Conduction of Supervised Classification Using Several Methods of Metric Learning

by

Bolun Xu

A Thesis

Presented to the Graduate and Research Committee

of Lehigh University

in Candidacy for the Degree of

Master of Science

in

Industrial and Systems Engineering

Lehigh University

05. 2017

Certificate of approval

This thesis is accepted and approved in partial fulfillment of the requirements for the Master of Science.

_____

Date

_____

Thesis Advisor

_____

Co-Advisor

_____

Chairperson of Department

Acknowledgements

First, I would like to thank my thesis advisor Professor Martin Takac of P.C. Rossin College of Engineering and Applied Science. The door to his office was always open whenever I ran into a trouble spot or had a question about my project.

I would also like to thank people in Exxon Mobil who were involved in the work for this project: Dimitri J. Papageorgiou and Krishnan Kumaran. Without their patient participation and theoretical instruction, the practice work could not have been successfully conducted.

Contents

Abstract

Nowadays, unsupervised classification and supervised classification are common ways in performing data analysis to perform a range of functions like anomaly detection and diagnosis, data segmentation and model development. Hence, there is a large body of research on these topics offering different solutions. This thesis states extension clustering work to a supervised classification method that is capable of learning the optimal distance/similarity metric directly from training data. This approach using an adaptive distance metric has the potential to provide results of better quality than state-of-the-art techniques like Support Vector Machines. Also, this thesis detailedly states method to implement models [e.g. 3.1, 4.1] in practice. Most of models are conducted in C++ environment with IBM CPLEX.

1. Problem statement

Suppose we are given N points $\mathbf{v}_i \in \mathbb{R}^p$ with class membership $\mathcal{C}_i$ for $i \in \mathcal{N} = \{1, \dots, N\}$. Our target is to find a distance metric $\mathbb{D} : \mathbb{R}^p \to \mathbb{R}$ such that the following conditions hold:

1. $\mathbb{D}(\mathbf{v}_i, \mathbf{v}_j) = \mathbf{a}^\top(\mathbf{v}_i - \mathbf{v}_j) + (\mathbf{v}_i - \mathbf{v}_j)^\top \mathbf{B}(\mathbf{v}_i - \mathbf{v}_j) \quad \forall i, j \in \mathcal{N}$, for some $\mathbf{a} \in \mathbb{R}^p$ and $\mathbf{B} \in \mathbb{R}^{p \times p}$.
2. $\mathbb{D}(\mathbf{v}_i, \mathbf{v}_j) \geq 0 \quad \forall i, j \in \mathcal{N}$
3. $\min_{j \in \mathcal{C}_i} \mathbb{D}(\mathbf{v}_i, \mathbf{v}_j) < \min_{k \notin \mathcal{C}_i} \mathbb{D}(\mathbf{v}_i, \mathbf{v}_j) \quad \forall i \in \mathcal{N}$

Notice that conditions above are not true distance metric since symmetry and the triangle property are not enforced. An important condition could be extended to include higher order terms. Let $\delta_{ij} = \mathbf{v}_i - \mathbf{v}_j$ for all $i, j \in \mathcal{N}$. Finding such a distance metric can be expressed as the following optimization problem:

$$\max_{\lambda, \mathbf{a}, \mathbf{B}, \mathbf{d}} \quad \lambda \tag{1a}$$

$$s.t. \quad \min_{j \in \mathcal{C}_i} d_{ij} + \lambda \leq \min_{k \notin \mathcal{C}_i} d_{ik} \qquad \forall i \in \mathcal{N} \tag{1b}$$

$$d_{ij} = \mathbf{a}^\top \boldsymbol{\delta_{ij}} + \boldsymbol{\delta_{ij}}^\top \mathbf{B}\, \boldsymbol{\delta_{ij}} \qquad \forall i, j \in \mathcal{N} \tag{1c}$$

$$d_{ij} \geq 0 \; \forall i, j \in \mathcal{N} \qquad \forall i, j \in \mathcal{N} \tag{1d}$$

$$\mathbf{a} \in \mathbb{R}^p \tag{1e}$$

$$\mathbf{B} \in \mathbb{R}^{p \times p}, \text{ symmetric} \tag{1f}$$

Let $\lambda_i = \min_{k \notin \mathcal{C}_i} d_{ik} - \min_{j \in \mathcal{C}_i} d_{ij}$ be the difference between the distance from point i to its nearest non-neighbor and the distance from point i to its nearest neighbor. In ideal

situatons, $\lambda_i > 0$ for all i. However, this might not be possible when the training data is not perfect. We call i an "outlier" if $\lambda_i < 0$. The thesis will give a discussion and possible solution to this situation in 4.1. In formulation (1), we would like to maximize the minimum separation over all points as can be seen by re-writing constraint (1b) as

$$\lambda \leq \lambda_i \quad \forall i \in \mathcal{N}$$

and noting that the objective function is to maximize λ.

2. Basic Environment and Parameter Settings

2.1 Statement

C++, is actually a very low level language for power, efficiency, control, and granularity. With C++, we can link directly by calling API of original solvers (e.g. CPLEX) without redundance and deal with very large scale of data. Hence, C++ is considered to be more effective environment than object-oriented programming language such as python or java. Here we use IBM cplex 12.7 in Microsoft Visual Studio 2013.

2.2 Basic settings linking to CPLEX

Before we start to implement formulations, Configuring the environment is a necessary

and vital procedure. If we do this wrong, no code will pass the compilation.

Using Microsoft Visual studio in windows-x64 with its own compiler:

a) Include the following files in “AdditionalIncludeDirectories” :

$\IBM\ILOG\CPLEX_Studio127\concert\include;

$\IBM\ILOG\CPLEX_Studio127\cplex\include;

b) Add the following preprocessors in “PreprocessorDefinitions”:

NDEBUG;_CONSOLE;IL_STD;

IL_STD is the most important. Several lines of crucial code call vector and map function in STD to achive our goals.

c) Add the following in “Linkers”:

$\cplex\lib\x64_windows \stat_mda\cplex1270.lib;

$\cplex\lib\x64_windows \stat_mda\ilocplex.lib;

$\IBM\ILOG\CPLEX_Studio127\concert\lib\x64_windows\stat_mda\concert.lib;

System Linux_x86-64 with compiler gcc:

It is similar with the settings above with those libs and preprocessors.

Before implementing any methods relating to cplex, we should include

<ilcplex/ilocplex.h> in the code.

3. Mixed-Integer Linear Optimization Approach

3.1 Max-Min Fomulations: Maximize the minimum separation with outliers:

Let $\mathcal{F}$ denote the feasible region associated with the distance metric parameters, i.e.

$$\mathcal{F} = \{(\mathbf{a}, \mathbf{B}, \mathbf{d}) \in \mathbb{R}^p \times \mathbb{R}^{p\times p} \times \mathbb{R}_+^{N\times N}: \mathbf{B} \text{ symmetric}, d_{ij} = \mathbf{a}^\top \boldsymbol{\delta}_{ij} + \boldsymbol{\delta}_{ij}{}^\top \mathbf{B}\, \boldsymbol{\delta}_{ij} \quad \forall i, j \in \mathcal{N}\} \quad (2)$$

Or

$$\mathcal{F} = \left\{ \mathbf{a}, \mathbf{B}, \mathbf{d}: \right.$$

$$d_{ij} = \sum_{k=0}^{p} \delta_{ijk} a_k + \sum_{k=0}^{p} \delta_{ijk}^2 b_{kk} + 2\sum_{k=1}^{p-1}\sum_{l=k+1}^{p} \delta_{ijk}\delta_{ijl}\, b_{kl} \quad \forall i, j \in \mathcal{N} \quad (3a)$$

$$0 \le d_{ij} \le 1 \quad \forall i, j \in \mathcal{N} \quad (3b)$$

$$\mathbf{a} \in \mathbb{R}^p \quad (3c)$$

$$b_{kl} \in \mathbb{R} \quad \forall k \in 1, \dots, \mathrm{p}, \mathrm{l} = \mathrm{k}, \dots, \mathrm{p} \quad \} (3d)$$

Model (1) can be expressed as a huge mixed-integer linear programming problem:

$$\max_{\boldsymbol{\lambda}, \mathbf{B}, \mathbf{d}, \mathbf{w}, \mathbf{y}, \mathbf{z}} \quad f(\lambda) = \sum_{i\in\mathcal{N}} (\lambda_i - \rho z_i) \quad (4a)$$

$$s.t. \quad \sum_{j\in\mathcal{C}_i} w_{ij} + \lambda_i \le d_{ik} + \mathrm{M} z_i \quad \forall i \in \mathcal{N}, k \in \mathcal{N}\backslash\mathcal{C}_i \quad (4b)$$

$$\lambda_i \le 1 - z_i \quad \forall i \in \mathcal{N} \quad (4c)$$

$$w_{ij} \le d_{ij} \quad \forall i \in \mathcal{N}, j \in \mathcal{C}_i \quad (4d)$$

$$w_{ij} \le y_{ij} \quad \forall i \in \mathcal{N}, j \in \mathcal{C}_i \quad (4e)$$

$$w_{ij} \le y_{ij} + d_{ij} - 1 \quad \forall i \in \mathcal{N}, j \in \mathcal{C}_i \quad (4f)$$

$$\sum_{j\in\mathcal{C}_i} y_{ij} = 1 \quad \forall i \in \mathcal{N} \quad (4g)$$

$$w_{ij} \ge 0 \quad \forall i \in \mathcal{N}, j \in \mathcal{C}_i \quad (4h)$$

$$y_{ij} \in \{0, 1\} \quad \forall i \in \mathcal{N}, j \in \mathcal{C}_i \quad (4i)$$

$$z_i \in \{0,1\} \qquad \forall i \in \mathcal{N} \qquad \text{(4j)}$$

$$\lambda_i \geq 0 \qquad \forall i \in \mathcal{N} \qquad \text{(4k)}$$

(3) (4l)

Formulation (4) assumes a linear reward for positive separation and penalty ρ for each outlier. Outliers are handled through the binary z_i decision variables. If z_i = 1(i is an outlier), then constraint (4c) ensures that the separation λ_i = 0 and hence ignored. In the mean time, constraint (4b) can be always true and meaningless since M in (4b) is a large number; Otherwise (if z_i = 0), Constraints (4b) and (4c) become effective.

Constraints (4d)-(4f) are the so-called McCormick envelopes needed to linearize the bilinear equation $w_{ij} = y_{ij} d_{ij}$.

3.3 Implement in C++:

Implementing a model includes three main procedures: define variables, set constraints and solve the problem.

a) Define variables: for our convenience to define Matrix variables, we should first define several types performing variable arrays in an array. The following code shows type definition with 2 dimensional variable Matrix:

```
typedef IloArray<IloNumArray>       NumMatrix;
typedef IloArray<IloNumVarArray> NumVarMatrix;
typedef IloArray<IloIntVarArray> IntVarMatrix;
```

typedef IloArray<NumMatrix> Num3Matrix;

Several functions shows definitions with different need:

public ***IloNumVar****(const IloEnv env, IloNum lb=0, IloNum ub=IloInfinity, IloNumVar::Type type=Float, const char * name=0);*

public ***IloNumVarArray****(const IloEnv env,const IloNumArray lb,const IloNumArray ub, IloNumVar::Type type=ILOFLOAT);*

public ***IloIntVar****(IloEnv env, IloInt vmin=0, IloInt vmax=IloIntMax, const char * name=0);*

public ***IloBoolVar****(IloEnv env, const char * name);*

With 4 functions and type definition about matrix above, we can define almost all the variables. Take the following vector $\mathbf{a}$ as an instance:

IloNumVarArray a(env, d, -IloInfinity, IloInfinity, ILOFLOAT);

Note that distance metric

$$\boldsymbol{B} = \begin{bmatrix} b_{00} & \cdots & b_{0k} \\ \vdots & \ddots & \vdots \\ b_{k0} & \cdots & b_{kk} \end{bmatrix},$$

if we define all of the variables in **B**, we get $p \times p = p^2$ variables. In constraint (3a), the

real variables we use is less than p^2 since **B** is a symmetric matrix. Thus we can modify **B** as an upper triangular matrix

$$\boldsymbol{B} = \begin{bmatrix} b_{00} & \cdots & b_{0k} \\ 0 & \ddots & \vdots \\ 0 & 0 & b_{kk} \end{bmatrix}.$$

Then we can discard p(p-1)/2 abundant variables. When p is very large, efficiency of problem solving will be promoted. Code is as followed:

```
NumVarMatrix B(env, d);
for (int i = 0; i < d; i++)
        {
        B[i] = (IloNumVarArray(env, d-i, -IloInfinity, IloInfinity, ILOFLOAT));////
define half top-right of the matrix B without over-define variables}
```

b) Set constraints: By setting loop over i and j, we can add one constraint at a time, until all the work is done. A simple way to add constraint is to use "add" function directly. In this model, we use "add" function. Code as example is as followed:

```
model.add(w[i][j] <= y[i][j]);
```

Also, we will meet problems that the expression of formulation is too long. To deal with the problem, we call function as is followed:

```
public IloExpr(const IloEnv env, IloNum val=0);
```

By calling the function above, we can get a temporary expression. The expression can be added multiple times without losing former formulation. Therefore, cleaning memory space is important when we are using this function.

c) Solve the problem: By calling “IloCplex”, we can set Cplex as the solver and solve the problem. Code is as followed:

```
IloCplex cplex(env);
cplex.solve();
```

If we want to get objective value from the solver, here is the function we can call:

```
cplex.getObjValue();
```

Get other values by calling:

```
cplex.getValue(z[i]);
```

Here we use z_i as an example.

4. Linear Optimization Approach

4.1 Iterative LP Algorithm

From 3.1, we can easily transform original problem into an MILP (Mixed Integer Linear Programming) Problem. Yet MILP method in 3.1 cannot candle data in large scale in a relatively short time. Data from project and work can be always in a large scale. Hence we need to formulate problem in an easy way for computer to solve.

One of the most effective way is to formulate the problem in several LPs (linear Programming).

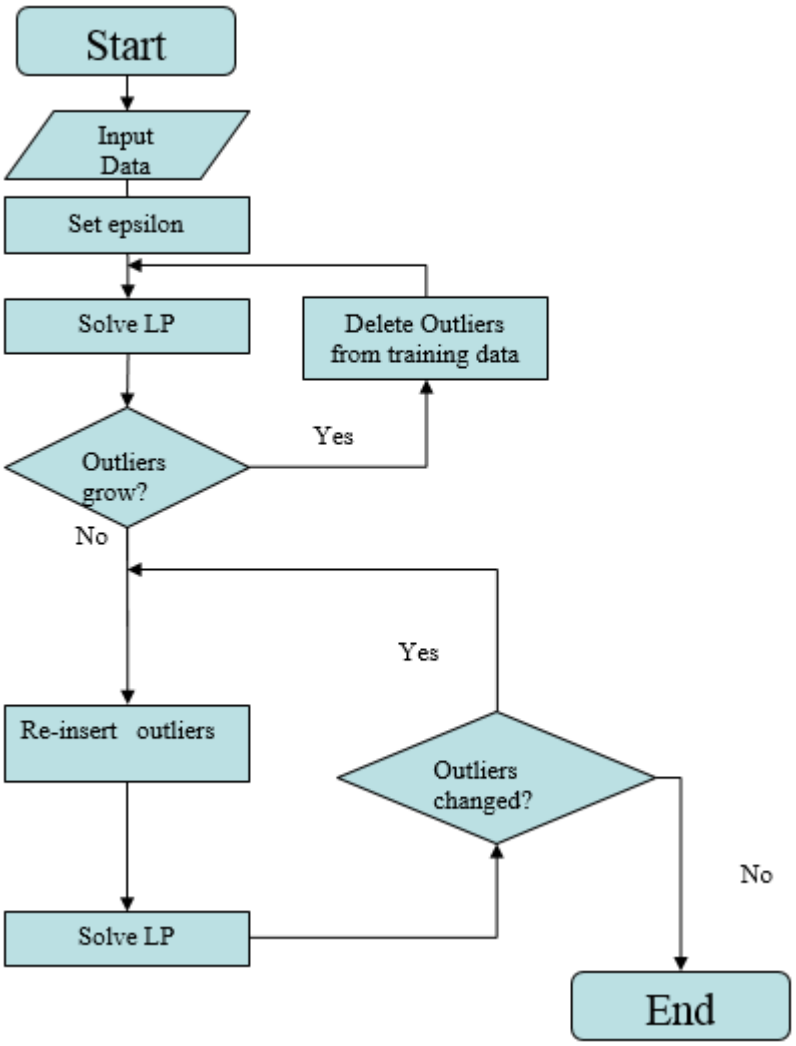


Figure 1 Iterative LP Algorithm

Figure 1 shows an iterative LP Algorithm to learn distance metric. In this chapter, the detail of the procedures before outliers re-insertion will be stated.

4.1 Learn Metric with LP Approach

4.1.1 Formulations

Given $\epsilon \in (0, \epsilon^{\max}]$, model (1) can be expressed as a linear programming feasibility

problem:

$$\max_{\mathbf{a},\mathbf{B},\mathbf{d},\mathbf{t}} \quad \sum_{i\in\mathcal{N}} t_i \tag{5a}$$

$$s.t.\ t_i + \epsilon \le d_{ik} \qquad \forall i \in \mathcal{N}, k \in \mathcal{N}\backslash\mathcal{C}_i \tag{5b}$$

$$t_i \le d_{ij} \qquad \forall i \in \mathcal{N}, j \in \mathcal{C}_i \tag{5c}$$

$$t_i \ge 0 \qquad \forall i \in \mathcal{N} \tag{5d}$$

$$(3) \tag{5e}$$

With $\epsilon \in (0, \epsilon^{\max}]$, formulation (5) is always feasible since solution $t_i = 0$ for all $\forall i \in \mathcal{N}$ is feasible. Note that ti represents the distance to nearest neighbor of point i since constraints (5c) ensure that $t_i \le \min_{j\in\mathcal{C}_i} d_{ij}$, while the objective function encourages $t_i = \min_{j\in\mathcal{C}_i} d_{ij}$ for all i. Meanwhile, constraint (5b) ensure that $\min_{k\in\mathcal{N}\backslash\mathcal{C}_i} d_{ik} - \min_{j\in\mathcal{C}_i} d_{ij} \ge \epsilon$. If model (5) returns a feasible solution with $t_i^* = \min_{j\in\mathcal{C}_i} d_{ij}^* > 0$ for all $i \in \mathcal{N}$, then the distance metric classify the points perfectly. Otherwise there exists a subset $\mathcal{O}$ of outlier points such that $t_i^* = \min_{k\in\mathcal{N}\backslash\mathcal{C}_i} d_{ik}^* < \min_{j\in\mathcal{C}_i} d_{ij}$ for all $i \in \mathcal{O}$. This situation shows that no distance metric guaranteeing a minimum separation exists for the choice of ϵ.

4.1.2 Implementation

By declaring several Boolean variables, we can set the expression of d_{ij} easily in constraint (3a). In the part of code below, If *include_first_order_terms* = 0, there is no

vector $\mathbf{a}$ in the model.

```
if (include_first_order_terms)
{
    D_ij_temp += fabs(features[i][k] - features[j][k]) * a[k];
}
D_ij_temp += (features[i][k] - features[j][k]) * (features[i][k] - features[j][k]) *
B[k][0];
}
for (int p = 0; p < d - 1; p++)
{
    for (int l = p + 1; l < d; l++)
    {
        D_ij_temp += 2 * (features[i][p] - features[j][p])* (features[i][l] -
features[j][l]) * B[p][l - p];
    }
}
    model.add(D[i][j] == D_ij_temp);
    D_ij_temp.end();
```

4.2 Learn Metric with LP Approach via Row and Column Generation (Fast LP)

A barrier in the LP above is the huge number of constraints if number of instances in training data is over thousand. For example, an instance with 1000 points, 10 classes, and 100 points per class would require 900,000 constraints for each (i, k) pair. With more points, the problem get worse.

Meanwhile, the fundamental theorem of linear programming asserts that if an optimal solution exists, there exists an optimal basic feasible solution that will involve n active constraints, where $n = N + (p + 1)p/2$ and m $(\mathcal{O}(N^2))$. Obviously, $n << m$.

Our target is to largely reduce the number of constraints expect for those must be included in LP. Let $\mathcal{N}_A \subseteq \mathcal{N}$ be the set of active points considered in the LP. Let $\mathcal{S}_i \subseteq \mathcal{C}_i$ and $\bar{\mathcal{S}}_i \subseteq \bar{\mathcal{C}}_i$ for each point $i \in \mathcal{N}$. Here, $\mathcal{S}_i$ ($\bar{\mathcal{S}}_i$) represents the set of co-class (non-class) neighbors that are currently considered in the LP.

4.2.1 Formulations

Consider the following reduced LP that maximizes both ϵ and the minimum distance variables t_i:

$$\max_{\mathbf{a},\mathbf{B},\mathbf{d},\mathbf{t}} \ \sum_{i\in\mathcal{N}} t_i \tag{6a}$$

$$s.t.\ t_i + \epsilon \le d_{ik} \qquad \forall i \in \mathcal{N}_A, k \in \bar{\mathcal{S}}_i \tag{6b}$$

$$t_i \le d_{ij} \qquad \forall i \in \mathcal{N}_A, j \in \mathcal{S}_i \tag{6c}$$

$$t_i \ge 0 \qquad \forall i \in \mathcal{N}_A \tag{6d}$$

$$(3) \tag{6e}$$

Algorithm LP Approach via Row and Column Generation (Fast LP)	
1:	Initialize: $\mathcal{N}_A = \mathcal{N}; \mathcal{O} = \emptyset;\ \mathcal{S}_i = \{\text{argmin}\{d_{ij}: j \in \mathcal{C}_i\}\};$

	$\overline{\mathcal{S}_i} = \{\text{argmin}\{d_{ik}: k \in \mathcal{N}\backslash\mathcal{C}_i\}\}$; Fix ϵ; **B** = Euclidean distance;
2:	DONE = False
3:	**While** DONE = False **do**
4:	DONE = True
5:	(Re-)Solve LP (6) for $\mathbf{B}^*, d_{ij}^*, t_i^*$. Fix the distance metric to d_{ij}^*.
6:	**For** $i \in \mathcal{N}_A$ **do**
7:	Set $\mathcal{V}_i = \{ j \in \mathcal{C}_i \backslash \mathcal{S}_i : t_i^* > d_{ij}^* \}$.
8:	Set $\bar{\mathcal{V}}_i = \{ k \in \bar{\mathcal{C}}_i \backslash \overline{\mathcal{S}_i} : t_i^* + \epsilon > d_{ik}^* \}$.
9:	Set $\mathcal{D}_i = \{ k \in \mathcal{N} : d_{ik}^* > 1 \}$.
10:	Set $\overline{\mathcal{D}_i} = \{ j \in \mathcal{N} : d_{ij}^* < 0 \}$.
11:	**If** $\lvert\mathcal{V}_i\rvert > 0$ **then** $\hat{\jmath} = \text{argmax}\{t_i^* - d_{ij}^*: j \in \mathcal{V}_i\}$ and set $\mathcal{S}_i = \mathcal{S}_i \cup \{\hat{\jmath}\}$; DONE = False;
12:	**If** $\lvert\bar{\mathcal{V}}_i\rvert > 0$ **then** $\hat{k} = \text{argmax}\{t_i^* + \epsilon - d_{ik}^*: k \in \bar{\mathcal{V}}_i\}$ and set $\overline{\mathcal{S}_i} = \overline{\mathcal{S}_i} \cup \{\hat{k}\}$; DONE = False;
13:	**If** $\lvert\overline{\mathcal{D}_i}\rvert > 0$ **then** $\hat{\jmath} = \text{argmin}\{d_{ij}^*: j \in \overline{\mathcal{D}_i}\}$ and set $\overline{\mathcal{D}_i} = \overline{\mathcal{D}_i} \cup \{\hat{\jmath}\}$; DONE = False;
14:	**If** $\lvert\mathcal{D}_i\rvert > 0$ **then** $\hat{k} = \text{argmax}\{d_{ik}^*: k \in \mathcal{D}_i\}$ and set $\mathcal{D}_i = \mathcal{D}_i \cup \{\hat{k}\}$; DONE = False;
15:	**end for**

16:	**End while**

Table 1 Algorithm: LP Approach via Row and Column Generation (Fast LP)

4.2.2 Implementation

In this problem, the key point is to create minimum number of variables D_ij. To implement the algorithm, “map” function in STL must be applied. Key code is as followed:

```
#include <map>
IloNumVarArray D_ij_array(env);
        void Set_new_pair_in_D_ij_map(int i, int j)
        {
            int sig = 0;
            int maptemp = 0;
            string strsig;
            sig = i*n + j;
            strsig = to_string(sig);
            maptemp = D_ij_map.size();
            D_ij_map.insert(std::make_pair(maptemp, strsig)); }
```

At first, we declare variables D_ij in an array form. Every time we add a variable D_ij in the Var-array, its sequence number will be stored as an index, and the unique value

sig = i*n+j will be paired to index. Both of them will be stored in “the Map” as a form of binding pair. The method guarantees that unless the variable is called in constraints for the first time, it will never be declared at all. In addition, with map function, we can now delete or modify certain constraints easily by locating index through the binding value(since we know the value = i*n + j).

4.2.3 Comparison with LP in 4.1

After implementation of 4.2, we can finally train some data to do a comparison between the two methods.

Dataset	**Dimension**	**class**	**points**	**Time consumed (Original LP)**	**Time consumed (Fast LP)**

	d=		n=	t= (sec)	t= (sec)
Simple data	2	2	20	0.115	0.112
syntheticNoOutliers	2	3	300	26.9	6.18
Synthetic With Outliers	2	3	198	3.69	1.81
Mnist(part)	20	3	980	>1.5hour	1400s=23.3 min
CIFAR	48	3	1497	>>10 hour	<10 hour

Table 2 Time consumed compared to the original LP method

From Table 2, note that Algorithm in 4.2 has great advantage in deal with large scale of dataset.

4.3 Finding Maximum Epsilon (ϵ)

With small dataset, we can use the following method to determine the maximum ϵ :

$$\max_{\mathbf{a},\mathbf{B},\mathbf{d},\epsilon} \quad \epsilon \tag{7a}$$

$$s.t.\ \epsilon \leq d_{ik} \qquad \forall i \in \mathcal{N}\ , k \in\ \mathcal{N}\backslash\mathcal{C}_i \tag{7b}$$

$$(3) \tag{7c}$$

The implementation of the problem is a small part of 4.1.2.

5. Outlier re-insertion

Figure 1 shows an iterative LP Algorithm to learn distance metric. In last chapter, the detail of LP methods have been discussed. In this chapter, several approaches to outlier re-insertion will be stated.

5.1 Outlier re-insertion in MILP

After running iterations of Model (4.2), a set of outliers $\mathcal{O}$ can be generated. Given B, a new model to re-insert non-outliers in $\mathcal{O}$ can be expressed as a mixed-integer linear programming problem:

$$\max_{\boldsymbol{\lambda},\mathbf{y},\mathbf{z}} \quad f(\lambda) = \sum_{i\in\mathcal{N}}(\lambda_i - \rho z_i) \tag{8a}$$

$$s.t. \quad \sum_{j\in\mathcal{C}_i} d_{ij}y_{ij} + \lambda_i \le d_{ik} + \mathrm{M}(z_i + z_k) \quad \forall i \in \mathcal{N}, k \in \mathcal{N}\backslash\mathcal{C}_i \tag{8b}$$

$$\lambda_i \le 1 - z_i \qquad \forall i \in \mathcal{N} \tag{8c}$$

$$z_i = 0 \qquad \forall i \in \mathcal{N}, j \in \mathcal{C}_i \tag{8d}$$

$$\sum_{j\in\mathcal{C}_i} y_{ij} = 1 \qquad \forall i \in \mathcal{N} \tag{8e}$$

$$y_{ij} \in \{0,1\} \qquad \forall i \in \mathcal{N}, j \in \mathcal{C}_i \tag{8f}$$

$$z_i \in \{0,1\} \qquad \forall i \in \mathcal{N} \tag{8g}$$

$$\lambda_i \ge 0 \qquad \forall i \in \mathcal{N} \tag{8h}$$

(3) (8i)

In this model, d_{ij} is constant values due to the fixed **B**. Yet the model can be tighter by replacing $\mathrm{M}(z_i + z_k)$ in (8b). Let

$$\mathcal{IK} = \{(\mathrm{i},\mathrm{k}) \in \mathcal{N} \times \mathcal{N} : \nexists j \in \mathcal{C}_i \backslash \mathcal{O} : d_{ij} + \epsilon \leq d_{ik}, k \in \bar{\mathcal{C}}_i\}.$$

Set $M_{ik} = 1 - d_{ik}$. For any outlier penalty parameter $\rho > 0$, the following MILP attempts to minimize the number of outliers given a fixed distance metric.

$$\min_{\mathbf{y},\mathbf{z}} \quad \rho \sum_{i \in \mathcal{O}} z_i \tag{9a}$$

$$s.t. \quad \sum_{j \in \mathcal{C}_i} d_{ij} y_{ij} + \epsilon \leq d_{ik} + M_{ik} z_k \qquad \forall (i,k) \in \mathcal{IK} \tag{9b}$$

$$z_i = 0 \qquad \forall i \in \mathcal{N} \backslash \mathcal{O} \tag{9c}$$

$$\sum_{j \in \mathcal{C}_i} y_{ij} = 1 - z_i \qquad \forall i \in \mathcal{O}, j \in \mathcal{C}_i \tag{9d}$$

$$\sum_{j \in \mathcal{C}_i} y_{ij} = 1 \qquad \forall i \in \mathcal{N} \backslash \mathcal{O}, j \in \mathcal{C}_i \tag{9e}$$

$$y_{ij} \leq 1 - z_i \qquad \forall i \in \mathcal{N}, j \in \mathcal{C}_i \tag{9f}$$

$$y_{ij} \in \{0,1\} \qquad \forall i \in \mathcal{N}, j \in \mathcal{C}_i \tag{9g}$$

$$z_i \in \{0,1\} \qquad \forall i \in \mathcal{N} \tag{9h}$$

Implementation of the problem is quite similar to method in 3.1.

5.2.1 Outlier re-insertion with ranking of R_i

Though the MILP problem is improved, when facing large scale of dataset, it will be out of the capability to solve in a short time. Here represents a new method with

$$R_i = \frac{\min\limits_{j \in \mathcal{C}_i} d_{ij}}{\min\limits_{k \in \bar{\mathcal{C}}_i} d_{ik}}$$

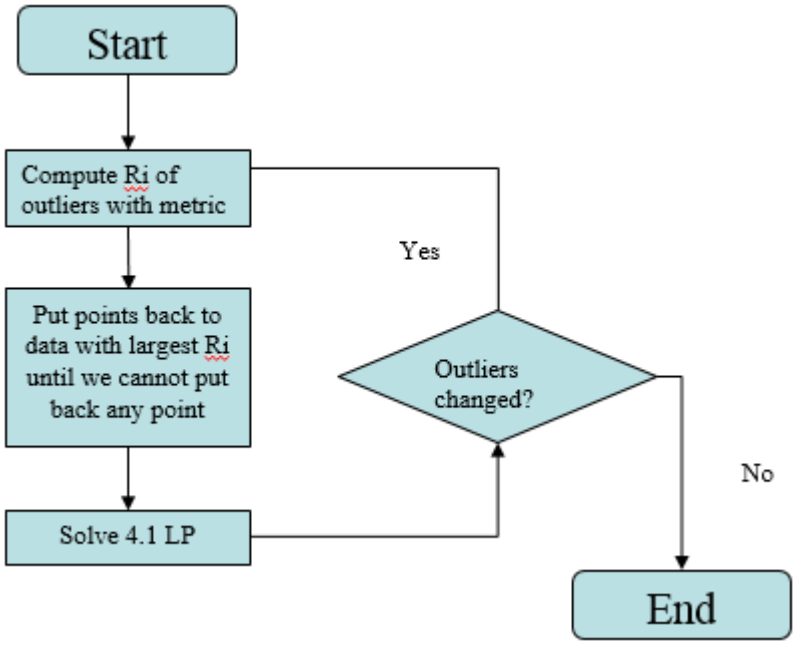


Figure 2 Algorithm performing method with R_i

Note that the iteration need to loop over several times to achieve the end. When the dataset is very large, it will be time consuming. To discard unnecessary procedures, introduce an improved re-insertion method:

Algorithm Improved Method of Outlier Re-insertion with Ranking of R_i';	
1:	Compute R_i for each outlier i;
2:	Identify all non-outliers j for which i is now the nearest co-class/non-class neighbor, i.e. all j whose R_i values are affected by insertion of i;
3:	Define $R_i' = \min(R_i, \min_j R_j)$;
4:	Sort outliers in the order of decreasing R_i';
5:	Insert the outliers according to decreasing ranking of R_i';
6:	Repeat steps 1-5 until no further re-insertion is possible, i.e. remaining outliers all have R_i'<1

Table 3 Outlier re-insertion with decreasing ranking of R_i'

With new method, we can insert outliers without changing distance metric as Figure 3 shows.

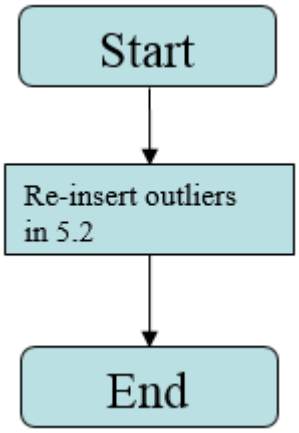


Figure 3 Algorithm performing improved method with R_i'

5.2.2 Implementation

Key code is as followed:

```
for (int j = 0; j < n; j++)
    {
        if (outliers[j] == 0)
        {
            R_j[j] = 0;
            R_j_1[j] = 0;
            temp_R_j = 999;
            R_j[j] = DistanceMetricLearner::Calculate_R(j);
            outliers[i] = 0;
            R_j_1[j] = DistanceMetricLearner::Calculate_R(j);
            if (R_j[j] - R_j_1[j]>threshold_1 && R_j_1[j] < temp_R_j)
                temp_R_j = R_j_1[j];
            outliers[i] = 1;
        }
    }
    if (temp_R_j < R[i])
    {
        R[i] = temp_R_j;
    }
    ////reinsert by order, until Ri<1;
    if ((outliers[i] == 1) && (temp < R[i]))
    {
        temp = R[i];
        ii = i;
    }
    if (R[ii]>1)
    outliers[ii] = 0;
```

Algorithm is to find $\min_j R_j$, then compare it with R_i and get the final value of R_i'.

6. Iterative LP Algorithm

6.1 Iterative LP Algorithm 1

Algorithm 1 is improved method in Figure 1.

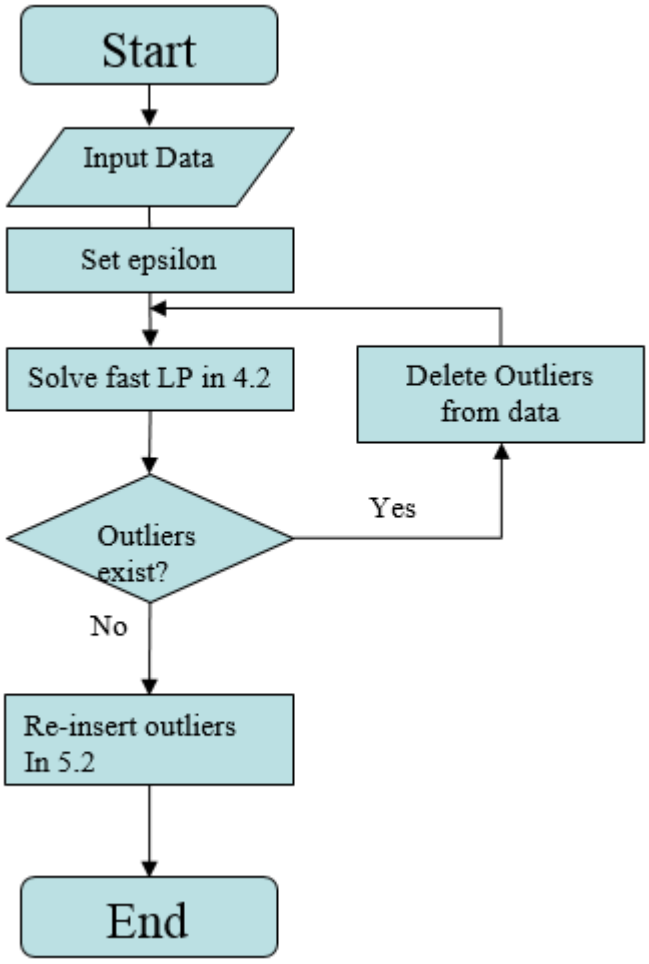


Figure 4 Algorithm 1

As Figure 4 shows, all the procedures from algorithm1 have been implemented.

6.2 Iterative LP Algorithm 2 with Row and Column Generation

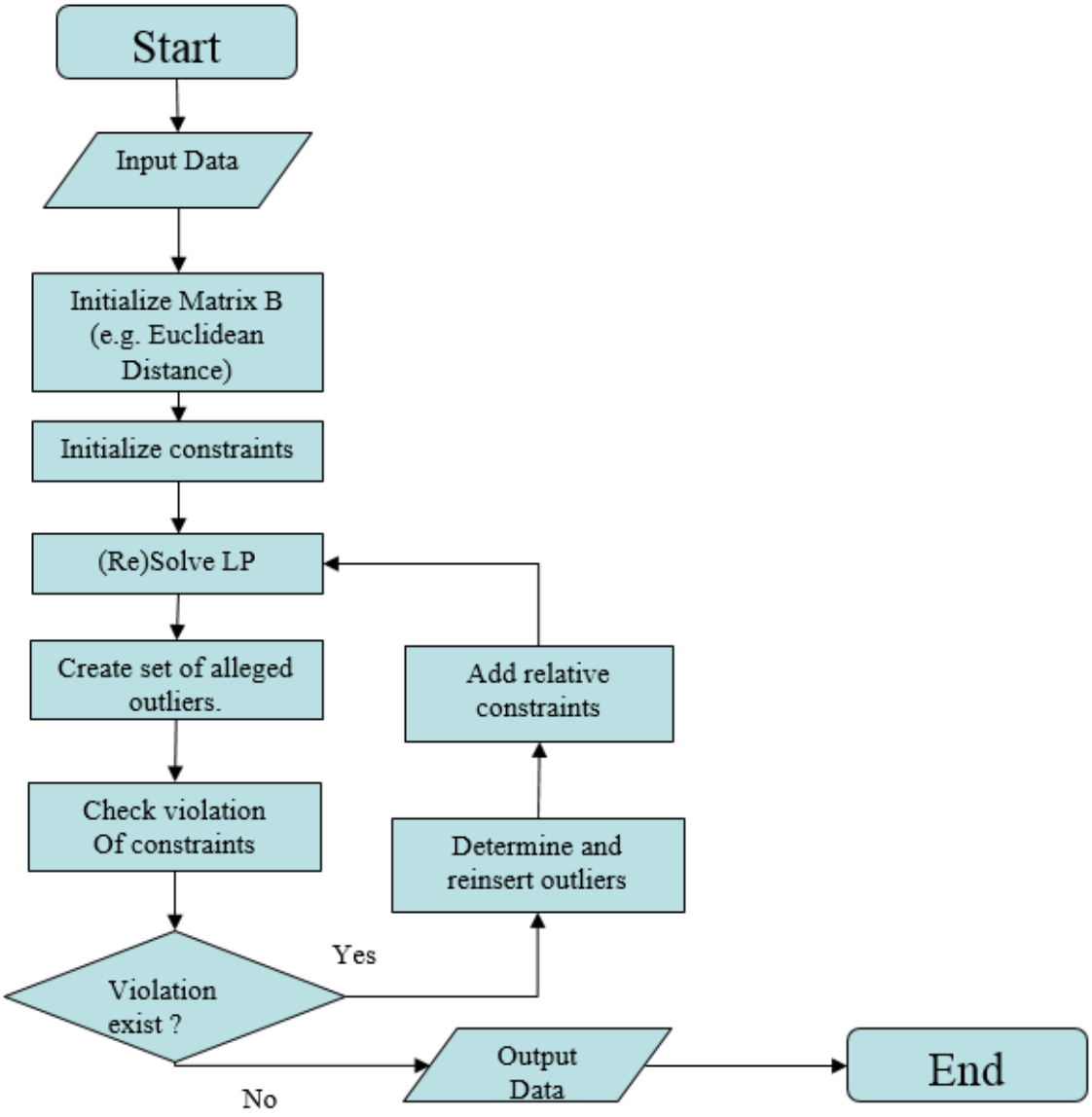


Figure 5 Algorithm 2

Different from Algorithm 1, Algorithm 2 includes outlier re-insertion inside the loop, as figure 5 shows.

Algorithm 2 Iterative LP with Row and Column Generation	
1:	The same with 1 in Table 1
2:	DONE = False
3:	**While** DONE = False **do**
4:	DONE = True
5:	(Re-)Solve LP (6) for $\mathbf{B}^*, d_{ij}^*, t_i^*$. Fix the distance metric to d_{ij}^*.
6:	**Step 1:** Create set of alleged outliers. Let $\mathcal{O}^{\text{alleged}} = \{i \in \mathcal{N}_A : t_i^* < \min_{j \in \mathcal{S}_i} d_{ij}^*\}$.
7:	**Step 2**: Check for violated constraints.
8:	**For** $i \in \mathcal{N}_A$ **do**
9:	The same with 7-10 in Table 1
13:	**If** $\lvert\mathcal{V}_i\rvert > 0$ **then** $\hat{\jmath}$ = argmax$\{t_i^* - d_{ij}^* : j \in \mathcal{V}_i\}$ and set $\mathcal{S}_i = \mathcal{S}_i \cup \{\hat{\jmath}\}$; DONE = False. **else** $(\lvert\mathcal{V}_i\rvert = 0)$, if $i \in \mathcal{O}^{\text{alleged}}$, **then** i is a true outlier: $\mathcal{N}_A = \mathcal{N}_A - \{i\}$ and $\mathcal{O} = \mathcal{O} \cup \{i\}$.
14:	The same with 12-14 in Table 1
17:	**end for**
18:	**Step 3**: Attempt to re-insert outliers.(here we use ranking R_i method)
19:	**Step 4**: Remove outliers: $\mathcal{N}_A = \mathcal{N}_A - \{\mathcal{O}\}$
20:	**End while**

Table 3 Detail of Algorithm 2

6.3 Implementation

Main difference between Algorithm 1 and Algorithm 2 is at step 13 Table 3. Code is attached.

7. Future Work

The thesis states several methods to implement Single Nearest Neighbor classification and the implementation of these methods. Next step is to compare methods in chapter 6 with LMNN and SVM, then analyze experimental results in aspects, e.g. efficiency, capability and accuracy. There is also a lot of work to formulate and implement KNN and more generalized metric learning method.

Refences

1) C. Domeniconi and D. Gunopulos. (2002) Adaptive nearest neighbor classification using support vector machines. In *Advances in Neural Information Processing Systems 14.* MIT Press.

2) Kilian Q. Weinberger and Lawrence K. Saul (2007) Fast Solvers and Efficient Implementations for Distance Metric Learning

3) Krishnan Kumaran and Dimitri J. Papageorgiou (2017) Adaptive Distance Metric Selection for Supervised Classification

4) Kilian Q.Weinberger and Lawrence K. Saul (2009) Distance Metric Learning for Large Margin

Appendix A: Results from real data - diabetes

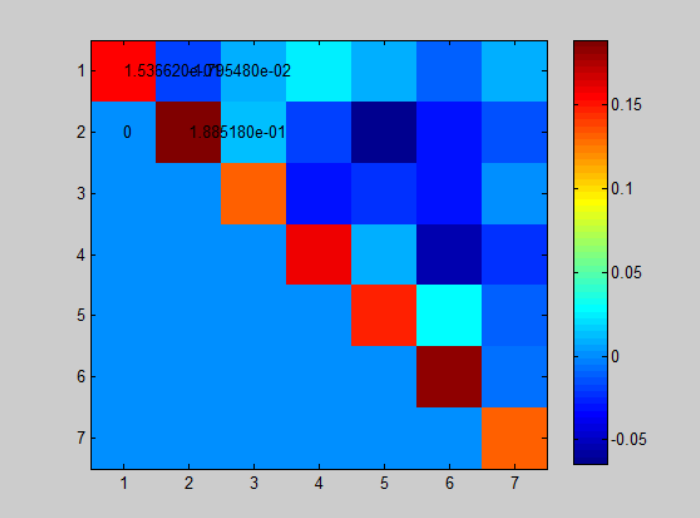
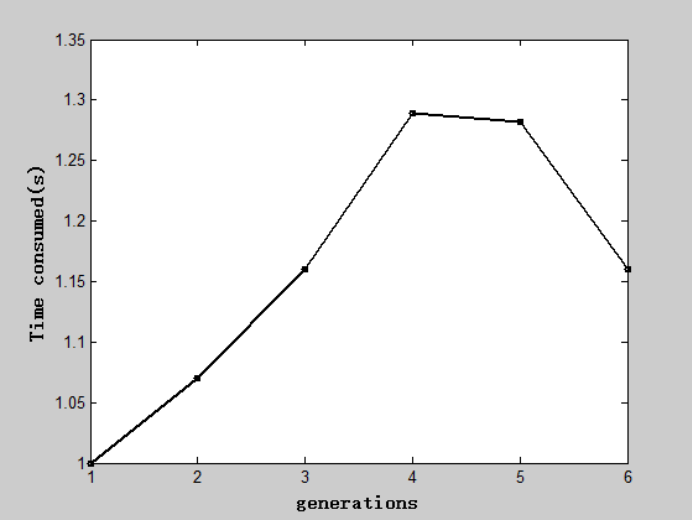


Figure of Metric B



Time consumed in generations

Appendix B: Results from real data – glass

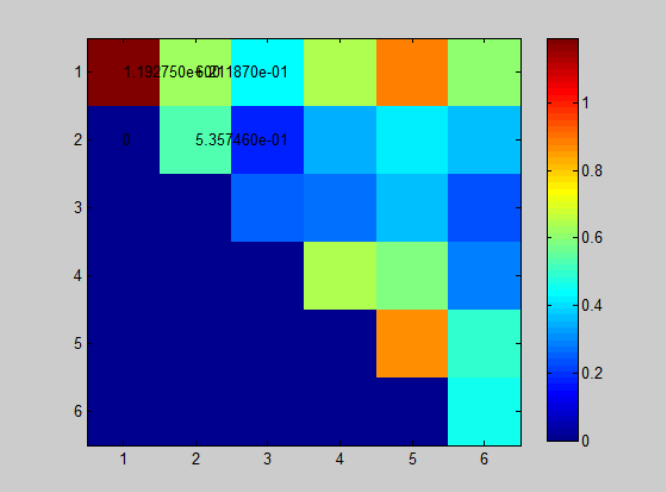
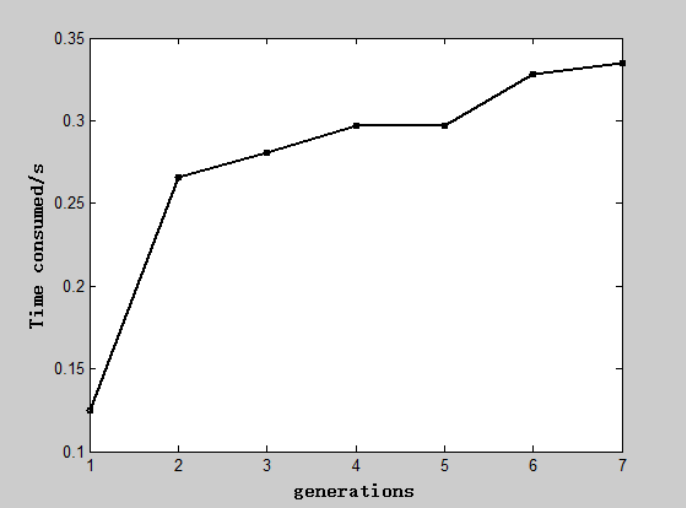


Figure of Metric B



Time consumed in generations

Appendix C: Results from real data - iris

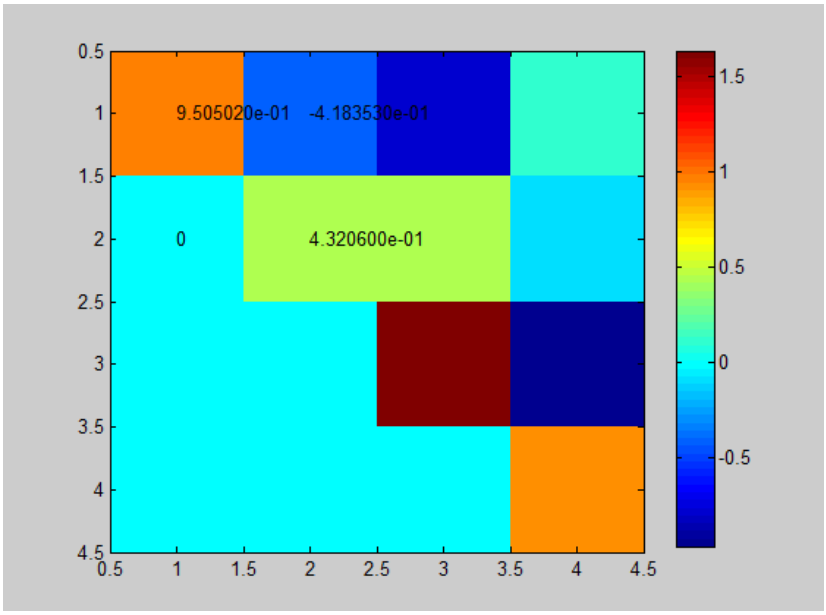
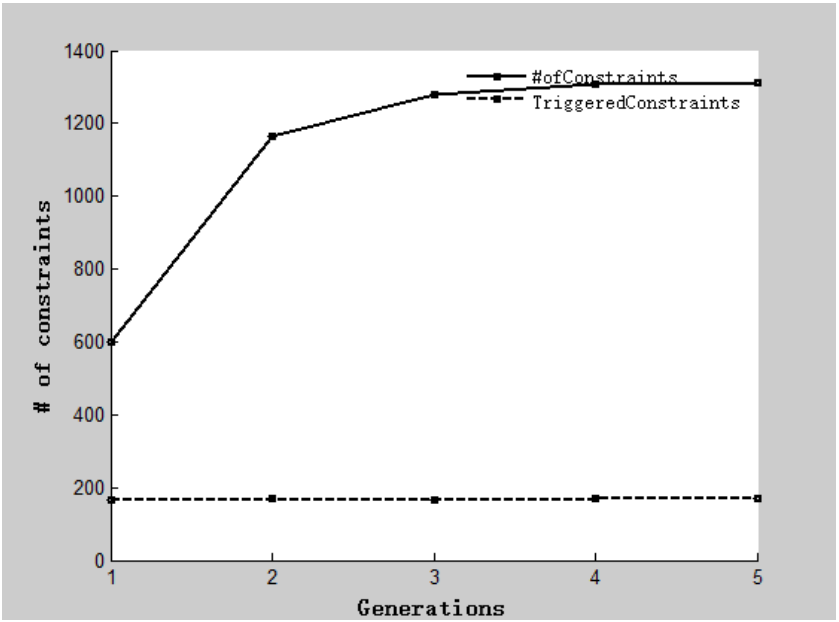


Figure of Metric B



Number of constraints triggered in generations

Vita

Bolun Xu was born in Yanan, Shaanxi Province, China on April 3th, 1993. Between 2011 and 2015 he studied Reliability and Systems engineering at Beihang University, China. Then he entered into P.C. Rossin College of Engineering and Applied Science of Lehigh University to pursue his Master of Science degree. His research area is data mining and machine learning.