|^2\right] = E\left[\sum_i \|A_i^T u\|^2\right]$$

and sends it back to the parties. At the end of the iteration, the party P_i updates α_i as

$$\begin{aligned} u &= \sum_i A_i\alpha_i^{(old)}, \\ \alpha_i^{(new)} &= \frac{A_i^T u}{\sqrt{\sum_i \|A_i^T u\|^2}}. \end{aligned} \tag{7}$$

The algorithm terminates when any one party P_i converges on α_i.

4 Analysis

4.1 Correctness

The protocol outlined in Section 3.2 is provably correct. The steps introduced in Section 3.3 do not modify the operation and hence the accuracy of the protocol in any manner.

4.2 Security

As a consequence of the procedures introduced in Section 3.3.2 the row spaces and null spaces of the parties are hidden from each another. In the multiparty scenario, the protocol

is also robust to collusion between parties with data, although not to collusion between Trent and any of the other parties. If two parties out of N collude, they will find information about each other, but will not learn anything about the data of the remaining $N-2$ parties.

What remains is the information which can be obtained from the sequence of u_i vectors. Alice receives the following two sets of matrices:

$$U = \{u_1, u_2, u_3, \ldots\}, \;\; U' = \{u'_1, u'_2, \ldots\}$$

representing the outcomes of valid iterations and the random vectors respectively. In the absence of the random data U', Alice only receives U. As mentioned in Section 3.3.4, $u_i = (MM^T)^i u_0$ which is a sequence of vectors from the Krylov space of the matrix $AA^T + BB^T$ sufficient to determine all eigenvectors of MM^T. For k-dimensional data, it is sufficient to have any sequence of k vectors in U to determine MM^T. Hence, if the vectors in U were not interspersed with the vectors in U', the algorithm essentially reveals information about all eigenvectors to all parties. Furthermore, given a sequence $u_i, u_{i+1}, u_{i+2}, \ldots, u_{i+k-1}$ vectors from U, Alice can *verify* that they are indeed from the Krylov space.[1] Introducing random scaling $r_i u_i$ makes it harder still to verify Krylov space. While solving for k vectors, Alice and Bob need to solve for another k parameters $r_1, \ldots, r_k$.

Security is obtained from the following observation: although Alice can *verify* that a given set of vectors forms a sequence in the Krylov space, she cannot *select* them from a larger set without exhaustive evaluation of all k sets of vectors. If the shortest sequence of k vectors from the Krylov space is embedded in a longer sequence of N vectors, Alice needs $\binom{N}{k}$ checks to find the Krylov space, which is a combinatorial problem.

4.3 Efficiency

First we analyze the computational time complexity of the protocol. As the total number of iterations is data dependent and proportional to $\left|\frac{\lambda_1}{\lambda_2}\right|$, we analyze the cost per iteration. The computation is performed by the individual parties in parallel, though synchronized and the parties also spend time waiting for intermediate results from other parties. The oblivious transfer introduces extra iterations with random data, on average the number of iterations needed for convergence increase by a factor of $\frac{1}{p}$, where p is the probability of Trent sending a non-random vector. As the same operations are performed in an iteration with a random vector, its the time complexity would be the same as an iteration with a non-random vector.

In the i^{th} iteration, Alice and Bob individually need to perform two matrix multiplications: $A\alpha_i$ and $A^T(A\alpha_i + B\beta_i)$, $B\beta_i$ and $B^T(A\alpha_i + B\beta_i)$ respectively. The first part involves multiplication of a $k \times m$ matrix by a m dimensional vector which is $O(km)$ operations for Alice and $O(kn)$ for Bob. The second part involves multiplication of a $m \times k$ matrix by a k dimensional vector which is $O(km)$ operations for Alice and $O(kn)$ for Bob. Calculating $\|A^T(A\alpha_i + B\beta_i)\|^2$ involves $O(m)$ operations for Alice and analogously $O(n)$ operations for Bob. The final step involves only a normalization by a scalar and can be again done in linear time, $O(m)$ for Alice and $O(n)$ for Bob. Therefore, total time complexity of computations performed by Alice and Bob is $O(km) + O(m) = O(km)$ and $O(kn) + O(n) = O(kn)$ operations respectively. Trent computes an element-wise product of two k dimensional vectors $A\alpha_i$ and $B\beta_i$ which is $O(k)$ operations. The multiplication of two encrypted scalar requires only one operation, making Trent's total time complexity $O(k)$.

[1] if the spectral radius of MM^T is 1.

In each iteration, Alice and Bob encrypt and decrypt two vectors and two scalar normalization terms which is equivalent to performing $k+1$ encryptions and $k+1$ decryptions individually, which is $O(k)$ encryptions and decryptions.

In the i^{th} iteration, Alice and Bob each need to transmit k dimensional vectors to Trent who computes $E(A\alpha_i + B\beta_i)$ and transmits it back: involving the transfer of $4k$ elements. Similarly, Alice and Bob each transmit one scalar norm value to Trent who sends back another scalar value involving in all the transfer of 4 elements. In total, each iteration requires the transmission of $4k + 4 = O(k)$ data elements.

To summarize, the time complexity of the protocol per iteration is $O(km)$ or $O(kn)$ operations whichever is larger, $O(k)$ encryptions and decryptions, and $O(k)$ transmissions. In practice, each individual encryption/decryption and data transmission take much longer than performing computation operation.

5 Experiments

In this section, we analyze the results of experiments with a prototype system implementing the privacy preserving protocol over simulated data and the USPS handwritten digits dataset. We mainly evaluate the protocol for accuracy as compared to standardized eigenvector computation and execution time.

5.1 Implementation

We created a prototype implementation of the protocol in C++ and used the variable precision arithmetic libraries provided by OpenSSL [13] to implement the Paillier cryptosystem. In the following experiments, we used the Paillier cryptosystem with 1024-bit keys. We performed the experiments on a Intel Core 2 Duo machine with 3 GB RAM running 64-bit Ubuntu.

5.1.1 Issues: Floating point numbers

The normalized eigenvectors typically contain floating point numbers and are central to our data representation. Most cryptosystems have no natural way of encrypting and decrypting floating point numbers. We addressed this issue by using a simple fixed precision representation. At each encryption operation, we multiply and floor a floating point number by a fixed constant that is a power of two for efficient arithmetic. After decryption, we divide by the same factor to get restore the floating-point number. For a constant c,

$$E[cx] \cdot E[cy] = E[c(x+y)].$$

By doing this, we restrict the precision to a fixed quantity by choosing only the first few digits ($\log_{10} c$) after the floating point. If c is too large, there is a risk of overflow, which we avoid by scaling the maximum allowable floating point numbers by $\frac{1}{c}$. In practice, this reduction in precision causes a trivially small error in the computation.

5.2 Simulated Data

In order to evaluate the protocol with a wide range of matrix sizes and configurations, we experimented random data matrices sampled from uniform and Gaussian distributions as described below.

5.2.1 Accuracy

We used data matrices A and B each of size 1000×1000 sampled from the uniform distribution. Similar to [6], we report the mean squared error (MSE) in calculating the principal eigenvector calculated by our protocol as compared to standard calculation done by MATLAB using the same input data matrices. If $v = [\alpha^T \beta^T]^T$ and v' is the principal eigenvector calculated by MATLAB,

$$MSE = \frac{1}{m+n} \sum_{i=1}^{m+n} (v - v')^2.$$

We use the convergence threshold of 10^{-6} between iterations as the termination criteria, *i.e.*, in the i^{th} iteration, $||\alpha_{i+1} - \alpha_i|| < 10^{-6}$ for Alice and $||\beta_{i+1} - \beta_i|| < 10^{-6}$ for Bob. For the multiplicative factor $c = 10^6$, we observed an MSE of 1.7341×10^{-12}. The protocol required 1078.94 s to execute till convergence, while the MATLAB execution took 2.64 s. We also report the MSE for other input matrices sampled from the standard uniform and Gaussian distributions in Table 1. We observe that the MSE in each case is negligible for practical applications. As the eigenvector computation over the data sampled from Gaussian distribution takes substantially more iterations to converge compared to the data sampled from uniform distribution, the corresponding MSE is higher in the former case.

Data Distribution	**k**	**m,m**	**n**	**MSE**
Uniform	1000	500	500	8.7467×10^{-16}
Uniform	1000	1000	1000	8.6705×10^{-16}
Uniform	1000	2000	2000	8.4310×10^{-16}
Uniform	2000	1000	1000	8.4980×10^{-16}
Uniform	2000	2000	2000	8.3430×10^{-16}
Gaussian	1000	500	500	5.1531×10^{-12}
Gaussian	1000	1000	1000	9.6460×10^{-12}
Gaussian	1000	2000	2000	1.2306×10^{-12}
Gaussian	2000	1000	1000	5.6775×10^{-12}
Gaussian	2000	2000	2000	4.4545×10^{-12}

Table 1: Mean squared error in calculating principal eigenvector using the protocol.

5.2.2 Execution Time

We analyze the execution time of the protocol with the size of the data matrices. As the number of iterations are data dependent, we report the average execution time per iteration. We again use data matrices sampled from the uniform distribution.

We measured the execution times for data matrix A and B both having fixed number of rows $k = 1000$ and varying number of columns m, n, and varying Paillier encryption key sizes as shown in Table 2. The execution time is consistently observed to increase with increasing encryption key size. It can be seen that the execution time per iteration increases almost linearly with the number of data samples, but the rate of increase is very small. This follows from our complexity analysis where the asymptotic computational cost is in $O(km)$ for Alice and $O(kn)$ for Bob. As the encryption cost $O(k)$ only depends on k, it remains constant for a fixed value of k. The increase in execution time is only because of the extra computation cost in this case.

$\mathbf{m = n}$	**time (s) 256-bit**	**time (s) 512-bit**	**time (s) 1024-bit**
1000	3.701	25.266	188.785
2000	3.843	25.348	188.925
3000	3.986	25.487	189.009
4000	4.118	25.559	189.082
5000	4.290	25.650	190.095

Table 2: Execution time per iteration for fixed k = 1000 and varying m and n using different Paillier encryption key sizes.

In Table 3 we show the execution times for data matrix A and B both having fixed number of columns $m = n = 1000$ and varying number of rows k. As before, the execution time increases with the increasing encryption key size. It can also be seen that the execution time per iteration increases linearly with k. This is coherent to our complexity analysis where we saw that the asymptotic encryption cost of the protocol is linear $O(k)$. The computation cost also increases for increasing values of k as the asymptotic computational cost is $O(km)$ for Alice and $O(kn)$ for Bob. Finally, we observe that the execution time increases sharply for increasing values of k as opposed to increasing m and n. This indicates that the encryption cost completely overwhelms the computation cost. This is summarized in Figure 3.

k	**time (s) 256-bit**	**time (s) 512-bit**	**time (s) 1024-bit**
1000	3.701	25.266	188.785
2000	7.381	50.334	380.272
3000	11.073	75.505	569.799
4000	14.805	100.579	763.727
5000	18.710	125.752	957.811

Table 3: Execution time per iteration for fixed $m = n$ = 1000 and varying k using different Paillier encryption key sizes.

5.3 USPS Handwritten Digits Data

The USPS handwritten digits dataset[2] consists of 1100 instances of each digits from '0' to '9' represented as 8-bit gray-scale images. We consider the scenario where the instances for a digit are split between two users. Finding the principal eigenvector of this data is a useful step in performing principal component analysis and other image processing tasks.

The data for a single digit can be represented as a matrix M of size 256 $\times$ 1100. We split it into two matrices A and B each of size 256 $\times$ 550 ($k = 256$, $m = n = 550$). We report the accuracy and execution time over this setup for the data instances representing the digit '0', but the results can also be generalized for other digits due to similar nature of the data.

[2]This dataset can be downloaded from `http://cs.nyu.edu/~roweis/data.html`

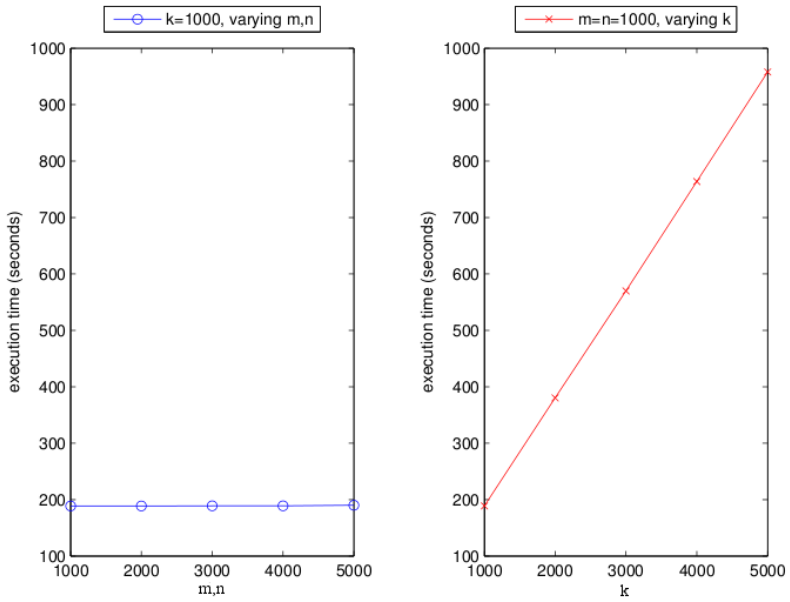

Figure 3: Execution time per iteration using 1024-bit Paillier encryption. The plot on the left represents the case with fixed k = 1000 and varying m and n. The plot on the right represents the case with fixed $m = n$ = 1000 and varying k.

5.3.1 Accuracy

We use the mean squared error (MSE) in calculating the principal eigenvector as compared to the computation done by MATLAB as a metric of accuracy as before. We used 1024-bit Paillier encryption and the convergence threshold of 10^{-6} and multiplicative factor $c = 10^6$. For the data representing the digit '0', we observed an MSE of 1.1037×10^{-8}. This computation converged in 7 iterations and required an execution time of 385.765 s. We observed the same MSE using 256-bit and 512-bit Paillier encryption. This error can be considered as negligible for practical applications.

5.3.2 Execution Time

We observed the average per-iteration execution time of the protocol with different Paillier encryption key-sizes. This is summarized in Table 4. As expected, the execution time for the protocol using uniformly random data of the same size ($k = 256$, $m = n = 550$) was approximately the same.

time (s) 256-bit	time (s) 512-bit	time (s) 1024-bit
0.923	6.417	55.109

Table 4: Execution time over USPS data for the digit '0' using different Paillier encryption key sizes.

6 Conclusion and Future Work

In this paper we proposed a protocol for computing the principal eigenvector of the combined data shared by multiple parties coordinated by a semi-honest arbitrator Trent. The data matrices belonging to individual parties and correlation matrix of the combined data is protected and cannot be reconstructed. We used randomization, data padding, and oblivious transfer to hide the information which the parties can learn from the intermediate results. The computational cost for each party is $O(km)$ where k is the number of features and m data instances along with $O(k)$ encryption/decryption operations and $O(k)$ data transfer operations. We observe similar behavior in experimental results using a prototype implementation of the protocol over simulated data and USPS handwritten digits data.

Potential future work include extending the protocol to finding the complete singular value decomposition, particularly with efficient algorithms such as thin-SVD [14]. Some of the techniques like data padding, oblivious transfer we applied to increase the security of the protocol can be used in other problems as well. We are working towards a unified theoretical model for applying and analyzing these techniques in general.

While we considered privacy under the semi-honest model, we are also interested in extending it to the malicious model in the future. In this model, the parties are not assumed to follow the protocol correctly and would try to gain as much information as possible, *e.g.* a party might send zero instead of the norm $\|A^T u_i\|$ and use this to infer the norm corresponding to the other party. Similarly, Trent might collude with a party and choose to not apply random scaling, allowing one party to identify the column space of the other data belonging to another party. The conventional method of ensuring that such malicious behavior does not take place is through *zero-knowledge proofs* (ZKPs). Kantarcioglu and Kardes [15] present simple two party protocols for equality, inner product, and set operations in the malicious model using primitives such as thresholded decryption [16]. While these primitives form an attractive basis to design and theoretically analyze privacy preserving algorithms, they are found to be many orders of magnitude expensive in practice as compared to the corresponding non-private computation. Duan, et al. [17] present P4P, which is a an alternative paradigm for designing protocols for computation involving iterations with additive updates with malicious parties using verifiable secret sharing (VSS) over small fields. This results in very efficient private arithmetic operations and the corresponding ZKPs which can be used to create protocols that can scale to large amount of data. Duan, et al. present an implementation of the privacy preserving protocol for SVD in the P4P framework. We plan to apply some of these ideas to our implementation of the principal eigenvector computation protocol and evaluate the performance as compared to this method.

References

[1] Manas Pathak and Bhiksha Raj. Privacy-preserving protocols for eigenvector computation. In *ECML/PKDD Workshop on Privacy and Security issues in Data Mining and Machine Learning*, 2010.

[2] W. F. Massy. Principal component analysis in exploratory data research. *Journal of the American Statistical Association*, 60:234–256, 1965.

[3] Ken Goldberg, Theresa Roeder, Dhruv Gupta, and Chris Perkins. Eigentaste: A constant time collaborative filtering algorithm. *Information Retrieval*, 4(2):133–151, 2001.

[4] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The pagerank citation ranking: Bringing order to the web. *Technical report, Stanford University, Stanford, CA*, 1998.

[5] Alexandre V. Evfimievski. Randomization in privacy-preserving data mining. *SIGKDD Explorations*, 4(2):43–48, 2002.

[6] Shuguo Han, Wee Keong Ng, and Philip S. Yu. Privacy-preserving singular value decomposition. In *IEEE International Conference on Data Mining*, pages 1267–1270. IEEE, 2009.

[7] Gene H. Golub and Charles F. Van Loan. *Matrix Computations*. The Johns Hopkins University Press, second edition, 1989.

[8] John Canny. Collaborative filtering with privacy. In *IEEE Symposium on Security and Privacy*, 2002.

[9] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *EUROCRYPT*, 1999.

[10] Michael Rabin. How to exchange secrets by oblivious transfer. Technical Report TR-81, Harvard University, 1981.

[11] Granville Sewell. *Computational Methods of Linear Algebra*. Wiley-Interscience, second edition, 2005.

[12] Athanasios Papoulis. *Probability, Random Variables, and Stochastic Processes*. McGraw-Hill, second edition, 1984.

[13] `http://www.openssl.org/docs/crypto/bn.html`.

[14] Matthew Brand. Fast low-rank modifications of the thin singular value decomposition. *Linear Algebra and its Applications*, 415(1):20–30, 2006.

[15] Murat Kantarcioglu and Onur Kardes. Privacy-preserving data mining in the malicious model. *J. Information and Computer Security*, 2(4):353–375, 2008.

[16] Ronald Cramer, Ivan Damgard, and Jesper B. Nielsen. Multiparty computation from threshold homomorphic encryption. In *EUROCRYPT*, 2001.

[17] Yitao Duan, John Canny, and Justin Zhan. P4P: Practical large-scale privacy-preserving distributed computation robust against malicious users. In *USENIX Security Symposium*, pages 207–222, 2010.

7 Appendix

Lemma 1. We have,

$$\bar{M}^T\bar{M} = \begin{bmatrix} M^TM & M^TP \\ P^TM & I \end{bmatrix}.$$

Multiplying by the eigenvector $\bar{v} = \begin{bmatrix} v_{t\times 1} \\ v'_{s\times 1} \end{bmatrix}$ gives us

$$\bar{M}^T\bar{M}\begin{bmatrix} v \\ v' \end{bmatrix} = \begin{bmatrix} M^TMv + M^TPv' \\ P^TMv + v' \end{bmatrix} = \lambda\begin{bmatrix} v \\ v' \end{bmatrix}.$$

Therefore,

$$M^T M v + M^T P v' = \lambda v, \tag{8}$$

$$P^T M v + v' = \lambda v'. \tag{9}$$

Since $\lambda \neq 1$, Equation (9) implies $v' = \frac{1}{\lambda - 1} P^T M v$. Substituting this into Equation (8) and the orthogonality of P gives us

$$M^T M v + \frac{1}{\lambda - 1} M^T P P^T M v = \frac{\lambda}{\lambda - 1} M^T M v = \lambda v.$$

Hence, $M^T M v = (\lambda - 1) v$. □