AN ABSTRACT OF THE PROJECT OF

Tez Deepthi Tammineni for the degree of Master of Science in Computer Science presented on March, 10 2011.

Title: On Convex Dimensionality Reduction for Classification.

Abstract approved: ______________________________

Raviv Raich

Dimensionality reduction (DR) is an efficient approach to reduce the size of data by capturing the informative intrinsic features and discarding the noise. DR methods can be grouped through a variety of categories, e.g. supervised/ unsupervised, linear/non-linear or parametric/non-parametric. Objective function based methods can be grouped into convex and non convex. Non convex methods have the tendency to converge to local optimal solutions which may differ from the desired global solution. To overcome this problem, one can frame a non convex problem as a convex problem. A direct transformation of a non convex formulation into a convex formulation is non trivial.

In this project, we chose a non convex, linear, supervised, multi-class, non parametric DR method's objective, which can be framed using convex formulation. We present the process of convex formulation, different methods of implementation, approaches to increase efficiency and numerical analysis results.

# On Convex Dimensionality Reduction for Classification

by

Tez Deepthi Tammineni

A PROJECT REPORT

submitted to

Oregon State University

in partial fulfillment of
the requirements for the
degree of

Master of Science

Presented March, 10 2011
Commencement June 2011

Master of Science project of Tez Deepthi Tammineni presented on March, 10 2011

APPROVED:

Major Professor, representing Computer Science

Director of the School of Electrical Engineering and Computer Science

Dean of the Graduate School

Tez Deepthi Tammineni, Author

# ACKNOWLEDGEMENTS

I thank my family for supporting my ambitions, constantly encouraging me and being there for me during all the times I needed them. I express my sincere gratitude to my advisor Dr.Raviv Raich for his guidance, advice and encouragement in research and academics. I thank almighty for his blessings.

Oregon State University has provided facilities and atmosphere that are required for the student to excel academically and professionally. I thank Dr. Bella Bose, Ferne Simendinger, Tina Batten for giving me an opportunity to work as EECS web developer during my masters.

I thank all my research group members for the valuable discussions. I extend my special thanks to Behrouz, Balaji and Madan who constantly supported, encouraged and helped me in research and academics. I thank Aarti, Ali, Aniket, Madhura, Nisha, Shalini and Vivek for discussions and combined effort on various academic projects. Thanks to Partiban for encouraging, supporting me. I thank my roommates Dung, Indu, Pratheeksha, Samina, Swetha for being cooperative and helpful. I thank many others for all the fun and for being wonderful friends during my stay at Oregon State University.

I would like to acknowledge contribution of the authors of [1], for providing required framework for preprocessing different datasets and classification algorithms which were used in numerical analysis in Chapter 8.

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

LIST OF FIGURES

## LIST OF TABLES

# 1. INTRODUCTION

## 1.1. Dimensionality Reduction in Machine Learning

Dimensionality reduction (DR) is an effective approach to reduce the size of data under consideration by capturing the intrinsic features and discarding the noise. Dimension is the number of features which represent the object. As the number of features increases, the amount of observations required for accurate analysis increases. Consequently, the memory required and time complexity increase as well. DR solves this problem by capturing the most useful features. Most of the machine learning algorithms and data mining techniques may be effective in the low number of dimensions required. Hence, DR is used for data pre-processing in most applications. DR is mainly used for better data visualization, data compression and noise removal. DR has its applications in various fields such as text classification [2], image retrieval [3], [4] and recommendation systems [5].

## 1.2. Different Types of Dimensionality Reduction

DR methods can be classified based on different categories such as the methods of learning (supervised, unsupervised), type of transformation (linear, non linear) used for DR, or the type of the data model (Gaussian, non Gaussian, non parametric) assumed. The examples of unsupervised DR methods are Principal Component Analysis (PCA) [6], Independent Component Analysis (ICA) [7], unsupervised Manifold learning algorithms [8], [9] and supervised DR methods include Linear Discriminant Analysis (LDA) [10], Canonical Correleation Analysis (CCA) [11]. The reader can refer to [12], for a detailed survey of DR methods. Based on transformation function used for DR; we can classify PCA, LDA, CCA into linear DR methods and Laplacian Eigenmaps [9], multidimensional scaling (MDS) [13], Local Linear Embedding [8] into non linear DR methods. A survey of linear and non linear DR methods is found in [14]. Based on the data model assumption made i.e., either implicit or explicit, DR methods like PCA, LDA can be grouped into methods that assume Gaussian data structure and methods like [15], [16], [17], SEPCA [18] can be grouped into methods that assume non Gaussian and use exponential family models while methods NCA [19], [1] can be grouped into methods that assume non parametric data model.

## 1.3. Convex and Non-Convex Methods in Dimensionality Reduction

Among the types of DR methods mentioned in the previous section, we can group PCA, CCA, LDA, [17], SEPCA [18] as methods having unique solution through the eigen value decomposition or through convex optimization problem. Methods like NCA, [1], [15], [16], are non convex in nature. Non convex problems will have multiple local optima and iterative solutions may converge to local optima. The problems with non convex methods are: they need multiple restarts with differnt initializations, the stopping criteria is not known, slow convergence rate and the global solution is not guaranteed. The advantages of convex methods are unique guaranteed solution and they do not require multiple restarts. Convex optimization is a well studied field. Different problems like linear programming, quadratic programming, Semi definite Programming (SDP) [20], Second Order Cone Programming (SOCP) [21] are some of well-studied canonical convex optimization problems. Methods such as interior point methods [22], ellipsoid method, sub-gradient methods are used to solve generic convex problems. Different softwares such as CVX [23],YALMIP, CVXOPT, CVXMOD, MOSEK, solver.com, SeDuMi, SDPT3 and OBOE are available to solve the convex problems efficiently. Convex optimization has gained importance due to its advantages and extended its applications to various fields such as communication, finance, signal processing and digital communication [24]. Convex optimization can be applied to dynamic programming [25], metric learning [26], dimensionality reduction [17], [18] problems.

Unique solution can be guaranteed if the DR method is using eigenvalue decomposition as in PCA, LDA or if its objective function is convex like in [17], SEPCA [18]. Framing a non convex method into convex function is one of the approaches used to get unique solution. For example a non-convex formulation of PCA into convex optimization method can be found in [17].

## 1.4. Our Approach

In our project, we have chosen a linear, supervised DR method whose objective is non-convex. This objective function can be formulated as relaxed convex function. In the subsequent chapters we will discuss about relaxed convex formulation of chosen non-convex objective function. We will also discuss about different methods we implemented in our project to solve the relaxed convex objective function, and the various machinery we tried to reduce time and space complexity and improve computational efficiency.

# 2. PROBLEM FORMULATION

## 2.1. Non-Convex Approach to Supervised Dimensionality Reduction

In [1] a non-convex linear, supervised, non-parametric, multi-class dimensionality reduction method for classification has been proposed. This method uses probability of error bound as criterion to optimize.

The data setup is assumed as follows. The $(X_i, c_i)$ is a (observation, class) pair where each observation or data point $X_i \in \mathbb{R}^D$ and total $N$ data points are present. According to linear DR method proposed in [1], we need to find a low dimension representation $(AX_i, c_i)$ for each $(X_i, c_i)$ through linear transformation of data points i.e., by multiplying with matrix $A$, which is $d \times D$ where $d$ is the required low dimension. The following was presented as an upper bound on the error probability for the multi-class problem where the underlying distribution is assumed to follow KDE (Kernel Density Estimation).

$$J(A) = \frac{1}{N} \sum_{i=1}^{N} \sum_{j=1}^{N} I(c_i \neq c_j) e^{-\frac{(X_i - X_j)^T A^T A (X_i - X_j)}{8\sigma^2}}. \tag{2.1}$$

The objective is to minimize (2.1) with respect to $A$. To minimize the $ij^{th}$ term in $J(A)$ one needs to maximize the distance between $AX_i$ and $AX_j$ given by $(X_i - X_j)^T A^T A (X_i - X_j)$. Intuitively, $J(A)$ gives priority to a close pair of points of different classes. The reader can refer to [1] for details of derivation of (2.1).

For convenience, replace $X_i = \sqrt{8\sigma^2}\tilde{X}_i$ in (2.1) and denote the difference between the normalized data points $\tilde{X}_i - \tilde{X}_j$ by $\delta\tilde{X}_{ij}$. Equation (2.1) is now transformed to

$$J(A) = \frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{N} I(c_i \neq c_j)e^{-(\delta\tilde{X}_{ij}^T A^T A \delta\tilde{X}_{ij})}. \tag{2.2}$$

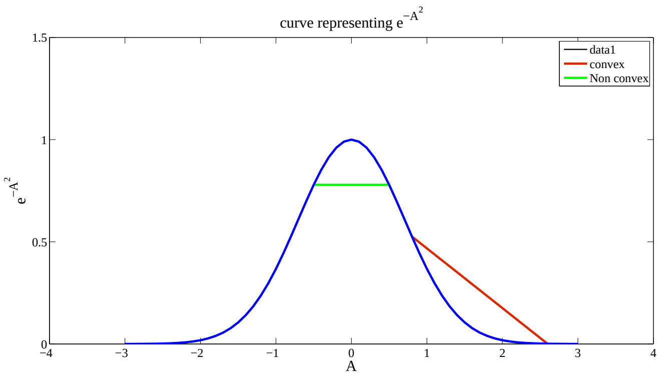
The minimization of (2.2) with respect to the orthogonal linear transformation $A$ is given by

$$\min_{A \in \mathbb{R}^{d\times D}} \frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{N} I(c_i \neq c_j)e^{-\delta\tilde{X}_{ij}^T A^T A \delta\tilde{X}_{ij}} \tag{2.3}$$

$$\text{subject to } AA^T = I_d.$$

The solution $A \sim S_{A_0} = \{A \mid A = RA_0\}$ where $R$ is a rotation matrix. Hence, the solution to (2.3) can be any $A$ in $S_{A_0}$. For the problem (2.3) the objective and orthogonality constraints are non-convex. To prove non-convexity of objective function, via a counter example, we consider a scalar version of the term $e^{-\delta\tilde{X}_{ij}^T A^T A \delta\tilde{X}_{ij}}$ w.r.t $A$ as $e^{-\delta^2 a^2}$ by setting $\delta\tilde{X}_{ij} = \delta$ and $A = a$. Taking the second derivative, we obtain $-2\delta^2 e^{-\delta^2 a^2}(1 - 2a^2\delta^2)$ which can be negative when $a = 0$. Hence, the objective function may be non-convex. For the function to be convex the double differentiation of it should be always non-negative. The plot of a curve of the form $e^{-x^2}$ is shown in Figure 2.1.

Similarly, the non-convexity of constraint $A^T A = I$ can be shown by assuming a scalar version for $A = a$, which yields $a = \pm 1$ a non-convex set. Hence, the set $\{A \mid A^T A = I\}$ is a non-convex set.



(a) Non convex objective function

FIGURE 2.1: Representing non-convex curve.

## 2.2. Convex Formulation

In this section, we describe the step by step process of formulating the non convex problem into equivalent relaxed convex problem. We subdivide the process of convex conversion into equivalent formulation followed by relaxed formulation.

### 2.2.1 Equivalent Formulation

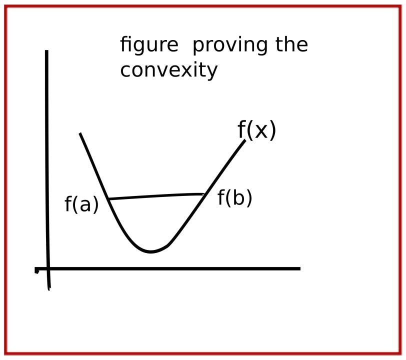
The equivalent formulation is done through change of variables. We replace $W = A^T A$ in (2.3) and obtain a new formulation

$$\widetilde{J}(W) = \frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{N} I(c_i \neq c_j)e^{-\delta\tilde{X}_{ij}^T W \delta\tilde{X}_{ij}}. \tag{2.4}$$

Note that (2.4) is convex in W. We can prove that (2.4) is convex by definition

$$f(\lambda a + (1-\lambda)b) \leq \lambda(f(a)) + (1-\lambda)f(b) \tag{2.5}$$

where $\lambda \in [0, 1]$.



(a) Convexity basic definition

FIGURE 2.2: Convex definition figure.

Let us assume our objective function of the form

$$f(W) = \sum_i e^{-\delta_i^T W \delta_i}. \tag{2.6}$$

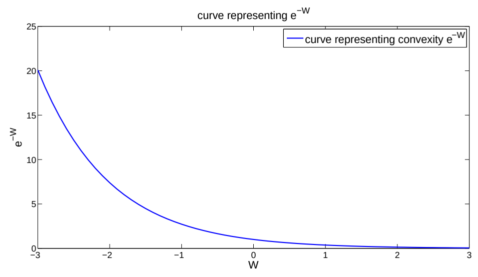
Since the summation maintains convexity [27], it is enough to prove $e^{-\delta_i^T W \delta_i}$ is convex. Consider two values $W_1$ and $W_2$ where $W_1 \neq W_2$, $0 \leq \lambda \leq 1$ and frame it in the basic convex definition

$$e^{-\delta_i^T(\lambda W_1 + (1-\lambda)W_2)\delta_i} \leq \lambda e^{-\delta_i^T W_1 \delta_i} + (1-\lambda)e^{-\delta_i^T W_2 \delta_i} \tag{2.7}$$

Let $t_1 = \delta_i^T \lambda W_1 \delta_i$ and $t_2 = \delta_i^T \lambda W_2 \delta_i$. Since $e^{-t}$ is convex, $e^{-\lambda t_1 + (1-\lambda)t_2} \leq \lambda e^{-t_1} + (1-\lambda)e^{-t_2}$ and hence, (2.7) holds. The plot representing the curve of the form $e^{-x}$ is shown in figure (2.3).

Substituting $W$ in the constraint set of $A$ yields a feasible set of $W$'s where the constraints set for $W$ given by $S_W = \{W \mid W = A^T A, A \in \mathbb{R}^{d \times D}, AA^T = I\}$

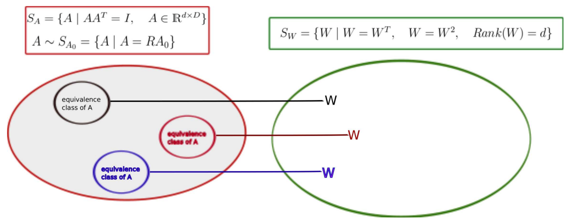
which is equivalent to $S_W = \{W \mid W = W^T, W = W^2, Rank(W) = d, W \in \mathbb{R}^{D\times D}\}$. Let us define $S_A = \{A \mid A \in \mathbb{R}^{d\times D}, AA^T = I_{d\times d}\}$ where $A \sim S_{A_0} = \{A \mid A = RA_0\}$



(a) Convex objective function

FIGURE 2.3: Representing convex curve.

and $R$ is rotation matrix. There is one to one mapping between sets $S_A$ and $S_W$ (see Figure (2.4)). This can be proved by showing that $\forall A \in S_{A_0} \subset S_A$ is mapped to unique $W$ in $S_W$ and each $W$ in $S_W$ is mapped to an equivalence class in set $S_A$. Reader can refer Appendix A for detailed proof.


$S_A = \{A \mid AA^T = I, \quad A \in \mathbb{R}^{d\times D}\}$
$A \sim S_{A_0} = \{A \mid A = RA_0\}$
$S_W = \{W \mid W = W^T, \quad W = W^2, \quad Rank(W) = d\}$
equivalence class of A
W
equivalence class of A
W
equivalence class of A
W

(a) Equivalence between sets

FIGURE 2.4: Representing the equivalence between sets $S_A$ and $S_W$

The objective function in (2.3) is transformed to

$$\begin{aligned} \min_{W\in\mathbb{R}^{D\times D}} \quad & \widetilde{J}(W) \\ \text{subject to}: \quad & W^2 = W \\ & W = W^T \\ & Rank(W) = d. \end{aligned} \tag{2.8}$$

We say that (2.3) and (2.8) are equivalent formulations as one to one mapping exists between $S_A$ and $S_W$.

### 2.2.2 Relaxed Formulation

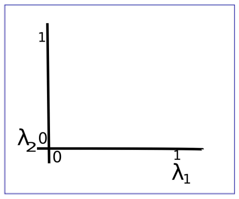
We notice that equation (2.8) is convex in the objective function but not in constraints. The two constraints that contribute to non convexity are $W = W^2$ (since $\lambda_l \in \{0, 1\}$) and $Rank(W) = d$.

The first non convex constraint $W = W^2$ can be replaced with a convex constraint by relaxing the eigenvalues to $0 \leq \lambda_l \leq 1$. So $\lambda_l \in \{0, 1\}$ and $\{W = W^2, \quad W = W^T \quad Rank(W) = d\}$ is relaxed to $0 \leq \lambda_l \leq 1$ and $\{W \geq 0, \quad I - W \geq 0, \quad W = W^T \quad Rank(W) = d\}$ (see Figure 2.5). We are guaranteed that proposed relaxation ($0 \leq \lambda_l \leq 1$) to the problem will result in the same solution as the original problem ($\lambda_l \in \{0, 1\}$). Our objective function is to minimize quantity of the form $e^{-\delta^T W \delta}$, i.e., we have to maximize the term $\delta^T W \delta$. If we express $W$ in terms of eigenvalue decomposition i.e., $U \Lambda U^T$, our objective is to maximize $\sum_{l=1}^{D} \lambda_l \mid (\delta_i)^T U_l \mid^2$. Suppose $0 < \lambda_l < 1$, setting $\lambda_l$ yields a smaller value to the objective function. Solutions with $0 < \lambda_l < 1$ are not optimal. Suppose we have $0 < \lambda_l < 1$, we can always set $\lambda_l = 1$ without affecting the rank constraint. Hence, we can say that our first relaxation will not affect the solution. After the first relaxation the problem (2.8) becomes

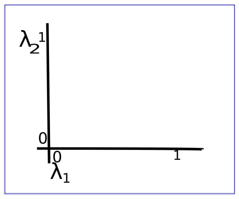
$$\begin{aligned} \min_{W \in \mathbb{R}^{D\times D}} \quad & \widetilde{J}(W) \\ \text{subject to:} \quad & W \geq 0 \\ & I - W \geq 0 \\ & W = W^T \\ & Rank(W) = d. \end{aligned} \tag{2.9}$$

And the problems (2.9) and (2.3) are equivalent. The similar relaxation is also done in [17]. For the second non convex constraint $Rank(W) = d$, we propose a heuristic relaxation, i.e., we replace $Rank(W) = d$ with $trace(W) \leq \gamma$ where $\gamma \leq d$ (see Figure 2.6). The trace is a linear operator in the space of matrix i.e., trace is the sum of eigenvalues (the eigenvalues in our problem are always positive), the linear combination of all positive values is convex set. This relaxation works in the ideal case when $\lambda_i \in \{0, 1\}$ the $trace(W) = Rank(W) = d$. And we relaxed the constraint as $trace(W) \leq \gamma$, as the problem with inequality constraints can be framed into smooth unconstrained problem and solved effectively. After the heuristic relaxation the problem (2.9) becomes

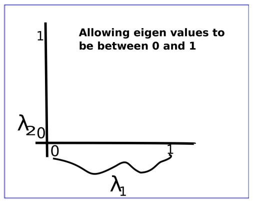
$$\begin{aligned} \min_{W \in \mathbb{R}^{D\times D}} \quad & \widetilde{J}(W) \\ \text{subject to:} \quad & W \geq 0 \\ & I - W \geq 0 \\ & W = W^T \\ & trace(W) \leq \gamma. \end{aligned} \tag{2.10}$$



(a) First eigenvalue non zero



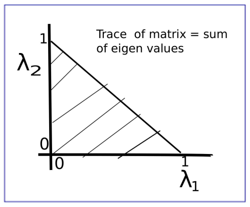
(b) second eigenvalue non zero



(c) Either of eigenvalues vary between 0 and 1

FIGURE 2.5: Representations of eigenvalues.

With this relaxation we are not guaranteed that solution to the problem (2.10) is the solution to the original problem (2.3). So once we get the solution, we then project the solution in to the feasible region of (2.3).

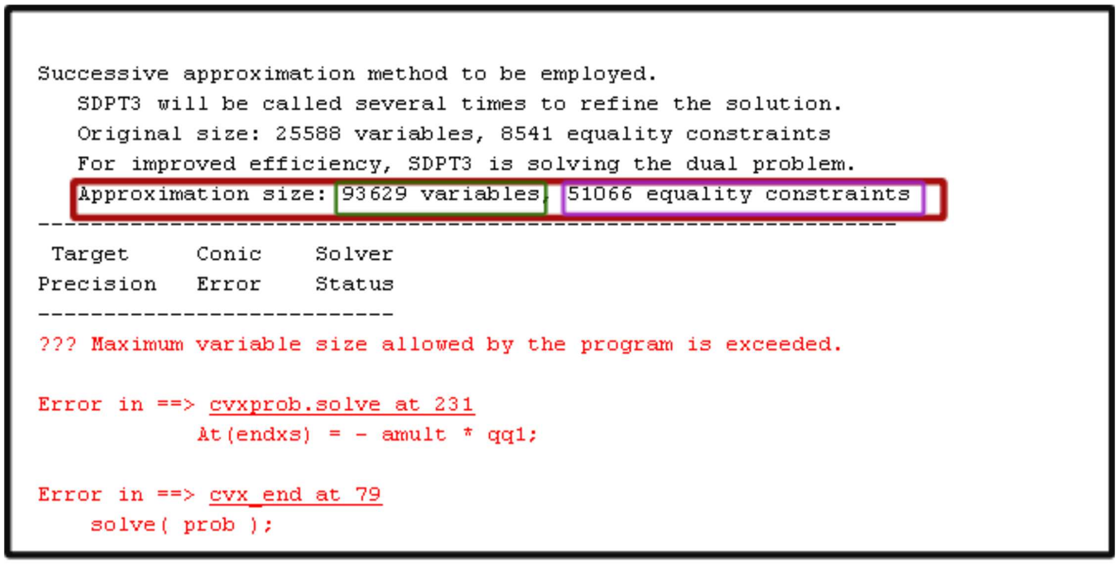


(a) Trace of W

FIGURE 2.6: Representing the trace feasibility region for $W$

# 3. IMPLEMENTATION

The canonical problems in convex optimization are, for example, linear programming (LP), quadratic programming (QP), semi definite programming (SDP) and second order cone programming (SOCP). Many years of research produced a variety of methods to solve such problems. We have general solvers to solve any convex problem. Some issues like memory requirement, time complexity exists with general solvers. For instance in Newton descent method, if we calculate the Newton direction ($H^{-1}l$ where $H$ represents Hessian and $l$ represents gradient) by explicitly spelling out the Hessian, calculating the inverse and then the product with gradient. The memory requirement is high and calculation of Hessian and inverse might be difficult if the matrix is ill conditioned. But, these issues can be resolved by iteratively solving the linear equations (e.g., Landweber iterations). To solve the convex problems we have softwares like YALMIP, CVX, CVXOPT, CVXMOD, MOSEK, solver.com and OBOE. We have used CVX to solve our problem. CVX uses successive approximation method to solve functions like log, exp, log-sum-exp that cannot be exactly represented by SOCP, SDP or LP. This technique can be slower. CVX solves our convex relaxed problem (2.10) through successive approximation using SDPT3 (a MATLAB software package for semidefinite-quadratic-linear programming). This resulted in creating large number of variables and equality constraints. We ran the experiment for 'Glass' data set which is of 8 dimensions having 214 data points. CVX created 93629 no of variables and 51066 equality constraints. The large number of variables resulted in reaching memory limits. Refer to the screen shot shown in Figure 3.1. We have designed an algorithm which handles the issues related to time complexity and memory issues effectively. The framework of the algorithm is described in Section 3.1. To overcome such problems and to increase



```
Successive approximation method to be employed.
   SDPT3 will be called several times to refine the solution.
   Original size: 25588 variables, 8541 equality constraints
   For improved efficiency, SDPT3 is solving the dual problem.
   Approximation size: 93629 variables, 51066 equality constraints
-----------------------------------------------------------------
 Target      Conic    Solver
Precision    Error    Status
---------------------------
??? Maximum variable size allowed by the program is exceeded.

Error in ==> cvxprob.solve at 231
            At(endxs) = - amult * qq1;

Error in ==> cvx_end at 79
    solve( prob );
```

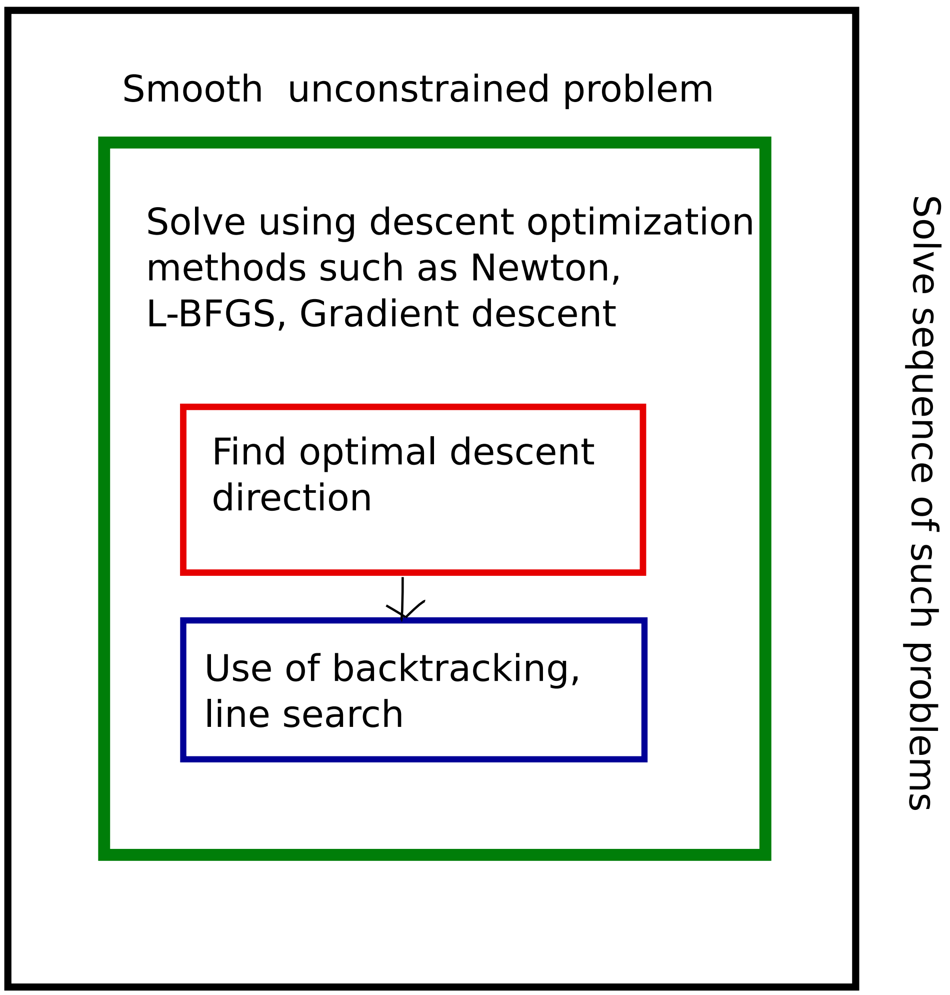
(a) cvx failure

FIGURE 3.1: Failed cvx screen shot

the computational efficiency consider a framework which is described in Section 3.1.

## 3.1. Framework

Our objective function (2.10) accompanied by inequality constraints is framed as smooth unconstrained problem, using the log barrier approach. Then we solve sequence of smooth unconstrained problems using descent methods such as Newton [28], Gradient descent [27], L-BFGS (limited memory Broyden-Fletcher-Goldfrab-Shanno) (a Quasi Newton method) [29] methods which involve calculating the direction and a step size in that direction. This approach of solving the problems is


Smooth unconstrained problem
Solve using descent optimization methods such as Newton, L-BFGS, Gradient descent
Find optimal descent direction
Use of backtracking, line search
Solve sequence of such problems

(a) Framework

FIGURE 3.2: Framework of our implementation

known as interior point method or also known as Sequential Unconstrained Minimization Technique (SUMT) [30]. The framework is shown in the Figure 3.2.

## 3.2. Log Barrier Method

Log barrier is one of the interior point methods [22] which forces the solution to be within the feasible set and frames the inequality constrained problem as smooth unconstrained problem. The standard log barrier [31], [27] is defined as follows
For convex problem

$$\begin{aligned} \text{minimize} \quad & f(x) \\ \text{subject to} \quad & f_i(x) \leq 0, i = 1, \ldots, m. \end{aligned} \tag{3.1}$$

Logarithmic barrier is defined as

$$\phi(x) = \begin{cases} \sum_{i=1}^{m} -log(-f_i(x)), & \text{if } f_i(x) \leq 0, \quad i = 1, \ldots, m. \\ \infty, & \text{otherwise.} \end{cases}$$

The unconstrained approximation to (3.1) is

$$\min_x . f(x) + \frac{1}{t}\phi(x), \tag{3.2}$$

where $\phi$ is convex, smooth on interior of feasible set. As $x$ approaches the boundary of feasible set, $\phi \to \infty$. From [31] the barrier method is simple to implement and shows good performance. Now the objective function (2.10) with log barrier method is transformed as follows

$$\begin{aligned}\widetilde{J}_t(W) =& \frac{1}{N}\sum_i \sum_j I(c_i \neq c_j) e^{-\delta \tilde{X}_{ij}^T W \delta \tilde{X}_{ij}} \\ & - \frac{1}{t}\big(\log(\det(W)) + \log(\det(I - W)) + \log(\gamma - tr(W))\big).\end{aligned} \tag{3.3}$$

The symmetric constraint is not explicitly incorporated into the objective function but it is guaranteed to maintain symmetric structure, if we initialize with symmetric matrix as the updates are always symmetric. We solve sequence of (3.3) by varying the log barrier $t$ and as $t \to \infty$ we are guaranteed that solution $W^{(t)} \to W^*$ which is the optimal solution.

In succeeding chapters, we will discuss in detail different optimization methods to solve sequence of smooth unconstrained problems.

# 4. METHOD -1

To solve each problem of the form (3.3) in the sequence of problems in the log barrier method we propose Newton method.

## 4.1. Newton Method

Newton method is one of the popular optimization methods. It is widely used as its order of convergence is quadratic [27]. Though the per iteration computation is expensive, the performance and convergence rate compensate it. The pseudo algorithm of Newton method is described in algorithm 1.

```
Algorithm 1: General Newton Algorithm
Input  : $f(\underline{x})$
Output: Optimal value of $\underline{x}$ that minimizes $f(\underline{x})$
begin
    $\underline{x}^{(0)} \leftarrow \mathbb{R}^{D\times 1}$
    repeat
        evaluate $\underline{x}^{(k+1)} = \underline{x}^{k} + \phi^{(k)}\Delta_{\underline{x}}^{(k)}$
        where $\Delta_{\underline{x}}^{(k)} = -H(\underline{x}^{(k)})^{-1}l(\underline{x}^{(k)})$ is Newton direction and $H(\underline{x})$ is
        Hessian and $l(\underline{x})$ is the gradient of $\underline{x}$ respectively and $\phi^{(k)}$ is the
        step size calculated using any line search algorithm.
    until convergence criteria is met ;
end
```

The challenge in this method is the calculation of Newton direction which involves Hessian. For low dimension, the Newton direction $\Delta_{\underline{x}} = -H(\underline{x})^{-1}l(\underline{x})$ can be calculated efficiently. For high dimension, the calculation of the Hessian and its inverse are expensive. So, we introduce an iterative algorithm which calculates the Newton direction. This process is shown in algorithm 2. During each update of the value of $\underline{x}$, the Newton direction $\Delta_{\underline{x}}$ is calculated through iterative process. The process of finding Newton direction (in Algorithm 2)

**Algorithm 2**: Newton Algorithm with Calculation of Newton Direction

```
Input  : $f(\underline{x})$
Output: Optimal value of $\underline{x}$ that minimizes $f(\underline{x})$
begin
    $\underline{x}^{(0)} \leftarrow \mathbb{R}^{D\times 1}$
    repeat
        evaluate $\underline{x}^{(k+1)} = \underline{x}^{k} + \phi^{(k)}\Delta_{\underline{x}}^{(k)}$
        find $\Delta_{\underline{x}}^{(k)}$
        repeat
            evaluate $\Delta_{\underline{x}}^{(k,l)} = \Delta_{\underline{x}}^{(k,l-1)} - \alpha_k \mathcal{H}(\underline{x}^{(k)}, \mathcal{H}(\underline{x}^{(k)}, \Delta_{\underline{x}}^{(k,l-1)}) + l(\underline{x}^{(k)}))$
            where $\mathcal{H}(x,\Delta) = H(x)\Delta = \frac{d}{dx}\langle l(x), \Delta\rangle$
        until convergence criteria is met ;
        update $\Delta_{\underline{x}}^{(k)} = \Delta_{\underline{x}}^{(k,l)}$
    until convergence criteria is met ;
end
```

will be discussed in detail in Section 4.2. The bilinear operator $\mathcal{H}(\underline{x}, \underline{\Delta})$ can also be generalized to matrices $\mathcal{H}(W, \Delta)$. The detailed proof of the bilinear operator $\mathcal{H}(\underline{x}, \underline{\Delta}) = H(\underline{x})\underline{\Delta} = \frac{d}{dx}\langle l(\underline{x}), \underline{\Delta}\rangle$ and generalization from vector to matrix form i.e., $\mathcal{H}(W, \Delta) = \frac{d}{dW}\langle L(W), \Delta\rangle$ can be found in Appendix B. Although the Algorithm 2 is shown for vectors, it can be easily generalized for matrices by replacing $\mathcal{H}(\underline{x}, \underline{\Delta})$ with $\mathcal{H}(W, \Delta) = \frac{d}{dW}\langle L(W), \Delta\rangle$. This change allows for an efficient calculation in terms of $W$. In Algorithm 3 we expressed the process in terms of matrix $W \in \mathbb{R}^{D\times D}$.

In Section 4.2., we will discuss different iterative methods used for calculating Newton direction.

**Algorithm 3**: Newton Algorithm with Calculation of Newton Direction, in Matrix Notation

```
Input  : $f(W)$
Output: Optimal value of $W$ that minimizes $f(W)$
begin
    $W^{(0)} \leftarrow \mathbb{R}^{D\times D}$  repeat
        evaluate $W^{(k+1)} = W^{(k)} + \phi^{(k)}\Delta_W^{(k)}$
        find $\Delta_W^{(k)}$
        repeat
            evaluate $\Delta_W^{(k,l)} = \Delta_W^{(k,l-1)} - \alpha^{(k)}\mathcal{H}(W^{(k)}, \mathcal{H}(W^{(k)}, \Delta_W^{(k,l-1)}) + l(W^{(k)}))$
            where $\mathcal{H}(W, \Delta_W) = \frac{d}{dW}\langle L(W), \Delta_W\rangle$
        until convergence criteria is met ;
        update $\Delta_W^{(k)} = \Delta_W^{(k,l)}$
    until convergence criteria is met ;
end
```

## 4.2. Newton Direction Calculation

The iterative methods such as one-step Landweber [32] and two-step accelerated Landweber iterations like [33],[34] are used to solve Newton direction iteratively when the matrix is of large size or ill conditioned.

### 4.2.1 Landweber Iteration

The Landweber method is an iterative approach to solve a set of linear equations of the form $y = Ax$. The solution $x$ is calculated by the following iteration

$$x^{(k+1)} = x^{(k)} + \alpha^{(k)} A^T (y - Ax^{(k)}). \tag{4.1}$$

An optimal step size is chosen to maximize reduction in the objective, i.e

$$\alpha^{(k)} = arg \min_{\alpha} \|A(x^{(k)} + \alpha^{(k)} A^T (y - Ax^{(k)})) - y\|^2. \tag{4.2}$$

Solving equation (4.2), by differentiating with respect to $\alpha$ and setting it to 0 yields

$$\alpha^{(k)} = \frac{\|A^T (y - Ax^{(k)})\|^2}{\|AA^T (y - Ax^{(k)})\|^2}. \tag{4.3}$$

We apply iteration (4.1) to

$$\mathcal{H}(W^{(k)}, \Delta_W) = -L(W^{(k)}) \tag{4.4}$$

to solve for $\Delta_W^k$ . We propose

$$\Delta_W^{(k,l)} = \Delta_W^{(k,l-1)} - \alpha_k \mathcal{H}(W^{(k)}, \mathcal{H}(W^{(k)}, \Delta_W^{(k,l-1)}) + L(W^{(k)})). \tag{4.5}$$

Notice that since the Hessian is square symmetric, the adjoint of $\mathcal{H}(W, \Delta_W)$ is equivalent to $\mathcal{H}(W, \Delta_W)$. Hence $\mathcal{H}(W, \Delta_W)$ is applied twice in (4.5). Due to the compact structure of the operator $\mathcal{H}(W, \Delta_W)$ (refer Appendix B) we do not explicitly calculate the Hessian. This operation reduces the time complexity and space requirement.

### 4.2.2 Two-Step Accelerated Landweber Method

To reduce computational complexity we consider the two-step accelerated Landweber method (e.g., [33], [34]), which makes use of last two previous values in update equation: $x^k$ and $x^{k-1}$. This use of history speeds up convergence rate [34]. If the equation is of the form $y = Ax$, then two step method increment will be as follows

$$x^{(k+1)} = (1-\beta^{(k)})x^{(k)} + \beta^{(k)}x^{(k-1)} + \alpha^{(k)}A^T(y - Ax^{(k)}). \tag{4.6}$$

Note that this equation can be reduced to the first order Landweber iterations by setting $\beta = 0$. The values of $\alpha^{(k)}$ and $\beta^{(k)}$ are chosen such that there is maximum decrease in per step iteration. We normalize the vectors in (C.1) in order to prevent numerical issues during implementation (refer to Appendix C for details). Equation (C.1) is now written as

$$x^{(k+1)} = x^{(k)} + \tilde{\beta}\frac{(x^{(k-1)} - x^{(k)})}{\|A(x^{(k-1)} - x^{(k)})\|} + \tilde{\alpha}\frac{A^T(y - Ax^{(k)})}{\|AA^T(y - Ax^{(k)})\|}. \tag{4.7}$$

The values of $\tilde{\alpha}^{(k)}$ and $\tilde{\beta}^{(k)}$ are:

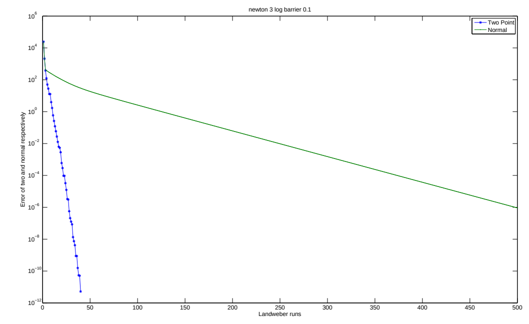
$$\begin{aligned}
\tilde{\alpha}^{(k)} &= \frac{\gamma_1 - \rho\gamma_2}{1-\rho^2} \\
\tilde{\beta}^{(k)} &= \frac{\gamma_2 - \rho\gamma_1}{1-\rho^2} \\
\gamma_1 &= \frac{\|A^T e^{(k)}\|^2}{AA^T e^{(k)}\|} \\
\gamma_2 &= \frac{-e^{(k)}(e^{(k-1)} - e^{(k)})}{\|e^{(k-1)} - e^{(k)}\|} \\
\rho &= -\frac{(e^{(k-1)} - e^{(k)})}{\|e^{(k-1)} - e^{(k)}\|}\frac{(AA^T e^{(k)})}{\|AA^T e^{(k)}\|} \\
e^{(k)} &= Ax^{(k)} - y
\end{aligned}$$

refer to Appendix C for details.

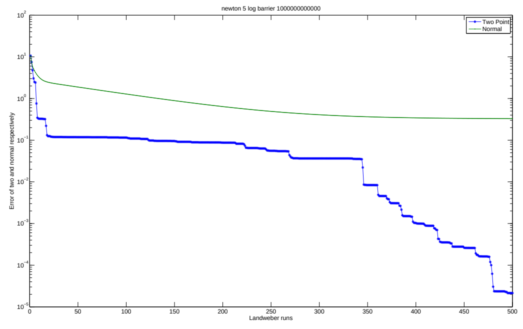
### 4.2.3 Comparison between One-Step Landweber and the Accelerated Two-Step Landweber

In two-step Landweber the convergence rate is faster compared to one-step Landweber iterations since the two previous values are used. The discussion related to convergence rate of two types (one-step and two-step) of Landweber iterations can be found in [34]. The author describes that number of iterations required to converge in one-step Landweber iterations are proportional to $\rho$ a condition number, but in accelerated two-step Landweber they are proportional to $\sqrt{\rho}$. We find the difference in convergence rates of two methods in our experiments. Figure 4.1 demonstrates the difference. Figure 4.1(a) represents the comparison between one-step and two-step Landweber method when the log barrier $t = 0.1$ in (4.4) and the Figure 4.1(b) when the log barrier $t = 10^{12}$. The figures are plotted based on the convergence rate of $\|\mathcal{H}(W^{(k)}, \Delta_W^{(k,l-1)}) + L(W^{(k)}))\|$ (the error term) as a function of $l$ (number of iterations) for the two methods. In accelerated two-step Landweber method we use the operator $\mathcal{H}(W, \Delta_W)$ thrice as compared to using it twice in one-step Landweber method. But the number of iterations required to reduce the error term by order of $10^4$ takes less than 50 iterations in accelerated two-step Landweber method while it takes atleast 250 iterations in one-step Landweber method (refer Figure 4.1(a)).

We used accelerated two-step Landweber iterations to calculate Newton direction iteratively. The use of bilinear operator $\mathcal{H}(W, \Delta) = \frac{d}{dW}\langle L(W), \Delta\rangle$ saves computational complexity as we need not explicitly calculate Hessian to find product of Hessian and a matrix, which is required in Landweber iterations.



(a) Two-point method and one-point Landweber convergence

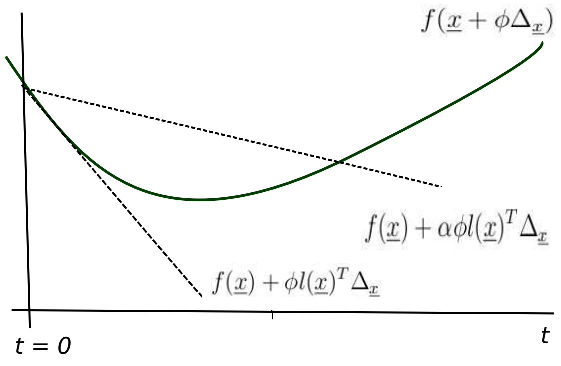


(b) Two point method and one-point Landweber convergence

FIGURE 4.1: Convergence rate versus no of iterations

## 4.3. Step Size using Backtracking

The next stage in our framework of implementation (see Figure 3.2 ) is step size calculation in optimal direction . This calculation can be done using various methods such as exact line search or backtracking. It is important to know the approximate step size, since small steps might take long time to reach the optimal step while large steps might cause to jump out of the optimal region. We need to choose a step size that gives sufficient decrease in function $f$ without spending a lot of time. We used the backtracking method in our implementation. The standard form of backtracking is found in [27], [31]. The pictorial representation is shown in figure (4.2) The step length is chosen to minimize function $f$ along $f(\underline{x} + \phi\Delta_{\underline{x}})$.


$f(\underline{x} + \phi\Delta_{\underline{x}})$
$f(\underline{x}) + \alpha\phi l(\underline{x})^T\Delta_{\underline{x}}$
$f(\underline{x}) + \phi l(\underline{x})^T\Delta_{\underline{x}}$
t = 0
t

(a) Backtracking

FIGURE 4.2: Representation of backtracking

The algorithm for backtracking is shown in algorithm 4. In backtracking method,

**Algorithm 4**: Backtracking Algorithm

```
Input  : a descent direction $\Delta_{\underline{x}}$ for $f$
         $\alpha \in (0, 0.5)$, $\beta \in (0, 1)$
Output: Optimal step size $\phi$
begin
    $\phi = 1$
    repeat
        update $\phi \leftarrow \beta\phi$
    until $f(\underline{x} + \phi\Delta_{\underline{x}}) \leq f(\underline{x}) + \alpha\phi l(\underline{x})^T\Delta_{\underline{x}}$ ;
end
```

initially the step size is chosen to be 1 and it is reduced by a factor of $\beta$ until the stopping criteria

$$f(\underline{x} + \phi\Delta_{\underline{x}}) \leq f(\underline{x}) + \alpha\phi l(\underline{x})^T\Delta_{\underline{x}} \tag{4.8}$$

is met. Equation (4.8) measures the decrease in the objective function $f$. The term $\alpha\phi l(\underline{x})^T \Delta_{\underline{x}}$ has negative slope (since $\Delta_{\underline{x}}$ is the descent direction), but the small positive value of $\alpha$ makes $f(\underline{x}) + \alpha\phi l(\underline{x})^T \Delta_{\underline{x}}$ lie above the curve $f(\underline{x} + \phi\Delta_{\underline{x}})$. Hence the step size calculated based on (4.8) will guarantee a decrease in objective function $f$ [27].

## 4.4. Time Complexity Analysis

In this section we describe the time complexity analysis of the operator $\mathcal{H}(W, \Delta_W)$ which is used in Landweber iterations to find Newton direction. We compare it to the time complexity involved in calculation of $-H(vec(W))^{-1}l(vec(W))$ which is a direct way of calculating Newton direction.

In our objective function Hessian multiplied by Newton step, is given by

$$\begin{aligned}\mathcal{H}(W, \Delta_W) = \frac{1}{N}\sum_i^N \sum_j^N I(c_i \neq c_j) e^{-\delta\tilde{X}_{ij}^T W \delta\tilde{X}_{ij}} \qquad (4.9)\\ (\delta\tilde{X}_{ij}\delta\tilde{X}_{ij}^T)(\delta\tilde{X}_{ij}^T \Delta_W \delta\tilde{X}_{ij})\\ - \frac{1}{t}(-(W^{-1}\Delta_W W^{-1})^T - ((I-W)^{-1}\Delta_W)(I-W))^T\\ - \frac{trace(\Delta_W)}{(\gamma - trace(W))^2} I).\end{aligned}$$

Let us split (4.9) into two parts:

$$\frac{1}{N}\sum_i^N \sum_j^N I(c_i \neq c_j) e^{-\delta\tilde{X}_{ij}^T W \delta\tilde{X}_{ij}} (\delta\tilde{X}_{ij}\delta\tilde{X}_{ij}^T)(\delta\tilde{X}_{ij}^T \Delta_W \delta\tilde{X}_{ij}) \qquad (4.10)$$

and

$$-\frac{1}{t}(-(W^{-1}\Delta_W W^{-1})^T - ((I-W)^{-1}\Delta_W)(I-W))^T - \frac{trace(\Delta_W)}{(\gamma - trace(W))^2}I). \quad (4.11)$$

Equation (4.10) yields a $D \times D$ matrix which has three major calculations, they are

$$\delta\tilde{X}_{ij}^T \Delta_W \delta\tilde{X}_{ij} \quad (4.12a)$$

$$\delta\tilde{X}_{ij}\delta\tilde{X}_{ij}^T \quad (4.12b)$$

$$e^{-\delta\tilde{X}_{ij}^T W \delta\tilde{X}_{ij}}. \quad (4.12c)$$

In this the time complexity of the term $(\delta\tilde{X}_{ij}^T \Delta_W \delta\tilde{X}_{ij})$ is $O(D^2)$ as $\delta\tilde{X}_{ij}^T$ is a vector of $1 \times D$ and $\Delta_W$ is a $D \times D$ matrix. The calculation of $\delta\tilde{X}_{ij}\delta\tilde{X}_{ij}^T$ is $O(D^2)$ and the calculation of $e^{-\delta\tilde{X}_{ij}^T W \delta\tilde{X}_{ij}}$ also takes $O(D^2)$. The double summation runs over all $N$ elements so the total time complexity is $O(N^2(D^2 + D^2 + D^2)) = O(N^2D^2)$. The time complexity for the log barrier part (4.11) is $O(D^3)$. $W$ is $D \times D$ matrix, the following table describes the time complexities for each of the operation. Hence the

| Operation on $D \times D$ matrix | Time Complexity |
|---:|---:|
| Inverse of a matrix | $O(D^3)$ |
| Trace of a matrix | $O(D)$ |
| Summation across all elements | $O(D^2)$ |

TABLE 4.1: Time complexity

total complexity of the whole equation is

$$O(D^3) + O(N^2D^2) = O(N^2D^2) \text{ when } N > D.$$

The time complexity of calculation of $-H^{-1}(vec(W))l(vec(W))$ is $O(N^2D^6)$. Since time complexity of $H^{-1}$ is $O(D^6)$ as $H$ has $D^2 \times D^2$ elements (refer table 4.1) and multiplication of $H^{-1}l$ is $O(D^4)$ as $H$ is $D^2 \times D^2$ and $l$ is $D^2 \times 1$ and this calculation has to be done for all $N \times N$ elements. Hence the total time complexity will be

$$O(N^2(O(D^4 + O(D^6)) = O(N^2D^6)$$

which is far higher than $O(N^2D^2)$ as $D$ increases.

Hence the time complexity of Landweber iterations to solve Newton direction is less compared to the direct calculation of Newton direction.

# 5. METHOD -2

To solve each problem of the form (3.3) in the sequence of problems in the log barrier method we propose L-BFGS method.

## 5.1. L-BFGS Quasi Newton Method

Though Newton method has advantages like fast convergence, it requires second order derivative and can be too expensive for large scale applications. The alternate to this is quasi Newton method, where second order derivative Hessian is calculated by approximation, which is based on corrections related to gradient $l(\underline{x})$ and position $\underline{x}$. Broyden-Fletcher-Goldfrab-Shanno (BFGS) is an effective algorithm, that calculates the inverse Hessian through approximation.For problems with large variables storing approximate Hessian inverse will cause issue related to space. L-BFGS (limited memory BFGS) addresses this problem by storing the approximate Hessian in compressed form that requires storing only constant $m$ vectors. We implemented L-BFGS (refer [29] [35]) method for optimization. The approximate Hessian inverse is calculate in the following way:

$$\begin{aligned} H_k^{-1} =& (V_{k-1}^T \cdots V_{k-m}^T)(H_k^{-1})^0(V_{k-m}^T \cdots V_{k-1}^T)+ \\ & \rho_{k-m}(V_{k-1}^T \cdots V_{k-m+1}^T)s_{k-m}s_{k-m}^T(V_{k-m+1}^T \cdots V_{k-1}^T)+ \\ & \cdots + \\ & \rho_{k-2}(V_{k-1}^T)s_{k-2}s_{k-2}^T(V_{k-1})+ \\ & \rho_{k-1}s_{k-1}s_{k-1}^T. \end{aligned} \tag{5.1}$$

where

$$
\begin{aligned}
s_k &= \underline{x}_{k+1} - \underline{x}_k \\
y_k &= l(\underline{x}_{k+1}) - l(\underline{x}_k) \\
\rho_k &= \frac{1}{y_k^T s_k} \\
V_k &= I - \rho_k y_k s_k^T \\
(H_k^{-1})^0 &= \text{ initial inverse Hessian approximation}
\end{aligned}
$$

The pair $\{s_{k-m}, y_{k-m}\}$ is the least recent pair and $\{s_{k-1}, y_{k-1}\}$ is the most recent pair. The matrix $H_k^{-1}$ is not formed explicitly but through $m$ values of $y$ and $s$.

## 5.2. Quasi Newton Direction

We have implemented L-BFGS two-loop recursion method in our project. The reader can refer to [29] for detailed discussion about the algorithm that describes the quasi-Newton direction calculation for L-BFGS efficiently.

## 5.3. Step Size using Backtracking

We calculate the appropriate step size in quasi -Newton direction using backtracking (refer Section 4.3.).

# 6. METHOD -3

To solve each problem of the form (3.3) in the sequence of problems in the log barrier method we use gradient descent method.

## 6.1. Gradient Descent Method

Gradient descent method is one of the oldest methods which minimizes the objective function along negative gradient. It has linear convergence rate. The advantage of this method is that it is simple to calculate and easy to use. The pseudo algorithm for gradient descent method is given in Algorithm 5. The gradient

**Algorithm 5**: General Gradient Descent Algorithm

```
Input  : $f(\underline{x})$
Output: Optimal value of $\underline{x}$ that minimizes $f(\underline{x})$
begin
    $\underline{x}^{(0)} \leftarrow \mathbb{R}^{D\times 1}$
    repeat
        evaluate $\underline{x}^{(k+1)} = \underline{x}^{k} + \phi^{(k)}\Delta_{\underline{x}}^{(k)}$
        where $\Delta_{\underline{x}}^{(k)} = -l(\underline{x}^{(k)})$ is descent direction, $l(\underline{x})$ is the gradient of $\underline{x}$ and $\phi^{k}$ is the
        step size calculated using any line search algorithm.
    until convergence criteria is met ;
end
```

descent algorithm in terms of $D \times D$ matrix $W$ can be represented as

**Algorithm 6**: Gradient Descent Algorithm in Terms of W

```
Input  : $f(W)$
Output: Optimal value of $W$ that minimizes $f(W)$
begin
    $W \leftarrow \mathbb{R}^{D\times D}$
    repeat
        evaluate $W^{(k+1)} = W^{(k)} + \phi^{(k)}\Delta_{W^{(k)}}$
        where $\Delta_{W^{(k)}} = -L(W^{(k)})$ is descent direction, $L(W)$ is the gradient of $W$ and $\phi^k$ is
        the step size calculated using any line search algorithm.
    until convergence criteria is met ;
end
```

## 6.2. Descent Direction

The descent direction is calculated by taking first order derivative of the objective function. For a $D \times D$ matrix the time and space complexity of Hessian calculation ($O(D^4)$) is greater than gradient ($O(D^2)$) calculation. The gradient calculation of our objective function (3.3) is $O(N^2D^2)$. The time complexity of Newton direction in Method -1 (refer chapter 4) of our implementation the time complexity of Newton direction calculation through Landweber iterations using the bilinear operator $\mathcal{H}(W, \Delta_W)$ is $O(N^2D^2)$, but it is calculated using constant number of iterations. We implemented gradient descent method to reduce time complexity. The convergence rate of gradient method is linear. The reader can refer to [36] for details related to convergence.

## 6.3. Step Size using Backtracking

We calculate the step size using backtracking (see Section 4.3.) in the descent direction obtained from the Section 6.2.

# 7. RANDOM SUBSET SELECTION METHOD

To overcome the time complexity and space complexity issues involved in the implementation, we tried a random subset selection method. The major contribution to time complexity is the distance calculation term in our objective function. To overcome that issue we randomly select $P$ data points out of total $N$ data points and calculate the distance between $P$ and $N$ points as opposed to calculating the distance between $N$ and $N$ points.

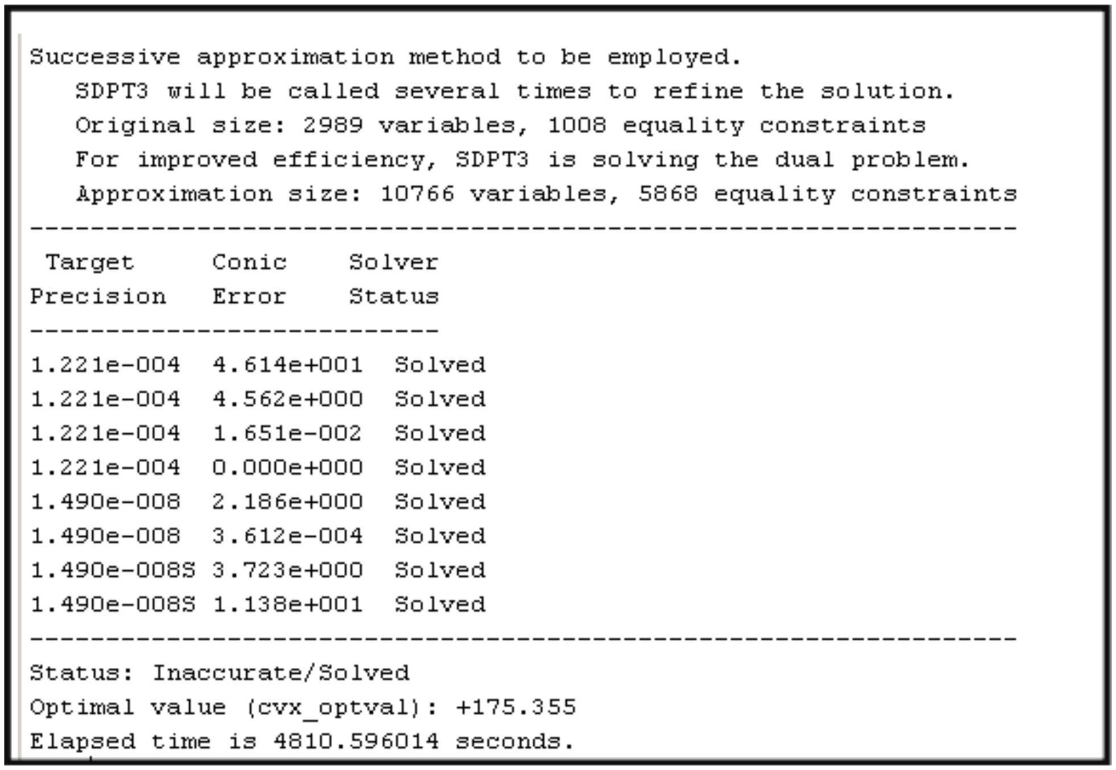
$$\widetilde{J}_p(W) = \frac{1}{P} \sum_{i=1}^{P} \sum_{j=1}^{N} I(c_i \neq c_j) e^{-\Delta \tilde{X}_{ij}^T W \Delta \tilde{X}_{ij}} \quad \text{where} \quad P \ll N \tag{7.1}$$

The time complexity (7.1) is $O(NPD^2)$ as dimension in equation (2.4) is reduced to $N \times P$ from $N \times N$.

We used CVX for solving (7.1). This method of selecting random subset of data points resolves the space complexity issue and CVX is executed successfully. The screen shot of it is present in Figure 7.1.

## 7.1. Complexity Comparisons between Total and Random Subset Data Points

Minimization of objective function (7.1) also reduces the space complexity significantly, as the space complexity of the distance matrix across all points of $X$



```
Successive approximation method to be employed.
   SDPT3 will be called several times to refine the solution.
   Original size: 2989 variables, 1008 equality constraints
   For improved efficiency, SDPT3 is solving the dual problem.
   Approximation size: 10766 variables, 5868 equality constraints
-----------------------------------------------------------------
 Target     Conic    Solver
Precision   Error    Status
--------------------------
1.221e-004  4.614e+001  Solved
1.221e-004  4.562e+000  Solved
1.221e-004  1.651e-002  Solved
1.221e-004  0.000e+000  Solved
1.490e-008  2.186e+000  Solved
1.490e-008  3.612e-004  Solved
1.490e-008S 3.723e+000  Solved
1.490e-008S 1.138e+001  Solved
-----------------------------------------------------------------
Status: Inaccurate/Solved
Optimal value (cvx_optval): +175.355
Elapsed time is 4810.596014 seconds.
```

(a) cvx success

FIGURE 7.1: Successful execution of cvx screen shot

where $X$ is $D \times N$ is $O(N^2)$ while the space complexity of distance matrix across $N$ points to $P$ points is $O(NP)$. This makes a huge difference when $P \ll N$. The classification accuracy calculated using K- Nearest Neighbours (KNN) classifier and Kernely Density Estimator (KDE) classifier for the minimization of (7.1) is comparable to that of original function (2.10). But we could find significant improvement in the time required for calculation, as $O(NPD^2) \ll O(N^2D^2)$ when $P \ll N$. This is illustrated in chapter 8.

# 8. NUMERICAL STUDY

For numerical analysis we have considered datasets from the UCI machine learning repository. Table 8.1 lists the datasets along with the following parameters: data dimension, data size (total number of available points), and the number of classes.

| Name | Dimension | Size | No:Classes |
|---:|---:|---:|---:|
| Wine | 13 | 178 | 3 |
| Glass | 8 | 214 | 5 |
| Balance | 4 | 625 | 3 |

TABLE 8.1: Data sets used in simulations

## 8.1. Analysis of Accuracy across Different Dimensions

The aim of this experiment is to analyze how the classification test error varies when we project the data from high dimension to different low dimensions.

### 8.1.1 Setup

We chose Method $-2$ (refer Chapter 5) framework of implementation with random initializations of $W$. We ran experiment for the Wine dataset for 10 different cross validations by projecting data into different low dimensions $(2 - 13)$. Using KDE and KNN classifiers we calculated the classification test error for each dimension $(2 - 13)$ across each of 10 validation sets. The Figure 8.1 and Figure 8.2 shows the mean and standard deviation of test error for each low dimension. We also ran the experiment just for one validation set for 10 times for each low dimension

$(2-13)$. The Figure 8.3 and Figure 8.4 shows the mean and standard deviation in classification test error calculated using KDE and KNN classifiers respectively.

### 8.1.2 Observations

We observe that there is not significant difference in the classification test error which is calculated using KDE and KNN classifiers across varied dimenisons for Wine dataset (refer Figure 8.1 and Figure 8.2).

### 8.1.3 Analysis

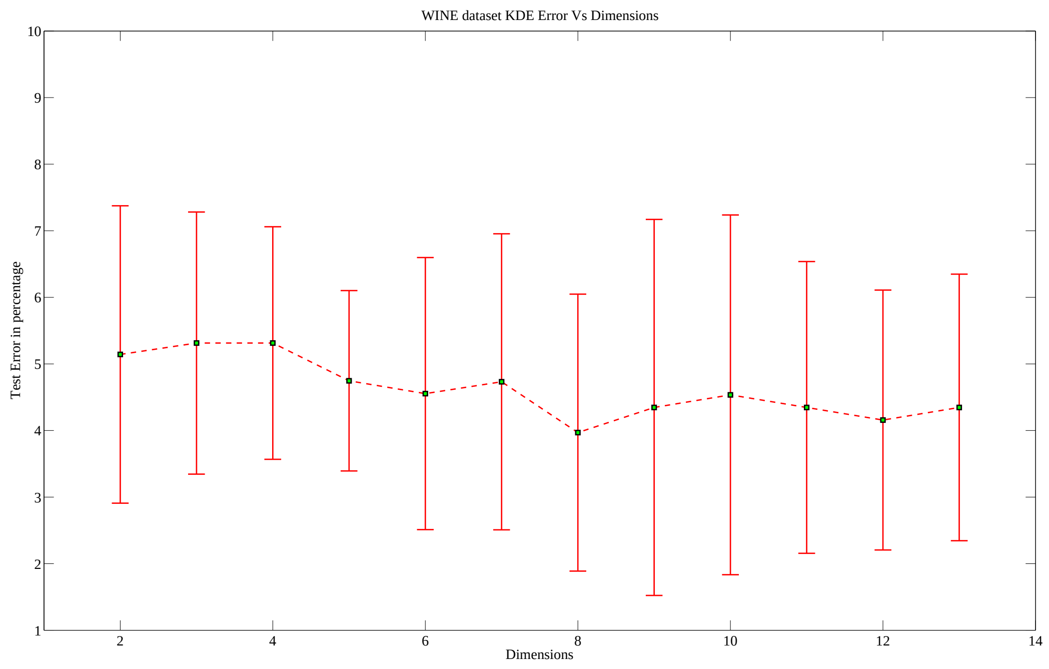
Based on the minimum value of classification test error in low dimensions (refer Figure 8.1) we can say that as dimension increase the error decrease, but based on the standard deviation we do not find significant difference in classification error across different dimensions for Wine dataset (refer Figure 8.1 and Figure 8.2). Figure 8.3 and Figure 8.4 represents classification test error for one validation set across different low dimenisons for KDE and KNN classifiers respectively. We cannot generalize the analysis based on one cross validation set.

## 8.2. Analysis of Run Time and Accuracy across Different Size of Random Subsets of Data

In this experiment we analyze the relation between run time, classification test error and number of training points.

### 8.2.1 Setup

We considered, the framework described in Chapter 7, i.e., we chose random subset of $P$ points from $N$ points. We implemented Method -2(refer chapter 5, L-



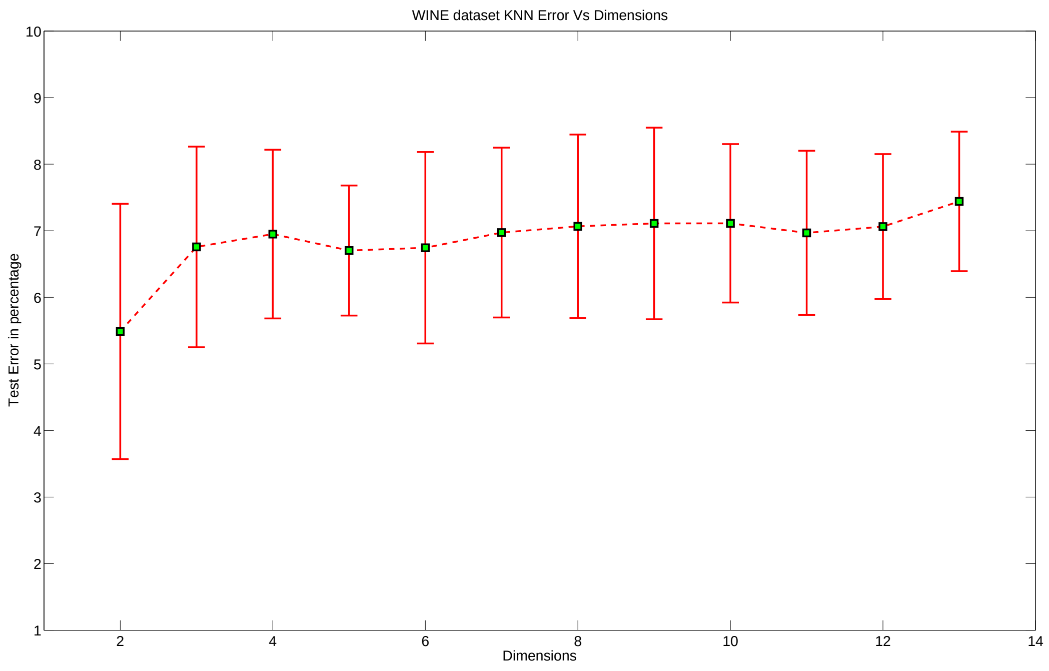
(a) Error versus Dimensions

FIGURE 8.1: Wine dataset KDE test error across different dimensions

BFGS optimization method) to solve (7.1) for random initializations of $W$. Analysis is done for the Balance data set by ranging the random subset of points ($P$) from 6 to 126 with step size of 10 for 10 cross validations. The mean and standard deviation of classification test error calculated by KDE and KNN classifiers for each random subset for 10 cross validations is recorded.

### 8.2.2 Observations

- We observes that run time increases almost linearly as the size of random subset of data points ($P$) increase (refer Figure 8.5).
- The classification test error calculated using KDE and KNN classifiers varies



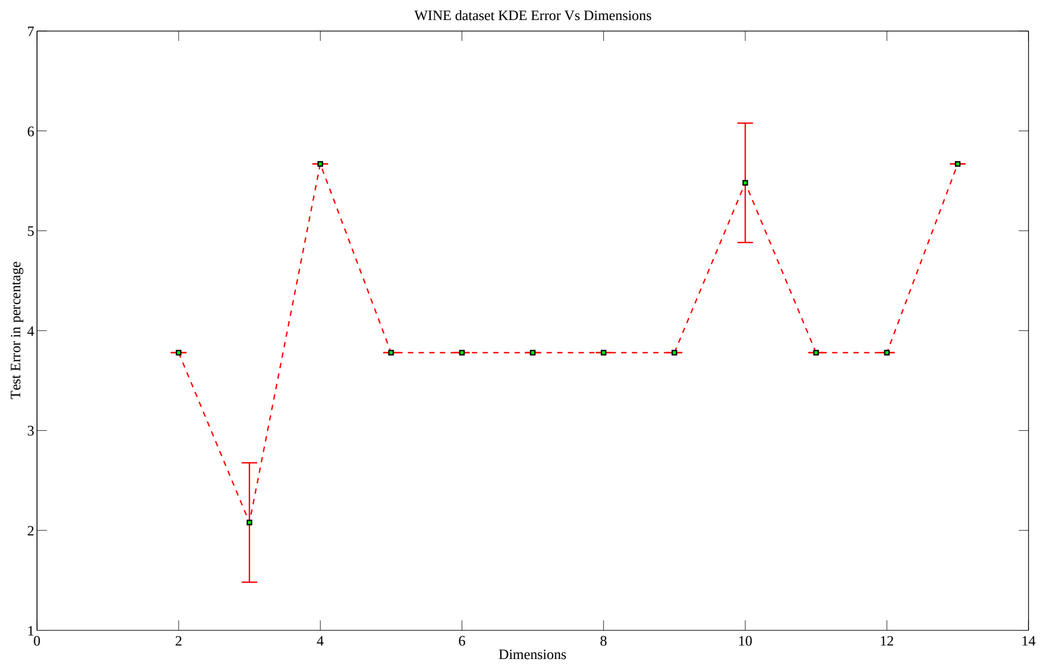
(a) Error versus Dimensions

FIGURE 8.2: Wine data set KNN test error across different dimensions

as the size of random subset data points vary (refer Figure 8.6 and Figure 8.7 respectively) and the standard deviation of classification test error is high when size of random subset $P$ is low.

### 8.2.3 Analysis

Though the run time decreases significantly when the size of random subset is small, we need to be watchful for accuracy rate we are compromising. We observe that standard deviation in the classification test error calculated using KDE and KNN classifiers for low value of $P$ is high when compared to large value of $P$. So we need to chose the correct proportion of subset where we can have advantage of



(a) Error versus Dimensions

FIGURE 8.3: Wine dataset KDE test error across different dimensions for same validation set

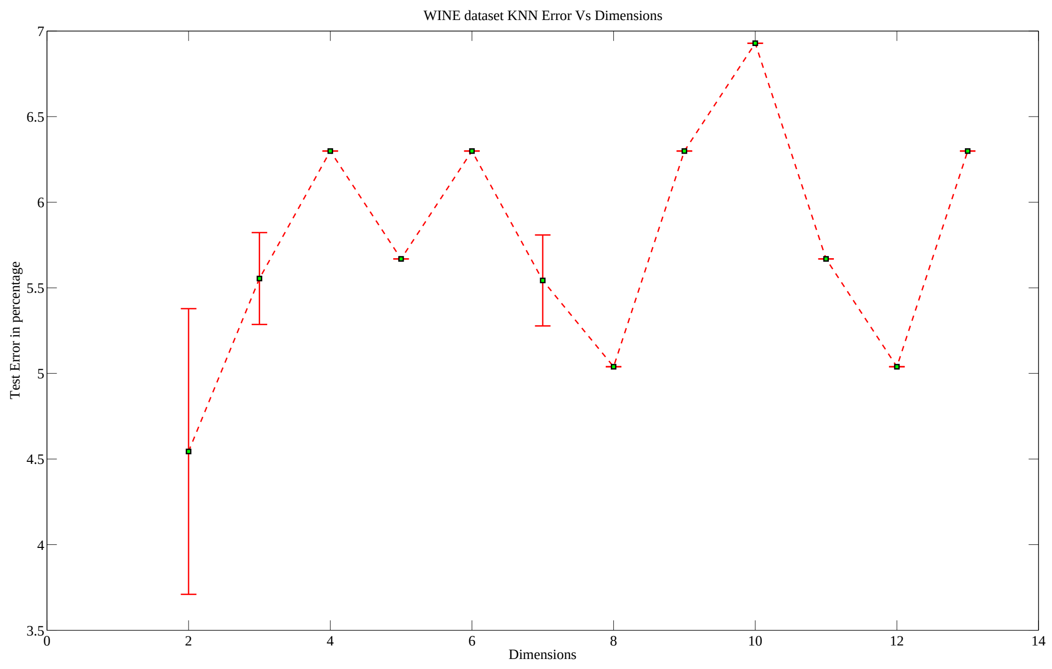
reducing time complexity and not compromise a lot regarding accuracy.

## 8.3. Analysis of Test Error for Multiple Runs

We analyzed the variation in classification test error in low dimension for different initializations.

### 8.3.1 Setup

We ran the experiment for convex problem (2.10) and non-convex problem (2.1). For convex problem we implemented Method-2 (refer Chapter 6) with random



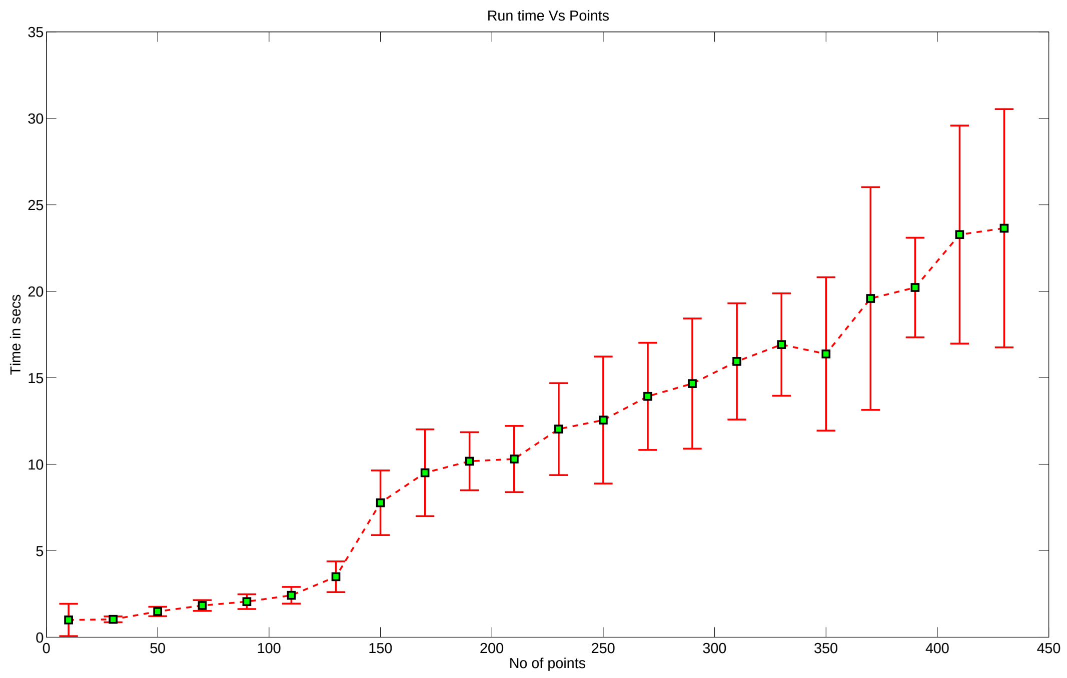
(a) Error versus Dimensions

FIGURE 8.4: Wine data set KNN test error across different dimensions for same validation set

initializations of $W$. In non-convex method we chose random initializations for $A$. The setup of experiment is to choose 10 random initializations for a particular cross validation set and the classification test error for different initialization across each validation set is recorded.

### 8.3.2 Observations

- When we used the convex objective function in this setup we observed that the classification test error calculated using KDE and KNN classifiers for each cross validation, for different initializations was quite constant i.e., standard deviation was very minimal which can be seen in Table 8.2.



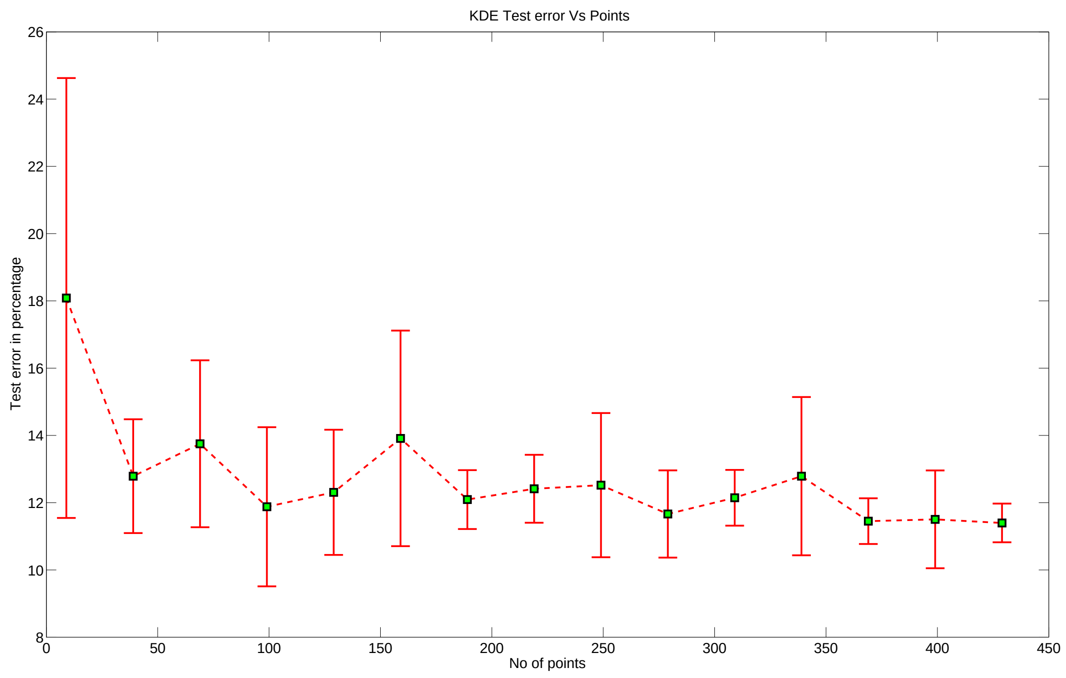
(a) Runtime Versus Training points

FIGURE 8.5: Balance data set run time for various random P training points

- In the same setup, when we used non convex objective function we find that standard deviation in classification test error calculated using KDE and KNN classifiers for each cross validation, for different initializations is high which is seen in Table 8.3.

### 8.3.3 Analysis

The convex function always guarantees a unique solution with in small deviation. The non-convex method is sensitive to initialization and often can converge to local optimal values.

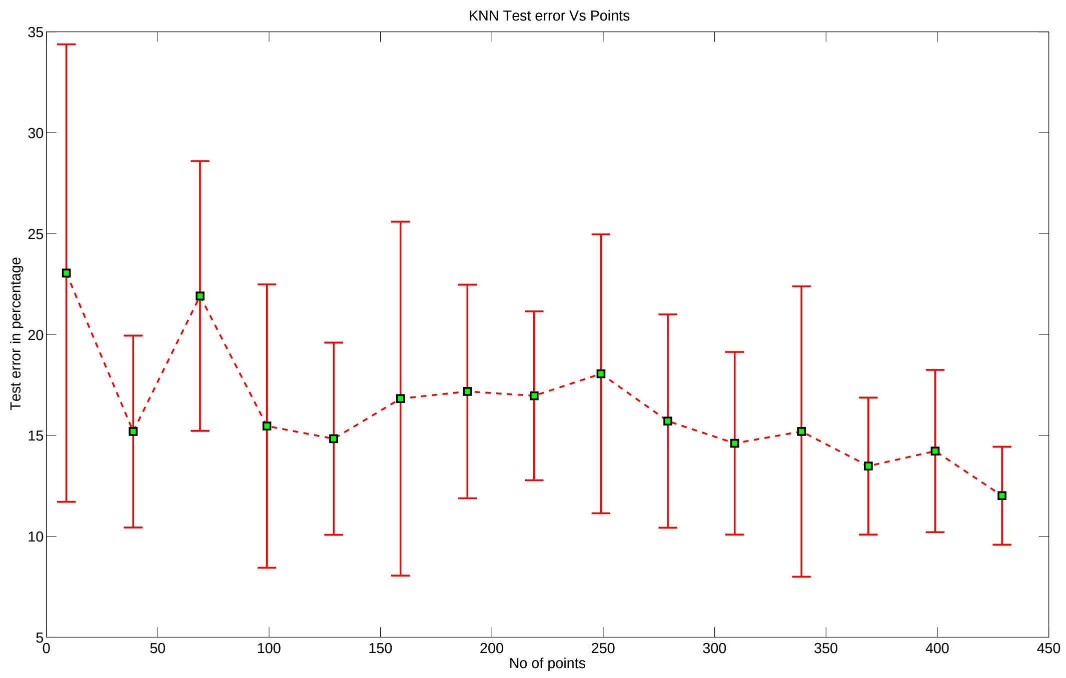


(a) KDE Test error Versus Training points

FIGURE 8.6: Balance data set KDE test error various random P training points

## 8.4. Analysis of Different Methods of Implementation

In this experiment, we analyze and compare run time and classification test error for different methods of implementation i.e., Method-1, Method-2, Method-3 (refer Chapter 4, Chapter 5, Chapter 6 respectively). We also analyze the relaxation in the eigenvalues of $W$. The eigenvalues of the original problem are $\lambda_1 = \lambda_2 = \cdots \lambda_d = 1; \lambda_{d+1} = \lambda_{d+2} = \cdots \lambda_D = 0$ where $d$ is required low dimension, and $D$ is the original high dimension we relaxed them to be between 0 and 1 i.e.,$0 \leq \lambda_1 = \lambda_2 = \lambda_3 = \cdots \lambda_D \leq 1$.



(a) KNN Test error Versus Training points

FIGURE 8.7: Balance data set KNN test error various random P training points

### 8.4.1 Setup

We ran the experiment for fixed cross validation set and same initialization of $W$ for fair comparison and recorded classification test error, run time and eigenvalues of $W$. The low dimension projection we used is 2.

### 8.4.2 Observation

- The eigenvalues of $W$ obtained using Method -1 implementation (refer Chapter 4, i.e., Newton method is used for optimization of objective function) are close to ideal solution $(\lambda_1 = \lambda_2 = \cdots \lambda_d = 1; \lambda_{d+1} = \lambda_{d+2} = \cdots \lambda_D = 0)$ (refer Table 8.5). The run time is higher compared to Method -2 (L-BFGS) and Method

| Cross Validation Set | Data Set | KDE Test Error | KNN Test Error |
|---|---|---|---|
| 1 | WINE | 3.7793±0 | 1.8896±2.34E-16 |
| 2 | WINE | 1.9274±0 | 3.8549±0 |
| 3 | WINE | 5.7067±0 | 3.85487±0 |
| 4 | WINE | 3.7793±0 | 1.9274±0 |
| 5 | WINE | 3.7793±0 | 5.7401±0 |
| 6 | WINE | 5.6689±0 | 3.7793±0 |
| 7 | WINE | 3.7793±0 | 3.7793±0 |
| 8 | WINE | 7.6675±0 | 5.7779±0 |
| 9 | WINE | 9.5905±0 | 9.5905±0 |
| 10 | WINE | 3.7793±0 | 3.7793±0 |

TABLE 8.2: Test error for different runs of convex method

| Cross Validation Set | Data Set | KDE Test Error | KNN Test Error |
|---|---|---|---|
| 1 | WINE | 2.8701±4.0815 | 3.0519±4.3417 |
| 2 | WINE | 4.8382±1.0015 | 4.6492±1.6352 |
| 3 | WINE | 10.9742±5.1479 | 10.3969±3.9785 |
| 4 | WINE | 4.5423±1.5898 | 4.7312±1.6107 |
| 5 | WINE | 6.1231±5.1940 | 6.8752±6.6618 |
| 6 | WINE | 8.8955±3.7132 | 9.2806±4.0852 |
| 7 | WINE | 4.1572±2.1453 | 5.6878±2.9705 |
| 8 | WINE | 4.9471±2.3889 | 4.7581±2.0391 |
| 9 | WINE | 7.9939±3.1261 | 7.805±3.2206 |
| 10 | WINE | 4.535±1.5935 | 4.3462±1.7926 |

TABLE 8.3: Test error for different runs of non convex method

-3(Gradient descent) implementations (refer Table 8.4).

- The eigenvalues of $W$ obtained using Method -2 (refer Chapter 5 ,i.e.,L-BFGS method is used for optimization of objective function) are similar to that of eigenvalues obtained by Method -1 (Newton method) (refer Table 8.5). The run time of Method-2 (L-BFGS) method is lower compared to that of Method -1 (Newton method) (refer Table 8.4).

- The eigenvalues obtained using Method -3 (refer Chapter 6, gradient descent method is used for optimization of objective function) are not close to the ideal solution ($\lambda_1 = \lambda_2 = \cdots \lambda_d = 1; \lambda_{d+1} = \lambda_{d+2} = \cdots \lambda_D = 0$) (refer Table 8.5). The run time of this method is lower compared to other two methods (Newton and L-BFGS methods) (refer Table 8.4)

### 8.4.3 Analysis

This experiment reaffirms that Newtom methods are more computationally more expensive compared to Gradient descent method. Gradient descent method requires more number of iterations to reach to ideal solution ($\lambda_1 = \lambda_2 = \cdots \lambda_d = 1; \lambda_{d+1} = \lambda_{d+2} = \cdots \lambda_D = 0$) in our problem. We find that our relaxation works and approximately gives the desired results (as $d = 2$ it makes top 2 eigenvalues almost to 1 and tries to make the rest of eigenvalues close to 0) (refer Table 8.5).

| Data Set | Method | KDE test error | KNN test error | Time in secs |
|---:|---:|---:|---:|---:|
| WINE | Gradient Descent | 0.0574 | 0.0578 | 157.760127 |
| WINE | L-BFGS | 0.0574 | 0.0385 | 289.64671 |
| WINE | Newton Method | 0.0574 | 0.0578 | 2534.924443 |

TABLE 8.4: Test error and time for different implementations

| Eigenvalues by Gradient descent | Eigenvalues by L-BFGS | Eigenvalues by Newton |
|---:|---:|---:|
| 0.889 | 0.9991 | 1 |
| 0.7402 | 0.829 | 0.8606 |
| 0.1802 | 0.1138 | 0.1165 |
| 0.1206 | 0.0492 | 0.0222 |
| 0.0359 | 0.0035 | 0.0002 |
| 0.0196 | 0.0015 | 0.0002 |
| 0.0145 | 0.0009 | 0.0001 |
| 0 | 0.0007 | 0.0001 |
| 0 | 0.0007 | 0 |
| 0 | 0.0007 | 0 |
| 0 | 0.0003 | 0 |
| 0 | 0.0003 | 0 |
| 0 | 0.0001 | 0 |

TABLE 8.5: Eigenvalues of solution for different implementations

# 9. CONCLUSION

## 9.1. Summary

In this project we provide a convex formulation for dimensionality reduction. We discussed the various methods of implementations like Newton method, L-BFGS method, gradient descent method. In Newton method we used Landweber iterations to calculate the Newton direction which significantly reduces the time complexity compared to direct Newton direction calculation. We also proposed another implementation method where we chose random subset of data points to reduce the time and space complexity. We presented a detailed numerical analysis which demonstrates the relations between the dimensions and accuracy, size of subset of data points and run time, size of subset of data points and test error and run time for different methods of implementation.

## 9.2. Future work

Currently in our implementation, the L-BFGS method is giving the desired results but time and space complexity might increase for large data sets. The random selection of subset of data points reduces the time and space complexity issues with a compromise in accuracy, which is not desirable. Future work will be to reduce the time complexity and space complexity issues without comprising in the solution.

# BIBLIOGRAPHY

1. M.K. Thangavelu, “On error bounds for linear feature extraction,” M.S. thesis, Oregon State University, 2009.
2. H. Kim, P. Howland, and H. Park, “Dimension reduction in text classification with support vector machines,” *The Journal of Machine Learning Research*, vol. 6, pp. 37–53, 2005.
3. E. Bingham and H. Mannila, “Random projection in dimensionality reduction: applications to image and text data,” in *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining.* ACM, 2001, pp. 245–250.
4. J. Ye, R. Janardan, and Q. Li, “GPCA: an efficient dimension reduction scheme for image compression and retrieval,” in *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining.* ACM, 2004, pp. 354–363.
5. B. Sarwar, G. Karypis, J. Konstan, and J. Riedl, “Analysis of recommendation algorithms for e-commerce,” in *Proceedings of the 2nd ACM conference on Electronic commerce*, 2000, pp. 158–167.
6. H. Hotelling, “Analysis of a complex of statistical variables into principal components,” *Journal of Educational Psychology*, vol. 24, pp. 417–441., 1933.
7. A. Hyvärinen and E. Oja, “Independent component analysis: algorithms and applications,” *Neural networks*, vol. 13, no. 4-5, pp. 411–430, 2000.
8. S.T. Roweis and L.K. Saul, “Nonlinear Dimensionality Reduction by Locally Linear Embedding,” *Science*, vol. 290, no. 5500, pp. 2323–2326, 2000.
9. M. Belkin and P. Niyogi, “Laplacian Eigenmaps for Dimensionality Reduction and Data Representation,” *Neural Computation*, vol. 15, no. 6, pp. 1373–1396, 2003.
10. W. L. Poston and D. J. Marchette, “Recursive dimensionality reduction using fisher’s linear discriminant - inference and applications to clustering,” *Pattern Recognition*, vol. 31, pp. 881–888(8), 31 July 1998.
11. D.R. Hardoon, S. Szedmak, and J. Shawe-Taylor, “Canonical correlation analysis: An overview with application to learning methods,” *Neural Computation*, vol. 16, no. 12, pp. 2639–2664, 2004.

12. I.K. Fodor, "A survey of dimension reduction techniques," *Livermore, CA: US DOE Office of Scientific and Technical Information*, vol. 18, 2002.

13. J.B. Kruskal, "Multidimensional scaling by optimizing goodness of fit to a nonmetric hypothesis," *Psychometrika*, vol. 29, no. 1, pp. 1–27, 1964.

14. L.J.P Van der Maaten, E.O Postma, and H.J Van den Herik, "Dimensionality reduction: A comparative review," *Published online*, vol. 10, no. February, pp. 1–41, 2007.

15. M. Collins, S. Dasgupta, and R.E. Schapire, "A generalization of principal component analysis to the exponential family," *Advances in neural information processing systems*, vol. 1, pp. 617–624, 2002.

16. O.A Sajama and A. Orlitsky, "Semi-parametric exponential family PCA," *Advances in Neural Information Processing Systems*, vol. 17, pp. 1177–1184, 2004.

17. Y. Guo and D. Schuurmans, "Efficient global optimization for exponential family PCA and low-rank matrix factorization," in *Communication, Control, and Computing, 2008 46th Annual Allerton Conference on*. IEEE, 2009, pp. 1100–1107.

18. Y. Guo, "Supervised exponential family principal component analysis via convex optimization," in *NIPS*, 2009, vol. 21, pp. 569–576.

19. Jacob Goldberger, Sam T. Roweis, Geoffrey E. Hinton, and Ruslan Salakhutdinov, "Neighbourhood components analysis," in *NIPS*, 2004.

20. L. Vandenberghe and S. Boyd, "Semidefinite programming," *SIAM review*, vol. 38, no. 1, pp. 49–95, 1996.

21. M.S. Lobo, L. Vandenberghe, S. Boyd, and H. Lebret, "Applications of second-order cone programming* 1," *Linear Algebra and its Applications*, vol. 284, no. 1-3, pp. 193–228, 1998.

22. Y. Nesterov and A. Nemirovskii, "Interior-point polynomial algorithms in convex programming," 1994.

23. M. Grant and S. Boyd, "CVX: Matlab software for disciplined convex programming, version 1.21," `http://cvxr.com/cvx`, Feb. 2011.

24. Z.Q. Luo, "Applications of convex optimization in signal processing and digital communication," *Mathematical Programming*, vol. 97, no. 1, pp. 177–207, 2003.

25. E.F. Arruda, M.D. Fragoso, and J.B.R. do Val, "An application of convex optimization concepts to approximate dynamic programming," in *American Control Conference, 2008*. IEEE, 2008, pp. 4238–4243.

26. E.P. Xing, A.Y. Ng, M.I. Jordan, and S. Russell, "Distance metric learning with application to clustering with side-information," *Advances in neural information processing systems*, pp. 521–528, 2003.

27. S.P. Boyd and L. Vandenberghe, *Convex optimization*, Cambridge Univ Pr, 2004.

28. T.J. Ypma, "Historical development of the Newton-Raphson method," *SIAM review*, vol. 37, no. 4, pp. 531–551, 1995.

29. S.J. Benson and J.J. Moré, "A Limited Memory Variable Metric Method in Subspaces and Bound Constrained Optimization Problems," *mathematical, Information and Computer Science Division, Argonne National Laboratory, ANL/MCS- P*, vol. 901, 2001.

30. A.V. Fiacco and G.P. McCormick, "The sequential unconstrained minimization technique for nonlinear programing, a primal-dual method," *Management Science*, vol. 10, no. 2, pp. 360–366, 1964.

31. H. Hindi, "A tutorial on convex optimization II: Duality and interior point methods," in *Proceedings of the 2006 American Control Conference*, 2006, pp. 686–696.

32. L. Landweber, "An iteration formula for Fredholm integral equations of the first kind," *American journal of mathematics*, vol. 73, no. 3, pp. 615–624, 1951.

33. AS Nemirovski and BT Polyak, "Iterative methods for solving linear ill-posed problems under precise information I, Moscow Univ," *Comput. Math. Cybern*, vol. 22, pp. 1–11, 1984.

34. M. Hanke, "Accelerated Landweber iterations for the solution of ill-posed equations," *Numerische mathematik*, vol. 60, no. 1, pp. 341–373, 1991.

35. D.C. Liu and J. Nocedal, "On the limited memory BFGS method for large scale optimization," *Mathematical programming*, vol. 45, no. 1, pp. 503–528, 1989.

36. J. Barzilai and J.M. Borwein, "Two-point step size gradient methods," *IMA Journal of Numerical Analysis*, vol. 8, no. 1, pp. 141–148, 1988.

# APPENDIX

## A Appendix 1

The following is a proof of one to one mapping between sets $S_A$ and $S_W$, where $S_A = \{A \mid A \in \mathbb{R}^{d\times D}, AA^T = I_{d\times d}\}$ and $A \sim S_{A_0} = \{A \mid A = RA_0\}$ where $R$ is rotation matrix. $S_W = \{W \mid W^T, \quad W^2 = W, \quad Rank(W) = d, \quad W \in \mathbb{R}^{D\times D}, \quad W = A^T A\}$. We split the proof into two steps. In the first step, we show mapping from $S_A$ to $S_W$ and in the second step we show mapping from $S_W$ to $S_A$.

Step 1: We need to prove $\forall A$ in $S_{A_0}$ are mapped to $W$ in $S_W$.

We know that $A^T A = W$, substituting any equivalent member which is formed by the rotation in place of $A$ results $(RA)^T(RA) = A^T R^T RA = A^T A = W$. Hence all $A$'s in the equivalence class are mapped to unique $W$ and that $W$ has properities such as $W = W^2, W = W^T, Rank(W) = d$. The proof of each property is show below.

Property 1: $W = W^2$ when $A^T A = W$

$$\begin{aligned}
W^2 &= (A^T A)(A^T A) \\
&= A^T A A^T A \text{ since } AA^T = I \\
&= A^T A \\
&= W
\end{aligned}$$

Hence proved.

Property 2: $W = W^T$ when $A^T A = W$

$$\begin{aligned} W^T &= (A^T A)^T \\ &= A^T (A^T)^T \\ &= A^T A \\ &= W \end{aligned}$$

Hence proved.

Property 3: $Rank(W) = d$ when $A^T A = W$

Let the Singular Value Decomposition (SVD) of $A$ be $V_1 S U_1^T$ where $V_1$ and $U_1$ are orthogonal matrices and $S = diag(\sigma_1, \cdots \sigma_d)$. $W = A^T A = U_1 S^2 U_1^T$ i.e., $W$ has an eigen value decomposition with $d$ non zero eigen values. Hence $Rank(W) = d$.

Step 2:

We need to prove that $W$ from $S_W$ is mapped to an equivalence class in $S_A$.

Proof: Since $W = W^T$, eigen value decomposition exists for $W$ i.e., $EVD(W) = U \Lambda U^T$ and as $W = W^2$ we get $(\Lambda)^2 = (\Lambda) \Rightarrow \lambda_i \in \{0, 1\}$ and also we know that $Rank(W) = d$ so based on $Rank(W) = d$ and $\lambda_i \in \{0, 1\}$. The eigen value decompositon of $W$ is expressed as

$$W = [U_1, U_2] \begin{bmatrix} I_d & 0 \\ 0 & 0 \end{bmatrix} [U_1 U2]^T \tag{A.1}$$

$$W = U_1 U_1^T$$

We can see that even if $U_1$ multiplied by any rotation matrix it will always yields the same result. Hence $W$ from $S_W$ is mapped to equivalence class of $A$.

Hence we proved the one to one mapping between $S_A$ and $S_W$.

## B Appendix 2

The value of by linear operator $\mathcal{H}(\underline{x}, \Delta) = H(\underline{x})\underline{\Delta}$ is shown as follows.

$$\begin{aligned}
[H(\underline{x})\underline{\Delta}]_i =& \sum_{j=1}^{N} \frac{d^2 f(\underline{x})}{dx_i dx_j} \underline{\Delta}_j \\
& \frac{d}{dx_i} \sum_{j=1}^{N} \frac{df(\underline{x})}{dx_j} \underline{\Delta}_j \\
& \frac{d}{dx_i} \sum_{j=1}^{N} l(\underline{x}) \underline{\Delta}_j \\
& \frac{d}{dx_i} \langle l(\underline{x}), \underline{\Delta} \rangle
\end{aligned}$$

Hence $\mathcal{H}(\underline{x}, \underline{\Delta}) = \frac{d}{dx} \langle l(\underline{x}), \underline{\Delta} \rangle$. The operator mapping is $\mathbb{R}^N \to \mathbb{R}^N$, where $N$ are the number of elements in vectors $l(\underline{x})$ and $\underline{\Delta}$ . This can be generalised for $W \in \mathbb{R}^{D \times D}$ as

$$\mathcal{H}(W, \Delta) = \frac{d}{dW}(trace(L(W)^T \Delta) = \frac{d}{dW} \langle L, \Delta \rangle \tag{B.1}$$

where $\mathcal{H}(W, \Delta), L, \Delta \in \mathbb{R}^{D \times D}$ and the mapping of operator $\mathcal{H}(W, \Delta)$ is $\mathbb{R}^{D \times D} \to \mathbb{R}^{D \times D}$.

The detail formulation of vector to matrix generalization is discussed below.

Let

$$W = \begin{pmatrix} w_{11} & w_{12} & w_{13} \cdots & w_{1n} \\ w_{21} & w_{22} & w_{23} \cdots & w_{2n} \\ \vdots & & & \\ w_{n1} & w_{n2} & w_{n3} \cdots & w_{nn} \end{pmatrix}$$

vectorizing a matrix is concatenating columns one below another.

$$vec(W) = \begin{pmatrix} w_{11} \\ w_{21} \\ w_{n1} \\ w_{12} \\ w_{22} \\ w_{n2} \\ \vdots \\ w_{1n} \\ w_{2n} \\ w_{nn} \end{pmatrix}$$

In the same way

$$vec(\Delta) = \begin{pmatrix} \Delta_{11} \\ \Delta_{21} \\ \Delta_{n1} \\ \Delta_{12} \\ \Delta_{22} \\ \Delta_{n2} \\ \vdots \\ \Delta_{1n} \\ \Delta_{2n} \\ \Delta_{nn} \end{pmatrix}$$

Then gradient matrix $L$ is

$$L = \begin{pmatrix} \frac{df}{dw_{11}} & \frac{df}{dw_{12}} & \frac{df}{dw_{13}} \cdots & \frac{df}{dw_{1n}} \\ \frac{df}{dw_{21}} & \frac{df}{dw_{22}} & \frac{df}{dw_{23}} \cdots & \frac{df}{dw_{2n}} \\ \vdots & & & \\ \frac{df}{dw_{n1}} & \frac{df}{dw_{n2}} & \frac{df}{dw_{n3}} \cdots & \frac{df}{dw_{nn}} \end{pmatrix}$$

vector form of gradient is $l$

$$vec(L) = l = \frac{df}{dvec(W)} \begin{pmatrix} \frac{df}{dw_{11}} \\ \frac{df}{dw_{21}} \\ \frac{df}{dw_{n1}} \\ \frac{df}{dw_{12}} \\ \frac{df}{dw_{22}} \\ \frac{df}{dw_{n2}} \\ \vdots \\ \frac{df}{dw_{1n}} \\ \frac{df}{dw_{2n}} \\ \frac{df}{dw_{nn}} \end{pmatrix}$$

Original matrices are represented with capital letters and vectorised forms are represented with lower case letters.

$$l^T \Delta w = \begin{bmatrix} \frac{df}{dw_{11}} & \frac{df}{dw_{21}} & \cdots & \frac{df}{dw_{1n}} & \cdots & \frac{df}{dw_{nn}} \end{bmatrix} \begin{bmatrix} \Delta_{11} \\ \Delta_{21} \\ \vdots \\ \Delta_{1n} \\ \vdots \\ \Delta_{nn} \end{bmatrix} \tag{B.2}$$

$$l^T \Delta = \frac{df}{dw_{11}} \Delta_{11} + \cdots \frac{df}{dw_{nn}} \Delta_{nn} \tag{B.3}$$

Now we need to find $trace(L^T\Delta)$. We know that

$$L = \begin{bmatrix} \begin{bmatrix} l_1 \end{bmatrix} \begin{bmatrix} l_2 \end{bmatrix} \cdots \begin{bmatrix} l_n \end{bmatrix} \end{bmatrix} \tag{B.4}$$

$$\Delta = \begin{bmatrix} \begin{bmatrix} \Delta_1 \end{bmatrix} \begin{bmatrix} \Delta_2 \end{bmatrix} \cdots \begin{bmatrix} \Delta_n \end{bmatrix} \end{bmatrix} \tag{B.5}$$

In (B.4) and (B.5) each column in the matrix is represented as vector.

$$L^T\Delta W = \begin{bmatrix} l_1^T \\ l_2^T \\ \vdots \\ l_n^T \end{bmatrix} \begin{bmatrix} \Delta_1 & \Delta_2 & \cdots\Delta_n \end{bmatrix} \tag{B.6}$$

$$L^T W = \begin{pmatrix} l_1^T\Delta_1 & l_1^T\Delta_2\cdots & l_1^T\Delta_n \\ l_2^T\Delta_1 & l_2^T\Delta_2\cdots & l_2^T\Delta_n \\ \vdots & & \\ l_n^T\Delta_1 & l_n^T\Delta_2\cdots & l_n^T\Delta_n \end{pmatrix}$$

where $l_i^T = [\frac{df}{dw_{1i}}\frac{df}{dw_{2i}}\cdots\frac{df}{dw_{ni}}]$ and $\Delta_i = \begin{bmatrix} \Delta_{1i} \\ \Delta_{2i} \\ \vdots \\ \Delta_{ni} \end{bmatrix}$. As we know the trace of a matrix is sum of the diagonal elements we proved that

$l^T\Delta = trace(L^T\Delta)$

Hence the vector to matrix generalization for the operator $\mathcal{H}(W, \Delta)$ is shown.

## C Appendix 3

To solve an equation of the form $y = Ax$, we propose the following iteration

$$x^{(k+1)} = (1 - \beta^{(k)})x^{(k)} + \beta^{(k)}x^{(k-1)} + \alpha^{(k)}A^T(y - Ax^{(k)}) \tag{C.1}$$

We are interested in reducing the norm of the error: $e^{(k)} = Ax^{(k)} - y$. We start by evaluating the error. Multiply (C.1) with $A$ and subtract $y$ :

$$Ax^{(k+1)} - y = Ax^{(k)} - y + A\beta^{(k)}(x^{(k-1)} - x^{(k)}) + \alpha^{(k)}AA^T(y - Ax^{(k)}). \tag{C.2}$$

Representing equation (C.2) in terms of $e^{(k)}$:

$$e^{(k+1)} = e^{(k)} + \beta^{(k)}(e^{(k-1)} - e^{(k)}) - \alpha AA^T e^{(k)}. \tag{C.3}$$

To minimize $\|e^{(k+1)}\|^2$ w.r.t $\alpha$ and $\beta$ one can minimize $\|e^{(k)} + \beta^{(k)}(e^{(k-1)} - e^{(k)}) - \alpha AA^T e^{(k)}\|^2$. To avoid numerical errors we propose the normalized reprsentation of the error.

$$e^{(k+1)} = e^{(k)} + \tilde{\beta}^{(k)}\frac{(e^{(k-1)} - e^{(k)})}{\|e^{(k-1)} - e^{(k)}\|} + \tilde{\alpha}\frac{-(AA^T e^{(k)})}{\|AA^T e^{(k)}\|}. \tag{C.4}$$

Now we have to

$$\text{minimize}||\underbrace{\widetilde{e^{(k)}}}_{y} - (\underbrace{[V_1 V_2]}_{H}\underbrace{[\tilde{\alpha}^{(k)}\tilde{\beta}^{(k)}]^T}_{\theta})||^2 \tag{C.5}$$

where $V_1 = \frac{(AA^T e^{(k)})}{\|AA^T e^{(k)}\|}$ and $V_2 = -\frac{(e^{(k-1)} - e^{(k)})}{\|e^{(k-1)} - e^{(k)}\|}$. Using Least squares (i.e.,$\hat{\theta}_{ls} = (H^T H)^{-1} H^T y$) we solve for $\tilde{\alpha}$ and $\tilde{\beta}$ as:

$$\begin{bmatrix} \tilde{\alpha}^{(k)} \\ \tilde{\beta}^{(k)} \end{bmatrix} = \begin{bmatrix} \begin{bmatrix} V_1^T \\ V_2^T \end{bmatrix} \begin{bmatrix} V_1 & V_2 \end{bmatrix} \end{bmatrix}^{-1} \begin{bmatrix} V_1^T e^{(k)} \\ V_2^T e^{(k)} \end{bmatrix} \tag{C.6}$$

$$\begin{bmatrix} \tilde{\alpha}^{(k)} \\ \tilde{\beta}^{(k)} \end{bmatrix} = \underbrace{\begin{bmatrix} \begin{bmatrix} V_1^T \\ V_2^T \end{bmatrix} \begin{bmatrix} V_1 & V_2 \end{bmatrix} \end{bmatrix}}_{M} \begin{bmatrix} V_1^T e^k \\ V_2^T e^k \end{bmatrix} \tag{C.7}$$

In the matrix $M$ the elements $M_{11}$ and $M_{22}$ value will be 1 as they are normalized and the value of $M_{12}$ and $M_{21}$ will be equal and the value is

$$\rho = -\frac{(e^{(k-1)} - e^{(k)})}{\|e^{(k-1)} - e^{(k)}\|} \frac{(AA^T e^{(k)})}{\|AA^T e^{(k)}\|}.$$

The values of $\tilde{\alpha}^{(k)}$ and $\tilde{\beta}^{(k)}$ are

$$\begin{aligned} \begin{bmatrix} \tilde{\alpha}^{(k)} \\ \tilde{\beta}^{(k)} \end{bmatrix} &= \begin{bmatrix} 1 & \rho \\ \rho & 1 \end{bmatrix}^{-1} \begin{bmatrix} \gamma_1 \\ \gamma_2 \end{bmatrix} \\ &= \tfrac{1}{1-\rho^2} \begin{bmatrix} 1 & -\rho \\ -\rho & 1 \end{bmatrix} \begin{bmatrix} \gamma_1 \\ \gamma_2 \end{bmatrix} \\ &= \tfrac{1}{1-\rho^2} \begin{bmatrix} \gamma_1 - \rho\gamma_2 \\ \gamma_2 - \rho\gamma_1 \end{bmatrix} \end{aligned} \tag{C.8}$$

This implies $\tilde{\alpha}^{(k)} = \frac{\gamma_1 - \rho\gamma_2}{1-\rho^2}$ and $\tilde{\beta}^{(k)} = \frac{\gamma_2 - \rho\gamma_1}{1-\rho^2}$ where $\gamma_1 = \frac{\|A^T e^{(k)}\|^2}{AA^T e^{(k)}\|}$ and $\gamma_2 = \frac{-e^{(k)}(e^{(k-1)} - e^{(k)})}{\|e^{(k-1)} - e^{(k)}\|}$ . Hence the update is

$$x^{(k+1)} = x^{(k)} + \tilde{\beta}\frac{(x^{(k-1)} - x^{(k)})}{\|e^{(k-1)} - e^{(k)}\|} - \tilde{\alpha}\frac{A^T(e^{(k)})}{\|AA^T e^{(k)}\|}. \tag{C.9}$$