The Pennsylvania State University
The Graduate School
Eberly College of Science

DISCLOSURE RISK AND SECURE LASSO ON VERTICALLY DISTRIBUTED DATABASES

A Thesis in
Statistics
by
Yuji Samizo

© 2016 Yuji Samizo

Submitted in Partial Fulfillment
of the Requirements
for the Degree of

Master of Science

August 2016

The thesis of Yuji Samizo was reviewed and approved* by the following:

Aleksandra B. Slavkovic
Professor of Statistics
Chair of Graduate Studies Program
Thesis Advisor

Naomi Altman
Professor of Statistics

David Hunter
Professor of Statistics
Head of Department of Statistics

*Signatures are on file in the Graduate School.

Abstract

Integrating multiple databases that are distributed among different data owners can be beneficial in numerous contexts of statistical analysis. Unfortunately, the actual sharing of data is often impeded by concerns about data confidentiality. A situation like this requires tools that can produce correct results while minimizing risk of disclosure. Over the past ten years a number of "secure" protocols have been proposed to solve specific statistical problems such as linear regression and classification in a distributed setting.

In this thesis, we first explore the disclosure risks associated with several existing protocols designed for the vertically partitioned database setting. We focus on the specific case where two parties are trying to perform logistic regression without actually combining their data. Although the protocols can be considered secure in the sense that there is no danger for either party's data to be fully exposed, there is information leakage resulting from the intermediate computations and also from the estimated coefficients. We provide detailed analysis of such cases.

Secondly we show how these previously proposed secure computation protocols can be applied to penalize regression methods, with a focus on the LARS algorithm used to do Lasso regression. A protocol for the vertically partitioned database setting is described, along with a thorough discussion on possible disclosure risks and computation. We also provide a detailed description on how to perform model selection and possible ways to expand our protocol to LARS-type algorithms for generalized linear models, such as logistic regression.

Table of Contents

List of Figures vii

List of Tables viii

Acknowledgments ix

Chapter 1
Introduction 1
1.1 Background 1
1.1.1 Vertically Partitioned Databases 2
1.2 Our Contribution and Structure of this Thesis 4

Chapter 2
Possible Disclosure Risks in Secure Logistic Regression Protocols 5
2.0.1 Outline of this Chapter 7
2.1 Forms of Security Risks 7
2.1.1 Loss of Protection and Constraints 7
2.1.2 Probabilistic (1 out of $\mathbf{m}$) Risks 9
2.1.3 Decryption Risks 9
2.2 Secure Protocols 10
2.2.1 Secure Summation 10
2.2.2 Secure Matrix Multiplication 10
2.2.3 Oblivious transfer 11
2.2.4 Homomorphic encryption 12
2.2.5 Random shares 13
2.3 Fienberg et al. (2009) 14
2.3.1 Case (a) for Fienberg et al. (2009) 14
2.3.2 Case (b) for Fienberg et al. (2009) 18
2.4 Lin and Karr (2010) 19
2.4.1 The Protocol (general) 20
2.4.2 Specific Case: Logistic Regression 20
2.4.3 Security breaches in case (a) for Lin and Karr (2010) 23
2.4.4 Security breaches in case (b) for Lin and Karr (2010) 25
2.5 Nardi et al. (2012) 26
2.5.1 Form of NR algorithm and assumptions 26
2.5.2 Secure NR algorithm for Cases (a) and (b) in Nardi et al. (2012) 27
2.6 Discussion 28

Chapter 3
Secure Penalized Regression Methods **30**
3.1 Introduction . . . 30
3.1.1 Outline of this Chapter . . . 32
3.2 Background . . . 33
3.2.1 Ridge Regression . . . 33
3.2.2 Lasso . . . 33
3.2.3 Elastic Net . . . 34
3.2.4 When to use Ridge Regression and Lasso . . . 34
3.3 The LARS Algorithm and Lasso . . . 35
3.3.1 The LARS Algorithm . . . 36
3.3.2 The LARS Algorithm in Detail . . . 37
3.4 The Secure Protocol . . . 42
3.4.1 Settings and Assumptions . . . 42
3.4.2 What is Learned . . . 42
3.4.3 Protocol for Lasso over Vertically partitioned data . . . 43
3.4.4 Computation Strategies . . . 48
3.4.5 Special case: $p > n$ . . . 49
3.5 Disclosure Risks . . . 50
3.5.1 The Two Party Case . . . 51
3.5.2 Two Party Case when $p > n$ . . . 53
3.5.3 Disclosure Risk from the Output . . . 54
3.6 Diagnostics and Model selection . . . 54
3.6.1 Diagnostics . . . 54
3.6.2 Cross-Validation . . . 55
3.6.3 Example: Diabetes Data (Tibshirani (1996)) . . . 56
3.7 Running Time and Error . . . 59
3.7.1 Dissecting the Secure Matrix Multiplication Algorithm . . . 59
3.7.2 Algorithms for QR Decomposition . . . 61
3.7.3 Computational Complexity and Memory Usage . . . 61
3.7.4 Simulation . . . 63
3.7.5 Accuracy of the Secure Protocol . . . 64
3.8 Extensions to Generalized Linear Models . . . 65
3.9 Discussion and Conclusion . . . 68
3.9.1 Ridge Regression . . . 68
3.9.2 Complex Partitions . . . 69
3.9.3 Alternative Methods for Matrix Multiplication . . . 69

Chapter 4
Concluding Remarks **70**
4.1 Summary and Conclusions . . . 70
4.2 Future Work . . . 71

Appendix
72
1 Code for Secure Matrix Multiplication . . . 72
2 Code for Example in Section 3.6.3 . . . 75
3 Code for Experiment in Section 3.7.4 (Results summarized in Tables 3.4 and 3.5) 87

4 Code for Experiment in Section 3.7.5 (Results summarized in Table 3.5) 89

Bibliography **93**

List of Figures

1.1 Horizontally Partitioned Databases, where X is a Matrix of Covariates and y is a Response Vector . . . 3
1.2 Vertically Partitioned Databases, where X is a Matrix of Covariates and y is a Response Vector . . . 3

3.1 LARS for Two Predictors . . . 36
3.2 Example of a Lasso path for 10 predictors (Diabetes Data, see Section 3.6.3 for details). . . . 43
3.3 Example of a Lasso path for 10 predictors, 7 Observations . . . 50
3.4 Diabetes Example: Lasso paths for each Training set of size 48 . . . 58
3.5 Diabetes Example: Mean prediction errors for each value of $\sum_{i=1}^{10} \hat{\beta}_i$ . . . 59
3.6 Diabetes Example: Lasso path for entire data set (Vertical line at $\sum_{i=1}^{10} \hat{\beta}_i = 1300$) 59

List of Tables

3.1 Lasso Estimates for Diabetes Data Based on Five-Fold Cross Validation, using our secure protocol 57
3.2 Breakdown of the Number of Arithmetic Operations in Secure Matrix Multiplication 62
3.3 Computing Times of $X_A^T X_B$ in Seconds, for Different Values of p_A (rows) and n (columns) 63
3.4 Computing Time in Seconds, Under Different Partitions when $n = 10000$ and $p = 8$ 65
3.5 Maximum Absolute Error of $X_A^T X_B$, for Different Values of p_A (rows) and n (columns) 66

Acknowledgments

I would like to thank my advisor, Dr. Aleksandra Slavkovic, for generously giving me her time, knowledge and advice to help me complete this thesis. I am also very grateful for her patience during times when my work progress was slow in the past year. Thanks also to the faculty, staff, and my colleagues at the Department of Statistics at Penn State University.

Chapter 1 Introduction

1.1 Background

Statistical analyses that combine data stored in multiple, distributed databases are of great interest to researchers. However, the actual integration of data is often impeded by concerns about data confidentiality, including legal or regulatory barriers. These concerns can still remain even if the data owners agree to perform the integrated analysis without breaking the confidentiality of each others' data, or trust their data to a third party to do the analysis. In many cases, however, it is possible to perform the analyses without actually integrating the data. For example, the only information needed to compute a given parameter of a statistical model is its corresponding sufficient statistic which can be computed in a secure way using techniques from the class of methods known as secure multi-party computation (SMPC), e.g., see Lindell and Pinkas (2009). In general, SMPC aims to compute the value of a function with multiple inputs where each participant holds only a subset of the inputs, while sharing as little information about those as possible using cryptographic tools.

Consider the situation where parts of an idealized database D are distributed among K *parties* (data holders) $A_1, A_2, \ldots, A_K$. The parties can be, say, government agencies, private companies, or research institutes who each own distinct parts of D. The parties want to build a linear regression model, but are reluctant to actually share and combine their data, due to concerns about confidentiality. Hence they need a way to obtain the same analysis results as if D existed, but without sharing their data with other parties.

The SMPC techniques that are needed to solve distributed database problems depend not only on the statistical model of interest, but also on the way the global database is partitioned. When all parties own the same set of variables but for different data subjects, the situation is called *horizontally partitioned databases* (Figure 1.1). On the other hand, when the parties hold disjoint sets of variables but for the same data subjects, we have the case of *vertically partitioned databases* (Figure 1.2). Beyond the two "pure" cases the partitions can be more complex, e.g., the variables may be held by different parties while not all data subjects are common to all parties (vertically partitioned partially overlapping databases).

A number of methods for performing statistical models have been developed with respect to different kinds of partitions. Karr et al. (2005) provide a comprehensive approach on horizontally partitioned data. A secure protocol to perform matrix multiplication is described, for example, in Karr et al. (2009) to compute the parts needed in addressing regression in the vertically partitioned case. These methods have been also applied in Fienberg et al. (2007) and Slavkovic et al. (2007), for example, to perform logistic regression in both of the "pure" partition cases. Combining SMPC with missing data techniques, Reiter et al. (2004) and Fienberg et al. (2006) reduce vertically partitioned, partially overlapping data to the pure vertical case. Hall et al. (2011) utilize stronger cryptography methods and describe an approach that not only provides a higher degree of security, but can also be applied to any partition scheme. The downside of the method by Hall et al. (2011), however is that it requires significantly more computation than the other approaches. Nardi et al. (2012) apply this approach to logistic regression, carrying over the same strengths and weaknesses. Protocols that do not use SMPC techniques have also been proposed; for example, in the DataSHIELD system by Gaye et al. (2014) individual data holders perform joint analyses by linking their computers to a central analysis computer.

1.1.1 Vertically Partitioned Databases

Our main focus in this thesis is on vertically partitioned databases, although we do discuss possible applications to both horizontal and complex partitions. We assume that $K \geq 2$ parties, denoted $A_1, \ldots, A_K$, are involved but note here that the case with $K = 2$ may have additional security implications. Assuming without loss of generality that party A_1 holds the response variable y, the data would look

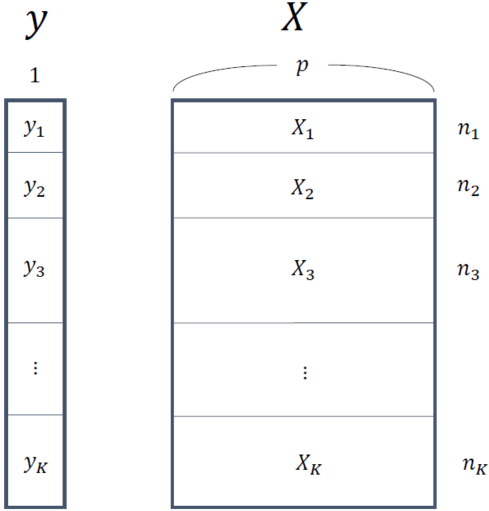

Figure 1.1: Horizontally Partitioned Databases, where X is a Matrix of Covariates and y is a Response Vector

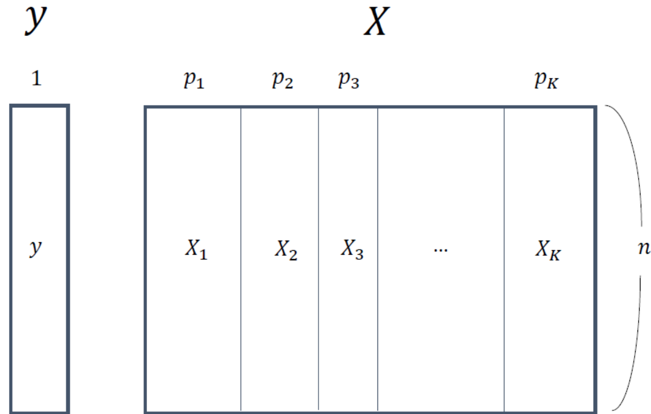

Figure 1.2: Vertically Partitioned Databases, where X is a Matrix of Covariates and y is a Response Vector

like:

$$\begin{array}{rccccc} \text{Data:} & [y,\, X_1\,] & X_2 & \cdots & X_{K-1} & X_K \\ \text{Held by party:} & A_1 & A_2 & \cdots & A_{K-1} & A_K \end{array}$$

where p_k is the number of variables that party A_k holds, hence each X_k is an $n \times p_k$ matrix. We denote $X = [X_1, \cdots, X_K]$ as the $n \times p$ matrix of all predictors where $p = p_1 + \cdots + p_K$. For vertically partitioned databases it is assumed that all parties have the same observations, and have matched up their data using a global

identifier such as social security numbers.

1.2 Our Contribution and Structure of this Thesis

This thesis serves two main purposes: First, we provide an in-depth analysis of secure protocols based on existing SMPC techniques, and second, we propose an application of these techniques to penalized regression methods, which has not yet been fully explored in previous literature. As mentioned before, the focus is on the vertically partitioned database setting. The structure of the thesis is as follows:

Chapter 2: Possible privacy breaches in the vertically partitioned data setting for the secure Logistic regression methods in Fienberg et al. (2009), Lin and Karr (2010), and Nardi et al. (2012) are discussed and compared. We focus on the two party case based on the logic that, if a protocol is secure under two parties it should also be secure when three or more parties are involved. For each of the three methods we explore the information leakage resulting from the intermediate computations and also from the estimated coefficients. The three methods each have its advantages and disadvantages in terms of security and efficiency; one method may be preferable to another depending on the situation. We look at different situations and analyze which of the three methods would be appropriate.

Chapter 3: Penalized regression methods such as ridge regression and Lasso in the vertically partitioned database setting are discussed. Our main contribution is in proposing a secure protocol for Lasso, based on the LARS algorithm by Efron et al. (2004). In addition to describing the secure protocol, computation strategies to make the procedure as efficient as possible are explained, along with security concerns and special cases such as $p > n$. For model selection we show how it is not unreasonable (in terms of computation) to use cross-validation. In addition we address concerns users may have other than security such as computation time and accuracy of the estimates, through simulations under various scenarios. Ways to expand our secure protocol to LARS-type algorithms for generalized linear models are also explored. One possibility is to use linear approximations of non-linear functions, and we show how to do this in the case of logistic regression.

Chapter 4: We summarize our findings and discuss future work.

Chapter 2
Possible Disclosure Risks in Secure Logistic Regression Protocols

This chapter aims to identify the disclosure risks when a secure logistic regression protocol is applied for the vertically partitioned data setting. We compare three different approaches from three different papers: Fienberg et al. (2009), Lin and Karr (2010), and Nardi et al. (2012). Our motivation for doing so is that, while these three articles do mention that there can be risks associated with their respective protocols, the magnitude or the specific cases in which there are risks are not necessarily explained in full detail; yet many practitioners aim at using those or similar methods with two parties, e.g. Gaye et al. (2014). In this chapter we examine the flow of the information from one party to another, and see the risks are associated with that information.

Fienberg et al. (2009) describe an approach that use the *secure summation protocol*, and *secure matrix multiplication* (Karr et al. (2009)). They show how secure matrix multiplication and summation can be used to construct the parts needed for the Newton-Ralphson algorithm. Subsequently, Slavkovic et al. (2007) use a similar algorithm for logistic regression in the "pure" horizontal and vertical settings. They show how it is possible to apply their method to perform some diagnostics such as Pearson's chi-squared or likelihood ratio deviance statistics. Some downsides to this approach are that there is some leakage of information resulting from the secure multiplication protocol, and that in extreme cases the Newton-Ralphson algorithm might iterate for too long while in each iteration leaking additional information to the other parties. Finally, it may be possible to obtain good estimates of another party's data since the estimated logistic regression coefficients $\hat{\beta}$ will be shared among all parties.

Lin and Karr (2010) outline a general approach in performing maximum likelihood estimation for exponential family models in the vertical setting. Their method utilizes the 1 out s oblivious transfer protocol by Yao (1986), or a more complex version of it by Atallah and Du (2001). The specific case on how to perform logistic regression in the vertical setting is not discussed in the article, but it is covered in this chapter (Section 2.4). We will see that the protocol is very similar to Fienberg et al. (2007) except that a matrix multiplication method based on a 1 out of s obvlivious transfer will be used instead of the secure multiplication protocol. Despite the similarities, the disclosure risks for this method are not directly comparable to Fienberg et al. (2007); we will see why this is the case in Section 2.4.

Nardi et al. (2012) utilize stronger cryptography methods and describe an approach that provides a higher degree of security. More specifically, they use *homomorphic encryption*, which allows users to obtain encrypted sums and products of encrypted values without decrypting the original values. Other techniques that are used include a special approximation technique by Guo and Higham (2006) for obtaining inverses of matrices from encrypted values, and the oblivious transfer protocol which is also used in the approach by Lin and Karr (2010). One major advantage of this approach is that, it does not require sharing any intermediate values in the process of calculating the final estimates of the logistic regression coefficients, so it guarantees a higher degree of security than the approaches mentioned above. Secondly, it can be applied to any kind of partitioning scheme whereas the two other approaches are only designed for the purely horizontal and vertical cases. The biggest drawback is that it is more computationally demanding than other methods. The authors indicate that it might not be practical for moderate to high dimensional problems. We should also note that the estimated regression coefficients $\hat{\beta}$ will still be shared among all parties so the method does not address any leakage which results from the output. The method also does not allow detailed statistical analysis since by default it is designed so that nothing but the coefficients will be estimated. The algorithm can be modified so that the parties can learn some intermediate statistics, but that will make it lose one of its major advantages, especially in the pure vertical setting.

In this chapter we look at the disclosure risks that can occur when using the

methods introduced above. These include disclosure that can happen from the computation process and also whatever information that can be learned from the output (the estimated coefficients) themselves. It is assumed that there is a global identifier that is common to all the databases, and that the parties do not use fake data in order to gain more information on the other party's data. We focus on the case where only two parties are involved in the process, since if a protocol is secure for two parties, we can assume it will also be safe when three or more parties are involved, given that the parties are semi-honest, i.e., there is no collusion.

2.0.1 Outline of this Chapter

In Sections 2.1 and 2.2 we introduce the SMPC techniques that are used in the three protocols in this chapter; these include secure summation and multiplication, oblivious transfer, homomorphic encryption, and random shares. Sections 2.3 to 2.5 provide descriptions of the protocols in each of the aforementioned articles while following the information that is exchanged in each step, examining the disclosure risks. In Section 2.6 we summarize our findings, and discuss which of the three approaches would be appropriate in various situations.

2.1 Forms of Security Risks

In this section we lay out different forms of security risks that are present in the secure multiparty protocols introduced in later sections. Our aim here is to try to give the reader a practical sense of the information that can be leaked from one party to another, and what kind of risks are associated with that information.

2.1.1 Loss of Protection and Constraints

Karr et al. (2009) introduce the concept of *loss of protection* to argue that their secure matrix multiplication protocol is safe. The loss of protection to one party (say, party A_1) is defined as the number of linearly independent *constraints* the other party has on A_1's data. Let us consider an example:

Suppose party A_1's data comprises p =3 variables with n_1 =4 observations each. The data can be represented in matrix form

$$X = \begin{pmatrix} x_{11} & x_{12} & x_{13} \\ x_{21} & x_{22} & x_{23} \\ x_{31} & x_{32} & x_{33} \\ x_{41} & x_{42} & x_{43} \end{pmatrix}$$

Let P be a 4×4 non-singular (invertible) matrix. If party A_2 does not know X but knows P and the product $B = P\,X$, it can invert P and obtain $X = P^{-1}\,B$. Knowing P and B is equivalent to knowing the 12 linear equations of the following format:

$$p_{i1}\,x_{1j} + p_{i2}\,x_{2j} + p_{i3}\,x_{3j} + p_{i4}\,x_{4j} = B_{ij} \qquad \text{for i=1,2,3,4 and j=1,2,3.}$$

We refer to each of such equations as *constraints*.

In general, if X is a $n \times p$ matrix, A_2 will need np unique constraints to know all elements of X. However, A_2 having fewer than np constraints does not mean that X is safe. For instance if A_2 knows the row vector $P = (\,0,\,0,\,c,\,0\,)$, then A_2 will learn the entire third row of X from P and $B = PX$. Note however, that relatively extreme cases like this are usually detectable in practice.

In any case, even if A_2 has fewer that np constraints on X, combining them with external information can help A_2 getting closer to learning X completely. Ultimately A_1 would want A_2 to know as few constraints on X as possible.

The secure logistic regression protocol by Fienberg et al. (2009) repeatedly uses two secure multi-party protocols, known as *secure summation* and *secure matrix multiplication*. Further details are given in Sections 2.3.1 and 2.3.2, where we also discuss the possible privacy breaches that may occur (i.e. how many constraints a party may let out) when these protocols are employed between only two parties.

2.1.2 Probabilistic (1 out of m) Risks

In the protocol by Lin and Karr (2010), one party A_i will "blind" a secret value z by sending m different values $z_1, \ldots, z_m$, one of which is z and the others random, to party B. Party B will be able to see the m potential values, but not know which z_i is the secret value z. The probability of B correctly guessing z is $1/m$; but this might be increased if B has any other information on z, allowing it to narrow down the potential values.

2.1.3 Decryption Risks

Nardi et al. (2012) use protocols based on public key cryptography, where a secret message (an integer value) is encrypted using a *public key*, which can only be decrypted using a *private key*. In the two party case, the public key is shared among both parties A and B but only one party, say party A, will know the private key. The private key consists of the pair of numbers (N, d) where N is the product of two randomly selected integers p and q, and d is a large integer relatively prime to $(p-1)(q-1)$. Only A can know the values of p,q, and d. The private key is the pair (N, e) where e is the modular multiplicative inverse of d, i.e.:

$$e \cdot d \equiv 1 \bmod (p-1)(q-1)$$

The possible security risk here is in B finding out the value of p, q, or d, which will allow them to decrypt the encrypted form of the secret message. However, this would be very difficult given that the only method to find the two prime factors of a large number N is to check the possibilities one by one. For example, if a 128 bit public key were used, the number of possible prime numbers is said to be around:

$$\frac{2^{128}}{\ln(2^{128})} > 2^{121} > 3 \cdot 10^{120}$$

hence more than $3 \cdot 10^{120}$ possibilities. Given such difficulty, in later sections we will assume that there are no disclosure risks when public key encryption is applied.

2.2 Secure Protocols

In this section we summarize the secure protocols that are used in later sections and chapters. We will assume that there are only two parties involved, and discuss the potential privacy risks.

2.2.1 Secure Summation

The secure summation protocol is used in Fienberg et al. (2009) and Lin and Karr (2010). The protocol is used when K parties $A_1, \ldots, A_K$ hold data matrices $X_1, \ldots X_K$ of same dimensions and want to calculate the sum $\sum_{i=1}^{K} X_i$ without sharing the data. In the two party case, the protocol proceeds as follows:

1. Party A_1 adds a random matrix R to its own matrix X_1, and then passes $(R + X_1)$ on to Party A_2.
2. Party A_2 adds X_2 and passes $(R + X_1 + X_2)$ back to A_1
3. Party A_1 subtracts R and obtains the sum $X_1 + X_2$, and shares it with A_2.

   Possible Security Risk:
   Party A_1 can learn the entire matrix X_2 by subtracting its own matrix X_1 from the sum. The second party will practically learn nothing about X_1, as long as R is chosen carefully.

2.2.2 Secure Matrix Multiplication

The secure matrix multiplication protocol is used in Fienberg et al. (2009). The goal is to calculate $X_1^T X_2$. We will assume that the amount of information exchanged is symmetric, i.e. $g = \lfloor (n - p_1)/2 \rfloor$. :

1. Party A_1 generates g orthonormal vectors $Z_1, Z_2, \ldots, Z_g$ such that $Z_i^T X_j^1 = 0$ for all i and j, and sends the matrix $Z = [Z_1, Z_2, \ldots, Z_g]$ to Party 2.
2. Party A_2 computes $(I - Z Z^T) X_2$ and sends it back to Party 1.
3. Party A_1 computes $X_1^T (I - Z Z^T) X_2 = X_1^T X_2$ and can either share or not share it with A_2.

Possible Security Risk:
Note that $(I - Z Z^T)$ has rank $(n-g) \approx n/2$, assuming n is much larger than p_1. So at step 2, Party A_1 cannot invert $(I - Z Z^T)$ to obtain X_2 but will get around $n/2$ constraints on Party 2's data. Also from the final product $X_1^T X_2$, party A_1 can obtain $p_1 p_2$ constraints on A_2's data (one from each element of $X_1^T X_2$). Hence A_1 obtains at most around $(n/2 + p_1 p_2)$ constraints (considering linear dependence) on X_2 in the end.

Similarly, Party A_2 knows that X_1 lies in the g-dimensional subspace orthogonal to Z, so it has $g \approx n/2$ constraints on X_1. Additionally, *if* the final product is shared in step 3, A_2 obtains (at most) $(g + p_1 p_2) \approx (n/2 + p_1 p_2)$ constraints on X_1 in the end.

2.2.3 Oblivious transfer

Lin and Karr (2010) introduce a protocol that utilize the method known as "oblivious transfer" to compute maximum likelihood estimates of generalized linear models in the vertically partitioned data setting. Oblivious transfer has many variations and uses; but in this chapter we will refer to it as any version of it that allows two parties to complete the following process:

(1) Party A_1 sends m messages (numerical values in our context) to party A_2

(2) Party A_2 receives exactly one out of the m messages, without seeing the other $m-1$ messages. Party A_1 does not know which message A_2 received.

We will see in Section 2.5 why this can be useful. First we provide an example on how to do this for $m = 2$. Suppose that party A_2 wants to receive one out of 2 numerical values r_0, r_1 from party A_1. We will refer to these values as *messages*. Party A_2 does not want A_1 to know which message they want to receive, but A_1 only wants A_2 to receive the relevant message. The protocol would be as follows:

1. If r_0, r_1 are not integers, A_1 applies some 1 to 1 transformation scheme to turn them into integers n_0, n_1. The 2s complement approach introduced later in Section 2.2.4 is one way to do this. We assume A_2 knows how to turn n_0, n_1 back to the original values.

2. A_1 generates the RSA public key (N, e) which is shared with B, and the private key (N, d) which is not shared with B. We refer the reader to Rivest et al. (1978) for details on RSA cryptography. For now we will just say that (N, e) is the information needed to encrypt any given message, and (N, d) is the information needed to decrypt an encrypted message.
3. A_1 also generates two random values x_0, x_1 and sends them to B.
4. A_2 picks b where n_b is the message that A_2 wants to receive ($b = 0\,\text{or}\,1$). They then generate a random value k, computes $v = (x_b + k)^e \; mod \; N$, and sends v to A_1. Note that A_1 will not be able to tell whether b is 0 or 1.
5. A_1 computes $k_0 = (v - x_0)^d \; mod \; N$ and $k_1 = (v - x_1)^d \; mod \; N$, one of which will equal k. Note that only B can tell which k_i equals to k.
6. A_1 then computes $n'_0 = n_0 + k_0$ and $n'_1 = n_1 + k_1$ and sends it back to A_2.
7. A_2 knows which k_i is k, so they can compute $n_b = n'_b - k_b$
8. If r_b was not an integer, B converts n_b back to r_b.

An example of a protocol for when m is greater than 2 (1 out of m oblivious transfer) is given in Rivest et al. (1978). Possible security risks when the method is applied to logistic regression are discussed in section 2.5.3.

2.2.4 Homomorphic encryption

Let us denote $Enc_N(a)$ as the encrypted form of a secret quantity a. The encryption scheme has the *homomorphic properties*, if the following holds:

$$Enc_N(a) \cdot Enc_N(b) \, mod \, N = Enc_N(a + b) \tag{2.1}$$

$$(Enc_N(a))^c \, mod \, N = Enc_N(a \cdot c) \tag{2.2}$$

These properties turn out useful in obtaining random shares and adding them back to compute a secret quantity, as will be discussed in the next section. Nardi et al. (2012) use a public-key cryptographic system by Paillier (1999). Paillier's scheme only works on non-negative integers $\mathbb{Z}_N = \{0, \ldots, N-1\}$ so if the secret quantities are real numbers, the need to be converted to integers through some 1-1

transformation scheme. Nardi et al. (2012) propose “2s complement” approach, where the mapping from $\mathbb{Z}_n$ to fixed precision real numbers is:

$$f : \mathbb{Z}_N \to \mathbb{R}, \quad f(a) = M^{-1} \begin{cases} a & \text{for } a \le \frac{N}{2} \\ N - a & \text{for } a \ge \frac{N}{2} \end{cases} \tag{2.3}$$

2.2.5 Random shares

The idea of *random shares* is to divide a quantity of interest a into K random numbers, where K is the number of parties involved, so that $\sum_{j=1}^{K} a_j = a$. Nardi et al. (2012) repeatedly use this construction to compute intermediate quantities secretly during the evaluation of Newton's method, and put them together at the last step to obtain the maximum likelihood estimates of the logistic regression coefficients.

Given below is a protocol for computing $x_{A_1} + x_{A_2} = x$. We will denote the encrypted form of a given quantity a as $Enc_N(a)$, and assume that it has the homomorphic properties discussed in 2.2.4. We assume that party A holds both the public and private key, but A_2 only holds the public key. In addition we can assume without loss of generality that x_{A_1} and x_{A_2} are both integer values, possibly through a fixed point approximation of real values .

1. Party A_1 encrypts x_{A_1} using the public key and sends $Enc_N(x_{A_1})$ to party A_2. Note that A_2 cannot decrypt this, since only A_1 has the private key.
2. Party A_2 encrypts x_{A_2} and calculates $Enc_N(x_{A_1} + x_{A_2})$ using homomorphic property (1). Specifically, A_2 computes:

$$Enc_N(x_{A_1} + x_{A_2}) = Enc_N(x_{A_1}) \cdot Enc_N(x_{A_2}) \; mod \, N \tag{2.4}$$

3. Party A_2 generates a random quantity r and then computes $Enc_N(x_{A_1} + x_{A_2} - r)$ by means of homomorphic property(1). This encrypted value is sent to party A.
4. Party A_1 decrypts $Enc_N(x_{A_1} + x_{A_2} - r)$ using the private key and obtains $x_{A_1} + x_{A_2} - r$. Party A_2 knows r. Note that neither party knows the value of

$x_{A_1} + x_{A_2}$ itself , and hence have obtained their random shares secretly.

Random shares of the product of two secret quantities can be obtained in a similar way using homomorphic property (2). The protocol for computing products will require some modification when x_{A_1} and x_{A_2} are fixed point approximations of real numbers; the modification needed depends on the approximation scheme. The details on when the 2s compliment approach is used for fixed point approximation are provided in Hall et al. (2011).

2.3 Fienberg et al. (2009)

2.3.1 Case (a) for Fienberg et al. (2009)

(a) Two parties A_1 and A_2.
A_1 holds response y and $n \times p_1$ matrix X_1, A_2 holds $n \times p_2$ matrix X_2.

Form of NR algorithm and assumptions

The form of the Newton-Raphson algorithm in Fienberg et al. (2009) is given as:

$$\hat{\beta}^{(s+1)} = \hat{\beta}^{(s)} + (X^T W^{(s)} X)^{-1} X^T y \tag{2.5}$$

where $W^{(s)} = diag(\pi_i^{(s)}(1 - \pi_i^{(s)}))$, $\pi_i^{(s)} = (1 + exp\{-x_i^T \beta^{(s)})\})^{-1}$, and $y^{(s)} = y - \pi^{(s)}$.

Assume we are in the s th iteration of the algorithm, and want to compute the components on the left of $\hat{\beta}^{(s)}$ in order to get $\hat{\beta}^{(s+1)}$. The authors define $I = X^T y^{(s)}$ and $II = (X^T W^{(s)} X)$.

Some assumptions are that:

(a) Party A_1 holds the response variable y and is not willing to share it

(b) A_1 and A_2 are both not willing to share intermediate values of their components of $\hat{\beta}$.

(c) The two parties *are* willing to share some summary statistics.

Secure NR algorithm

Recall that A_1 holds response y and $n \times p_1$ matrix X_1, and A_2 holds $n \times p_2$ matrix X_2. Below we outline the steps of the algorithm as given in Fienberg et al. (2009), and explain possible information leakages:

1. The two parties choose initial values $\beta_1^{(0)}$ and $\beta_2^{(0)}$, respectively. There is no exchange of information here.

2. The two parties obtain $\pi^{(s)}$ by applying secure summation to $X^T\beta^{(s)} = X_1^T\beta_1^{(s)} + X_2^T\beta_2^{(s)}$.

   Information flow (if A_1 initiates secure summation):
   (1) A_1 sends $(R + X_1^T\beta_1^{(s)})$ to A_2, (2) A_2 sends $(R + X_1^T\beta_1^{(s)} + X_2^T\beta_2^{(s)})$ back to A_1, (3) A_1 shares $X^T\beta^{(s)}$ with A_2

   In Section 2.2.1 we saw that the party who initiates the secure summation protocol, say A_1, will know the value of $X_2^T\beta_2^{(s)}$. Under assumption (c) this value is practically useless in terms of finding out X_2, since $\beta_2^{(s)}$ will always be unknown to A_1. However a problem may arise after the NR algorithm converges. Typically the parties would decide that the algorithm has converged when:

   (a) $\hat{\beta}_i^{(s+1)} \approx \hat{\beta}_i^{(s)}$ holds for all $i = 1, \ldots, p$, or:

   (b) $|\hat{\beta}^{(s+1)} - \hat{\beta}^{(s)}| < \epsilon$ for some small $\epsilon > 0$, where $|\hat{\beta}^{(s+1)} - \hat{\beta}^{(s)}|$ is the L^1 or L^2 norm of $\hat{\beta}^{(s+1)} - \hat{\beta}^{(s)}$.

   Note that even when (b) is the convergence criteria, the value of ϵ would usually be chosen to be small enough so that (b) implies (a). Now suppose that the algorithm converges at the $(s+1)$th step and the value of $\hat{\beta} = \hat{\beta}^{(s+1)}$ is shared among all parties. Then A_1, who knows the value of $\hat{\beta}_2$ and $X_2^T\beta_2^{(s)}$, will obtain a close estimate:

   $$X_2^T\hat{\beta}_2 \approx X_2^T\beta_2^{(s)}$$

   This is equivalent to obtaining n constraints on party A_2's data. In the case

that A_2 only holds one variable, A_1 learns A_2's data entirely.

3. Let $I = (I_1 \; I_2)^T$. A_1 computes $I_1 = X_1^T y$ locally and does not share it with A_2. A_2 obtains $I_2 = (X_2)^T y$ via secure matrix multiplication with A_1.

   Information flow:
   (1) A_2 sends the set of vectors $Z_1, Z_2, \ldots, Z_g$ to A_1 (see section 1.2), (2) A_1 computes and sends $(I - Z Z^T) y$ back to A_2, (3) A_2 obtains $I_2 = (X_2)^T y$ and does not share it with A_1.

   From Section 2.2 we know that A_2 can obtain at most $(n/2 + p_2)$ constraints on y. Note that the secure multiplication here only needs to be done in the first iteration of the algorithm. Hence for a sufficiently large n, the response vector y can be considered safe.

4. Both parties obtain their own diagonal sub-matrix of II by computing $X_1^T W^{(s)} X_1$ and $X_2^T W^{(s)} X_2$ locally. For the off-diagonal $X_1^T W^{(s)} X_2$, secure matrix multiplication is applied. (Note that $W^{(s)}$ is known to both parties.)

   Information flow:
   A_2 has already sent $Z_1, Z_2, \ldots, Z_g$ to A_1 in step 3, so the only steps are: (1) A_1 computes and sends $(I - Z Z^T) W^{(s)} X_1$ back to A_2, and (2) A_2 computes $X_2^T W^{(s)} X_1$ and shares it with A_1.

   A_1 gets to know $X_2^T W^{(s)} X_1$, so it obtains at most $(n/2 + p_1 p_2)$ constraints on X_2. A_2 learns approximately the same amount of information about X_1. Note that for $s \neq t$, the corresponding rows of $X_1^T W^{(s)}$ and $X_2^T W^{(t)}$ are linearly dependent so multiple iterations of the algorithm will not give away additional constraints on X_1 or X_2.

5. Each party inverts II. Suppose:

$$II^{-1} = \begin{pmatrix} A_{11}^{(s)} & A_{12}^{(s)} \\ A_{21}^{(s)} & A_{22}^{(s)} \end{pmatrix}$$

   A_1 obtains $A_{11}^{(s)} X_1^T y + A_{12}^{(s)} X_2^T y$ using secure summation by initiating the protocol, and not sharing the result with A_2. From Section 2.1 we know that

A_1 will learn the value of $A_{12}^{(s)}\, X_2^T\, y$. Similarly, party A_2 will learn the sum $A_{21}^{(s)}\, X_1^T\, y\, +\, A_{22}^{(s)}\, X_2^T\, y$ and the value of $A_{21}^{(s)}\, X_1^T\, y$.

Information flow: Secure summation for both A_1 and A_2, similar to step 2.

As for loss of protection, A_1 obtains p_1 constraints on $X_2^T y$ from learning $A_{12}^{(s)}\, X_2^T\, y$, and A_2 obtains p_2 constraints on $X_1^T y$ from learning $A_{21}^{(s)}\, X_1^T\, y$.

6. Both parties update their share of $\hat{\beta}$. No information is exchanged.

Summary of Information Leakages

If the NR algorithm took s_1 steps to converge,

(a) A_1 obtains at most around $(n/2\, +\, p_1 p_2)$ constraints on X_2. (from step 4)

(b) A_2 obtains at most around $(n/2\, +\, p_1 p_2)$ constraints on X_1. (from step 4)

(c) A_2 obtains at most around $(n/2\, +\, p_2)$ constraints on y. (from step 3)

(d) A_1 obtains at most around $s_1 p_1$ constraints on $X_2^T y$. (from step 5)

(e) A_2 obtains at most around $s_1 p_2$ constraints on $X_1^T y$. (from step 5)

Note that (d) and (e) are unique to the two-party case, and (a), (b), and (c) will happen even if three or more parties are involved. As long as n is sufficiently large compared to p_1 or p_2, and if X_1 or X_2 both contain at least two columns, we can assume that y or X_1 or X_2 are unlikely to be revealed. However, notice that the constraints given away in (d) and (e) are a multiple of s_1. Even with linear dependence taken into account, it is possible in the two-party case that enough iterations of the algorithm would reveal $X_1^T y$ or $X_2^T y$ to the other party. This still does not reveal y or X_1 or X_2, but nevertheless a party might consider it a security breach.

Additionally we saw that the party who initiates the secure summation protocol in step 2, say party A_1, will obtain an additional n constraints on X_2. In the

case that X_2 only contains one column, A_1 will learn A_2's data entirely. This can be avoided by letting the party who has fewer predictors (assuming at least one of the parties has two or more predictors) initiate the protocol; but it will cause some unfairness in terms of loss of protection. Note, if both parties only hold one predictor the entire protocol is not secure.

2.3.2 Case (b) for Fienberg et al. (2009)

(b) Two parties A_1 and A_2.
A_1 only holds response y. A_2 holds $n \times p_2$ matrix X.

Form of NR algorithm and assumptions
Same as in case (a).

Secure NR algorithm
Recall that A_1 only holds response y, while A_2 holds $n \times p_2$ matrix X. The algorithm, as given in the article, is as follows:

1. A_2 chooses initial value $\beta^{(0)}$, and doesn't tell A_1. No information exchanged.
2. A_2 obtains $\pi^{(s)}$ by computing $X^T \beta^{(s)}$ locally. No information exchanged.
3. A_2 obtains $I = (X)^T y$ via secure matrix multiplication with A_1.

   Information flow:
   (1) A_2 sends the set of vectors $Z_1, Z_2, \ldots, Z_g$ to A_1 (see section 1.2), (2) A_1 computes and sends $(I - Z Z^T) y$ back to A_2, (3) A_2 obtains $I_2 = X^T y$ and does not share it with A_1.

   From Section 2.2.2 we know that A_2 can obtain at most $(n/2 + p_2)$ constraints on y. Note that the secure multiplication here only needs to be done in the first iteration of the algorithm.
4. A_2 obtains II by computing $X^T W^{(s)} X$ locally. No information exchanged.
5. A_2 inverts II. No information exchanged.

6. A_2 updates $\hat{\beta}$.

Summary of Information Leakages

- A_2 obtains at most around $(n/2 + p_2)$ constraints on y. (from step 3)

If A_1 only holds the response vector, no information about A_2's data is leaked to A_1. Furthermore we can make a more general statement that if there are k parties ($k \geq 2$) involved where A_1 only holds the response, A_1 would learn nothing about the other parties' data, while the information leaked between A_s and A_t ($s, t \geq 2$) is the same as in the normal (A_1 holds some explanatory variables) case. Hence again, the protocol is safe as long as n is sufficiently larger than the number of predictors that each party holds.

2.4 Lin and Karr (2010)

Lin and Karr (2010) propose a general protocol that can be used for exponential family models. Suppose that each *row* X_i from the combined data X is from an exponential family where the density is of the form:

$$f(x_i; \theta) = b(x_i) \exp[\, a(\theta)^T t(x_i) - c(\theta)\,] \tag{2.6}$$

Then the log-likelihood would be:

$$l(\theta|X) = \sum_{i=1}^{n} \log b(X_i) + \sum_{i=1}^{n} [\, a(\theta)^T t(X_i) - c(\theta)\,] \tag{2.7}$$

and the maximum likelihood estimate is:

$$\hat{\theta} = \arg\max_{\theta} \; a(\theta)^T \sum_{i=1}^{n} t(x_i) - nc(\theta) \tag{2.8}$$

Notation for Sections 2.4.1 to 2.4.4
In this section (Section 2.4) we use slightly different notation from other sections, mainly so that we can use subscripts to indicate observation numbers. We denote the two parties A and B whereas they were called parties A_1 and A_2 in other sections, and also refer to their data as X^A and X^B. In this section we assume that X^A and X^B are both vectors of length n; so for example X_i^A will indicate the

ith element of party A's data.

2.4.1 The Protocol (general)

Consider the vertically partitioned data setting where two parties A and B hold data X^A and X^B respectively ($X = [X^A\, X^B]$). For simplicity assume that both parties only hold one variable, hence X^A and X^B are both vectors of length n. In order to obtain $\hat{\theta}$ in equation (4), the parties would need to compute $\sum_{i=1}^{n} t(X_i^A, X_i^B)$ securely. The protocol is as follows:

For each data record i,

1. Party A generates a vector Z of length m, one component of which is X_i, and the other $m-1$ of which are random, and sends it to party B.
2. Party B computes $t(Z_1, X_i^B), \ldots, t(Z_m, X_i^B)$, generates a random value ϵ_i, and calculates $t(Z_1, X_i^B) - \epsilon_i, \ldots, t(Z_m, X_i^B) - \epsilon_i$.
3. Party A obtains $t(X_i^A, X_i^B) - \epsilon_i$ from these using 1 out of m oblivious transfer.

Once the process is complete for each i, party A has $\sum_{i=1}^{n} [\, t(X_i^A, X_i^B) - \epsilon_i \,]$ and B has $\sum_{i=1}^{n} \epsilon_i$, which add up to $\sum_{i=1}^{n} t(X_i^A, X_i^B)$.

2.4.2 Specific Case: Logistic Regression

Now we apply Lin and Karr (2010) to the specific case of logistic regression. Note that no specific case is discussed in Lin and Karr (2010), so what follows is our own interpretation of how to apply their method. We assume the same two-party setting as in the general case, but this time we will assume matrices X^A and X^B have more than one column. As in Section 2.3, the parties want to obtain the maximum likelihood estimator (MLE) $\hat{\beta}$ using the Newton Ralphson algorithm:

$$\hat{\beta}^{(s+1)} = \hat{\beta}^{(s)} + (X^T W^{(s)} X)^{-1} X^T y \tag{2.9}$$

where $W^{(s)} = diag(\pi_i^{(s)}(1 - \pi_i^{(s)}))$, $\pi_i^{(s)} = (1 + \exp\{-x_i^T \beta^{(s)}\})^{-1}$, and $y^{(s)} = y - \pi^{(s)}$.

Assume we are in the sth iteration of the algorithm, and want to compute the components on the left of $\hat{\beta}^{(s)}$ in order to get $\hat{\beta}^{(s+1)}$. The main idea is to use the oblivious transfer approach to calculate $(X^A)^T W^{(s)} X^B$ at each step, which will allow the parties to obtain $X^T W^{(s)} X$. The protocol to update $\beta^{(s)}$ is as follows. Note that the only difference from Fienberg et al. (2009) is in steps 3 and 4, where the parties need to perform matrix multiplication securely.

1. Each party picks an initial value for their share of $\beta_j^{(0)}$, $j = 1, \ldots, p$.
2. Both parties obtain $\pi_i^{(s)}$ after using secure summation to obtain $x_i^T \beta^{(s)}$. Recall that we have assumed that both parties hold more than one predictor; otherwise this step would not be secure.
3. The parties obtain the components of $X^T W^{(s)} X$ through the following process:

   For every $(i,\, j)$ where $i = 1, \ldots, n$ and $j = 1, \ldots, p_A$,

   3.1 Party A generates a vector $\mathbf{Z}_{i,j} = (\, Z^1_{i,j}, \ldots, Z^m_{i,j}\,)$ of length m where one component of which is $X^A_{i,j}$ and the other $m-1$ of which are random, and sends it to party B.

   3.2 For all k where $k = 1, \ldots, p_B$,
   Party B computes $(Z^1_{i,j})^T\, W_i^{(s)} X^B_{i,k}, \,\ldots,\, (Z^m_{i,j})^T\, W_i^{(s)} X^B_{i,k}$, generates a random value $\epsilon_{i,j,k}$, and computes $(Z^1_{i,j})^T\, W_i^{(s)} X^B_{i,k} - \epsilon_{i,j,k}, \,\ldots,$ $(Z^m_{i,j})^T\, W_i^{(s)} X^B_{i,k} - \epsilon_{i,j,k}$

   3.3 Party A obtains $(X^A_{i,j})^T\, W_i^{(s)} X^B_{i,k} - \epsilon_{i,j,k}$ from these using 1 out of m oblivious transfer, for all $k = 1, \ldots, p_B$.

   When this process is complete, the parties will be able to obtain $(X^A_j)^T\, W^{(s)}\, X^B_k$ for all $j = 1, \ldots, p_A$ and $k = 1, \ldots, p_B$. That is, party A will have $\sum_{i=1}^n [\, (X^A_{i,j})^T\, W_i^{(s)} X^B_{i,k} - \epsilon_{i,j,k}\,]$ and B will have $\sum_{i=1}^n \epsilon_{i,j,k}$, and add them together to get:

$$(X^A_j)^T\, W^{(s)}\, X^B_k \;=\; \sum_{i=1}^n (X^A_{i,j})^T\, W_i^{(s)} X^B_{i,k}$$

Hence the parties now have the off-diagonal elements of $X^T W^{(s)} X$. The diagonal elements, $X_j^T W^{(s)} X_j$, $j = 1, \ldots, p$, are computed locally by each party and are shared with the other party. This entire process has to be done in every iteration, but it is not necessary for party A to send a different $\mathbf{Z}_{i,j}$ for each iteration.

Note that in the interest of making the above description simple, we have let one party A do the sending of the m values for each of its elements but this does not have to be the case. In fact it may make more sense for the work to be shared so that the information exchanged is equal, in a similar manner to the *inequity* concept introduced in Karr (2007). Since there will be a total of $np_A p_b$ multiplications to be done, party A should do the initializing in [3.1] for $g_A = \frac{n p_A}{p_A + p_B}$ rows for all of its columns, while party B does the initializing for the rest ($g_B = n - g_A$ rows). In the end party A will be doing [3.1] for $p_A g_A$ of its elements, and B will be doing [3.1] for $p_B g_B$ elements.

4. The parties use the same approach as in step [3.] to obtain their shares of $X^T y$. Each party will hold their shares privately. Unlike step [3.], this step only needs to be done once. Note that for logistic regression each y_i can only take the values 0 or 1, and the other party is aware of this. Assume that party A holds y. Step[3.] can be modified as follows:

   For every (i, k) where $i = 1, \ldots, n$ and $k = 1, \ldots, p_B$:

   4.1 Party B generates a random value $\epsilon_{i,k}$, and computes $X_{i,k}^B - \epsilon_{i,k}$ and $-\epsilon_{i,k}$.

   4.2 Party A obtains $X_{i,k}^B - \epsilon_{i,k}$ from these using 1 out of 2 oblivious transfer.

   When this process is complete, party A will have $\sum_{i=1}^n [\, (X_{i,k}^B)^T y_i - \epsilon_{i,j,k} \,]$ and B will have $\sum_{i=1}^n \epsilon_{i,k}$. Party A will send $\sum_{i=1}^n [\, (X_{i,k}^B)^T y_i - \epsilon_{i,j,k} \,]$ to B, who will obtain:

$$(X_k^B)^T y \;=\; \sum_{i=1}^{n} (X_{i,k}^B)^T y_i$$

for each $k = 1, \ldots, p_B$. Party B does not share these values with A. Party A (the y holder) can compute $(X^A)^T y$ locally and will not share it with B.

5. Each party inverts $X^T W^{(s)} X$. Let:

$$(X^T W^{(s)} X)^{-1} = \begin{pmatrix} \mathcal{A}_{AA}^{(s)} & \mathcal{A}_{AB}^{(s)} \\ \mathcal{A}_{BA}^{(s)} & \mathcal{A}_{BB}^{(s)} \end{pmatrix}$$

   Party A obtains $\mathcal{A}_{AA}^{(s)} (X^A)^T y + \mathcal{A}_{AB}^{(s)} (X^B)^T y$ using secure summation by initiating the protocol, and not sharing the result with B. Similarly, party B will learn the sum $\mathcal{A}_{BA}^{(s)} (X^A)^T y + \mathcal{A}_{BB}^{(s)} (X^B)^T y$.

6. Each party updates its own share of $\hat{\beta}$.

2.4.3 Security breaches in case (a) for Lin and Karr (2010)

Here we look at the possible security breaches under case (a). Again, note that the notation is slightly different from case (a) in Section 2.3 where we discussed the method from Fienberg et al. (2007), but the setting is basically the same:

(a) Two parties A and B.
A holds $(n \times p_A)$ matrix X^A and response y, B holds $(n \times p_B)$ matrix X^B.

Note that steps 1, 2, 5, and 6 are exactly the same as in Fienberg et al. (2007) and so are the possible security breaches. Hence we will focus on looking at steps 3 and 4.

3. (The parties compute $X^T W^{(s)} X$.)
   We refer the reader to Section 2.4.2 of this chapter for details of this step and flow of information. The main risk associated with this step is that party B may correctly guess which of the m elements of $\mathbf{Z}_{i,j}$ is $X_{i,j}^A$. If B has no other information on $X_{i,j}^A$, the probability of this is $1/m$ [1]. The risk of party B making the correct guess will be higher if they have any prior information on $X_{i,j}^A$. These may include (but are not limited to):

   – B knows a specific digit of the value of $X_{i,j}^A$.

[1] This probability can be further reduced with more complex versions of oblivious transfer.

- $X^A_{i,j}$ can only take a finite set of values which B has information on.
- $X^A_{i,j}$ can be estimated from B's data X^B

There is also some risk on party B's side, which may come from party A obtaining $(X^A_{i,j})^T W_i^{(s)} X^B_{i,k} - \epsilon_{i,j,k}$. Since A already knows the value of $(X^A_{i,j})^T W_i^{(s)}$, they will be able to figure out $X^B_{i,k}$ if $\epsilon_{i,j,k}$ is not sufficiently random. Nevertheless, this can easily be avoided as long as B is aware of the risk.

Also note that since the entire $X^T W^{(s)} X$ is shared, each party will learn the inner products $(X^A_j)^T W^{(s)} X^B_k$ for $j = 1, \ldots, p_A$ and $k = 1, \ldots, p_B$, which results in giving them $p_A p_B$ constraints on the other party's data.

4. (Parties compute $X^T y$.)
   There is no information leakage resulting from the 1 out of 2 oblivious transfer protocol because party B already knows that each y_i is either 0 or 1. Aside from that, party B obtains p_A constraints on y from $(X^B)^T y$. Party A does not learn anything more about X^B than it already has.

5. (Parties compute $(X^T W^{(s)} X)^{-1} X^T y$)
   A obtains p_A constraints on $(X^B)^T y$ from learning $\mathcal{A}^{(s)}_{12} (X^B)^T y$, and B obtains p_B constraints on $(X^A)^T y$ from learning $\mathcal{A}^{(s)}_{21} (X^A)^T y$.

Summary

If the NR algorithm took s_1 steps to converge,

(a) Party A learns m possible values of $X^B_{i,k}$, for $\frac{n p_B}{p_A + p_B}$ different values of i and $k = 1, \ldots, p_B$. (from step 3)

(b) Party B learns m possible values of $X^A_{i,j}$, for $\frac{n p_A}{p_A + p_B}$ different values of i and $j = 1, \ldots, p_A$. (from step 3)

(c) Party A obtains $p_A p_B$ linear constraints on X^B. (from step 3)

(d) Party B obtains $p_A p_B$ linear constraints on X^A. (from step 3)

(e) Party B obtains p_B linear constraints on y. (from step 4)

(f) A obtains at most $s_1 p_A$ linear constraints on $(X^B)^T y$. (from step 5)

(g) B obtains at most $s_1 p_B$ linear constraints on $(X^A)^T y$. (from step 5)

As before, (f) and (g) are unique to the two-party case but the rest will happen even if three or more parties are involved. As long as n is sufficiently large compared to p_A or p_B, and if X^A or X^B both contain at least two columns, we can assume that y or X^A or X^B are unlikely to be revealed. The constraints given away in (f)(g) are a multiple of s_1, so enough iterations of the algorithm can reveal $(X^A)^T y$ or $(X^B)^T y$ entirely to the other party; but this still won't be enough to find out y or X^A or X^B.

The main difference from the method based on secure multiplication is in (a) and (b). In (a) for example, instead of obtaining linear constraints on the B's data, party A learns m possible values for $x^B_{i,j}$. In theory party A will have a $1/m$ chance of guessing the correct value of each $x^B_{i,j}$, which can be increased further if they have other information on the values. For continuous variables, party B will have to make sure that m is sufficiently large (in exchange making the process more computationally expensive) and that the m values are chosen carefully. For variables that can only take a finite set of values (e.g. categorical variables), m should equal the size of the set in order to mitigate information leakage, that is, assuming that the other party knows this finite set.

2.4.4 Security breaches in case (b) for Lin and Karr (2010)

(b) Two parties A and B.
A holds response y. B holds $n \times p_B$ matrix X.

Form of NR algorithm and assumptions
Same as in case (a).

Secure NR algorithm
Recall that B only holds response y, while A holds $n \times p_A$ matrix X. The algorithm is as follows:

1. B chooses initial value $\beta^{(0)}$, and doesn't tell A. No information exchanged.
2. B obtains $\pi^{(s)}$ by computing $X^T \beta^{(s)}$ locally. No information exchanged.

3. B computes $X^TW^{(s)}X$ locally. No information exchanged.

4. B obtains $I = X^T y$ using the same method as in step [4.] for case (a). We refer the reader to section 2.5.3 for the information flow. In section 2.5.3 we have seen that B will obtain p_B constraints on y but learn nothing else, and that this process only needs to be done in the first iteration of the algorithm.

5. B computes $(X^TW^{(s)}X)^{-1}X^Ty$. No information exchanged.

6. B updates $\hat{\beta}$.

Summary

- B obtains at most around p_B constraints on y. (from step 4)

In the case that A only holds the response vector, no information about B's data is leaked to A. Furthermore we can make a more general statement that if there are k parties ($k \geq 2$) involved where A only holds the response, A would learn nothing about the other parties' data, while the information leaked between A_s and A_t ($s, t \geq 2$) is the same as in case (a). We can conclude that as long as n is sufficiently larger than p_B, the process would be secure.

2.5 Nardi et al. (2012)

2.5.1 Form of NR algorithm and assumptions

$$\beta_{s+1} = \beta_s - 4(X^T X)^{-1} \nabla l$$
$$\text{where: } \nabla l = X^T \left(y - \sigma(X^T\beta_s)\right)$$

This is a well-studied approximation and is known to eventually converge to the correct parameter value. The pieces of the algorithm are computed as random shares of encrypted values. Note that $\sigma(\beta_s^T X)$ is a linear approximation of the logistic function evaluated at $\beta_s^T X$. The approximation is used in order to make use of the homomorphic properties of Paillier's encryption scheme which allows multiplication and addition in encrypted form. For the same reason, $X^T X$ is

inverted in a way such that only matrix multiplication and addition is required. Details of this approximation is given in Guo and Higham (2006). Convergence is determined by checking if:

$$(\nabla l)^T \,(-4(X^T X)^{-1} \nabla l) \,\leq \epsilon$$

This is evaluated using a variant of the oblivious transfer protocol by Blake and Kolesnikov (2004).

2.5.2 Secure NR algorithm for Cases (a) and (b) in Nardi et al. (2012)

(a) Two parties A_1 and A_2.
A_1 holds response y and $n \times p_1$ matrix X_1, A_2 holds $n \times p_2$ matrix X_2.

(b) Two parties A_1 and A_2.
A_1 only holds response y. A_2 holds $n \times p_2$ matrix X.

The secure Newton Ralphson algorithm is as follows:

1. Each party picks an initial value for their share of $\beta_j^{(0)}$, $j = 1, \ldots, p$. There is no exchange of information here.

2. Each party converts their data into integers using the 2s complement approach described in Section 2.3.4, and then encrypt those values using Paillier's cryptosystem. No exchange of information.

3. (Only in case (a)) The parties obtain random shares of $X^T y$ using the constructions described in Section 2.3.5, where we have seen that the only information exchanged are encrypted values. There is no security risk as long as the encrypted values cannot be broken.

4. (In each iteration s of the algorithm) The parties obtain random shares of $\beta_s^T X$ using the same method as in [3.]. Again there is no security risk since the only information exchanged are encrypted values.

5. The parties obtain random shares of $\sigma(\beta_s^T X) = \sigma(\beta_s^T x_1), \ldots, \sigma(\beta_s^T x_n)$ where $\sigma(\beta_s^T x_i)$ is the linear approximation of the logistic function evaluated at $\beta_s^T x_i$. There is no security risk here as well.

6. The parties first compute their shares of $X^T W^{(s)} X$, and obtain random shares of the (approximation of the) inverse using the method by . No security risk.

7. The parties check for convergence (evaluate $(\nabla l)^T (-4(X^T X)^{-1} \nabla l) \leq \epsilon$) by following the protocol by Blake and Kolesnikov (2004) . If convergence has been reached, the parties combine their shares of random sums to obtain $\hat{\beta}$.

Summary of Information Leakage

Neither party can learn anything about other party's data from the intermediate calculations, as long as all the encrypted values are secure. If the parties agree to share some intermediate statistics such as $X^T X$, say for model diagnostic purposes, the algorithm can be modified to do so. This will cause some information leakage similar to what we have seen in previous sections; but in general the loss of information will be smaller than in Fienberg et al. (2009) and Lin and Karr (2010).

2.6 Discussion

In this chapter we have explored the potential disclosure risks in the secure protocols from Fienberg et al. (2009), Lin and Karr (2010), Nardi et al. (2012) to perform logistic regression under the vertically partitioned database setting. We did this by carefully following the flow of the information from one party to another while analyzing the risks are associated with that information, which was not done in the original articles. The risk of one party making inferences on another party's data using the final model is common to all three methods, since they all assume that the final results (the logistic regression coefficients) are shared. The risks resulting from intermediate computations differ but are not directly comparable; the appropriate method depends on the size and types of data the parties hold. The protocol from Nardi et al. (2012) allows the information leakage incurred from intermediate computations to be as small as the users wish, with some trade-off in terms of computational cost and information that can be used in diagnostics. Even if the protocol is modified so that say, $X^T X$ is shared, the information leakage will not be larger than the other two methods. Hence the protocol works best if computational and communication cost is not of concern, and if the users want

as high a degree of security as possible.

The disclosure risk of the protocol by Lin and Karr (2010) can also be somewhat controlled under ideal situations. That is, if the predictors in the model can take infinite possible values the disclosure risk can be made arbitrarily small by making m large in 1 out of m oblivious transfer. The obvious drawback is that increasing m requires significantly more computation and communication. It is also very possible in real world situations for one party to have enough information to narrow down another party's data to a few candidate values, in which case increasing m would not help much. The method suits the case that many of the variables involved can only take a finite set of values (e.g., categorical variables), since there will be no information leakage if m is taken to be the size of the set.

The protocol by Fienberg et al. (2009) is the simplest of the three, requiring the least amount of computation and communication, but the disclosure risk largely depends on how large the sample size n is compared to the number of predictors p_A and p_B. The method should not be used if n is close to either p_A or p_B or both, since in worst cases it can even lead to one party learning another party's data entirely. However if n is sufficiently larger than the number of predictors, the information leakage from the intermediate computations will be too small to matter in terms of exposing the other party's data. Hence the method serves best if the parties are willing to share some information as long as it does not expose any particular values of their data.

Chapter 3 Secure Penalized Regression Methods

3.1 Introduction

This chapter aims to expand on the literature on performing linear regression on vertically partitioned databases, to penalized versions of linear regression. For ordinary linear regression, Karr et al. (2005) describe an approach that uses the *secure summation protocol*, and *secure matrix multiplication* . They use secure matrix multiplication to calculate the off-diagonal blocks of the full data covariance matrix and describe how some diagnostics can be performed without any further loss of privacy. The approach does have some security risks; for example, one party can obtain some information about another party's data from the full data covariance matrix ($X^T X$) or covariance with the response ($X^T y$), since the approach requires those statistics to be shared among all parties. There is also leakage resulting from the process of performing the secure multiplication protocol, which we have discussed in detail in the previous chapter. Finally, some information about a party's data can be learned from the computation results themselves. It may be possible to obtain good estimates of another party's data since the estimated regression coefficients $\hat{\beta}$ will be shared among all parties.

Hall et al. (2011) describe a more computationally intensive approach that utilizes (a modified version of) a classical method from the cryptography literature, known as *homomorphic encryption*. Their method also requires using a special approximation technique by Guo and Higham (2006) that allows them to obtain inverses of matrices from encrypted values. There are two main advantages to this approach. First of all, it does not require sharing intermediate values $X^T X$ and

$X^T y$ so it guarantees a higher degree of security than the approach by Karr et al. (2009). Secondly, it can be applied to any kind of partitioning scheme whereas the aforementioned approach is only for the purely vertical case. The biggest drawback is that it is more computationally demanding than other methods, as the authors indicate that it might not be practical for moderate to high dimensional problems. Another drawback is that the method does not allow detailed statistical analysis since by default the parties do not have access to the statistics $\{X^T X, X^T y\}$. The algorithm can be modified so that the parties can learn these statistics but that will take out one of its major advantages, especially in the pure vertical setting.

Ordinary least squares (OLS) linear regression is perhaps the most widely used statistical method in almost any field, but researchers are not always satisfied with it. One of the reasons is in prediction accuracy: OLS estimates will often have low bias but large variance, especially when there is multicollinearity between the predictors. Secondly, when there is a large number of predictors the OLS estimates can be difficult to interpret: researchers would often like to determine a smaller subset of the predictors which have the strongest effects. One method that deals with the first issue is *Ridge regression* (Hoerl and Kennard (1970)). Ridge regression is a continuous process that shrinks the OLS estimates of the coefficients. The idea is to sacrifice a little bias to reduce the variance of the predicted values by shrinking some of the coefficients, and hence improve overall prediction accuracy. However, it does not set any of the coefficients to zero so the second issue still remains.

The *Lasso* proposed by Tibshirani (1996), shrinks some coefficients while setting others to zero and hence deals with both issues described above. Lasso is widely used in modern statistical applications and is especially effective in the settings of low to moderate multicollinearity where the solution is believed to be sparse. Tibshirani (1996) point out that the performance of Lasso is suboptimal outside of the sparse and low to moderate multicollinearity setting, but the model selection capabilites of the Lasso can still be useful even then.

Lasso has become extremely popular in statistics even though there is no closed form solution to compute the Lasso estimates, and a great emphasis has been put on developing algorithms that are capable of efficiently computing the Lasso solu-

tions. One of the most well known of such algorithms is the *LARS* (Least angle regression) algorithm, developed by Efron et al. (2004). LARS itself can be viewed as a form of stagewise variable selection algorithm but with a little modification, it efficiently computes the entire sequence of Lasso solutions by exploiting the geometry of the Lasso problem. If implemented in the way that the authors propose, LARS requires only the same order of magnitude of computational effort as OLS applied to the full set of variables.

Our work in this chapter is closely related to that of the secure linear regression protocol Karr et al. (2009) and the LARS algorithm (Efron et al. (2004)). In particular we use secure matrix multiplication (Karr et al. (2009)) along with secure summation to address the problem of running the LARS algorithm when the columns in a design matrix are distributed among multiple sources. Since we use the same SMPC techniques, the security risks are similar to the ones in Karr et al. (2009).

3.1.1 Outline of this Chapter

In Section 3.2 we will provide some background on penalized methods and explain the situations in which they are useful. Section 3.3 describes the LARS algorithm, with a little more detail than in Tibshirani (1996). The secure Lasso procedure for vertically partitioned data is proposed and described in Section 3.4. In Section 3.5 we analyze the security risks in our protocol. Section 3.6 discusses diagnostics, model selection (cross-validation) and gives an example for 4 parties. Section 3.7 addresses non-security related concerns such as computational complexity and correctness of the algorithm. In section 3.8 we explore the possibility of expanding our secure Lasso protocol to generalized linear models. Finally in Section 3.9 we summarize our results along with some discussion on future directions.

3.2 Background

3.2.1 Ridge Regression

Ridge regression Hoerl and Kennard (1970) is a constrained version of ordinary least squares (OLS) that shrinks estimated regression coefficients away from the maximum likelihood estimates. Given a response vector $y \in \mathbb{R}^n$ and a predictor matrix $X \in \mathbb{R}^{n \times p}$, the ridge regression coefficients are defined as:

$$\hat{\beta}^{\text{ridge}} = \underset{\beta \in \mathbb{R}^p}{\arg\min} \; ||y - X\beta||_2^2 + \lambda \, ||\beta||_2^2$$

or equivalently:

$$\hat{\beta}^{\text{ridge}} = \underset{\beta \in \mathbb{R}^p}{\arg\min} \; ||y - X\beta||_2^2 \quad \text{subject to} \;\; ||\beta||_2^2 \leq t$$

where $t \geq 0$ and $\lambda \geq 0$ are tuning parameters that controls the amount of shrinkage that is applied to the estimates. As λ increases (or t decreases) , more shrinkage is employed to the coefficients. The main motivation for using ridge regression over OLS is in improving overall prediction accuracy - it sacrifices some bias to reduce the variance of predicted values. One drawback is that it does not set any coefficients to zero, which may not be desirable for researchers who are interested in interpretation of the model.

Karr et al. (2009) suggest that with methods such as secure summation and secure matrix multiplication, it is straightforward to compute the ridge regression estimates in the distributed setting. Hall et al. (2011) also discuss applications of their methods (homomorphic encryption and random shares) to ridge regression. Given that ridge regression has already been covered in the distributed setting, we focus on Lasso which has not yet been discussed to the best of our knowledge.

3.2.2 Lasso

The Lasso is also a constrained version of OLS, with a similar form to ridge regression:

$$\hat{\beta}^{\text{Lasso}} = \underset{\beta \in \mathbb{R}^p}{\arg\min} \; ||y - X\beta||_2^2 + \lambda \, ||\beta||_1$$

or equivalently:

$$\hat{\beta}^{\text{Lasso}} = \underset{\beta \in \mathbb{R}^p}{\arg\min} \; ||y - X\beta||_2^2 \quad \text{subject to} \;\; ||\beta||_1 \leq t.$$

The only difference between Lasso and ridge regression is that the former uses the l_1 penalty $||\beta||_1$ instead of the l_2 penalty $||\beta||_2^2$ of the latter. Despite the similarities, the two solutions behave very differently. The form of Lasso allows it to shrink some coefficients while setting others to zero, hence it is able to perform variables selection while retaining the good features of ridge regression. As λ increases (or t decreases), more coefficients are set to zero while more shrinkage is employed to the nonzero coefficients. Setting the coefficients to zero may not always be desirable; for example, if some predictor variables are highly correlated Lasso chooses one of those while setting the coefficients of the other variables to zero. The chosen variable can easily change if the data are modified in some way. In this case ridge regression might work better, since it will include all of the correlated variables in the model with the coefficients distributed among them depending on the correlation.

3.2.3 Elastic Net

Zou and Hastie (2005) proposed the Elastic Net which combines shrinkage and variable selection while overcoming the limitations of the Lasso in the cases discussed above. The Elastic Net encourages grouping of variables: groups of highly correlated variables tend to be selected together, instead of only selecting one variable of the group as in Lasso. In addition, in the case where there are more predictor variables than the number of observations ($p > n$, also see Section 3.4.5), Lasso is limited because at most n variables can be selected. Zou and Hastie (2005) show that this is not the case for the Elastic Net. Again, for this chapter we focus on Lasso and leave exploring the possibility of the Elastic Net in distributed settings as future work.

3.2.4 When to use Ridge Regression and Lasso

Tibshirani (1996) examines the relative merits of traditional subset selection, ridge

regression, and Lasso in three different scenarios and reports the following:

- For a small number of large effects, subset selection performs best, Lasso not quite as well and Ridge regression performs quite poorly.
- For a moderate number of moderate-sized effects, Lasso does best, followed by Ridge regression and then subset selection.
- For a large number of small effects, Ridge regression does best by a good margin, followed by Lasso and then subset selection.

3.3 The LARS Algorithm and Lasso

The form of Lasso as seen in Section 3.2.2 suggests that obtaining the solution $\hat{\beta}^{\text{Lasso}}$ is a quadratic programming problem. While this straightforward approach does do the job, quadratic programming generally requires a considerable amount of computation especially when the number of variables involved (in our case the coefficients in β) is large. A computationally efficient alternative is *Least Angle regression* (LARS), introduced by Efron et al. (2004). LARS is not necessarily designed for use in Lasso - it is more of a computationally efficient version of traditional forward selection methods. However, it has been shown in Efron et al. (2004) that with a simple modification LARS can calculate all possible Lasso solutions with less computing time than quadratic programming and most other methods.

The secure Lasso algorithm described in the next section is entirely based on the LARS algorithm. In the current section we will describe the LARS algorithm closely following Efron et al. (2004) but with a little more mathematical detail, to prepare the reader for the secure algorithm in Section 3.4. We will assume that the covariates are linearly independent and have been standardized to have mean 0 at unit length, and that the response has mean 0:

$$\sum_{i=1}^{n} y_i = 0, \quad \sum_{i=1}^{n} X_{ij} = 0, \quad \sum_{i=1}^{n} X_{ij}^2 = 1 \qquad \text{for } j = 1, 2, \ldots, p$$

Note that in this chapter we follow the same notation as described in Section 1.1.1: There are a total of K parties, party A_1 holds the response variable y (a vector)

and matrix of predictor variables X_1, parties $A_2, \ldots, A_K$ hold matrices $X_2, \ldots, X_K$, respectively. Each X_k is an $n \times p_k$ matrix, and we denote $X = [X_1, \cdots, X_K]$ as the $n \times p$ matrix of all predictors where $p = p_1 + \cdots + p_K$. We work with vertically partitioned databases so it is assumed that all parties have the same observations, and have matched up their data using some global identifier.

3.3.1 The LARS Algorithm

Let $\hat{\mu} = X\hat{\beta}$ denote the estimate of y. LARS starts out with $\hat{\beta} = 0$[1] and first finds the predictor variable, say x_{j_1}, that is most correlated with the current residual $y - \hat{\mu}$ (which is initially just y since we let $\hat{\mu}_0 = 0$). It then moves in the direction of x_{j_1} until some other variable, say x_{j_2}, has as much correlation with the residual as x_{j_1} has. LARS then proceeds in the direction equiangular between x_{j_1} and x_{j_2} until a third variable x_{j_3} has as much correlation with the current residual, then proceeds equiangularly between x_{j_1}, x_{j_2}, and x_{j_3} until another variable x_{j_4} enters, and so on.

The LARS algorithm with only $p = 2$ predictors is illustrated in Figure 3.1 Here $\bar{y}_1$ is the projection of y into $L(x_1)$ and $\bar{y}_2$ is the projection of y into $L(x_1, x_2)$. Here $L(x_1)$ is the Euclidean space consisting of the vector x_1, $L(x_1, x_2)$ is the Euclidean space consisting of vectors x_1 and x_2, and so on. Beginning at $\hat{\mu}_0 = 0$, the residual vector $\bar{y}_2 - \hat{\mu}_0$ has more correlation (has a smaller angle) with x_{j_1} than with x_{j_2}, so LARS proceeds in the direction of x_{j_1}. The next LARS estimate is $\hat{\mu}_1 = \hat{\mu}_0 + \hat{\gamma}_1 x_1$, where $\hat{\gamma}_1$ is chosen so that $\bar{y}_2 - \hat{\mu}_1$ bisects the angle between x_{j_1} and x_{j_2}.

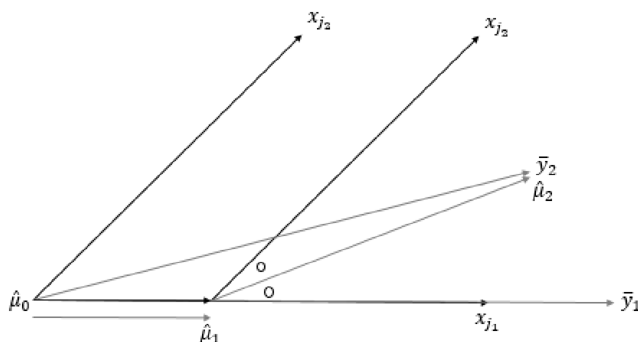

Figure 3.1: LARS for Two Predictors

Following Hastie et al. (2007), the LARS algorithm can be summarized as follows:

[1] Recall that we had the mean subtracted from the response

1. Start with all coefficients equal to zero: $\beta_j = 0$ for $j = 1, \ldots, p$
2. Find the predictor x_j most correlated with the current residual $y - \hat{\mu}$.
3. Move β_j from zero in the direction of its least squares coefficient until some other predictor x_k has as much correlation with the current residual as x_j.
4. Move (β_j, β_k) in the direction of their joint least squares coefficient until some other predictor x_l has as much correlation with the current residual.
5. Keep doing this until all p predictors have entered the model. After p steps, one arrives at the full least squares solution.

In the addition to steps 1 to 5, the full set of Lasso solutions can be produced by a small modification of the algorithm:

5a. If a non-zero coefficient hits zero, remove it from the active set of predictors and recompute the joint direction.

Steps 1,2, and 3 are only for the initial iteration of the algorithm. Steps 4,5, and 5a repeatedly occur in the subsequent iterations. In the following section we explain the calculations required in each step.

3.3.2 The LARS Algorithm in Detail

Following the steps of the LARS algorithm, as introduced in Section 3.2, are the necessary algebraic calculations:

1. **Start with all coefficients equal to zero:** $\beta_j = 0$ **for** $j = 1, \ldots, p$

   Let X be the $n \times p$ matrix of all predictors. Recall that we denote $\hat{\mu} = X\hat{\beta}$ to be the current estimate. Hence the initial estimate is $\hat{\mu}_0 = 0$. Let $c(\hat{\mu})$ be the length p vector of current correlations:

   $$\hat{\mathbf{c}} = c(\hat{\mu}) = X^T(y - \hat{\mu}) \tag{3.1}$$

   So the initial vector of correlations is just $c(\hat{\mu}_0) = X^T y$

2. **Find the predictor x_j most correlated with the current residual $y - \hat{\mu}$.**

   Let $\hat{C} = \max_j \{|\hat{c}_j|\}$. The *active set* $\mathcal{A}$ is the set of indices corresponding to the predictors with greatest absolute current correlations, $\mathcal{A} = \{j : |\hat{c}_j| = \hat{C}\}$. Note that with a slight abuse of notation, we will also use the phrase "active set" to mean the active set of predictors, $\{x_j : |\hat{c}_j| = \hat{C}\}$. In each step of the LARS algorithm we add a new predictor to the active set $\mathcal{A}$. With the Lasso modification we will also occasionally remove, instead of add, a predictor from $\mathcal{A}$.

   The active set $\mathcal{A}$ starts out empty so in the initial step of the algorithm, adding a new predictor to $\mathcal{A}$ simply amounts to finding the index j such that $|\hat{c}_j| = \hat{C}$. In subsequent steps we will want to find the index not already in $\mathcal{A}$ that satisfies this. This is done as part of step 4 and will be explained later on.

3. **Move β_j from zero in the direction of its least squares coefficient until some other predictor x_k has as much correlation with the current residual as x_j.**

   This amounts to augmenting our initial estimate $\hat{\mu}_0 = 0$ in the direction of x_j, to

   $$\hat{\mu}_1 = \hat{\mu}_0 + \hat{\gamma}_1 x_j$$

   where $\hat{\gamma}_1$ is chosen so that the residual $y - \hat{\mu}_1$ is equally correlated with x_j and x_k. Note that x_k must be chosen so that $\hat{\gamma}_1$ is small as possible. How to choose x_k and the value of $\hat{\gamma}_1$ will be explained later on. After the initial step, we will have $\hat{\beta}_j = \hat{\gamma}_1$ and $\hat{\beta}_k = 0$ for $k \neq j$.

4. **Move (β_j, β_k) in the direction of their joint least squares coefficient until some other predictor x_l has as much correlation with the current residual.**

   In a similar manner to step 3, we want to augment our current estimate $\hat{\mu}_{\mathcal{A}}$ in the equiangular direction between all the predictors that are currently

in the active set. We will write this as:

$$\hat{\mu}_{\mathcal{A}_+} \; = \; \hat{\mu}_{\mathcal{A}} + \hat{\gamma}\, \mathbf{u}_{\mathcal{A}}$$

where $\mathbf{u}_{\mathcal{A}}$ is called the *equiangular vector*, the unit vector making equal angles (less than 90 degrees) with all predictors in the active set. We shall see how to construct $\mathbf{u}_{\mathcal{A}}$, and then how to find $\hat{\gamma}$.

Let $s_j = \text{sign}\{\hat{c}_j\}$ and define the matrix

$$X_{\mathcal{A}} = (\cdots \; s_j x_j \; \cdots)_{j \in \mathcal{A}} \tag{3.2}$$

We can think of this as making each predictor in the active set $\mathcal{A}$ face the direction of their correlation with the current residual, and collecting them in a matrix $X_{\mathcal{A}}$. Letting $m_{\mathcal{A}}$ to be the number of elements in $\mathcal{A}$, $X_{\mathcal{A}}$ is an $n \times m_{\mathcal{A}}$ matrix. We want $\mathbf{u}_{\mathcal{A}}$ to have equal angles with the columns in $X_{\mathcal{A}}$ so it must satisfy:

$$X_{\mathcal{A}}^T \mathbf{u}_{\mathcal{A}} \; = \begin{pmatrix} a_{\mathcal{A}} \\ a_{\mathcal{A}} \\ \vdots \\ a_{\mathcal{A}} \end{pmatrix} = a_{\mathcal{A}} \, \mathbf{1}_{\mathbf{m}_{\mathcal{A}}} \tag{3.3}$$

for some scalar $a_{\mathcal{A}}$. Solving for $\mathbf{u}_{\mathcal{A}}$ gives us:

$$\mathbf{u}_{\mathcal{A}} \; = \; a_{\mathcal{A}} \, X_{\mathcal{A}} \, (X_{\mathcal{A}}^T X_{\mathcal{A}})^{-1} \, \mathbf{1}_{\mathbf{m}_{\mathcal{A}}} \tag{3.4}$$

Now since $\mathbf{u}_{\mathcal{A}}$ is a unit vector, we want

$$||\mathbf{u}_{\mathcal{A}}||^2 \; = \; \mathbf{u}_{\mathcal{A}}^T \mathbf{u}_{\mathcal{A}} \; = \; 1 \tag{3.5}$$

so by solving this for $a_{\mathcal{A}}$ we get:

$$a_{\mathcal{A}} \; = \; [{\mathbf{1}_{\mathbf{m}_{\mathcal{A}}}}^T \, (X_{\mathcal{A}}^T X_{\mathcal{A}})^{-1} \, \mathbf{1}_{\mathbf{m}_{\mathcal{A}}}]^{-1/2} \tag{3.6}$$

To summarize, the equiangular vector $\mathbf{u}_{\mathcal{A}}$ can be written as:

$$\mathbf{u}_{\mathcal{A}} \; = \; X_{\mathcal{A}} \, w_{\mathcal{A}} \qquad \text{where} \;\; w_{\mathcal{A}} \; = \; a_{\mathcal{A}} \, (X_{\mathcal{A}}^T X_{\mathcal{A}})^{-1} \, \mathbf{1}_{\mathbf{m}_{\mathcal{A}}} \tag{3.7}$$

Now we are left with finding the value of $\hat{\gamma}$. First we define the *inner product vector* $\mathbf{v}$ to be the vector of inner products between each predictor and $\mathbf{u}_{\mathcal{A}}$:

$$\mathbf{v} \equiv X^T \mathbf{u}_{\mathcal{A}} \tag{3.8}$$

When a new predictor x_k where $k \notin \mathcal{A}$ has as much correlation with the residual with the residual $y - \hat{\mu}_{\mathcal{A}_+}$, the correlation would be:

$$\begin{aligned}
\hat{c}_{k+} &= x_k^T (y - \hat{\mu}_{\mathcal{A}_+}) \\
&= x_k^T (y - (\hat{\mu}_{\mathcal{A}} + \hat{\gamma}\, \mathbf{u}_{\mathcal{A}})) \\
&= x_k^T (y - \hat{\mu}_{\mathcal{A}}) - \hat{\gamma}\, x_k^T \mathbf{u}_{\mathcal{A}} \\
&= \hat{c}_k - \hat{\gamma}\, v_k
\end{aligned}$$

Similarly, for a predictor x_j such that $j \in \mathcal{A}$, the correlation would be:

$$\hat{c}_{j+} = \hat{C} - \hat{\gamma}\, a_{\mathcal{A}}$$

Among all $k \notin \mathcal{A}$, we want to find the smallest positive value of $\hat{\gamma}$ such that $|\hat{c}_{k+}| = |\hat{c}_{j+}|$. Hence:

$$\hat{\gamma} = \overset{+}{\min_{k \in \mathcal{A}^c}} \{ \frac{\hat{C} - \hat{c}_k}{a_{\mathcal{A}} - v_k}, \frac{\hat{C} + \hat{c}_k}{a_{\mathcal{A}} + v_k} \} \tag{3.9}$$

where the + over min indicates that the minimum is taken over only positive components within each choice of k. $\hat{\gamma}$ that satisfies (3.2.9) is the smallest positive value of $\hat{\gamma}$ such that some new index k joins the active set $\mathcal{A}$. The new active set would be $\mathcal{A}_+ = \mathcal{A} \cup \{k\}$, and the new maximum absolute correlation would be:

$$\hat{C}_+ = \hat{C} - \hat{\gamma}\, a_{\mathcal{A}} \tag{3.10}$$

At the end of each iteration, for every predictor x_j where $j \in \mathcal{A}$, the new estimate of $\hat{\beta}_j$ will be:

$$\hat{\beta}_j^+ = \hat{\beta}_j + \hat{\gamma}\, s_j\, (w_{\mathcal{A}})_j \tag{3.11}$$

5. **Keep repeating steps 2 and 4 until all p predictors have entered the model. After p iterations, one arrives at the full least squares**

solution.

In the final step of the LARS algorithm, the active set contains all predictors. The maximum correlation with the residuals will be zero so from (3.2.10) we get:

$$\hat{\gamma} = \frac{\hat{C}}{a_{\mathcal{A}}} \tag{3.12}$$

From this we get the final $\hat{\beta}$, which equals the OLS estimate.

5a. If a non-zero coefficient hits zero, remove it from the active set of predictors and recompute the joint direction.

This is saying that if $\hat{\beta}_j + \gamma' s_j (w_{\mathcal{A}})_j = 0$ in (3.11) for some positive $\gamma' < \hat{\gamma}$, we should take j out of the active set and return to step 4. Hence for each $j \in \mathcal{A}$ we will compute:

$$\hat{\beta}_j + \gamma_j s_j (w_{\mathcal{A}})_j = 0 \quad \Leftrightarrow \quad \gamma_j = \frac{-\hat{\beta}_j}{s_j (w_{\mathcal{A}})_j} \tag{3.13}$$

and let:

$$\gamma' = \min_{j \in \mathcal{A}, \gamma_j > 0} \{\gamma_j\} \tag{3.14}$$

If $\gamma' < \hat{\gamma}$, we will stop the ongoing LARS iteration at $\gamma = \gamma'$ and take the minimizing j in (4.2.14) out from the active set. Specifically, we will have $\mathcal{A}_+ = \mathcal{A} - \{j\}$ and the following updates:

$$\hat{\mu}_{\mathcal{A}_+} = \hat{\mu}_{\mathcal{A}} + \gamma' \mathbf{u}_{\mathcal{A}} \tag{3.15}$$

$$\hat{C}_+ = \hat{C} - \gamma' a_{\mathcal{A}} \tag{3.16}$$

$$\hat{\beta}_j^+ = \hat{\beta}_j + \gamma' s_j (w_{\mathcal{A}})_j \tag{3.17}$$

3.4 The Secure Protocol

3.4.1 Settings and Assumptions

We work under the vertically partitioned data setting described in Chapter 1 and assume that the data have been standardized as discussed in the beginning of Section 3.3. The jth column of party A_k's data X_k will be denoted as x_{k_j} where $j = 1, \ldots, p_k$. The response variable, which we assume that party A_1 holds, will simply be denoted as y. Each party is not willing to share any of their individual columns to any other party, not to mention their entire data matrices.

The parties *are* willing to share the correlation values (calculated via secure matrix multiplication) between their data and other parties' data, but only with the party involved. For example parties A_1 and A_2 will share $X_1^T X_2$ with each other but not with any other party. The parties will also share to every other party, all intermediate values of the coefficients associated with their data. Lastly it is also assumed that the parties are semi-honest, i.e. they will follow the protocol sincerely and use their true data values. Parties may retain values from intermediate calculations but will not collude with other parties to obtain more information about some other party's data.

3.4.2 What is Learned

In the end every party will obtain enough information to make a plot similar to that in Figure 3.2, which we will refer to as the *Lasso path*. The algorithm starts with all estimates of coefficients set to zero and terminates when the sum of absolute coefficients reaches the maximum, which happens at the OLS estimate of $\hat{\beta}$. For example, in Figure 3.2, the algorithm first chooses the variable *bmi*[2] held by party A_1 as most correlated with the initial residuals (which is just y). It then increases the coefficient for *bmi* until a second variable *ltg*, held by party A_3 enters the model, and so on. The points at which a variable enters or leaves the current model will be referred to as *steps*. If the parties wish they can terminate the algorithm at some intermediate step but every party will have to withdraw at that point. It is also possible to calculate the residuals for any intermediate estimate of $\hat{\beta}$ in the

[2] We provide more details on these variables when we discuss the example in more detail; see Section 3.6.3.

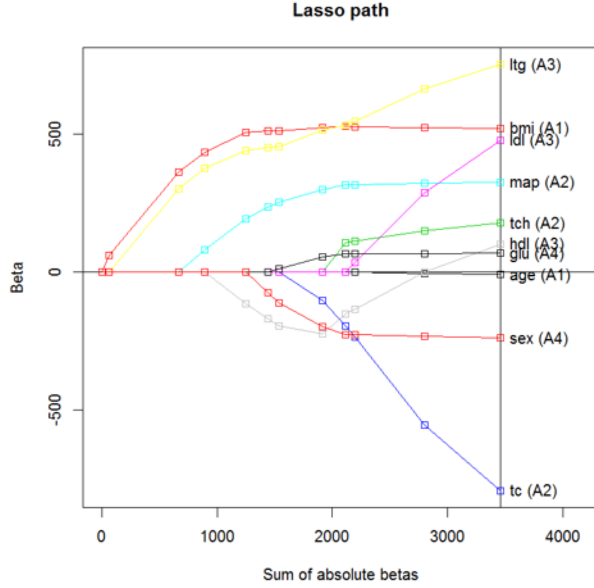

Figure 3.2: Example of a Lasso path for 10 predictors (Diabetes Data, see Section 3.6.3 for details).

Lasso path with the help of secure summation.

3.4.3 Protocol for Lasso over Vertically partitioned data

We now propose and describe a secure protocol that each party follows to obtain all Lasso solutions in a vertically partitioned setting. For simplicity we focus on explaining the algebraic calculations the parties need to perform; the reader is referred to the previous section for details on the derivations of the calculations and the underlying geometry. Also note that items (i) to (vii) below do not necessarily correspond to the steps in the previous section.

(i) Calculating the initial vector of current correlations

$$\hat{\mathbf{c}} = X^T (y - \hat{\mu}_0) = X^T y \tag{3.18}$$

and also

$$\hat{C} = \max_j \{|\hat{c}_j|\} \quad \text{and} \quad \mathcal{A} = (j : |\hat{c}_j| = \hat{C}) \tag{3.19}$$

This is done by having party A_k, for every $k = 2, \ldots K$, apply secure matrix

multiplication with party A_1 to obtain $X_k^T y$. Each party then finds the maximum value of $|\hat{c}_j|$ within their own share of $\hat{\mathbf{c}}$, and shares it with, say, party A_1 only. Party A_1 can compute $X_1^T y$ locally and then find out $\hat{C}$.

In the end:

- Party A_1 learns every other party's maximum value of $|\hat{c}_j|$. It uses this to find $\hat{C}$ and $\mathcal{A}$, which is shared with everyone else. ($\mathcal{A}$ in the initial step only includes one element.)
- Party A_k, for every $k = 2, \ldots K$, learns $X_k^T y$ and also in the process$s_{k_j} = \pm 1$ for each of their predictors x_{k_j}, where $j = 1 \ldots p_k$. It learns $\hat{C}$ and $\mathcal{A}$ from party A_1.

Note that every party sharing their maximum value of $|\hat{c}_j|$ with A_1 is not absolutely necessary, it is just one way of finding the maximum value of $|\hat{c}_j|$ among all parties. If preferred the parties can use other strategies to find $\hat{C}$, e.g., invoking $K(K-1)/2$ instantiations of the MillionaresâĂŹ problem protocol **?**. However, in most cases sharing the above information will not cause a serious privacy breach, even if a party holds only one variable.

We remind the reader that part (i) only applies to the first iteration of the algorithm. Parts (ii) and beyond (discussed below) applies to every iteration.

(ii) Calculating $X_{\mathcal{A}}^T X_{\mathcal{A}}$ and $a_{\mathcal{A}}$ and $w_{\mathcal{A}}$

Let us denote the columns in $X_{\mathcal{A}}$ as $(X_{\mathcal{A}})_i$ where $i = 1, \ldots, m_{\mathcal{A}}$. We remind the reader that each $(X_{\mathcal{A}})_i = (s_{\mathcal{A}})_i \, (x_{\mathcal{A}})_i$ where $(x_{\mathcal{A}})_i$ is the data column in its original form and $(s_{\mathcal{A}})_i$ is the sign of its correlation with the current residual. $X_{\mathcal{A}}^T X_{\mathcal{A}}$ can be written as:

$$X_{\mathcal{A}}^T X_{\mathcal{A}} = \begin{pmatrix} (X_{\mathcal{A}})_1^T (X_{\mathcal{A}})_1 & (X_{\mathcal{A}})_1^T (X_{\mathcal{A}})_2 & \cdots & (X_{\mathcal{A}})_1^T (X_{\mathcal{A}})_{m_{\mathcal{A}}} \\ (X_{\mathcal{A}})_2^T (X_{\mathcal{A}})_1 & (X_{\mathcal{A}})_2^T (X_{\mathcal{A}})_2 & \cdots & (X_{\mathcal{A}})_2^T (X_{\mathcal{A}})_{m_{\mathcal{A}}} \\ \vdots & \vdots & \cdots & \vdots \\ (X_{\mathcal{A}})_{m_{\mathcal{A}}}^T (X_{\mathcal{A}})_1 & (X_{\mathcal{A}})_{m_{\mathcal{A}}}^T (X_{\mathcal{A}})_2 & \cdots & (X_{\mathcal{A}})_{m_{\mathcal{A}}}^T (X_{\mathcal{A}})_{m_{\mathcal{A}}} \end{pmatrix} \quad (3.20)$$

All the diagonal elements can be calculated locally within each party. Secure matrix multiplication will be used in the off-diagonal elements where two

predictors $(X_{\mathcal{A}})_k$ and $(X_{\mathcal{A}})_j$ are held by different parties. Note that for any given k and j, matrix multiplication only needs to be done once in the entire algorithm to compute $(X_{\mathcal{A}})_k^T\,(X_{\mathcal{A}})_j$ for each iteration. To see what this means, notice that (3.20) can be rewritten as:

$$\begin{pmatrix} s_1 & & \\ & \ddots & \\ & & s_{m_{\mathcal{A}}} \end{pmatrix} \begin{pmatrix} (x_{\mathcal{A}})_1^T\,(x_{\mathcal{A}})_1 & (x_{\mathcal{A}})_1^T\,(x_{\mathcal{A}})_2 & \cdots & (x_{\mathcal{A}})_1^T\,(x_{\mathcal{A}})_{m_{\mathcal{A}}} \\ (x_{\mathcal{A}})_2^T\,(x_{\mathcal{A}})_1 & (x_{\mathcal{A}})_2^T\,(x_{\mathcal{A}})_2 & \cdots & (x_{\mathcal{A}})_2^T\,(x_{\mathcal{A}})_{m_{\mathcal{A}}} \\ \vdots & \vdots & \cdots & \vdots \\ (x_{\mathcal{A}})_{m_{\mathcal{A}}}^T\,(x_{\mathcal{A}})_1 & (x_{\mathcal{A}})_{m_{\mathcal{A}}}^T\,(x_{\mathcal{A}})_2 & \cdots & (x_{\mathcal{A}})_{m_{\mathcal{A}}}^T\,(x_{\mathcal{A}})_{m_{\mathcal{A}}} \end{pmatrix} \begin{pmatrix} s_1 & & \\ & \ddots & \\ & & s_{m_{\mathcal{A}}} \end{pmatrix}$$

where each $(s_{\mathcal{A}})_j$ depends on the iteration but $(x_{\mathcal{A}})_k^T\,(x_{\mathcal{A}})_j$ does not. Hence the parties can keep reusing $(x_{\mathcal{A}})_k^T\,(x_{\mathcal{A}})_j$ while updating the $(s_{\mathcal{A}})_j$'s in each iteration.

Once this is done, all the elements in $X_{\mathcal{A}}^T\,X_{\mathcal{A}}$ will be shared with all parties who have at least one predictor in the active set. Each party can then invert[3] $X_{\mathcal{A}}^T\,X_{\mathcal{A}}$ to obtain:

$$a_{\mathcal{A}} \;=\; [\mathbf{1_{m_{\mathcal{A}}}}^T\,(X_{\mathcal{A}}^T\,X_{\mathcal{A}})^{-1}\,\mathbf{1_{m_{\mathcal{A}}}}]^{-1/2} \tag{3.21}$$

and also compute:

$$w_{\mathcal{A}} \;=\; a_{\mathcal{A}}\,(X_{\mathcal{A}}^T\,X_{\mathcal{A}})^{-1}\,\mathbf{1_{m_{\mathcal{A}}}} \tag{3.22}$$

The scalar $a_{\mathcal{A}}$ and the entire vector $w_{\mathcal{A}}$ is shared among all parties.

(iii) (Not) calculating the equiangular vector $\mathbf{u}_{\mathcal{A}}$

From (3.7), we have:

$$\mathbf{u}_{\mathcal{A}} \;=\; X_{\mathcal{A}}\,w_{\mathcal{A}} \;=\; (w_{\mathcal{A}})_1\,(X_{\mathcal{A}})_1 + (w_{\mathcal{A}})_2\,(X_{\mathcal{A}})_2 + \cdots + (w_{\mathcal{A}})_{m_{\mathcal{A}}}\,(X_{\mathcal{A}})_{m_{\mathcal{A}}} \tag{3.23}$$

where $(w_{\mathcal{A}})_i$ is the ith element of the vector $w_{\mathcal{A}}$.

It is tempting to use secure summation to calculate $\mathbf{u}_{\mathcal{A}}$. However, notice that this will cause security issues if there are only two parties in the active set. For example, suppose that at the second iteration of the algorithm the active set of predictors is $\{(X_{\mathcal{A}})_1,\,(X_{\mathcal{A}})_2\}$ where party A_k holds $(X_{\mathcal{A}})_1$ and party A_j holds $(X_{\mathcal{A}})_2$. Then the two parties will use secure summation to

[3] To save on computation, Efron et al. (2004) recommend updating the cholesky decompostion of $X_{\mathcal{A}}^T\,X_{\mathcal{A}}$ found at the previous step, using the method of Golub and Van Loan (2012). There are no security concerns to this since all parties involved already know the entirety of $X_{\mathcal{A}}^T\,X_{\mathcal{A}}$

calculate:

$$\mathbf{u}_{\mathcal{A}} = (w_{\mathcal{A}})_1 (X_{\mathcal{A}})_1 + (w_{\mathcal{A}})_2 (X_{\mathcal{A}})_2$$

but since both parties know the values of both $(w_{\mathcal{A}})_1$ and $(w_{\mathcal{A}})_2$, using secure summation to compute $\mathbf{u}_{\mathcal{A}}$ will let one party learn the entire column of the other.

To look at another example, say the active set of predictors is $\{(X_{\mathcal{A}})_1, (X_{\mathcal{A}})_2, (X_{\mathcal{A}})_3\}$ where party A_k holds $(X_{\mathcal{A}})_1$ and party A_j holds $(X_{\mathcal{A}})_2$ and $(X_{\mathcal{A}})_3$. In this case the above problem can be avoided if party A_k initiates the secure summation protocol, but there is uneven exchange of information.

Fortunately, the protocol does not require computing $\mathbf{u}_{\mathcal{A}}$ explicitly (in fact, it does not make use of secure summation at all). We will see this in the next step.

(iv) Computing the inner product vector $\mathbf{v} = X^T \mathbf{u}_{\mathcal{A}}$

Recall that $X = [X_1 \cdots X_K]$ is the entire $n \times p$ matrix of predictors where each X_k is the $n \times p_k$ matrix of party A_k's data. We will denote the ith column of party A_k's data as $(X_k)_i$. The inner product vector $\mathbf{v}$ will have length p, where $p = p_1 + \cdots + p_K$.
Now observe:

$$\begin{aligned}
\mathbf{v} &= X^T \mathbf{u}_{\mathcal{A}} \\
&= \begin{pmatrix} (X_1)_1^T \\ (X_1)_2^T \\ \vdots \\ (X_K)_{p_K}^T \end{pmatrix} [\,(w_{\mathcal{A}})_1 (X_{\mathcal{A}})_1 + (w_{\mathcal{A}})_2 (X_{\mathcal{A}})_2 + \cdots + (w_{\mathcal{A}})_{m_{\mathcal{A}}} (X_{\mathcal{A}})_{m_{\mathcal{A}}}\,] \\
&= \begin{pmatrix} (w_{\mathcal{A}})_1 (X_1)_1^T (X_{\mathcal{A}})_1 + (w_{\mathcal{A}})_2 (X_1)_1^T (X_{\mathcal{A}})_2 + \cdots + (w_{\mathcal{A}})_{m_{\mathcal{A}}} (X_1)_1^T (X_{\mathcal{A}})_{m_{\mathcal{A}}} \\ (w_{\mathcal{A}})_1 (X_1)_2^T (X_{\mathcal{A}})_1 + (w_{\mathcal{A}})_2 (X_1)_2^T (X_{\mathcal{A}})_2 + \cdots + (w_{\mathcal{A}})_{m_{\mathcal{A}}} (X_1)_2^T (X_{\mathcal{A}})_{m_{\mathcal{A}}} \\ \vdots \\ (w_{\mathcal{A}})_1 (X_K)_{p_K}^T (X_{\mathcal{A}})_1 + (w_{\mathcal{A}})_2 (X_K)_{p_K}^T (X_{\mathcal{A}})_2 + \cdots + (w_{\mathcal{A}})_{m_{\mathcal{A}}} (X_K)_{p_K}^T (X_{\mathcal{A}})_{m_{\mathcal{A}}} \end{pmatrix}
\end{aligned}$$

Hence each party can calculate their share of $\mathbf{v}$ by applying secure matrix matrix multiplication to their data and other parties' data. Note that most of these calculations may have already been done earlier in the algorithm. It is

not necessary for each party A_k to share v_k with other parties, though doing so will not cause a serious privacy breach[4].

(v) Calculating $\hat{\gamma}$

Recall that:

$$\hat{\gamma} = \min_{k \in \mathcal{A}^c}^{+} \left(\frac{\hat{C} - \hat{c}_k}{a_{\mathcal{A}} - v_k}, \frac{\hat{C} + \hat{c}_k}{a_{\mathcal{A}} + v_k} \right) \tag{3.24}$$

so each party who holds a predictor x_k that is not in the active set ($k \in \mathcal{A}^c$) will need to know the four components: $a_{\mathcal{A}}$, v_k, $\hat{C}$, and $\hat{c}_k$. As discussed earlier, $a_{\mathcal{A}}$ is shared among everyone and each party knows v_k if they hold x_k. We also saw in part (i) that the initial value of $\hat{C}$ is shared among all parties and in subsequent iterations it is updated as:

$$\hat{C}_{+} = \hat{C} - \gamma \, a_{\mathcal{A}} \tag{3.25}$$

where $\gamma = \{\hat{\gamma}\} \cup \{\gamma'\}$. As long as each value of $\hat{\gamma}$ or γ' is shared, we can assume every party knows $\hat{C}$ as well. Similarly, the initial value of $\hat{c}_k = x_k^T \, y$ and it is updated as:

$$\hat{c}_j + = \hat{c}_k - \gamma \, v_k \tag{3.26}$$

so every party can update their value of $\hat{c}_k$ on their own.

Now the parties will have to share their minimum positive value of $\frac{\hat{C}-\hat{c}_k}{a_{\mathcal{A}}-v_k}$ or $\frac{\hat{C}+\hat{c}_k}{a_{\mathcal{A}}+v_k}$ with the other parties to find the minimum positive value among all predictors not in the active set. The minimizing index k and value will be revealed but except in very special cases (e.g. $p > n$) this carries very little information on x_k itself. The bottom line is that $\hat{\gamma}$ can found securely with respect to each party, and the value will be shared among all parties.

(vi) Computing γ' for the Lasso modification.

We want to securely compute:

$$\gamma' = \min_{j \in \mathcal{A}, \gamma_j > 0} \{\gamma_j\} \quad \text{where} \quad \gamma_j = \frac{-\hat{\beta}_j}{s_j \, (w_{\mathcal{A}})_j} \tag{3.27}$$

[4] This goes for the two-party case as well, which is discussed in section 3.5.1

We have assumed that all intermediate values of the coefficients $\hat{\beta}$ will be shared among all parties. We also saw that s_j and $(w_{\mathcal{A}})_j$ can be computed securely and will be shared among all parties in the active set. Hence every party in the active set can calculate γ' without any additional exchange of information.

(vii) Updating the coefficients and the active set

If $\gamma' < \hat{\gamma}$, where j is the minimizing index in (3.24):

$$\mathcal{A}_+ = \mathcal{A} - \{j\} \tag{3.28}$$

$$\hat{\beta}_j^+ \;=\; \hat{\beta}_j \;+\; \gamma' \, s_j \, (w_{\mathcal{A}})_j \tag{3.29}$$

Otherwise (if $\gamma' > \hat{\gamma}$), where j is the minimizing index in (4.3.7):

$$\mathcal{A}_+ = \mathcal{A} + \{j\} \tag{3.30}$$

$$\hat{\beta}_j^+ \;=\; \hat{\beta}_j \;+\; \hat{\gamma} \, s_j \, (w_{\mathcal{A}})_j \tag{3.31}$$

3.4.4 Computation Strategies

Note that throughout the entire algorithm, secure computation is only used when two parties calculate the inner product $x_k^T \, x_j$. If the parties have agreed that the goal is to obtain the entire sequence of Lasso solutions (as we have assumed so far), the best strategy is to compute the entire cross product matrix in the beginning:

$$X^T X \;=\; \begin{pmatrix} X_1^T X_1 & X_1^T X_1 & \cdots & X_1^T X_K \\ X_2^T X_1 & X_2^T X_2 & \cdots & X_2^T X_K \\ \vdots & \vdots & \cdots & \vdots \\ X_K^T X_1 & X_K^T X_2 & \cdots & X_K^T X_K \end{pmatrix} \tag{3.32}$$

using secure matrix multiplication, and let each party learn $X^T X$ entirely. This is best because every party will learn $X^T X$ entirely at some point in the algorithm anyway, and it saves communication cost. In fact, if the parties are also willing to share their values of $X_k^T \, y$ with everyone else, then any party can carry out the

entire algorithm on their own without further communication with other parties. Otherwise the parties will have to communicate some intermediate values in each iteration of the algorithm. In the "usual" case where $n > p$, sharing $X_k^T y$ can be recommended for the following reasons:

- It saves communication cost, as mentioned above.
- For party A_j where $j \notin \{1, k\}$, knowing $X_k^T y$ does not reveal much information about X_k or y itself. Even party A_1, who holds y, will not learn much about X_k by knowing $X_k^T y$ (see Section 2.2.2).
- Depending on the order of the variables entering the model some parties, but not all, will be able to reverse calculate some of the $x_k^T y$ values anyway.

There do exist situations where calculating $X^T X$ entirely in the beginning may not be optimal. For example, if for some reason the parties have a previously discussed agreement to terminate the algorithm at some threshold $||\hat{\beta}|| < t$, then some of the inner product calculations will never be carried out. Hence in this case the strategy would be counter-productive both security-wise and computation-wise. For similar reasons the strategy is not optimal in the case where $p > n$, which will be discussed in the next section.

3.4.5 Special case: $p > n$

As discussed in Efron et al. (2004), the LARS algorithm still works when there are many more predictors than observations ($p > n$). This holds for our proposed secure protocol for vertically partitioned data as well. When $p > n$, the algorithm terminates after $n - 1$ variables enter the model[5]. Note that although the model does not involve more than $n - 1$ variables at any time, the number of different variables to have ever entered the model can be (and typically is) greater than $n - 1$. Let us look at a simple example: Figure 3.3 shows the Lasso path where we applied our secure algorithm on a dataset with $p = 10$ and $n = 7$. It is based on the same data as in Figure 3.2 in Section 3.4.2, except that only 7 (randomly chosen) observations were used while keeping all 10 predictors. We can see that the algorithm terminates with 6 variables, while at one point the model included a variable that is not one of the 6 variables in the final model, hence there were 7

[5] It is $n - 1$ rather than n because the columns of X have been mean centered; the row-rank of X is $n - 1$.

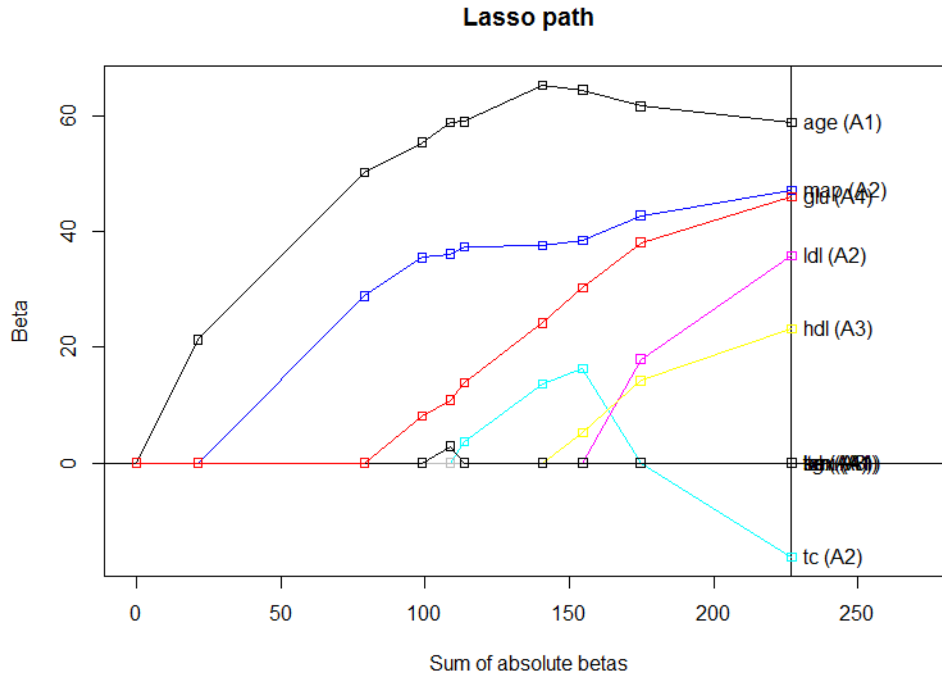

Figure 3.3: Example of a Lasso path for 10 predictors, 7 Observations

variables that ever entered the model.

As discussed earlier, in this special case the secure inner product computations of $x_k^T x_j$ should be carried out need-based since the algorithm will terminate before some of the $x_k^T x_j$ values are needed. There is also greater security concern when $p > n$. For example if some party A_k holds $p_k > n$ predictors, it can easily find out the entire column y by knowing $X_k^T y$. In this case party A_k will also be able to learn the entire columns of some predictors that other parties own as well.

3.5 Disclosure Risks

In Section 3.2 we discussed how much information is leaked when the secure summation and multiplication are applied between two parties. In the current section we analyze how much of a party's data will be revealed to another party when they run through the entire secure Lasso algorithm. We have assumed that the

parties are semi-honest, so we can think that the parties will not cooperate with each other to reveal another party's data. This allows us to focus on the case where there are only two parties involved. If the parties' data are safe under the two party case, we can induce that the data will still be safe when more parties are involved. We will also briefly discuss some issues in residual diagnostics and the special case $p > n$.

3.5.1 The Two Party Case

We will first consider the following scenario:

Data:	$[Y, X_1]$	X_2
Held by party:	A_1	A_2

where X_1 and X_2 are $n \times p_1$ and $n \times p_2$ matrices, respectively. We should also assume that at least one of p_1 or p_2 is greater than or equal to 2; neither variable would be set to zero in the process if $p_1 = p_2 = 1$. Furthermore, for now we will assume that $n > p_1 + p_2$; in Section 3.5.2 we discuss a scenario where this is not the case.

We now follow the protocol of Section 3.4.3.

- Step(i)
  Party A_2 obtains $X_2^T y$ via secure matrix multiplication with A_1. Both parties find their own maximum value of $|x_i^T y|$ (looking only at its own variables) and then compare it with each other to find the maximum among both parties. Let's say k was the maximizing index and variable x_k belonged to party A_2. In the end both parties will learn $\hat{C} = |x_k^T y|$ and $\mathcal{A} = \{k\}$.

  Information flow:

  - (1) Party A_2 sends the set of vectors $Z = [Z_1, Z_2, \dots, Z_g]$ to A_1 where $g = \lfloor (n - p_2)/2 \rfloor$, (2) A_1 computes and sends $(I - Z^T Z)y$ back to A_2, (3) A_2 obtains $X_2^T y$ and does not share it with A_1.
  - A_2 tells A_1 the value of $\hat{C} = |x_k^T y|$ and the index k.

From Section 2.2.2 we know that A_2 can obtain at most around $(n/2 + p_2)$ constraints on y. This is the only instance where A_2 will have any interaction with y, so for a sufficiently large n the response y can be considered safe. A_1 will learn around $n/2$ constraints on X_2. A_1 also obtains one extra constraint on predictor x_k by learning the absolute value of $x_k^T y$.

- Steps (ii) to (vii)

  From here we assume the parties follow the strategy discussed in section 3.4.4, i.e. compute $X^T X$ entirely in the beginning. As discussed before, the amount of information exchanged will be exactly the same whether or not the parties follow the strategy. Now in addition to this, we also assume that the parties decide to share $X_1^T y$ and $X_2^T y$ with each other. We explain later why this would not cause much of a problem. Once $X^T X$ and $X^T y$ are computed and shared, each party can proceed with the entire protocol without further interactions with the other party.

  Information flow:

  - A_2 has already sent $Z_1, Z_2, \ldots, Z_g$ to A_1 in (i), so the only steps are: (1) A_1 computes and sends $(I - Z Z^T) X_1$ back to A_2, and (2) A_2 computes $X_2^T X_1$ and shares it with A_1.
  - A_1 and A_2 share $X_i^T X_i$ and $X_i^T y$ $(i = 1, 2)$ with each other.

  A_1 gets to know $X_2^T X_1$, so it obtains at most $(n/2 + p_1 p_2)$ constraints on X_2. Even at the individual column level A_1 will obtain at most $(n/2 + p_1)$ constraints on any given column x_k. A_2 learns approximately the same amount of information about X_1. By learning $X_2^T y$, A_1 obtains one extra constraint on each of the columns in X_2. A_2 practically learns nothing from $X_1^T y$. Learning $X_i^T X_i$ of the other party is also useless for learning X_i itself.

In summary:

(a) A_1 obtains at most around $(n/2 + p_1 p_2)$ constraints on X_2, with possibly at most around $(n/2 + p_1)$ constraints for each individual column.

(b) A_2 obtains at most around $(n/2+, p_1p_2)$ constraints on X_1, with possibly at most around $(n/2 + p_2)$ constraints for each individual column.

(c) A_2 obtains at most around $(n/2 + p_2)$ constraints on y.

(d) A_1 learns $X_2^T X_2$ and $X_2^T y$ entirely.

(e) A_2 learns $X_1^T X_1$ and $X_1^T y$ entirely.

Note that none of these are unique to the two party case: they will happen even if three or more parties are involved (assuming the parties share $X_i^T y$). Having only two parties does make a small difference when the parties decide not to share $X_i^T y$. In the two party case, both parties can back-calculate the other party's $X_i^T y$ entirely; if more parties are involved, they may not be able to calculate some of the values of $X_i^T y$. Between parties A_1 and A_2 this only affects A_2 as A_1 will learn one constraint for each of A_2's variables by learning $X_2^T y$. Obviously this would be a minor issue as long as n is sufficiently large.

If A_1 only holds the response y, and A_2 holds all the predictors X, all that is needed is for A_2 to obtain $X^T y$ via secure summation with A_1. There is no need for A_2 to share the result with A_1. Then A_2 can finish the remaining process all by itself, and share the final results with A_1 if it wishes. In the end:

(a) A_1 obtains at most around $n/2$ constraints on X.

(b) A_2 obtains at most around $(n/2 + p_2)$ constraints on y.

3.5.2 Two Party Case when $p > n$

Serious privacy breaches can occur when there are more predictors than observations, and if there are only a few parties involved. To look at a simple example, if we have $p_2 > n$ under the scenario in Section 3.6.1, party A_2 can choose n of its predictors to form a non-singular $n \times n$ matrix and learn y entirely, since the algorithm requires A_2 to know $X_2^T y$. It would also be possible for A_2 to learn some of the columns in X_1, but the number of columns that A_2 learns will be smaller if there were more parties involved.

Even if the number of predictors that each party holds (p_i) is less than n, we have seen that each party has a chance to learn up to around $n/2 + p_ip_j$ constraints

about another party's data through the secure matrix multiplication protocol. In general the algorithm cannot be considered secure if the product of any two parties' number of predictors ($p_i p_j$) is anywhere close to $n/2$.

3.5.3 Disclosure Risk from the Output

So far we have only been concerned with the loss of information resulting from the computation (secure matrix multiplication, in particular) itself but not from what can be learned from the output (the coefficient values in the Lasso path). For example, if there are only two parties involved, the party that holds the response y (A_1) can create a model for any of the predictors that the other party A_2 holds. By performing residual analysis (Sections 3.7.1 and 3.7.2), the parties can even tell how good those models are. Hall et al. (2011) provides a rigorous analysis of this issue (in both purely vertical and horizontal cases) and their conclusion is that in general, the regression coefficients themselves will reveal less (a subset of) information that the data covariance matrix ($X^T X$) will reveal. Hence if the parties have agreed that it is okay to share their components of $X^T X$, they will learn nothing new from sharing $\hat{\beta}$.

3.6 Diagnostics and Model selection

3.6.1 Diagnostics

To evaluate the fit of a candidate model the parties can perform residual analysis using secure summation, but it should be done with caution. For example if the parties want to calculate the residuals for the model $\hat{y} = \hat{\beta}_5 x_5$ where x_5 is owned by A_2, secure summation cannot be used without a third party. Now say the parties want to calculate the residuals for the model $\hat{y} = \hat{\beta}_5 x_5 + \hat{\beta}_6 x_6 + \hat{\beta}_{11} x_{11}$ where x_5,x_6, and x_{11} are all owned by A_2. In this case A_2 can send the sum $\hat{\beta}_5 x_5 + \hat{\beta}_6 x_6 + \hat{\beta}_{11} x_{11}$ to A_2 sacrificing some constraints on (x_5,x_6,x_{11}), but A_1 cannot share the entire set of residuals with A_2 since that would result in revealing y. They can, however, share summary statistics with A_2 such as the sum of the squared residuals without risking exposure of y. These situations tend to happen at small values of $\sum ||\beta_1||$,

where it is likely that only two parties have a predictor in the model.

3.6.2 Cross-Validation

Once the parties obtain the full set of Lasso solutions they would want to choose the tuning parameter t from

$$\hat{\beta}^{\text{Lasso}} = \underset{\beta \in \mathbb{R}^p}{\arg\min} \; ||y - X\beta||_2^2 \quad \text{subject to} \;\; ||\beta||_1 \leq t$$

that will give them $\hat{\beta}$ that best fits their purposes. Here we assume that the parties want to choose the right t for prediction purposes. The standard tool for choosing t is cross-validation, which would not be unreasonable to apply since the Lars algorithm (and our secure protocol that is based on it) is computationally inexpensive.

Next we describe how K parties can efficiently execute an M-fold cross-validation securely. Each party k divides their data X_k into M roughly equal parts:

$$X_k = \begin{pmatrix} X_k^1 \\ X_k^2 \\ \cdots \\ X_k^M \end{pmatrix}$$

where each X_k^i $(i = 1, \ldots, M)$ contains roughly n/M observations. Let us denote the training set for party A_k as $X_k^{(-i)}$ when the testing set is X_k^i. Following the strategy in section () where all parties calculate $X^T X$ entirely in the beginning, each party A_k will use secure matrix multiplication with party $j = 1, \ldots, K$ $(j \neq k)$ to obtain $(X_k^i)^T X_j^i$ for subsets $i = 1, \ldots, M$ and store the results. Then for the ith training set, $(X_k^{(-i)})^T X_j^{(-i)})$ can be calculated as:

$$(X_k^{(-i)})^T X_j^{(-i)}) = \sum_{h \neq i} (X_k^h)^T X_j^h \tag{3.33}$$

and similarly to obtain $(X_k^{(-i)})^T y^{(-i)}$, they would calculate:

$$(X_k^{(-i)})^T y^{(-i)} = \sum_{h \neq i} (X_k^h)^T y^i. \tag{3.34}$$

In the end, each party will do M secure matrix multiplications with the remaining $(K-1)$ parties. Note that the computation amount in secure matrix multiplication largely depends on the dimensions of the matrices. Here the row lengths are roughly n/M, so the computation required for each multiplication is smaller than for the full set.

Security concerns

Recall from the previous chapter that when $X_k^T X_j$ is calculated where X_k is $n \times p_k$ and X_j is $n \times p_j$, each party may be able to learn up to around $n/2+p_j$ or $n/2+p_k$ constraints on the other party's individual columns. Roughly speaking, we do not want any party to learn more than n (linearly independent) constraints on any particular column so we would want p_k and p_j to be significantly less than $n/2$. Now if the parties were applying secure multiplication on subsets for an M-fold cross validation, the row lengths would be roughly n/M and each party may be able to learn up to around $n/2M + p_j$ or $n/2M + p_k$ constraints on some column. The parties should not learn more than n/M constraints so p_j and p_k should be significantly less than $n/2M$. In other words, suppose $n > n_0$ is the requirement of n for the secure Lars algorithm without cross validation to be considered secure. Then $n > M \cdot n_0$ would be the requirement for the algorithm when M-fold cross validation is to be done. Once the secure computations have been done and the results are shared, the parties can then obtain the full set of Lasso solutions for each of their training sets. They can then decide on a grid of candidate values for t and calculate prediction errors securely as discussed in Section 3.6.1.

3.6.3 Example: Diabetes Data (Tibshirani (1996))

Here we walk through an example to demonstrate how the proposed protocol can be applied in practice. We use the diabetes dataset[6] from Tibshirani (1996) comprising response variable y, a quantitative measure of disease progression, and $p = 10$ predictor variables with sample size $n = 442$. The ten predictor variables are: *age*, *sex*, *bmi* (body mass index), *map* (mean arterial pressure), and six serial measurements *tc*, *ldl*, *hdl*, *tch*, *ltg*, *glu*. We will refer to the matrix of all 10 predictors as X, an $n \times p$ matrix where $n = 442$ and $p = 10$. By location and scale transformations all the predictors have been standardized to have mean 0 and unit

[6]Note that all the variables in the datset, as provided, were already standardized as discussed in Section 3.3; we did not have access to the data in their orignial form.

length, and the response has mean 0.

The data are vertically partitioned and distributed across four parties: A_1 holds the response y and two predictors *age*, *bmi*, A_2 holds *tch*,*tc*, *map*, A_3 holds *ldl*, *ltg*, *hdl*, and A_4 holds *glu*, *sex*. We can write the design matrix X as $X = [X_1, X_2, X_3, X_4]$ where X_1, X_2, X_3, X_4 have 2, 3, 3, 2 columns, respectively. Now suppose the 4 parties have agreed to do a 5-fold cross validation. Then following the procedure in Section 3.6.2, (for example) party A_1's data X_1 will be separated into $(X_1^1, X_1^2, X_1^3, X_1^4, X_1^5)$, where each X_1^i has 88 or 89 rows and 2 columns. Each party would then do $(4-1)\cdot 5 = 15$ secure matrix multiplications to obtain $(X_k^{(-i)})^T X_j^{(-i)})$ for training set i and party j. Note that all the secure matrix multiplications can be considered safe here since at most $442/(2 \cdot 5) + 3 \approx 48$ constraints will be released for each subset of variable, which has at least 88 elements. Once all these calculations are done and the results are shared, the parties follow the procedures in sections 3.4.2 and 3.6.2 and obtain the Lasso paths for each training set, as shown in Figure 3.4.

From the results in Figure 3.4 the parties know that $\sum ||\beta||_1$ ranges from 0 to somewhere around 4000. Suppose the parties agree to test on a grid of 35 values $t = 500, 600, 700, \ldots, 4000$. The parties would then use secure summation on the testing data to calculate the mean prediction errors for each value of t, and share results. Figure 3.5 shows a plot of the mean prediction errors for our example. The minimum is obtained at $t = 1300$. Figure 3.6 shows the Lasso path for using all observations and we can see that at $t = 1300$ five variables have non-zero coefficients. With some simple calculation the parties can find (on their own) the coefficient values and the corresponding standard errors for the prediction model at this minimum. The results are summarized in Table 3.1.

Variable	Coefficient	Std. Error	Variable	Coefficient	Std. Error
age	0.0000	–	ldl	0.0000	–
bmi	507.1349	80.3212	ltg	442.5185	81.2831
tch	0.0000	–	hdl	-128.5240	81.1892
tc	0.0000	–	glu	0.0000	–
map	202.3924	74.9761	sex	-19.4302	68.8901

Table 3.1: Lasso Estimates for Diabetes Data Based on Five-Fold Cross Validation, using our secure protocol

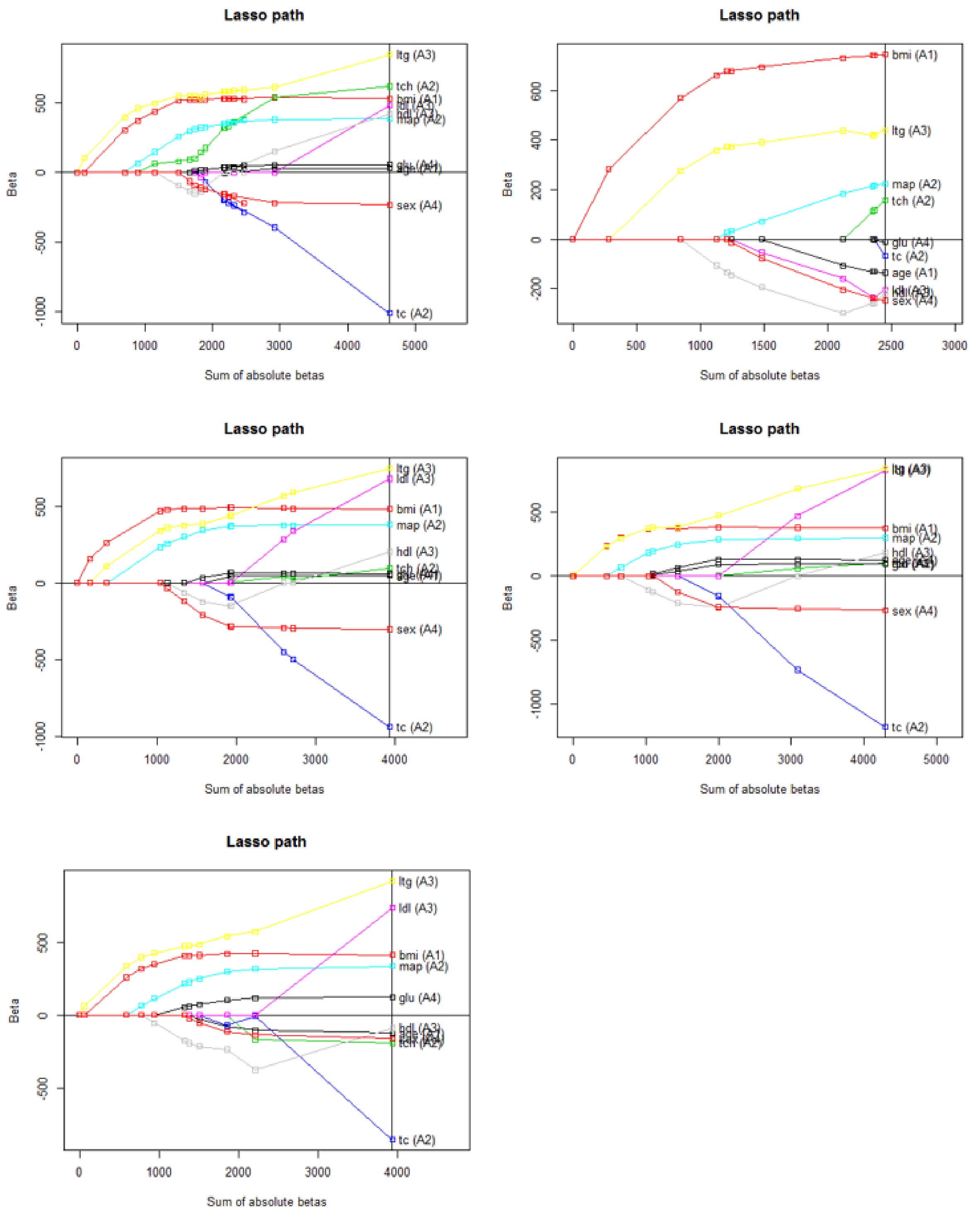

Figure 3.4: Diabetes Example: Lasso paths for each Training set of size 48

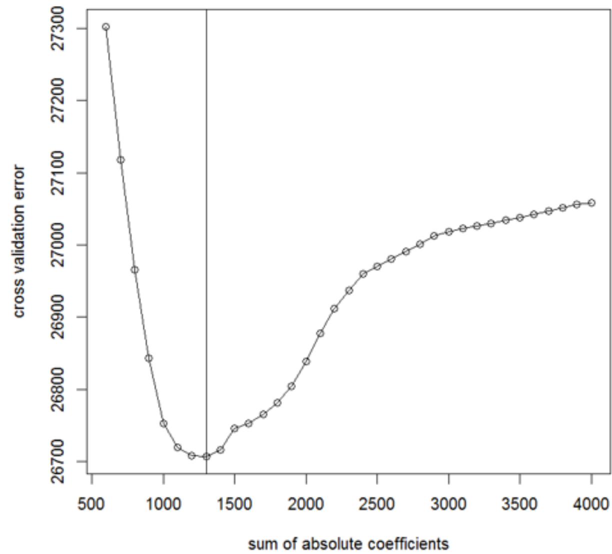

Figure 3.5: Diabetes Example: Mean prediction errors for each value of $\sum_{i=1}^{10} \hat{\beta}_i$

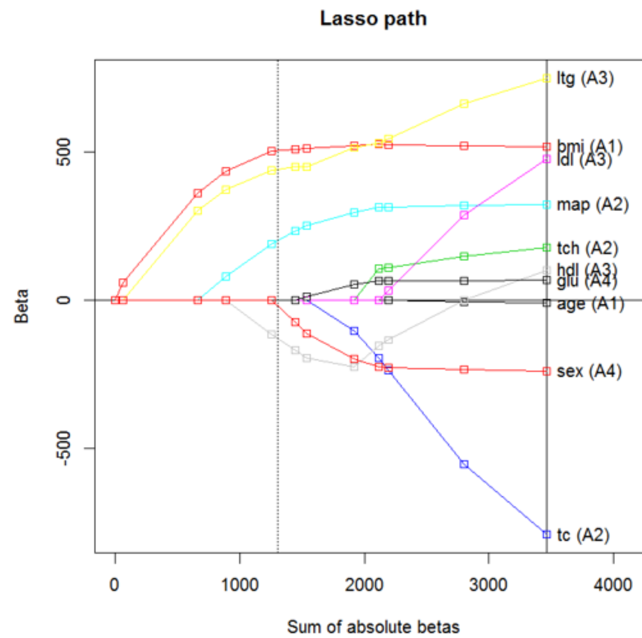

Figure 3.6: Diabetes Example: Lasso path for entire data set (Vertical line at $\sum_{i=1}^{10} \hat{\beta}_i = 1300$)

3.7 Running Time and Error

3.7.1 Dissecting the Secure Matrix Multiplication Algorithm

Some factors that may concern researchers in any kind of secure protocol are the amount of additional computational time and the accuracy of the results produced,

relative to the original algorithm without any secure measure. In the case of our secure Lasso protocol for vertically partitioned data, these entirely depend on the step where each party applies secure matrix multiplication with other parties to obtain $X^T X$ or $X^T y$ (we will show this later on). As we have discussed before, the parties can carry on with the rest of the algorithm individually once the multiplication results are shared. Hence whatever additional computation time or error from the secure protocol stems from secure matrix multiplication. Here we look at how much computation the secure matrix multiplication procedure requires under different scenarios.

Suppose parties A and B wish to calculate $X_A^T X_B$ where the matrix sizes are $n \times p_A$ and $n \times p_B$. If one of the parties, say party B, owns the response y we can assume it attached to the last column of X_B. The crucial step of secure matrix multiplication is where party A generates g orthogonal vectors $Z_1, \ldots Z_g$ that satisfy $(X_A)_i^T Z_j = 0$ for all i and j. We will assume that $g = \frac{p_A}{p_A + p_B} n$ which is the optimal choice of g when we want the loss of protection to be shared equally among the two parties, as discussed in Karr et al. (2009). The running time almost entirely depends on the method that A uses to produce $Z = [Z_1, \ldots Z_g]$ (we will explain this in detail later on). The accuracy depends on whether $X_A^T Z = 0$ holds exactly. To produce Z, Karr et al. (2009) recommend using the QR-decomposition of X_A, which is given by:

$$X_A = \mathbf{QR}$$

where $\mathbf{Q}$ is an $(n \times n)$ orthonormal matrix and $\mathbf{R}$ is an $(n \times p_A)$ matrix. $\mathbf{Q}$ can be decomposed as

$$\mathbf{Q} = [\, \mathbf{Q_1} \, \mathbf{Q_2} \,]$$

where $\mathbf{Q_1}$ consists of the leftmost p_A columns of $\mathbf{Q}$ and is the orthogonal basis for the range of X_A. $\mathbf{Q_2}$ consists of the rightmost $(n - P_A)$ columns of $\mathbf{Q}$ and is the orthogonal basis for the null space of X_A. The set of vectors Z is obtained by selecting g columns from $\mathbf{Q_2}$.

3.7.2 Algorithms for QR Decomposition

There are two popular algorithms to construct a QR decomposition, one based on *Householder transformations*, and another based on *Givens rotations*. Both algorithms are known to be fast and numerically accurate. For our simulations we used the one based on Householder transformations, mainly because it is easier to make general statements about its performance. However, it is worth noting that the algorithm based on Givens rotations will be faster than the alternative when X_A has a large number of zero components, and is also more easily parallelized. For purposes of computing the matrix Z, we can slightly simplify both algorithms since we do not need the $\mathbf{R}$ part of the QR decomposition, but the gain in efficiency is negligible. For a description of the algorithm based on Householder transformations, see for example, Businger and Golub (1965).

3.7.3 Computational Complexity and Memory Usage

We should warn the user that since secure matrix multiplication uses the QR decomposition of X_A, at one point the algorithm will have to allocate an $(n \times n)$ matrix $\mathbf{Q}$, no matter how many columns (p_a) in X_A. This can become a problem if the user's computing environment has limited memory space. For example, suppose a user wants to securely calculate $X_A^T X_B$ where $n = 50000$, $p_A = 3$, $p_B = 4$. In practice, data frames of such dimensions are at most considered moderately sized, and one would think that it can be easily handled on any personal computer. However, the matrix $\mathbf{Q}$ produced in the algorithm will have dimension (50000×50000), which has the same number of elements, for instance, as a matrix with 25 million rows and 100 columns. For the commonly used statistical software **R**, this takes up roughly 18.6 gigabytes of memory space. Continuing with this example, party A will then choose g rows where

$$g = \frac{p_A}{p_A + p_B} n = \frac{3}{3+4} \cdot 50000 \approx 21400$$

from the matrix $\mathbf{Q}$ to obtain the (50000×21400) matrix Z. Party A will then also calculate ZZ^T, which also requires a large number of operations that produces another (50000×50000) matrix, before finally sending the result to party B. What we wanted to illustrate through this example is that secure matrix multiplication can require a lot of memory space and computing time even when the data itself

Procedure	Party	Dimension of input(s)	Total number of operations	Dimension of output
$X_A = Q_A R_A$ (via QR Householder)	A	$n \times p_A$	$2np_A^2 - 2p_A^3/3$	$n \times n$
$Z\,Z^T$	A(B)	$n \times g$	$O(n^3 p_A/(p_A + p_B))$	$n \times n$
$I - ZZ^T$	A(B)	$n \times n$	n	$n \times n$
$(I - ZZ^T)\,X_B$	B	$n \times n$ and $n \times p_B$	$np_B(2n-1)$	$n \times p_B$
$X_A\,[(I - ZZ^T)\,X_B]$	A	$n \times p_A$ and $n \times p_B$	$p_A p_B(2n-1)$	$p_A \times p_B$

Table 3.2: Breakdown of the Number of Arithmetic Operations in Secure Matrix Multiplication

is not too large[7].

In Table 3.2 we provide a breakdown of the number of arithmetic operations throughout the entire process of secure matrix multiplication. Again we assume the general scenario where parties A and B wish to calculate $X_A^T X_B$, and we take $g = p_A/(p_A + p_B)$. The column "Procedure" indicates which part of the secure multiplication is being performed, and the column "Party" tells which of the two parties handles that procedure. "A(B)" means that in terms of security it does not matter which party handles the corresponding procedure, it would not change the loss of protection for either side.

There are a few things which we should take note from 3.2. First of all, both parties A and B will have to deal with matrices with the size of $n \times n$ at some point so they will need a computing environment that can handle that. Secondly, in terms of the number of arithmetic operations, the dominant procedure is in where either party A or B computes ZZ^T, where the order is $O(n^3 p_A/(p_A + p_B))$. Note that according to Efron et al. (2004), the LARS algorithm with $n > p$ requires $O(p^2 n + p^3)$ operations. Hence for the secure Lasso algorithm when $n > p$ (as we have assumed so far), the secure matrix multiplication part dominates the rest of the algorithm in terms of computation time. In the next section we provide some simulation results to get an idea on how much time it takes to run secure matrix multiplication.

[7]Note that the $(n \times n)$ matrix discussed here is produced when we apply a straightforward implementation of the Householder algorithm. However, for a reasonably small value of g, there may be ways to modify the algorithm to do similar computations without having to use such a large matrix, given that the rank of X_A is less than n.

3.7.4 Simulation

We consider computing the product $X_A^T X_B$ under the following scenarios:

- n = 200, 500, 1000, 2000, 5000, 10000, and 20000
- p_A = 5, 10, 20, 50, 100, and 200
- $p_B = 10$ (without loss of generality, since the value of p_B has little contribution to the total computational cost (see 3.2).)

Under each scenario, X_A and X_B were filled with independent values sampled from a uniform distribution, and the total computing time it took to securely compute $X_A^T X_B$ were recorded. The results are provided in Table 3.3, in seconds. The rows correspond to the value of p_A, and the columns correspond to n, so for instance, it took roughly 4800 seconds or 1 hour and 20 minutes to compute $X_A^T X_B$ when $n = 20000$, $p_A = 10$, and $p_B = 10$. When we run the LARS algorithm under the same scenario with two parties but *without* using secure matrix multiplication, we get the results almost immediately - this is consistent with our previous statement that secure matrix multiplication is the dominant factor in terms of computing time.

	500	1000	2000	5000	10000	20000
5	0.063	0.371	2.646	47.019	371.637	3256.21
10	0.115	0.580	4.667	73.110	556.704	4801.01
20	0.172	0.938	7.125	100.310	755.96	6428.16
50	0.371	1.574	9.937	135.747	1044.01	8183.06
100	0.751	2.714	14.620	165.668	1103.26	9200.98
200	1.693	5.941	23.237	222.639	1318.54	11474.7

Table 3.3: Computing Times of $X_A^T X_B$ in Seconds, for Different Values of p_A (rows) and n (columns)

Thus far we have only been considering the total computing time between two parties but how about if there were more than two parties involved? This will depend on the operational protocol for computing $X^T X$, i.e. the order in which the parties calculate the product $X_{A_i}^T X_{A_j}$ for $i \neq j$. This does not matter at all from a security standpoint, but it will not hurt to make the procedure as efficient as possible. Let's suppose parties $A_1, \ldots A_K$ wish to compute the full correlation

matrix $X^T X$.

Perhaps one efficient protocol for doing this is:

1. Each party A_i $(i = 1, \dots K)$ computes $X_{A_i}^T X_{A_i}$ locally.
2. For $i = 1, \dots K-1$, party A_i performs QR decompostion on X_{A_i} and sends g_j vectors $z_1, \dots z_{g_j}$ to each party A_j where $j = i+1, \dots, K$.
3. Each party A_j computes $(I - ZZ^T)X_j$ and sends it back to the party which they received the vectors in Z.
4. Each party A_i $(i = 1, \dots K-1)$completes the secure matrix multiplication and shares the results with every other party to obtain $X^T X$.

Simulation for Different Partitions

Following the above protocol, we performed another simulation to look at the computing times to calculate $X^T X$ under different partitions. In all situations we have $n = 10000$, and $p = 8$, meaning that all partitions add up to 8. For example, partition $(2, 2, 4)$ means that there are three parties A_1, A_2, and A_3 where the parties hold 2,2, and 4 variables, respectively. The results are summarized in Table 3.4. What we can first observe from the table is that the total computing time increases dramatically as the number of partitions increases, e.g. it takes roughly 2.25 hours to compute $X^T X$ for partition $(2, 1, 1, 1, 1, 2)$ but around 3.4 hours for partition (2,1,1,1,1,1,1). Note that here we are not considering any loss in utility that arises from the communication cost between the parties. In a real life situation, having more parties will require more communication in addition to the increase in computing times. Another observation we can make from Table 3.4 is that in general, the computing time is faster when the last party A_K holds many variables. This makes sense since in our protocol above, A_K does not perform any QR decomposition.

3.7.5 Accuracy of the Secure Protocol

The accuracy of the secure LARS algorithm, i.e., whether it produces the same results as in the non-secure case, entirely depends on whether secure matrix multiplication can produce the same results as usual matrix multiplication. This in turn depends on whether the quality of the QR decomposition of X_A, i.e., whether

Partition	Time	Partition	Time
(2,6)	285.595	(2,1,1,4)	2923.894
(3,5)	427.553	(2,1,2,3)	2999.342
(4,4)	570.346	(2,2,2,2)	3232.501
(2,1,5)	1260.703	(2,1,1,1,3)	5006.531
(2,2,4)	1314.648	(2,1,1,2,2)	5224.128
(2,3,3)	1463.308	(2,1,1,1,1,2)	8158.937
(3,1,4)	1559.314	(3,1,1,1,1,1)	9503.753
		(2,1,1,1,1,1,1)	12359.812

Table 3.4: Computing Time in Seconds, Under Different Partitions when $n = 10000$ and $p = 8$

the algorithm can produce a $\mathbf{Q}$ that contains columns that are close to being orthogonal to X_A as possible. If the columns in $\mathbf{Q}$ are only close to being orthogonal, it may affect the product of $X_A^T X_B$ itself and ultimately the entire Lasso path.

When we conducted the simulation experiment in Section 3.7.4, we also compared the results from the secure protocol to the results from usual matrix multiplication. For each element in the matrices, we recorded the absolute relative error defined as:

$$e_{i,j} = \left| \frac{x_{i,j} - \hat{x}_{i,j}}{x_{i,j}} \right|, \tag{3.35}$$

where $x_{i,j}$ is the (i,j)th element of $X_A^T X_B$ computed normally and $\hat{x}_{i,j}$ is the (i,j)th element of $X_A^T X_B$ computed securely. A complete summary of the maximum absolute errors, defined as the maximum value of $e_{i,j}$ among all $i = 1, \ldots, p_A$ and $j = 1, \ldots, p_B$ in each scenario is provided in Table 3.5. We can see that in every case the errors are extremely small, though there is some tendency for the errors to grow larger along with the dimensions of the matrices. Our conclusion here is that secure matrix multiplication will provide very accurate results as long as a reliable algorithm for QR decomposition such as the Householder method or Givens rotation method is used, and this will let the entire secure Lasso procedure provide accurate results as well.

3.8 Extensions to Generalized Linear Models

In their discussion of the Lars algorithm, Madigan and Ridgeway (2004) briefly mention one way to extend a LARS-type algorithm to logistic regression, using a

	500	1000	2000	5000	10000	20000
5	1.40042e-14	1.83723e-14	9.01718e-15	5.94185e-15	1.59099e-14	3.59016e-14
10	1.99712e-14	4.37232e-14	1.79257e-13	5.39473e-13	4.80661e-13	1.77894e-12
20	2.1019e-13	2.51346e-13	4.52951e-13	6.09989e-13	8.91064e-13	1.52471e-12
50	1.88323e-12	2.9712e-13	2.2382e-13	9.45871e-13	1.57692e-12	7.18132e-13
100	7.0009e-13	4.93034e-13	6.36274e-13	2.64608e-12	1.09933e-12	2.18584e-12
200	5.98019e-12	9.28439e-13	2.34969e-11	1.00633e-11	4.4112e-11	2.28193e-12

Table 3.5: Maximum Absolute Error of $X_A^T X_B$, for Different Values of p_A (rows) and n (columns)

form of linear approximation. Here we outline their method while showing how it can be applied in the vertically partitioned setting in a secure way.

Consider the logistic log-likelihood for a linear mean function $\mu(\mathbf{x})$:

$$l(\mu(\mathbf{x})) \;=\; \sum_{i=1}^{n} \left[\, y_i\, \mu(\mathbf{x}_i) \;-\; \log(1+\exp(\mu(\mathbf{x}_i)))\,\right] \tag{3.36}$$

where $\mu(\mathbf{x}_i)$ is the current estimate of the mean:

$$\mu(\mathbf{x}_i) \;=\; \beta_0 + \beta_{\mathcal{A}_1}\mathbf{x}_{i,\mathcal{A}_1} +, \ldots, +\beta_{\mathcal{A}_k}\mathbf{x}_{i,\mathcal{A}_M}$$

and each predictor $x_{\mathcal{A}_m}$ is in the active set $\mathcal{A}$. We can initialize as $\mu(\mathbf{x}) = \log(\bar{y}/(1-\bar{y}))$ if the response holder agrees to share $\bar{y}$. In the vertical setting, $\mu(\mathbf{x})$ in each step of the algorithm can be obtained using secure summation, but it should not be calculated when there are only two parties in the active set. For some γ, we want to find a new predictor $x_j \in \mathcal{A}^c$ that increases the Logistic log-likelihood $l(\mu(\mathbf{x}) + x_j^T\gamma)$ the most. To find such a $\mathbf{x}_j$, the parties can compute the fractional derivative for each of their variables j and find the maximum:

$$\begin{aligned} j^* \;&=\; \arg\max_j \left| \frac{d}{d\,\gamma}\, l(\mu(\mathbf{x}) + x_j^T\gamma) \right|_{\gamma=0} \\ &=\; \arg\max_j \left| \mathbf{x}_j^T \left(y \;-\; \frac{1}{1+\exp(-\mu(\mathbf{x}))} \right) \right| \\ &=\; \arg\max_j \left| \mathbf{x}_j^T\, y \;-\; \mathbf{x}_j^T\,(1+\exp(-\mu(\mathbf{x}))^{-1} \right| \end{aligned}$$

Note that there is a slight abuse of notation here, $(1+\exp(-\mu(\mathbf{x}))^{-1}$ indicates a vector of scalar inverses. As before, the parties would compute $\mathbf{x}_j^T\, y$ using secure summation with the party that holds y and compute $\mathbf{x}_j^T\,(1+\exp(-\mu(\mathbf{x})))^{-1}$ on

their own (recall that $\mu(\mathbf{x})$ is obtained by secure summation). They will then compare their own maximum absolute values with other parties to find the maximizing index j^*. Now we want to increase γ while it satisfies:

$$(s_{j^*}\mathbf{x}_{j^*} - s_j\mathbf{x}_j)^T \left(y - \frac{1}{1+\exp(-\mu(\mathbf{x}) - \mathbf{x}_j^T\gamma)} \right) \geq 0 \tag{3.37}$$

for all $j \in \mathcal{A}^c$. Note that s_j indicates the sign of the correlation between x_j and the current residual as in the normal LARS case. Increasing γ while maintaining the constraint in (3.37) would involve a nonlinear optimization. One way to get around this is to apply a linear approximation to the fraction in the second term of (3.37). Specifically, we would apply a Taylor expansion around $\gamma = 0$ and get:

$$\begin{aligned}
\frac{1}{1+\exp(-\mu(\mathbf{x}) - \mathbf{x}_j^T\gamma)} &\approx \frac{1}{1+\exp(-\mu(\mathbf{x}))} + \frac{\gamma\exp(-\mu(\mathbf{x}))}{(1+\exp(-\mu(\mathbf{x})))^2} \\
&= \frac{1}{1+\exp(-\mu(\mathbf{x}))} + \gamma\frac{1}{1+\exp(-\mu(\mathbf{x}))}\frac{\exp(-\mu(\mathbf{x}))}{(1+\exp(-\mu(\mathbf{x})))} \\
&= p(\mathbf{x}) + \gamma\, p(\mathbf{x})\,(1-p(\mathbf{x}))
\end{aligned}$$

Now from (3.37) and the approximation above, we can find $\hat{\gamma}$ which is:

$$\hat{\gamma} = \min_{k\in\mathcal{A}^c}^{+} \left[\frac{(s_{j^*}\mathbf{x}_{j^*} - s_k\mathbf{x}_k)^T\,(y-p(\mathbf{x}))}{(s_{j^*}\mathbf{x}_j - s_k\mathbf{x}_k)^T\,p(\mathbf{x})(1-p(\mathbf{x}))\mathbf{x}_{j^*}} \right]. \tag{3.38}$$

Note that the quantity in (3.38) can be obtained with the combination of secure summation and secure matrix multiplication. For example, observe that:

$$(s_{j^*}\mathbf{x}_{j^*} - s_k\mathbf{x}_k)^T\,(y-p(\mathbf{x})) = s_{j^*}\mathbf{x}_{j^*}^T y - s_{j^*}\mathbf{x}_{j^*}^T p(\mathbf{x}) - s_k\mathbf{x}_k^T y + s_k\mathbf{x}_k^T p(\mathbf{x}) \tag{3.39}$$

If it is safe to calculate $p(\mathbf{x})$ explicitly (i.e. there are more than two parties involved in the active set), $p(\mathbf{x})$ will be shared so the second and fourth terms can be computed locally within each party. Otherwise those terms will require secure multiplication. The calculation of the denominator is similar. As in the normal LARS case, in the end the parties would end up sharing X^TX and X^Ty so it may be better to calculate those quantities in the beginning in order to save communication cost.

The loss of information incurred by this method is the same as in the normal LARS case. One concern is that a linear approximation is used which may affect the estimates, but Madigan and Ridgeway (2004) show via simulation that the method (without secure protocol) can produce fairly good results. Hence we can say the same for the secure version as long as the secure matrix multiplication produces accurate values, and we have seen in Section 3.7.5 that this is indeed the case. We believe that the secure LARS algorithm can be applied to other models in the class of GLMs but for now leave that as future work.

3.9 Discussion and Conclusion

3.9.1 Ridge Regression

Sometimes researchers might think it would be better to perform ridge regression over Lasso, especially when it is believed that the appropriate model would have a large number of small effects. When the penalty parameter λ is specified, performing ridge regression using secure matrix multiplication and secure summation is straightforward since the ridge estimator $\hat{\beta}_{Ridge}$ has the closed form solution:

$$\hat{\beta}_{Ridge} = (X^T X + \lambda I)^{-1} X^T y \tag{3.40}$$

hence $X^T X$ and $X^T y$ can be calculated securely and each party can perform the calculation in (3.40). Some difficulty lies in choosing the right value of λ. In a non-distributed setting a researcher would use cross validation, but this requires searching through a grid of λ values. Usually this grid is decided based on some trial and error, and some information the researcher has about the covariates. This would be difficult in the vertically partitioned setting where each party holds a separate set of covariates; the parties would have no means to guess candidate values for λ. Perhaps one solution to this is to use sample-based estimates of λ such as the modified HKB estimator based on works of Hoerl and Kennard (1981) and Thisted (1976)

$$\lambda_{HKB} = \frac{(p-2)\,\hat{\sigma}^2}{\hat{\beta}^T \hat{\beta}} \tag{3.41}$$

which can be computed using secure summation and multiplication. The parties can use these estimates of λ itself, or a guideline in choosing a range of values of λ to search over in cross validation.

3.9.2 Complex Partitions

In this chapter we have only considered pure vertical cases in performing the LARS algorithm securely. We have not considered pure horizontal cases, but application to this is relatively straightforward. In a real world situation the partitioning of data across different parties is likely to be more complex. One way of dealing with such situations is to view the data as an incomplete dataset, using missing value techniques to fill in some spots in order to apply protocols for pure cases. Reiter et al. (2004) and Fienberg et al. (2007) propose a secure EM algorithm based approach to perform linear or logistic regression on vertically partitioned, partially overlapping data. It is natural to think that we can apply it to our secure Lasso protocol as well, but for now we will leave it as future work.

3.9.3 Alternative Methods for Matrix Multiplication

Throughout this chapter we have suggested the method by Karr et al. (2009) to be used as the means for secure matrix multiplication. All the discussions in this chapter have been based on this. However, in Chapter 2 we have seen that there are other secure ways to compute $X^T X$ or $X^T y$. For example, there is the method based on 1 out of m oblivious transfer by Lin and Karr (2010), which might provide a higher degree of security if most of the parties' data are categorical variables. Random shares (Hall et al. (2011), Nardi et al. (2012)) can also be used to calculate $X^T X$ and $X^T y$, and perhaps almost any of the intermediate quantities in our protocol. The drawbacks for both of these methods is the immense amount of communication and computational cost, but it may be worth keeping in mind as there may be users who are willing to incur these costs in the light of more security.

Chapter 4 Concluding Remarks

4.1 Summary and Conclusions

In this thesis we first provided an in-depth analysis on the disclosure risks of secure protocols based on existing SMPC techniques (Chapter 2), and then proposed an application of these techniques to Lasso, a popular penalized regression method (Chapter 3). Secure protocols for penalized regression methods in the distributed dataset setting has not yet been fully explored in previous statistical disclosure literature at the time of this writing.

In Chapter 2 we looked at three different methods that aim to perform logistic regression under the vertically partitioned database setting. For each method we considered the case where only two parties are involved, since it can be considered the most "risky" case; it is similar to where there are 3 or more parties involved but every other party cooperates to expose one particular party's data. While the three methods can all be considered secure in the sense that there is no danger for either parties' data to be fully exposed as long as the protocols are carried out carefully and that certain conditions hold, there is some information leakage that happens between the two parties. The method by Nardi et al. (2012) causes the least amount of information leakage, but the computation and communication cost to carry out the protocol is immense. The types of information leakage that occur in Karr et al. (2009) and Lin and Karr (2010) are quite different, and one may be preferable to the other depending on the situation (Section 2.6). In all three methods parties may build estimates of other party's data based on the estimated coefficients but there is no way to avoid this, hence it is a risk that the parties must accept.

In Chapter 3 we proposed a protocol for conducting statistical analysis under the vertically partitioned data setting for Lasso, based on the LARS algorithm. While the methods we used were similar to the ones in Karr et al. (2009) discussed above, the other two methods can be applied to our protocol as well. As we have done in Chapter 2, we have shown that if the protocol is carried out properly and some conditions such as $n > p$ are met, the protocol is safe even when there are only two parties involved, in the sense that each party will not be able to reveal another party's data. Our protocol does also work when $n < p$, but the algorithm will terminate when $n - 1$ variables are in the model and there will be serious security issues unless n is fairly close to p. Though we have assumed that the parties do not collude with each other to reveal one party's data, violating this would only cause problems when performing diagnostics. We have demonstrated that the algorithm will produce accurate results but the computational cost is higher than what the user might expect from the data dimensions, especially when there is a large number of parties involved. Finally we outlined an approach in expanding our protocol to Logistic regression and possibly other models in the class of GLMs.

4.2 Future Work

In Chapter 2 we analyzed disclosure risks by only looking at the secure protocols themselves, but there may be more findings if we use real life data and attempt a simulation from one party's viewpoint, trying to expose the other party's data under the situations which we considered dangerous. In Chapter 3 we have mainly focused on conducting Lasso in the purely vertical setting, but as a next step we can attempt to apply a secure EM algorithm based approach such as that of Reiter et al. (2004) or Fienberg et al. (2007) to perform analyses on more complex partitions. From a security viewpoint it may be worth designing a protocol where $X^T X$ and $X^T y$ don't have to be shared, like Hall et al. (2011) or Nardi et al. (2012) have done for linear and logistic regression, respectively. We have assumed that there is a global identifier that links all the records; in real life situations this is often not the case: an investigation on how our method conforms with record linkage methods may also be an interesting direction for future research.

Appendix

1 Code for Secure Matrix Multiplication

```
#################################
# Secure Matrix Multiplication  #
#################################

## QR Decomposition using Householder Transformations
# Some simple functions follow:

## create (1,0,0,....)^T
efunc = function(n){
  ee1 = matrix(0,n,1)
  ee1[1] = 1
  return(ee1)
}

rfunc = function(x1){
  x1=as.matrix(x1)
  r11 = -sign(x1[1,1])*sqrt(t(x1)%*%x1)
  return(as.numeric(r11))
}

L2func = function(x1,r1){
  x1=as.matrix(x1)
  ress = sqrt(t(x1)%*%x1 +r1^2 - 2*r1*x1[1,1])
  return(as.numeric(ress))
}

wfunc = function(x1,r1){
  x1 = as.matrix(x1)
  w1 = (x1-r1*efunc(dim(x1)[1]))/L2func(x1,r1)
  return(as.matrix(w1))
```

```
}

Hfunc = function(w1){
  w1 = as.matrix(w1)
  lent = dim(w1)[1]
  H1 = -2*w1%*%t(w1)
  for (m in 1:lent)   H1[m,m] = H1[m,m]+1
  return(H1)
}

HX_func = function(X1,w1){
  # Householder Transformation
  X1 = as.matrix(X1)
  w1 = as.matrix(w1)
  HX1 = X1 - w1%*%(2*t(t(X1)%*%w1))
  return(as.matrix(HX1))
}


########################################
## Main function for QR decomposition ##
## Returns QR decomp of input X       ##
## Uses functions defined above       ##
########################################
Q_Rfunc = function(X){
  X = as.matrix(X)
  n1 = dim(X)[1]
  p1 = dim(X)[2]
  ws = vector("list",p1)
  r11 = rfunc(X[,1])
  w1 = wfunc(X[,1],r11)
  ws[[1]] = w1
  H1 = Hfunc(w1)
  HX1 = HX_func(X,w1)
  if(p1>1){
    Xn1 = HX1[2:n1,2:p1]
    for (i in 2:p1){
      Xn1 = as.matrix(Xn1)
      xn1 = as.matrix(Xn1[,1])
      r11 = rfunc(xn1)
      w1 = wfunc(xn1,r11)
      w2 = matrix(0,n1,1)
      w2[i:n1,] = w1
      ws[[i]] = w2
      if (i<p1){
```

```
        temp = matrix(NA,n1,p1)
        temp[1:(i-1),] = HX1[1:(i-1),]
        temp[i:n1,] = HX_func(HX1[i:n1,],w1)
        HX1 = temp
        Xn1 = HX1[(i+1):n1,(i+1):p1]
      }
    }
    QT = H1
    for (j in 2:p1){
      QT = HX_func(QT,ws[[j]])
    }
  }else{
    QT=H1
  }
  return(t(QT))
}


## Find 'g' orthogonal vectors from QR decomposition
findorths = function(x,g){
  g1 = g
  if (g > (dim(x)[1]-dim(x)[2]))  g1 = dim(x)[1]-dim(x)[2]
  Q1 = Q_Rfunc(x)
  st1 = dim(x)[1]-g1+1
  en1 = dim(x)[1]
  Q1 = Q1[,st1:en1]
  return(Q1)
}


################################################################
## Function that performs secure matrix multiplication        ##
## Computes t(x)%*%y securely, using functions defined above. ##
## If secure == FALSE, performs normal matrix multiplication  ##
################################################################
sec_prod = function(x,y, secure = FALSE){
   if (secure == TRUE){
          x1 = t(as.matrix(x))
          y1 = as.matrix(y)
          NN = nrow(x1)
          pa = ncol(x1)
          pb = ncol(y1)
          g1 = trunc(pa*NN/(pa+pb))

          Zs = findorths(x1,g1)
```

```
          dddd2 = -Zs%*%t(Zs)
          for (l in 1:dim(dd2)[1])   dddd2[l,l] = 1 + dddd2[l,l]
          result1 = t(x1)%*%(dddd2%*%y)
          return(result1)
   }
   else{
          x1 = as.matrix(x)
          y1 = as.matrix(y)
          return(x1%*%y1)
   }
}
```

2 Code for Example in Section 3.6.3

```
####################
# Diabetes Example #
####################

################## Prepare data ###################
load("diabetes.Rdata")
attach(diabetes)

### convert diabetes data into dataframe
temp1 = x
resp1 = y
detach(diabetes)
da_ta = matrix(0,442,11)
da_ta[,1] = resp1
   for(i in 2:11) da_ta[,i] = temp1[,(i-1)]
#da_ta = da_ta[1:7,]
da_ta = as.data.frame(da_ta)
names(da_ta) = c("Y",dimnames(temp1)[[2]])
names(da_ta)


##########################################################
## Function for secure summation (not really necessary) ##
##########################################################
sec_sum = function(x,randomorder=TRUE,secure =TRUE){
 if (security == TRUE){
    if (is.list(x)==TRUE){
        if (randomorder==FALSE) index3 = 1:length(x)
        else index3 = sample(1:length(x),length(x))
```

```
        nelements = dim(x[[1]])[1]*dim(x[[1]])[2]
        m = 2^10
        R = matrix(sample(1:m,nelements, replace=TRUE),
                    nrow = dim(x[[1]])[1], ncol=dim(x[[1]])[2])

        s1 = (R + as.matrix(x[[index3[1]]]))

        for (i in 2:length(x)){
              s1 = (s1+ as.matrix(x[[index3[i]]]))
        }
    }
    else if ((is.list(x)==FALSE)&&(is.vector(x)==TRUE)){
           if (randomorder==FALSE) index3 = 1:length(x)
          else index3 = sample(1:length(x),length(x))
          nelements1 = length(x)
             m = 2^10
             R = sample(1:m,1)
             s1 = (R + x[index3[1]])
             for (i in 2:length(x)){
                 s1 = (s1 + x[index3[i]])
             }
    }
        return((s1-R))
 }
 else {
   if (is.list(x)==TRUE){
       sum1 = Reduce('+', x)
   }
   else {
      sum1 = sum(x)
   }
 return(sum1)
 }
}


## Split data vertically, into "int1" equal pieces
split_ver=function(data, seed=NULL, int1, response=1,rand=FALSE) {
    if (!is.null(seed)) set.seed(11111)
    lis1=vector("list",int1)
    lis2=vector("list",int1)
    quo=(dim(data)[2]-1)%/%int1
    rem=(dim(data)[2]-1)%%int1
       if (response==1){
```

```
            index1=2:dim(data)[2]
        }else if (response==dim(data)[2]){
            index1 = 1:(dim(data)[2]-1)
        }else{
            index1 = c(1:(response-1),(response+1):(dim(data)[2]))
        }

    if (rand==FALSE){
        index2 = index1
    }else{
        index2=sample(index1,length(index1))
    }

    if (rem==0) {
       for (i in 1:int1){
       lis1[[i]]=index2[((i-1)*quo+1):(i*quo)]
       }
    }else {
       for (i in 1:int1){
          if (i<(rem+1)){
          lis1[[i]]=index2[((i-1)*(quo+1)+1):(i*(quo+1))]
          }else{
          lis1[[i]]=index2[(rem+(i-1)*quo+1):(rem+i*quo)]
          }
       }
       if(rand!=FALSE){
          cc=lis1[[1]]
          lis1[[1]]=lis1[[(rem+1)]]
          lis1[[(rem+1)]]=cc
       }
    }
        lis1[[1]]=c(response,lis1[[1]])
          for (k in 1:int1){
          lis2[[k]]=as.data.frame(data[,lis1[[k]]])
          names(lis2[[k]])= names(data[lis1[[k]]])
          }
    return(lis2)
}


## (horizontal) split data into 'int1' equal pieces
split_hor=function(data, seed=NULL, int1) {
  if (!is.null(seed)) set.seed(11111)
  index1=sample(1:nrow(data),nrow(data))
  lis1=vector("list",int1)
```

```
  lis2=vector("list",int1)
  quo=nrow(data)%/%int1
  rem=nrow(data)%%int1
  if (rem==0) {
    for (i in 1:int1){
      lis1[[i]]=index1[((i-1)*quo+1):(i*quo)]
    }
  }
  else {
    for (i in 1:int1){
      if (i<(rem+1)){
        lis1[[i]]=index1[((i-1)*(quo+1)+1):(i*(quo+1))]
      }
      else{
        lis1[[i]]=index1[(rem+(i-1)*quo+1):(rem+i*quo)]
      }
    }
  }
  for (k in 1:int1){
    lis2[[k]]=data[lis1[[k]],]
  }
  return(lis2)
}

## Function for equation (3.9)
find_gam = function(A_A,a_a,Ac,Chat,chat){
if (length(Ac)==0){
gam = Chat/A_A
ind2 = NULL
}else{
for (i in 1:length(Ac)){
ind1 = Ac[i]
t1 = (Chat-chat[ind1])/(A_A-a_a[ind1])
t2 = (Chat+chat[ind1])/(A_A+a_a[ind1])
if (t1>0 & t2>0){
t3=min(t1,t2)
}else if (t1<0&t2>0){
t3=t2
}else if (t2<0 & t1>0){
t3=t1
}else{
t3=1/0
}

if(i==1){
```

```
ind2=ind1
gam=t3
}else if(t3<gam){
ind2=ind1
gam=t3
}
}
}
return(c(gam,ind2))
}

## Function for equation (3.14)
gamfunc2 = function(gamm,aset){
ind2 = NULL
gam2 = 1/0
for (i in 1:length(gamm)){
if(gamm[i]>0 && gamm[i]<gam2){
gam2 = gamm[i]
ind2 = aset[i]
}
}
return(c(gam2,ind2))
}


########################################
## Main LARS function, for K parties  ##
########################################

ver_lasso = function(da_ta, K, seed=NULL,cross ="OFF", xTx1=NULL){
# Ignore last parameter (xTx1=NULL)

security1 = FALSE     # secure summation
security2 = FALSE    # secure matrix multiplication
spdata = split_ver(da_ta,seed,K,1,rand=FALSE)
# split into K parts vertically, response is in column 1

sapply(spdata,names)  # check which party has which variables
name_set = c()
nlength = 0
Y = spdata[[1]][,1]
spdata[[1]] = spdata[[1]][,2:length(spdata[[1]])]
X1 = spdata[[1]]
for (i in 2:K) X1 = cbind(X1,spdata[[i]])
```

```
for (i in 1:K){
name_set = c(name_set, names(spdata[[i]]))
nlength = nlength + length(spdata[[i]])
}
name_set = name_set[1:nlength]
p = length(name_set)
index_set = vector("list",K)
st1=1
for (i in 1:K){
index_set[[i]]=st1:(st1+length(spdata[[i]])-1)
st1 = st1+length(spdata[[i]])
}

## start protocol

### calculate entire XTX in the beginning
if (cross=="OFF"){
  xTx = matrix(NA,p,p,dimnames=list(name_set,name_set))
  start1=1
  start2=1
for (k in 1:K){ # row party
for (l in k:K){ # column party
rpl = dim(spdata[[k]])[2]
cpl = dim(spdata[[l]])[2]
if (k==l){
securityA = "OFF" # local computation
tempp = sec_prod(t(spdata[[k]]),spdata[[l]],securityA)
xTx[start1:(start1+rpl-1),start2:(start2+cpl-1)] = tempp
}else{
securityA = security2 # secure multiplication
tempp = sec_prod(t(spdata[[k]]),spdata[[l]],securityA)
xTx[start1:(start1+rpl-1),start2:(start2+cpl-1)] = tempp
xTx[start2:(start2+cpl-1),start1:(start1+rpl-1)] = t(tempp)
}
start2 = start2 + cpl
}
  start1 = start1 + rpl
  start2 = start1
}
}else{
   xTx = xTx1
  }
## first iteration
coefs = matrix(0,1,p,dimnames=list(c(),name_set))
 # matrix of Lasso path (move rows)
```

```
f_set = 1:length(name_set)        # full set

# Each party computes initial c_hat
# using secure multiplication with y
temp1 = sapply(spdata,function(x) sec_prod(t(x),Y,security2))
c_hat = c()
if (!is.list(temp1)){
  for (i in 1:K) c_hat = c(c_hat,temp1[,i])
}else{
  for (i in 1:K) c_hat = c(c_hat,temp1[[i]])
}
c_hat = t(as.matrix(c_hat))
#dimnames(c_hat)=dimnames(coefs)   # possible problem

# Determine C_hat (2.9) and initial variable in A_set
C_hat = max(abs(c_hat))             # share
a_set = which(abs(c_hat) == C_hat)       # share
s_set = sign(c_hat[a_set])     # (2.10) share

# Initial G_A and A_A (2.4)(2.5)
# everything calculated locally, share AA and w_A with everyone
G_A = (s_set)*xTx[a_set,a_set]*s_set
# (2.5) share (doesn't matter)

AA = (sum(sum(solve(G_A))))^(-0.5)        # (2.5) share
one_A = matrix(1,length(a_set),1)         # (2.5') doesn't matter
w_A = AA*solve(G_A)%*%one_A               # (2.6) share

# Each party calculates a_j locally using w_A and xTx values, not share
aa = vector("numeric",p)
for (i in 1:p){
aa[i] = xTx[i,a_set]*s_set*w_A
}

A_c = f_set[which(!(f_set%in%a_set))]     #  A^c (2.13)

# Find value of gamma and next active set,
# share gamma and beta_j
gams = find_gam(AA,aa,A_c,C_hat,c_hat)    # share
reduced = FALSE
  uninvert =FALSE
gammm = gams[1]
coefs = rbind(coefs, matrix(0,1,p))
coefs[2,a_set] = coefs[1,a_set]+ gammm    # fill in second row
directions = matrix(0,1,p)
```

```
directions[1,a_set]=1

## interations 2~
step = 2
while(!(((length(A_c)==0)&&(reduced==FALSE)) |  (uninvert==TRUE ) ) )
# while active set is not full
{
# each party updates c_hat using secure multiplication,
# but doesn't share:
c_hat = c_hat - gammm*aa

# next active set and \hat{C}
if (reduced == FALSE){
   a_set = sort(c(a_set,gams[2]))
   # new active set
   C_hat = C_hat - gammm*AA   #  p.414
}else{
  a_set = sort(a_set[which(!(a_set%in%red_gam[2]))])
  # new active set
  C_hat = C_hat - gammm*AA   #  p.414
}

# share signs
s_set = sign(c_hat[a_set])

# parties in active set calculate G_A using xTx and sign set
# everything calculated locally
G_A = diag(s_set)%*%xTx[a_set,a_set]%*%diag(s_set)

# Everyone in active set knows G_A, invert it and calculate A_A
# also calculate w_A
invGA = try(solve(G_A),silent=TRUE)
# use cholesky in practice

    if (is.matrix(invGA)){
  AA = (sum(sum(invGA)))^(-0.5)         # (2.5)
   one_A = matrix(1,length(a_set),1)         # (2.5')
  w_A = AA*invGA%*%one_A            # (2.6)

# Each party calculates a_j locally using w_A and xTx values,
# not share
aa = vector("numeric",p)
for (i in 1:p){
aa[i] = xTx[i,a_set]%*%diag(s_set)%*%w_A
}
```

```
# each party who holds a variable not in
# active set compares gamma values to get \hat{gamma}
A_c = f_set[which(!(f_set%in%a_set))]     #  A^c (2.13)
gams = find_gam(AA,aa,A_c,C_hat,c_hat)
#  find gam value and share

# Lasso step
d_hat = s_set*w_A # everyone knows
beta_hat = coefs[step,a_set] # everyone knows
gams2 = -beta_hat/d_hat # (3.4) share
red_gam = gamfunc2(gams2,a_set) # (3.5)

# see if add a variable or take away a variable from active set
# and then calculate \hat{\beta}
if(red_gam[1] < gams[1]){
reduced = TRUE
gammm=red_gam[1]
}else{
reduced = FALSE
gammm=gams[1]
}

  coefs = rbind(coefs, matrix(0,1,p))
   coefs[(step+1),a_set] = gammm*w_A*s_set
   directions = rbind(directions,matrix(0,1,p))
   directions[step,a_set]=w_A*s_set
   coefs[(step+1),] = coefs[(step+1),]+coefs[step,]
   step = step+1
    }else{
      uninvert = TRUE
    }
}
return(list(coefs=coefs,directions=directions,inds=index_set,xx=X1,yy=Y))
}


#################################
## Function to plot lasso path ##
#################################
plot_lasso = function(x1,ind=NULL){
bet = apply(abs(x1),1,sum)
if (!is.null(ind)) size1 = 1.2
else size1 = 1.1
plot(bet, x1[,1],pch=22,col=1,type="o",xlim=c(0,size1*max(bet)),
```

```
ylim=c(min(x1),max(x1)), ylab="Beta",xlab="Sum of absolute betas"
, main ="Lasso path")
abline(h=0,v=max(bet))

for (i in 2:dim(x1)[2]){
  lines(bet,x1[,i],pch=22,col=i,type="o")
}
aa=sort(x1[dim(x1)[1],],index.return=TRUE)
labs1=labels(x1)[[2]]

if (!is.null(ind)){
  labs2 = vector("character",dim(x1)[2])
   for (k in 1:length(ind)){
   temp1 = paste0("(A",k,")")
   for (i in min(ind[[k]]):max(ind[[k]])){
  labs2[i]=paste(labs1[i],temp1)
   }
  }
   text(x=rep(max(bet),dim(x1)[2]),y=sort(x1[dim(x1)[1],]), pos=4
  , labels=labs2[aa$ix])
}else{
   text(x=rep(max(bet),dim(x1)[2]),y=sort(x1[dim(x1)[1],]), pos=4
  , labels=labs1[aa$ix])
}
}


############################################
## Function to back-calculate beta values ##
## for target sum of absolute beta        ##
############################################
beta_func = function(coef, direction, target){
bet = apply(abs(coef),1,sum)
mark =1
if (target>=max(bet)){
betas = coef[dim(coef)[1],]
}else{
while (target > bet[mark]){
mark = mark+1
}
mark = mark-1
aa=0
bb=0
for (i in 1:(dim(coef)[2])){
if(coef[mark,i]<0){
```

```
aa = aa + coef[mark,i]
bb = bb - direction[mark,i]
}else if(coef[mark,i]>0){
aa = aa - coef[mark,i]
bb = bb + direction[mark,i]
}else{
if(direction[mark,i]<0){
bb = bb - direction[mark,i]
}else{
bb = bb + direction[mark,i]
}
}
}
gmmm = (target+aa)/bb
betas = coef[mark,] + gmmm*direction[mark,]
}
return(betas)
}

## Lasso using LARS on entire data, 4 parties
seed = 11111
K=4
entire1 = ver_lasso(da_ta,K,seed)


################################
## Cross valuation experiment ##
################################

## create test sets and train sets
folds = 5
horiz_data=split_hor(da_ta,seed,folds)
tests = vector("list",folds)
trains = vector("list",folds)
for (i in 1:folds){
tests[[i]] = horiz_data[[i]]
tempp = matrix(NA,0,length(da_ta))
for (j in 1:folds){
if(j==i){
tempp = tempp
}else{
tempp = rbind(tempp,horiz_data[[j]])
}
}
trains[[i]] = tempp
```

```
}


## for each training set
crossv_stuff = vector("list",folds)
for (i in 1:folds){
crossv_stuff[[i]] = ver_lasso(trains[[i]],K,seed)
}


## make each testing set compatible with analyzed training set
testXs=vector("list",folds)
testYs=vector("list",folds)
for (i in 1:folds){
spda = split_ver(tests[[i]],seed,K,1)
YY1 = spda[[1]][,1]
spda[[1]] = spda[[1]][,2:length(spda[[1]])]
XX1 = spda[[1]]
for (j in 2:K){
XX1 = cbind(XX1,spda[[j]])
}
testXs[[i]] = XX1
testYs[[i]] = YY1
}

## Figure 3.4
par(mfrow=c(2,3))
for (i in 1:folds){
plot_lasso(crossv_stuff[[i]][[1]],crossv_stuff[[i]][[3]])
}

plot_lasso(entire1[[1]],entire1[[3]])
dim(crossv_stuff[[1]][[4]])

len_gths = vector("numeric",folds)
# number of observations for each fold

for (i in 1:folds) len_gths[i] = length(testYs[[i]])
t_length = sum(len_gths)


cbetas = seq(600,4000,100)    # length 35

p_errors = vector("list",length(cbetas))
# sum of sq prediction errors
```

```
cbeta_errors = vector("numeric",length(cbetas))
for (i in 1:length(cbetas)){
cbeta_errors[i] = 0
for (j in 1:folds){
bets = beta_func(crossv_stuff[[j]][[1]],crossv_stuff[[j]][[2]], cbetas[i])
res1 = testYs[[j]] - as.matrix(testXs[[j]])%*%as.matrix(bets)
ssqres1 = sum(res1^2)
p_errors[[i]][[j]] = ssqres1
cbeta_errors[i] = cbeta_errors[i] + ssqres1
}
}


## Figure 3.5
av1=cbeta_errors/t_length
plot(cbetas,av1,type="o")

beta_func(entire1[[1]],entire1[[2]],1300)


## Figure 3.3
da_ta2 = as.matrix(da_ta[sample(1:370,7),])
for (i in 2:11)    da_ta2[,i] = scale(da_ta2[,i])

da_ta2 =as.data.frame(da_ta2)
entire2 = ver_lasso(da_ta2,K,seed)
plot_lasso(entire2[[1]],entire2[[3]])
```

3 Code for Experiment in Section 3.7.4 (Results summarized in Tables 3.4 and 3.5)

```
Ns = c(200,500,1000,2000,5000,10000,20000)
nl = length(Ns)
pAs = c(5,10,20,50,100,200)
np = length(pAs)
pb =10

ti_mes1 = matrix(NA,np,nl, dimnames=list(pAs,Ns))
err_ors1 = matrix(NA,np,nl, dimnames=list(pAs,Ns))
err_ors2 = matrix(NA,np,nl, dimnames=list(pAs,Ns))
```

```
for (i in 1:nl){
  XB = matrix(rnorm(Ns[i]*pb),Ns[i],pb)
  for (j in 1:np){
    pa = pAs[j]
    XA = matrix(rnorm(Ns[i]*pa),Ns[i],pa)
    g1 = trunc(Ns[i]*pa/(pa+pb))
    tt=rep(0,6)

    for (k in 1:6){
    ptm = proc.time()

    Zs = findorths(XA,g1)

    dd2 = -Zs%*%t(Zs)
    for (l in 1:dim(dd2)[1])   dd2[l,l] = 1 + dd2[l,l]
    result1 = t(XA)%*%(dd2%*%XB)

    ttt = proc.time()-ptm
    tt[k] = ttt[3]
    }

    tmedian = signif(median(tt))
    tmax = signif(max(tt))
    ti_mes1[j,i] = paste0(tmedian," (",tmax,")")
    truu = t(XA)%*%XB
    a_diffs = abs((truu - result1)/truu)
    dmean = signif(mean(a_diffs))
    dmax = signif(max(a_diffs))
    err1 = length(a_diffs[a_diffs<.1])/length(a_diffs)
    err01 = length(a_diffs[a_diffs<.01])/length(a_diffs)
    err001 = length(a_diffs[a_diffs<.001])/length(a_diffs)
    err_ors1[j,i] = paste0(dmean," (",dmax,")")
    err_ors2[j,i] = paste0(err001," (",err01,")", " [",err1,"]" )
    print("Case1 errors1, mean (max):")
    print(err_ors1)
    print("Case1 errors2, <.001 (<.01):")
    print(err_ors2)
    print("Case1 times, median (max):")
    print(ti_mes1)
  }
}
```

4 Code for Experiment in Section 3.7.5 (Results summarized in Table 3.5)

```
##############################################
pars = list("vector", 13)
pars[[1]]=c(2,6)
pars[[2]]=c(3,5)
pars[[3]]=c(4,4)
pars[[4]]=c(2,1,5)
pars[[5]]=c(2,2,4)
pars[[6]]=c(2,3,3)
pars[[7]]=c(3,1,4)
pars[[8]]=c(2,1,1,4)
pars[[9]]=c(2,1,2,3)
pars[[10]]=c(2,2,2,2)
pars[[11]]=c(2,1,1,1,3)
pars[[12]]=c(2,1,1,2,2)
pars[[13]]=c(2,1,1,1,1,1)
##################################################

compfunc = function(n,pa,pb){
  g1 = pa*n/(pa+pb)
  ssum = 2*n*pa^2 -2*pa^3/3 + n^2*(2*g1-1) + n + n*pb*(2*n-1) +pa*pb*(2*n-1)
  return(ssum)
}

cfunc1 = function(n,pa){
  ssum = 2*n*pa^2 -2*pa^3/3
  return(ssum)
}

cfunc2 = function(n,pa,pb){
  g1 = pa*n/(pa+pb)
  ssum = n^2*(2*g1-1) + n + n*pb*(2*n-1) +pa*pb*(2*n-1)
  return(ssum)
}

comptimefunc = function(n, pars1){
  summ = 0
  for (i in 1:(length(pars1)-1)){
    summ = summ  + cfunc1(n, pars1[i])
    for (j in (i+1):length(pars1)){
      summ = cfunc2(n,pars1[i],pars1[j])
```

```
    }
  }
  return(summ)
}

n11 = 5000
v11 = rep(NA, length(pars))
for (i in 1: length(pars)){
  v11[i]= comptimefunc(n11,pars[[i]])
}

for (i in 1:length(pars)){
  cat("partition = ", pars[[i]],"   ", "time = ",ti_mes2[i], "\n")
}

ti_mes2 = rep(NA,length(pars))
for(t in 1:length(pars))

splitt = pars[[t]]
hh1 = split_ver2(housing5,splitt)
hh2 = vector("list",length(hh1))
KK = 5
for (i in 1:length(hh1)){
  hh2[[i]] = split_hor2(hh1[[i]], KK)
}

ti_mes1 = matrix(NA,KK,length(splitt)-1)
va_ls = vector("list", (length(splitt)))    # store XTX
for (i in 1:length(va_ls)){
  va_ls[[i]] = vector("list",length(splitt))
  for (j in 1:length(va_ls[[i]])){
    va_ls[[i]][[j]] = vector("list",KK)
  }
}

yva_ls = vector("list", (length(splitt)))   # store xTy
for (i in 1:length(yva_ls))  yva_ls[[i]] = vector("list",KK)

for (m in 1:length(splitt)){
  for (b in 1:KK){
    if (m==1){
        res1 = t(hh2[[m]][[b]])%*%hh2[[m]][[b]]
        res2 = res1[2:dim(res1)[1],2:dim(res1)[2]]
        va_ls[[m]][[m]][[b]] = res2
        res3 = res1[1,2:dim(res1)[2]]
```

```
        yva_ls[[m]][[b]] = t(res3)
    }else{
        va_ls[[m]][[m]][[b]] = t(hh2[[m]][[b]])%*%hh2[[m]][[b]]
    }
  }
}

for (i in 1:(length(splitt)-1)){
  pa = splitt[i]
  pb = max(splitt[-i])
  for (j in 1:KK){
    g1 = trunc(dim(hh2[[i]][[j]])[1]*pa/(pa+pb))
    ptm = proc.time()
    Zs = findorths(hh2[[i]][[j]],g1)
    ttt = proc.time()-ptm
    ti_mes1[j,i] = ttt[3]
       for (k in (i+1):length(splitt)){
          g2 = trunc(dim(hh2[[i]][[j]])[1]*pa/(pa+splitt[k]))
          Zs1 = Zs[,1:g2]
          dd2 = -Zs1%*%t(Zs1)
             for (l in 1:dim(dd2)[1])   dd2[l,l] = 1 + dd2[l,l]
                  result1 = t(hh2[[i]][[j]])%*%(dd2%*%hh2[[k]][[j]])
                  if (i==1){
                       result2 = result1[2:dim(result1)[1],]
                       yva_ls[[k]][[j]] = t(result1[1,])
                       va_ls[[i]][[k]][[j]] = result2
                       va_ls[[k]][[i]][[j]] = t(result2)
                  }else{
                      va_ls[[i]][[k]][[j]] = result1
                      va_ls[[k]][[i]][[j]] = t(result1)
                 }
               }
         }
}
tot_time = proc.time() - ptm1

calc = matrix(NA,dim(housing5)[2]-1,dim(housing5)[2]-1)

st1 = 1
sp1 = splitt[1]-1
for (i in 1:length(splitt)){
  st2 = 1
  sp2 = splitt[1]-1
  for (j in 1:length(splitt)){
    sm1 = va_ls[[i]][[j]][[1]]
```

```
      for (k in 1:KK){
        sm1 = sm1 + va_ls[[i]][[j]][[k]]
      }
    calc[st1:sp1,st2:sp2] = sm1
    if (j!=length(splitt)){
        st2 = sp2 + 1
        sp2 = sp2 + splitt[j+1]
    }
  }
  if(i!=length(splitt)){
      st1 = sp1  + 1
      sp1 = sp1 + splitt[i+1]
  }
}

truuX = t(housing5[,2:8])%*%housing5[,2:8]
truuXy = t(housing5[,2:8])%*%housing5[,1]
a_diffs = abs((truuX - calc)/truuX)

a_diffs; ti_mes1; tot_time
truuX; calc; truuXy; ti_mes2
```

Bibliography

Atallah, M. J. and W. Du (2001). Secure multi-party computational geometry. In *Algorithms and Data Structures*, pp. 165–179. Springer.

Blake, I. F. and V. Kolesnikov (2004). Strong conditional oblivious transfer and computing on intervals. In *Advances in Cryptology-ASIACRYPT 2004*, pp. 515–529. Springer.

Businger, P. and G. H. Golub (1965). Linear least squares solutions by householder transformations. *Numerische Mathematik 7*(3), 269–276.

Efron, B., T. Hastie, I. Johnstone, R. Tibshirani, et al. (2004). Least angle regression. *The Annals of statistics 32*(2), 407–499.

Fienberg, S., A. Karr, Y. Nardi, and A. Slavkovic (2007). Secure logistic regression with distributed databases. In *Proceedings of the 56th Session of the ISI, The Bulletin of the International Statistical Institute.*

Fienberg, S. E., W. J. Fulp, A. B. Slavkovic, and T. A. Wrobel (2006). "secure" log-linear and logistic regression analysis of distributed databases. In *Privacy in statistical databases*, pp. 277–290. Springer.

Fienberg, S. E., Y. Nardi, and A. B. Slavković (2009). Valid statistical analysis for logistic regression with multiple sources. In *Protecting Persons While Protecting the People*, pp. 82–94. Springer.

Gaye, A., Y. Marcon, J. Isaeva, P. LaFlamme, A. Turner, E. M. Jones, J. Minion, A. W. Boyd, C. J. Newby, M.-L. Nuotio, et al. (2014). Datashield: taking the analysis to the data, not the data to the analysis. *International journal of epidemiology*, dyu188.

Golub, G. H. and C. F. Van Loan (2012). *Matrix computations*, Volume 3. JHU Press.

Guo, C.-H. and N. J. Higham (2006). A schur-newton method for the matrix\ boldmath p th root and its inverse. *SIAM Journal on Matrix Analysis and Applications 28*(3), 788–804.

Hall, R., S. E. Fienberg, and Y. Nardi (2011). Secure multiple linear regression based on homomorphic encryption. *Journal of Official Statistics 27*(4), 669.

Hastie, T., J. Taylor, R. Tibshirani, G. Walther, et al. (2007). Forward stagewise regression and the monotone lasso. *Electronic Journal of Statistics 1*, 1–29.

Hoerl, A. E. and R. W. Kennard (1970). Ridge regression: Biased estimation for nonorthogonal problems. *Technometrics 12*(1), 55–67.

Hoerl, A. E. and R. W. Kennard (1981). Ridge regressionâĂŤ1980: Advances, algorithms, and applications. *American Journal of Mathematical and Management Sciences 1*(1), 5–83.

Karr, A. F., X. Lin, A. P. Sanil, and J. P. Reiter (2005). Secure regression on distributed databases. *Journal of Computational and Graphical Statistics 14*(2), 263–279.

Karr, A. F., X. Lin, A. P. Sanil, and J. P. Reiter (2009). Privacy-preserving analysis of vertically partitioned data using secure matrix products. *Journal of Official Statistics 25*(1), 125.

Lin, X. and A. F. Karr (2010). Privacy-preserving maximum likelihood estimation for distributed data. *Journal of Privacy and Confidentiality 1*(2), 6.

Lindell, Y. and B. Pinkas (2009). Secure multiparty computation for privacy-preserving data mining. *Journal of Privacy and Confidentiality 1*(1), 5.

Madigan, D. and G. Ridgeway (2004). Least angle regression: Discussion. *The Annals of Statistics 32*(2), 465–469.

Nardi, Y., S. E. Fienberg, and R. J. Hall (2012). Achieving both valid and secure logistic regression analysis on aggregated data from different private sources. *Journal of Privacy and Confidentiality 4*(1), 9.

Paillier, P. (1999). Public-key cryptosystems based on composite degree residuosity classes. In *Advances in Cryptology-EUROCRYPT 99*, pp. 223–238. Springer.

Reiter, J., C. Kohnen, A. Karr, X. Lin, and A. Sanil (2004). Secure regression for vertically partitioned overlapping data. Technical report, Technical report.

Rivest, R. L., A. Shamir, and L. Adleman (1978). A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM 21*(2), 120–126.

Slavkovic, A. B., Y. Nardi, and M. M. Tibbits (2007). "secure" logistic regression of horizontally and vertically partitioned distributed databases. *Data Mining Workshop, 2007, ICDM Workshops 2007, Seventh IEEE International Conference on Data Mining*, 723–728.

Thisted, R. A. (1976). *Ridge regression, minimax estimation, and empirical Bayes methods.* Department of Statistics, Stanford University.

Tibshirani, R. (1996). Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B (Methodological)*, 267–288.

Yao, A. (1986). How to generate and exchange secrets. In *Foundations of Computer Science, 1986., 27th Annual Symposium on*, pp. 162–167. IEEE.

Zou, H. and T. Hastie (2005). Regularization and variable selection via the elastic net. *Journal of the Royal Statistical Society: Series B (Statistical Methodology) 67*(2), 301–320.