University of Toronto

Industrial Engineering

MIE335: Algorithms and Numerical Methods

Project

Quadratic Programming Optimization

Quadratic Programming and an Application to the Disputed Fedralist Papers

Team Members:
Minjia Zhu 1001286761
Kenno Nagari 1001086718
Pongasakorn Rahman 1002352483
Lesley Hwang 1002490020

Professor:
Merve Bodur

April 4, 2020

Contents

1 Introduction

In this project, a quadratic programming problem must be solved in order to determine authorship of disputed Federalist Papers. Quadratic programming is a linearly constrained optimization problem with a quadratic objective function. Finance, economics, inventory production, and other fields are few examples that use quadratic programming applications for optimization purposes.

The quadratic programming problem that the team must solve takes on the following form:

$$\begin{aligned} \min \quad & \frac{1}{2}x^T Hx + c^T x \\ \text{s.t} \quad & Ax \leq b \\ & x \geq 0 \end{aligned} \tag{1}$$

where x represents the continuous decision variables in vector form, and H is a positive definite matrix within the equation to be minimized.

Quadratic programming is to be applied to a support vector machine with the end goal to solve the unknown authorships behind disputed Federalist Papers. These Federalist Papers were written in 1787-1788 by Hamilton, Jay, and Madison for the U.S. Constitution, and resulted in a remaining 12 out of 162 papers left in dispute. The disputed papers are written by either Hamilton or Madison, and the team is to use data provided by the papers with known authorship to determine the actual author of each of the disputed papers. In order to do this, the vocabularial habits each author must be analyzed and compared against the disputed papers.

A linear support vector machine (SVM) will be computed to determine the authorship of each disputed paper, where the known authorship papers will be split into a training set and a tuning set, while the disputed papers are the testing set. The SVM will run and compute each of the testing set papers against the training set and tuning set, and determine whether the author of the paper is Hamilton or Madison. For this computation, the SVM quadratic programming formulation is as shown:

$$\begin{aligned} \min \quad & \frac{1}{N}\sum_{i=1}^{N} s_i + \frac{\mu}{2}\omega^\top \omega \\ \text{s.t} \quad & y_i(\omega^\top x_i + b) \geq 1 - s_i \quad \forall i \in \{1, ..., N\} \\ & s \geq 0 \end{aligned} \tag{2}$$

In the above formulation, ω and b are decision variables that must be determined, s is an error variable, and N is the number of training examples. xi are feature vectors in R70, where 70 is the number of floating point numbers that represent the relative frequency of 70 function words from the papers. During computation, y_i for Hamilton will be 1, while y_i for Madison will be set as -1. The output from the decision variables are used to compute a linear classifier f in the equation as shown:

$$f(x_i) = \omega^\top x_i + b \tag{3}$$

While this problem is to be solved using quadratic programming, there are many algorithms

that can be implemented to optimally find the solution. The team decided to choose Hildreth and Conjugate Gradient algorithm for this problem. In short, Hildreth is an algorithm that takes advantage of the relationship between the primal and dual forms, solving the dual solutions to obtain the primal solutions. Hildreth is chosen due to its strength in the use of element-by-element search, neglecting the need to create matrix conversions. The team favoured Hildreth also due to the team's background knowledge in the dual and primal theory studied in the previous course, Operations Research 1. As for Conjugate Gradient, it is an algorithm that starts from a point in the problem space and computes direction and line searches based on conjugates to obtain the optimal solution. Conjugate Gradient is implemented because of its ability to start the search at any point and simple calculations required to compute search directions and the solutions. These calculations will result in low memory space needed and has an expectation of shorter run time. Furthermore, the team realized that these algorithms will not compute with SVM instances until later, due to their complexity and concept. Explanation as to why the algorithms do not run with SVM instances will be addressed in Algorithms Comparison section below. However, since there were QP instances available to check these algorithms' functionality, their correctness can still be assessed. Due to the time constraint for this project and effort outputted, the team decided to stick with these methods.

The criteria being used to compare the two algorithms are Speed, Accuracy, Feasibility, and Memory. Speed is an important criteria as time is one of the crucial factors in real world problems. It is necessary for problems to be solved as fast as possible. Accuracy is also considered as the team not only wants the algorithm to be correct but also the most optimal and correct solution. Feasibility is also significant as the algorithm should be able to return a reasonable answer most of the time so findings and conclusions can be extracted at the end. Lastly, the team decided to make Memory as a criteria as well since space is still a limited resource in this world and an algorithm should require minimum space.

The remaining sections in the report will emphasize focus on the algorithms chosen, the comparison between algorithms, SMV application results, and final recommendations.

2 Algorithms

The following section talks about the two algorithms in detail. Their pseudo code and complexity analysis are also illustrated. The predictions on performance will also be discussed. Hildreth's quadratic algorithm will first be addressed, followed by the Conjugate Gradient method. The solver that was used to be the team's reference is the MATLAB *quadprog* function. MATLAB *quadprog* function is chosen due to the team's known capabilities and experience with MATLAB code language. The team is also especially familiar with reading the outputs of MATLAB code allowing for thorough understanding of results.

2.1 Hildreth's Quadratic Programming Algorithm

The Hildreth's Quadratic Programming algorithm uses the connection between primal and dual forms of problems to solve Quadratic Programming problems. This method involves finding the solution for the dual and then using the dual solutions to obtain the primal solutions. The large number of constraints make the computational time large when solving for primal. Therefore, solving in dual form allows for systematic identification of inactive constraints, making the computation time lower. In summary, Hildreth's Quadratic Programming algorithm follows the convergence process of duality theory to acquire the optimal solution. There were a few variations with different modifications found online, however most possessed similar concepts

starting off with converting the primal variables to dual variables. The team decided to choose the variation and source that provided the most equations to be used as references to make it convenient for the team to understand. This variation followed clear steps and will be discussed below.

The Hildreth algorithm starts with converting the problem to its dual form. A and c from the primal form are first calculated and expressed as P and d respectively to be the counterpart representative in its dual form. P and d are calculated using the following formula:

$$P = AH^{-1}A^T$$
$$d = AH^{-1}c + b$$

With P and d, the final dual will be calculated using the formula expressed below:

$$\tfrac{1}{2}P\lambda - \lambda^T d$$
$$s.t\ \lambda \geq 0$$

The lambda is treated as the decision variables here. The optimal dual variables are to be found using the following formula. Objective function and optimal x values can be found for the primal problem after optimal dual variables are found.

$$\lambda_i^{k+1} = max(0, \frac{-1}{p_{ii}}(d_i + \sum_{j=1}^{i-1} P_{ii}\lambda_j^{k+1} + \sum_{j=i+1}^{n} P_{ij}\lambda_j^k)) \tag{4}$$

With the solutions for the dual variables found, $x*$ and the objective function value ($z*$) can be found for the primal problem.

2.1.1 Pseudo-Code

Algorithm 1 portrays Hildreth's Quadratic Programming pseudo-code.

Algorithm 1 Hildreth's Quadratic Programming Algorithm

Input: QP instance
Output: x, z

1: Load QP instance
2: Convert into dual matrices:
3: $P = AH^{-1}A^T$
4: $D = AH^{-1}f + b$
5: Initialize lambda vectors
6: **while** *count* $<$ max iterations **do**
7: **for** i=1:Number of iterations **do**
8: $\lambda_i^{count+1} = max(0, \frac{-1}{p_{ii}}(d_i + \sum_{j=1}^{i-1} P_{ii}\lambda_j^{k+1} + \sum_{j=i+1}^{n} P_{ij}\lambda_j^k))$
9: **end for**
10: $count = count + 1$
11: **end while**
12: Calculate Primal solutions:
13: $x = -H^{-1}f - H^{-1}A^T\lambda$
14: $z = \frac{1}{2}x^THx + f^Tx$

2.1.2 Complexity Analysis

Time Complexity
The complexity of the time for the algorithm is addressed below. The time is measured assuming a worst case scenario where the algorithm runs until it reaches maximum number of iterations. Table 1 below defines the run-time for each line in the pseudo-code.

Table 1: Time-Complexity of Hildreth's Quadratic Programming Algorithm Method

Pseudo-Code Line	Step	Max. No. Operations
2	Calculate $P = AH^{-1}A^T$	$3n^3 + n^2$
3	Calculate $D = AH^{-1}f + b$	$3n^3 - n^2$
6	Calculate for lambda	$n^4 + 4n^3 + n2$
11	Calculate for $x = -H^{-1}f - H^{-1}A^T\lambda$	$3n^3 + 2n^2 - 2n$
12	Calculate for $z = \frac{1}{2}x^THx + f^Tx$	$4n^2 - 3n$
Totals	1 iteration	$13n^3 + 5n^2 + n^4 - 2n$

Therefore, the overall time complexity is:
$O_n(\text{HQP}) = 13n^3 + 5n^2 + n^4 - 2n \approx kn^4$,
where k is the maximum number of iterations.

Space Complexity
The space required to execute this algorithm are illustrated below in Table 2. The space requirement for each variable for each step in the algorithm is identified.

Table 2: Space Complexity of Hidreth's Quadratic Programming

Variable	Size of Variable	Space Complexity
A	$2m$ x n	$128mn$
H(Q)	n x n	$64n^2$
f	n x 1	$64n$
b	$(m+n)$ x 1	$64m + 64n$
x	m x 1	$64m$
A-dual	m x m	$64m^2$
f-dual	m x 1	$64m$
Length	1 x 1	64
dual-Solution	m x 1	$64m$
z	1 x 1	64
lastDualSolution	m x 1	$64m$
Stop	1 x 1	64
Totals		$128mn + 64n^2 + 64m^2 + 128n + 256m + 192$

Therefore, total space complexity of Hildreth's Quadratic Programming algorithm is:
$O_n = 128mn + 64n^2 + 64m^2 + 128n + 256m + 192 \approx 64n^2 + 64m^2$

2.1.3 Predictions About Performance

Speed
Overall, this algorithm is expected to be slow. This is due to the fact that the run-time will keep increasing as the size of the problem becomes larger, making the convergence rate of dual variables slower. Furthermore, when the algorithm is unable to find a best solution, it has to run the maximum number of iterations to acquire the near-optimal solution.

Accuracy
Hildreth's algorithm should perform with high accuracy, assuming that it is able to run as many iterations as it needs to have convergence on dual variables. This is because the algorithm will compute the objective function simultaneously until there is a convergence of dual variables.

Feasibility
A feasible result that satisfy the constraints will return only if there is a convergence of dual variables. However, it will not return a feasible answer if there is no convergence and the algorithm is stopped once reaching the maximum number iterations instead. Therefore, it is more probable for this algorithm to perform poorly in terms of feasibility.

Memory
The memory space of Hildreth algorithm depends on the size of the problem but there is no storage of new additional space as for every iteration, the matrices are refreshed. This algorithm is expected to not require a lot of memory for computation.

2.2 Conjugate Gradient Method

The second method used, Conjugate Gradient Method, is a steepest descent algorithm that iteratively performs line searches on x_k in direction d_k to determine x_{k+1} until the optimal solution is obtained. The variation of Conjugate Gradient that was chosen followed steps with clear, concise, and pre-set equations and reflected material that the team was familiar with.

There were two complete variations the team encountered and was confident about for Conjugate Gradient Method. The first variation did not take into consideration A and b which are the constraints. However, this problem possesses constraints so it was not chosen. The second variation walked through similar steps except it also took in consideration of the constraints, therefore it was chosen. The Conjugate Gradient Method variation used is outlined and discussed below with its detailed steps and computations.

The directions are determined so that each line search will result in a minimization of the objective function. This method utilizes conjugates by calculating directions so that each one is conjugate to the previous direction, where $d_i^\top A d_j = 0$ for $i \neq j$. Utilizing conjugates allows the method to ensure that the search directions computed will not affect one another. The method requires problem size n, computing a maximum of n number of line searches before finding the optimal minimum. However, due to memory restraint, Conjugate Gradient Method only computes the required search directions. The direction calculations, along with the necessary unknowns, are as shown:

$$d_0 = - \bigtriangledown f(x_0) = -(Hx_0 + f) \quad \text{[for the first direction]} \tag{5}$$

$$\beta_{k-1} = \frac{\bigtriangledown f(x_k)^\top H_{adj} d_{k-1}}{d_{k-1}^\top H_{adj} d_{k-1}} \tag{6}$$

where β_{k-1} is calculated with respect to $d_k^\top H_{adj} d_{k-1} = 0$

$$d_k = - \bigtriangledown f(x_k) + \beta_{k-1} d_{k-1} \quad \text{[for all other search directions]} \tag{7}$$

As stated above, once the search direction is determined, line search is performed x_k in the direction of d_k in order to find x_{k+1}. x_{k+1} and necessary unknowns needed are calculated as shown:

$$x_1 = x_0 + \alpha_0 d_0 \quad \text{[for the first } x_k\text{]} \tag{8}$$

$$\alpha_k = -\frac{d_k^\top H_{adj} x_k + d_k^\top b}{d_k^\top H_{adj} d_k} = -\frac{d_k^\top \bigtriangledown f(x_k)}{d_k^\top H_{adj} d_k} \tag{9}$$

where α_k is calculated with respect to $d_k^\top H_{adj} d_{k-1} = 0$

$$x_{k+1} = x_k + \alpha_k d_k \quad \text{[for all other } x_k\text{]} \tag{10}$$

The process follows an iterative structure, where x_{k+1} is continuously computed until the objective function value converges and the number of iterations stay within n. Upon convergence, the x_{k+1} value is the optimal solution x.

2.2.1 Pseudo-Code

Algorithm 2 portrays Conjugate Gradient Method pseudo code.

Algorithm 2 Conjugate Gradient Method

Input: H,c,A,b,ω,x_0
Output: z^*, x^*

while $\omega_{prev} < \omega$ **do**
  Generate H_{adj}, $c_{adj}^\top$
  $H_{adj} = H + 2\omega A^\top A$
  $c_{adj}^\top = c^\top - 2\omega b^\top A$
  $d_0 = - \bigtriangledown f(x_k) = -(Hx_0 + f)$
  Set $k = 0$
  for iteration from i to n **do**
    Calculate z_{old}
    $\alpha_k = -\frac{d_k^\top H_{adj} x_k + d_k^\top b}{d_k^\top H_{adj} d_k} = -\frac{d_k^\top \bigtriangledown f(x_k)}{d_k^\top H_{adj} d_k}$
    $x_{k+1} = x_k + \alpha_k d_k$
    $\beta_k = \frac{\bigtriangledown f(x_{k+1})^\top H_{adj} d_k}{d_k^\top H_{adj} d_k}$
    $d_{k+1} = - \bigtriangledown f(x_{k+1}) + \beta_k d_k$
    Increment $k = k + 1$
    Set $x_k = x_{k+1}$
    Calculate z_{new}
  end for
  $\omega = z_{new}$
  $z* = z_{new}$
  $x* = x_k$
end while

2.2.2 Complexity Analysis

Time-Complexity
Conjugate Gradient Method time complexity is shown in Table 3. This table shows the complexity for each line of the pseudo code, where the steps are iterated for a maximum of 76 times (n times) in order to find $x*$.

Table 3: Time Complexity of Conjugate Gradient Method

Pseudo-Code Line	**Step**	**Max. No. Operations**
3, 4 (outer)	Calculating H_{adj} and $c_{adj}^{\top}$	$2n^3 + (2m+1)n^2 + 5n + 4m$
5 (outer)	Calculating d_0 (gradient $\bigtriangledown$)	$2n^2$
8	Calculating z_{old}	$2n^2 + 3n + 1$
9	Calculating α	$4n^2 + 3n$
10	Calculating line search x_k	$2n$
11	Calculating β_k	$4n - 1$
12	Calculating search direction d_{k+1}	$2n^2 + 2n$
13	Increment k	1
15	Calculating z_{new}	$2n^2 + 3n + 1$
Totals	1 iteration of Conjugate Gradient	$(2n^3 + (2m+3)n^2 + 5n + 4m)$ $+(10n^2 + 17n + 2)$

Therefore, the overall time complexity for the outer loop is:
$O_n = \mathbf{k}(2n^3 + (2m+3)n^2 + 5n + 4m) \approx k(2n^3)$
The overall time complexity for the inner loop is:
$O_n = \mathbf{n}(10n^2 + 17n + 2) \approx 10n^3$

Space Complexity
The space complexity of the Conjugate Gradient algorithm is calculated according to the variables used in the method and defined in the pseudo code. Table 4 below shows the space complexity for each variable, including its name and size.

Table 4: Space Complexity of Conjugate Gradient Method

Variable	**Size of Variable**	**Space Complexity**
H	n x n	$64n^2$
c	n x 1	$64n$
A_{n-1}	m x n	$64mn$
b_{n-1}	m x 1	$64m$
A	$(m+n)$ x n	$64n(m+n)$
$H_{adjusted}$	n x n	$64n^2$
$c_{adjusted}$	n x 1	$64n$
Constant	1 x 1	64
ATA	n x n	$64n^2$
bTA	n x 1	$64n$
bTb	1 x 1	64
w_{prev}	1 x 1	64
w	1 x 1	64
n	1 x 1	64
tol	1 x 1	64
x_k	n x 1	$64n$
x_{k+1}	n x 1	$64n$
Steepest descent	n x 1	$64n$
New steepest	n x 1	$64n$
Search direction	n x 1	$64n$
Conjugate gradient	n x 1	$64n$
α	1 x 1	64
β	1 x 1	64
k	1 x 1	64
z_{old}	1 x 1	64
z_{new}	1 x 1	64
z_{adj}	1 x 1	64
$\bigtriangledown_{old}$	1 x 1	64
$\bigtriangledown_{new}$	1 x 1	64
Totals		$64(4n^2+10n+2mn+2m+14)$

Therefore, total space complexity of Conjugate Gradient Method is:
$O_n = 64(4n^2+10n+2mn+14) \approx n^2$

2.2.3 Predictions About Performance

Speed
The Conjugate Gradient Method is expected to reach convergence and obtain the optimal solution x without prolonged running time. Although the method considers the size of the problem, n, to be the maximum number of line searches, it computes only the required direction searches and minimizes the number of operations that need to be repeated.

Accuracy
The accuracy of this method is predicted to be high and capable of returning an optimal solution within the required 5%. However, due to the SVM problem that must be solved, it is uncertain that the Conjugate Gradient Method will be able to produce an accurate enough

output as the objective function value.

Feasibility
The method is expected to return a feasible solution. Feasibility is guaranteed as long as there is convergence between the x values within n number of iterations. However, since Conjugate Gradient Method minimizes computations by calculating only the required search directions, the number of iterations will unlikely reach n and a feasible solution is guaranteed.

Memory
A fixed number of variables and unknowns are utilized in order to perform computations in each iteration, where these values are overridden each time. Due to this, an excess amount of memory, as well as any sort of memory overflow will not be expected. The performance, in terms of memory usage and space, of this algorithm is predicted to be high.

3 Algorithm Comparison

This section compares the results outputted from after running both algorithms, Hildreth and Conjugate Gradient, to the results acquired from MATLAB's *quadprog* solver. The team used the QP instances that was provided in Portal. The SVM application instances were not used as the two algorithms were not designed to be able to inverse the positive definite matrix H that was described in the Project Description. However, both Hildreth and Conjugate Gradient algorithms compute correctly using the QP instances. The comparison between the two algorithms and the MATLAB *quadprog* function are compared using QP instances.

The 3 criteria the team is using to compare and evaluate are Speed, Accuracy, and Feasibility. Time computation will be evaluated for Speed for the two algorithms and MATLAB's *quadprog* function. For Accuracy, the results of the two algorithms will be compared to the results of the solver. Lastly, the team will measure how many non-negative constraints the two algorithms and solver violated for Feasibility.

3.1 Speed

When testing the speed of the two algorithms and MATLAB's *quadprog* solver, the team used the QP instances that was provided. The time it takes to compile a result for each method is recorded and shown in Table 5 below. The values in the table are in unit seconds.

There are also a total of 3 graphs to display the speed performance of the algorithms below to visually depict the performances for effective interpretation. These 3 represent the 3 different QP instances that was used to run the algorithms. Only 3 are shown to provide just enough evidence for different instances so all 5 are not needed. For each QP instance, the number of iterations were adjusted for the Hildreth Quadratic Programming algorithm since the algorithm's performance depends on its number of iterations. Conjugate and solver follow a different concept that does not depend on the number of iterations so they possess constant time results for each QP instances. Time H represents the time for Hildreth, Time Solver is the time for Solver, and Time C is the time for Conjugate Gradient Algorithm.

Table 5: Time Computation of Two Algorithms Compared to Solver

QP Instance 1			
Iterations	Time H	Time Solver	Time C
100	0.0067	0.008	0.0054
1000	0.0778	0.008	0.0054
10000	0.5194	0.008	0.0054
QP Instance 2			
Iterations	Time H	Time Solver	Time C
100	0.0236	0.0073	0.0057
1000	0.06	0.0073	0.0057
10000	0.5204	0.0073	0.0057
QP Instance 3			
Iterations	Time H	Time Solver	Time C
100	0.0239	0.0067	0.0089
1000	0.0752	0.0067	0.0089
10000	0.5255	0.0067	0.0089

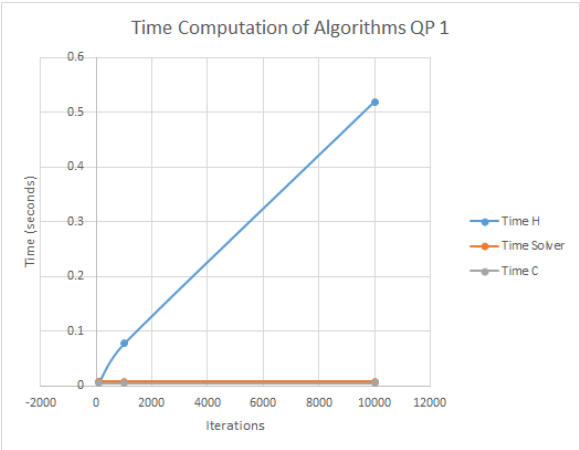

Figure 1

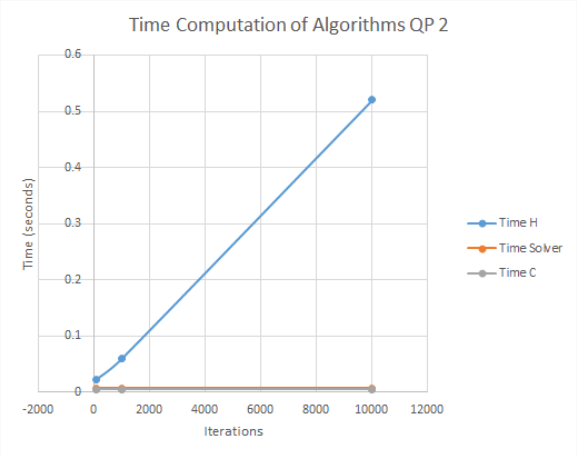

Figure 2

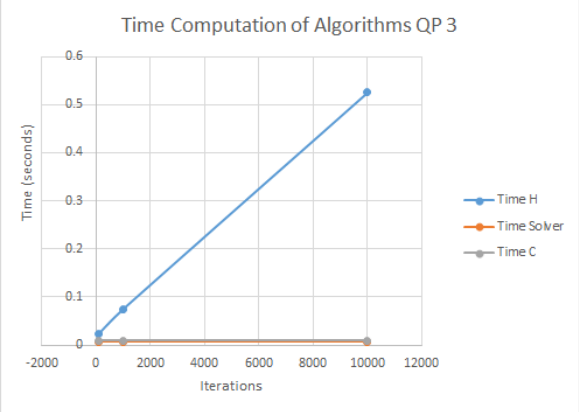

Figure 3

As the table suggests, the Conjugate Gradient Method computes with the lowest time making it the overall fastest algorithm. The purpose of running each of the instances with different iteration numbers was to analyze the Hildreth's performance in speed. It is evident that there is dependence on the number of iterations. For when iteration was 100, the computation time for Hildreth was 0.0067 for QP instance 1, which is considered fast compared to the other algorithms having 0.008 and 0.0054. However, for when the iteration became 1000, a drastic change in time computation occurred and is visibly shown in the graphs above. In summary, Hildreth is concluded to have poor performance in terms of speed and Conjugate Gradient Method performs considerably well.

3.2 Accuracy

This section measures how accurate the two algorithms are when compared to MATLAB *quadprog* function. It is an assumption that quadprog solver is projecting the correct results and is to be used as the referenced solution. Similarly to the Speed section, 3 QP instances are used to compute the two algorithms and the solver. Also, for the purpose of effectively measuring Hildreth's algorithm, several computations were done using different iterations. The table below displays the results from each method. Graphs to visually display the accuracy are also formed for easier analysis.

Acc H represents the accuracy for Hildreth, Acc Solver represents the accuracy for the solver, and Acc H represents the accuracy for Conjugate Gradient Method. The values in the table are results outputted from the algorithms.

Table 6: Accuracy of Algorithms Compared to Solver

QP Instance 1			
Iterations	Acc H	Acc Solver	Acc C
100	-908.51	-956.757	-940.56
1000	-960.3	-956.757	-940.56
10000	-956.76	-956.757	-940.56
QP Instance 2			
Iterations	Acc H	Acc Solver	Acc C
100	-612.188	-575.793	-580.94
1000	-559.541	-575.793	-580.94
10000	-575.793	-575.793	-580.94
QP Instance 3			
Iterations	Acc H	Acc Solver	Acc C
100	2953	-551.58	-545.24
1000	-714.16	-551.58	-545.24
10000	-551.58	-551.58	-545.24

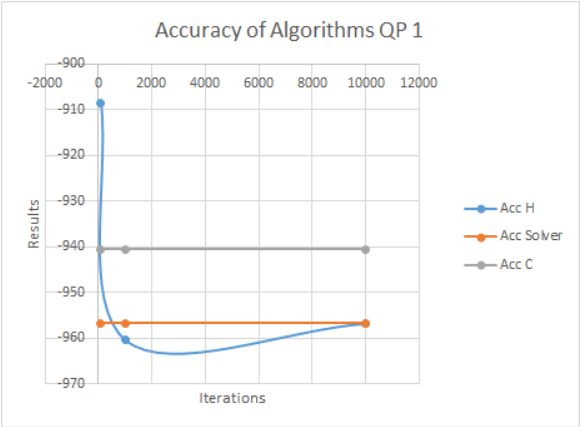

Figure 4

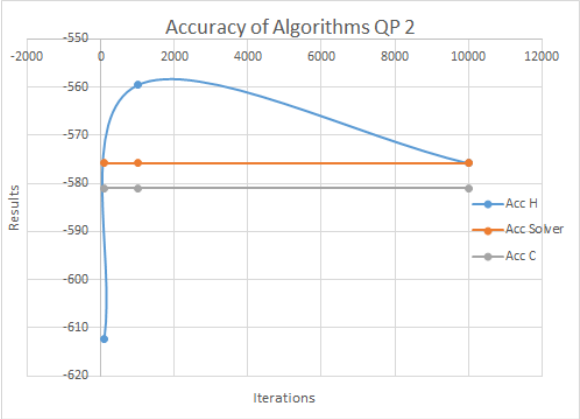

Figure 5

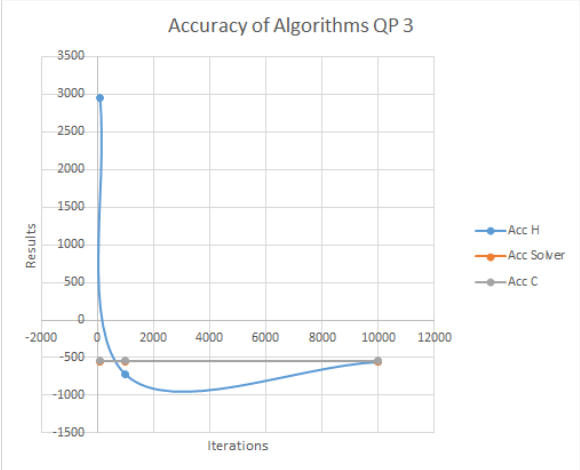

Figure 6

The way how the 2 algorithms perform are clearly shown in the table and graphs above. For low iterations, Hildreth performs extremely poorly, but when there are many iterations, it performs extremely well to the point where its solution matches the quadprog exactly. Conjugate may project decent outputs for low iterations, but it does not project the exact results as the quadprog solution.

To provide further insight into the algorithm's accuracy and efficiency, another graph is generated to show the difference in the results with a 5% threshold limit to compare with. The cyan colored stars represent the 5% threshold for each solver solution running each of the QP instances. The higher the point is in the graph, the worse the answer becomes. If the point is below the 5% threshold, it means the algorithm is accurate. If it is above the limit then the algorithm is considered to be inaccurate. The blue circle represents the deviation results from Hildreth algorithm. The reason there are 3 blue circles for each instance run as those represent the Hildreth results for each iteration level (100, 1000, 10000). The red circle symbolizes the Conjugate Gradient Method results. See graph below.

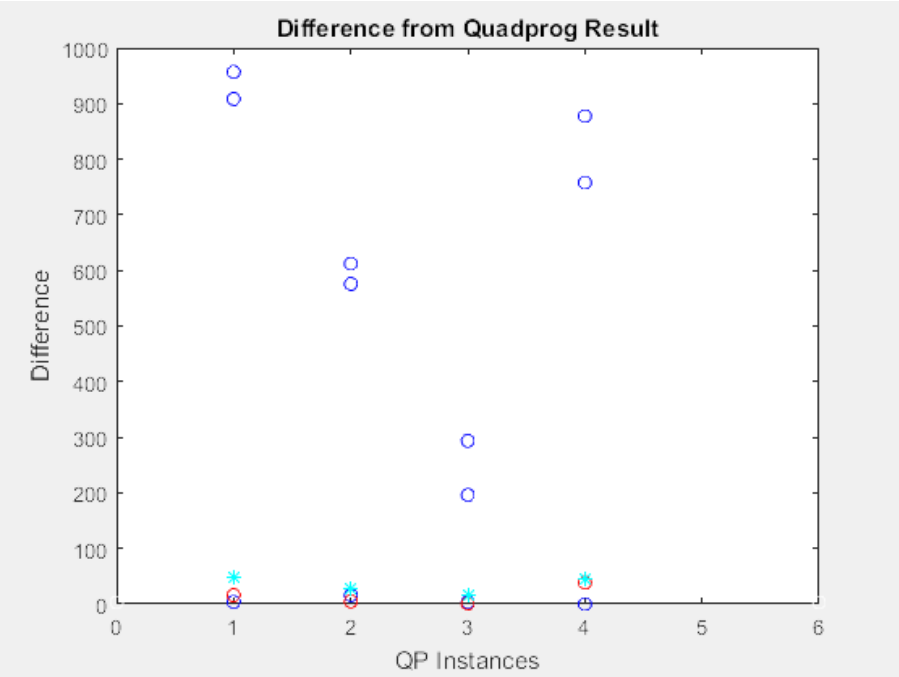

Figure 7

As the graph above suggests, Hildreth Quadratic Program algorithm breaks the 5% limit by a large margin for two different iteration runs. In contrast, when iteration is 10000, Hildreth performed exceedingly well. It also makes sense that the 3 blue circles exist in very different locations. This is because the difference in iteration number is also large. Each iteration is increased by a factor of 10, having a significant increase for each run. Conjugate Gradient Method performs well as well having all results to be within the 5% threshold. Hildreth still performs better at its best compared to Conjugate Gradient Method, but much worse at other conditions.

3.3 Feasibility

This section analyzes the algorithm's ability to obtain feasible solutions. The team is going to measure this through calculating the number of violations against non-negative constraint. The QP instances are the running tests these methods will use to compute. There are a total of 40 trials for each QP instance run, so the number of violations will be measured out of 40. Initially the team intended to use Linear Constraints to be one of the feasibility assessments, but because of how the QP instances are provided there are no linear constraints to be assessed. The QP instances that were used to compute testings did not contain any linear constraints.

3.3.1 Non-negative Constraints

This feasibility assessment describes how many times the two algorithms violate the non-negative constraints. This assessment emphasizes attention to the non-negative constraint equation from the quadratic programming formula, $x \geq 0$. If the variable ends up being negative, then there is a violation. If the variable results in a positive number then it is feasible.

The graph below portrays a comparison of 2 methods being assessed in terms of non-negative constraint violation. The x axis represents each QP instances and the y axis serves as the number of violation times.

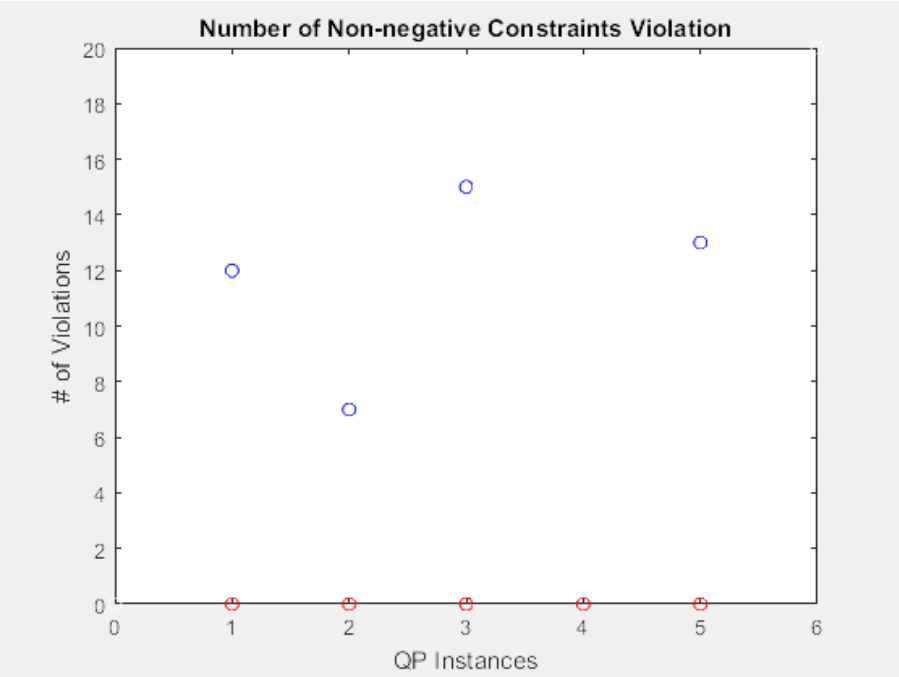

Figure 8

As the graph proposes, out of the 5 QP instances, the worst Hildreth performed with the non-negative constraints was 15 violations out of the 40 trials, so 37.5%. On average, Hildreth violates 9 constraints so 22.5% violation. This proves that Hildreth is performing fairly poor. However, Conjugate Gradient Method is computing much more optimally as for each of the 5 QP instance runs, there were no negative variables outputted so this algorithm did not violate any non-negative constraints. Conjugate Gradient Method performs very well in terms of non-negative feasibility while Hildreth performs poorly.

3.4 Prediction VS. Results

After obtaining the results, the prediction beforehand can now be perceived if they were accurate or not. Speed, accuracy, and feasibility are the performance criterias being observed. Memory was one of the performance criteria used as a prediction, however, due to the limited capabilities of MATLAB, memory was not able to be extracted after running the algorithms. The table below effectively summarizes the actual performance and the predicted performance. Note that the Hildreth Quadratic Programming algorithm performance takes into consideration the different iteration amounts. So even though Hildreth may perform well for large iterations, if it performs poorly for certain iterations, then the overall result will be decent level.

(prediction level/results level)
Level: Bad(worst) - Decent - Good(best)

Table 7: Comparing Prediction and Results

Methods	Speed	Accuracy	Feasibility
Hildreth	Bad/Bad	Good/Decent	Decent/Bad
Conjugate	Good/Good	Decent/Good	Good/Good

4 SVM Application Results (Q1-Q4)

This section displays the answers to the 4 SVM application questions using the MATLAB *quadprog* function as the computation solver. The compressed answers are shown below, however, the full complete answers are in the form of large tables and those can be found in the appendix section.

4.1 Question 1

The objective QP function values, the b values, and the ω values for the different values of μ are shown in the tables below. Note that ω values in the table are averaged values from 70 data points. The full data point results are accessible in Appendix A.

μ **value**	0	0.001	0.01	0.1	1	10	100
$F(x)$	1.06E-9	2.0E-5	0.000204	0.002043	0.020433	0.15192	0.385348
b value	5.8071	-5.05	-5.0711	-5.0708	-5.0707	-4.146	-1.0093
ω **value (avg)**	-0.00583	0.004353	0.00437	0.004379	0.004381	0.003809	0.000713

The number of misclassified points from the training set are also expressed here. The result exists for each value of μ.

μ **value**	0	0.001	0.01	0.1	1	10	100
Misclassified Points	0	0	0	0	0	1	14

So for when μ is 10, there is 1 misclassified point. When μ is 100, there are 14 misclassified points.

The number of misclassified points from the *tuning* set are also expressed here. The result exists for each value of μ.

μ **value**	0	0.001	0.01	0.1	1	10	100
Misclassified Points	10	10	10	10	10	10	12

When μ is 100, there are 12 misclassified points. For every other μ, there are 10 misclassified points.

Lastly, the predictions of authorship for the 12 disputed papers are generated and shown below.
M = Madison / H = Hamilton

Paper	$\mu = 0$	$\mu = 0.001$	$\mu = 0.01$	$\mu = 0.1$	$\mu = 1$	$\mu = 10$	$\mu = 100$
1	M	M	M	M	M	M	M
2	M	M	M	M	M	M	M
3	H	M	M	M	M	M	M
4	M	M	M	M	M	M	M
5	M	M	M	M	M	M	M
6	M	M	M	M	M	M	M
7	M	M	M	M	M	M	M
8	M	M	M	M	M	M	M
9	M	M	M	M	M	M	M
10	M	M	M	M	M	H	M
11	M	H	H	H	H	H	H
12	M	M	M	M	M	M	H

So there are 11 papers written by Madison and 1 paper (paper number 3) written by Hamilton for all values of μ except when μ is 10 and 100, where there are 2 papers written by Hamilton and 10 papers written by Madison.

4.2 Question 2

Based on the results from Question 1, the effect μ has on ω is that the higher the value of μ is, the more the ω values decreases. As μ increases, the objective value function also increases. Furthermore, the number of misclassified points increases in proportion to μ, implying that the quality of the classifying hyperplane decreases when μ is large. Since μ of 0.001 has the minimum misclassified points, it is considered to be the best μ.
μ is the parameter in the minimization of the SVM objective function and its value increases by multiples of 10 as shown:

μ **Value**	0	0.001	0.01	0.1	1	10	100

4.3 Question 3

Having the μ value fixed to be 0.1, the 2 most effective attributes out of the 70 total attributes are the words *'also'* (3rd attribute) and *'upon'* (60th attribute). The number of misclassified points was calculated to be 0. This was determined through finding the minimum number of misclassified points to get the optimal pair of attributes from the training set, then finding the objective function value ($F(x)$) from the tuning set using the pair of attributes. Then lastly, comparing the $F(x)$ against less than 0 (Madison = 2) and greater than 0 (Hamilton = 1). The full dataset solution can be seen in Appendix B.

4.4 Question 4

The predictions of authorship for the 12 disputed papers using the optimal classifier from question 3 are shown below.
M = Madison / H = Hamilton

Paper Number	1	2	3	4	5	6	7	8	9	10	11	12
Author	H	M	M	M	M	M	H	M	M	M	M	M

So Hamilton wrote 2 papers (papers 1 and 7) and Madison wrote 10 papers.
The graph below displays the prediction of authorship for question 4. This graph was generated using MATLAB.

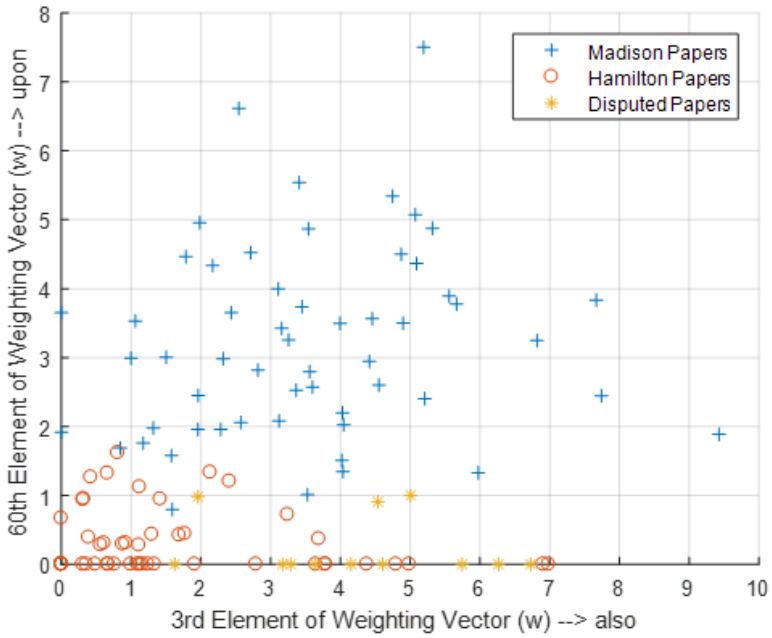

Figure 9: Prediction of Authorship representing Question 4

5 Conclusions

According to the results obtained, overall the team recommends Conjugate Gradient Method as the optimal algorithm for quadratic programming problems and for this disputed Federalist papers application problem. This algorithm outputs tolerable results that are not extreme on any of the performance criteria. However, depending on what performance criteria are weighted more importantly and the nature of the problem, the optimal choice could be either.

If the sample size or the number of iterations of the problem is large and time computation is not a strict condition, Hildreth's Quadratic Program algorithm is the optimal choice. This is because this algorithm has the best accuracy even though the time computation becomes much worse as the iterations increases. Conjugate may perform better for different scenarios but if the sample size is extremely large, which is a specific case, then Hildreth is the best choice.

On the contrary, if the sample size is not large enough and there is strong attention to time, Conjugate Gradient Method is the best option. This algorithm is able to execute and project results considerably well and consistently in speed. Constraints are hardly being violated using Conjugate as well, while Hildreth does not perform well for feasibility.

Regarding the disputed Federalist papers, it is evident that Madison is the deserving author who wrote the majority of the disputed papers. Questions 1 and 4 from SVM Application section concluded that Hamilton wrote only 1 and 2 papers respectively, while Madison wrote the rest of the 12 papers. Quadratic Programming became a successful tool to effectively determine the authorship of the disputed papers.

References

[1] C. Hildreth, "A quadratic programming procedure", 2018. [Online]. Available: https://onlinelibrary.wiley.com/doi/full/10.1002/nav.3800040113. [Accessed: 03- Apr- 2018].

[2] *Rose-hulman.edu*, 2018. [Online]. Available: https://www.rose-hulman.edu/~bryan/lottamath/congrad.pdf. [Accessed: 03- Apr- 2018].

[3] *Xn–vjq503akpco3w.top*, 2018. [Online]. Available: http://www.xn--vjq503akpco3w.top/literature/Nocedal_Wright_Numerical_optimization_v2.pdf. [Accessed: 03- Apr- 2018].

[4] "Quadratic Programming", *Mathworks.com*, 2018. [Online]. Available: https://www.mathworks.com/discovery/quadratic-programming.html. [Accessed: 03- Apr- 2018].

[5] A. Iusem and A. De Pierro, "On the convergence properties of Hildreth's quadratic programming algorithm", 2018. [Online]. Available: https://link.springer.com/content/pdf/10.1007/BF01580851.pdf. [Accessed: 03- Apr- 2018].

[6] T. Sugimoto, M. Fukushima and T. Ibaraki, "A parallel relaxation method for quadratic programming problems with interval constraints", 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/037704279400093G. [Accessed: 03- Apr- 2018].

[7] *Stanford.edu*, 2018. [Online]. Available: https://stanford.edu/class/ee364b/lectures/conj_grad_slides.pdf. [Accessed: 03- Apr- 2018].

[8] *Webdocs.cs.ualberta.ca*, 2018. [Online]. Available: https://webdocs.cs.ualberta.ca/~rgreiner/C-466/SLIDES/painless-conjugate-gradient.pdf. [Accessed: 04- Apr- 2018].

[9] 2018. [Online]. Available: https://www.quora.com/What-is-an-intuitive-explanation-of-quadr [Accessed: 04- Apr- 2018].

Appendices

Appendix A: ω Values

u = 0	u = 0.001	u = 0.01	u = 0.1	u = 1	u = 10	u = 100
w values	w values	w values	w values	w values	w values	w values
0.0283	0.0129	0.0129	0.0129	0.0129	0.026	0.0075
-0.1516	0	0	0.0001	0.0001	0.0054	0.0029
0.0329	0.0095	0.0096	0.0096	0.0096	0.0078	0.003
-0.1213	-0.0526	-0.0527	-0.0528	-0.0528	-0.0351	-0.0096
-0.1575	-0.0042	-0.0041	-0.004	-0.004	-0.0113	-0.0183
-0.0805	-0.0062	-0.0061	-0.0061	-0.0061	-0.0053	0.0159
0.2189	0.0374	0.0376	0.0376	0.0376	0.029	0.0041
0.0306	-0.0166	-0.0162	-0.0161	-0.0161	-0.0232	-0.0081
-0.2856	-0.0222	-0.0221	-0.0221	-0.022	-0.0244	-0.0008
0.0227	-0.0063	-0.0061	-0.0061	-0.006	-0.0072	0.0043
-0.2087	-0.0128	-0.0128	-0.0128	-0.0128	-0.0138	-0.0001
-0.2136	-0.0071	-0.0072	-0.0072	-0.0072	0.0025	0.0007
-0.0486	-0.0869	-0.0868	-0.0868	-0.0868	-0.0706	-0.0148
-0.2632	0.0122	0.0121	0.0121	0.0121	0.0157	0.007
-0.9195	0.0031	0.003	0.003	0.003	0.0002	-0.0006
1.678	0.0041	0.0042	0.0042	0.0042	0.0023	0.0002
-0.3929	0.0026	0.0024	0.0024	0.0024	0.0033	0.001
0.0227	-0.0142	-0.0145	-0.0146	-0.0146	-0.011	-0.0009
-0.0628	-0.0046	-0.0046	-0.0045	-0.0045	-0.0025	-0.0024
0.0554	-0.001	-0.0005	-0.0005	-0.0005	0.0079	0.0002
0.4515	0.0216	0.0217	0.0217	0.0218	0.0128	-0.0009
0.2422	-0.0239	-0.024	-0.024	-0.024	-0.0015	-0.0017
0.2058	0.0055	0.0055	0.0055	0.0055	0.0084	0.0052
0.0604	0.007	0.0068	0.0068	0.0068	0.0024	0.0001
-0.1927	-0.0281	-0.028	-0.0279	-0.0279	-0.0135	-0.0048
0.0422	0.0408	0.0409	0.0409	0.0409	0.0246	0.0152
0.0544	0.0348	0.035	0.0351	0.0351	0.006	-0.0157
-0.3389	0.0225	0.0227	0.0227	0.0227	0.0214	0.0067
-0.0217	0.087	0.0871	0.0871	0.0871	0.0427	0.016
-0.2028	-0.0841	-0.0842	-0.0843	-0.0843	-0.0585	-0.0179
0.0915	-0.0344	-0.0343	-0.0343	-0.0343	-0.0066	0.0021
0.1545	-0.0183	-0.0183	-0.0183	-0.0183	-0.0197	0.0013
0.1793	-0.0073	-0.0073	-0.0073	-0.0073	-0.0055	-0.0004
-0.1758	0.0238	0.0238	0.0238	0.0238	-0.0007	0.0013
-0.3907	0.0001	0.0002	0.0002	0.0002	0.0003	0
0.1721	0.0097	0.0096	0.0096	0.0096	0.0064	-0.0016
-0.0308	-0.0122	-0.012	-0.012	-0.012	-0.0048	-0.0053
-0.7089	0.0058	0.0058	0.0058	0.0058	0.0075	0.0001
0.0126	0.0511	0.0511	0.0511	0.0511	0.0417	0.0133
-0.1592	-0.0672	-0.0669	-0.0668	-0.0668	-0.0587	-0.0158
0.0538	-0.0021	-0.0018	-0.0018	-0.0018	-0.0073	-0.0014
0.623	0.0515	0.0514	0.0514	0.0514	0.0132	0.0015
0.0181	0.0254	0.0254	0.0254	0.0254	0.0216	0.0017
-0.0469	0.0378	0.038	0.0379	0.0379	0.0306	0.0023
-0.1668	0.0376	0.0377	0.0377	0.0377	0.0235	0.0013
-0.2405	-0.0091	-0.0092	-0.0092	-0.0092	0.0038	0.0017
0.1351	-0.0421	-0.0423	-0.0423	-0.0423	-0.0161	-0.0019
-0.5236	-0.0275	-0.0275	-0.0275	-0.0275	-0.0057	-0.0017
-0.0355	0.0023	0.0022	0.0022	0.0022	-0.0016	-0.0009
-0.4025	-0.0119	-0.0121	-0.0121	-0.0121	-0.0014	-0.0007
-0.0792	-0.0382	-0.0382	-0.0382	-0.0382	-0.0086	0.0014
-0.0405	-0.0215	-0.0215	-0.0215	-0.0215	-0.0181	-0.0076
0.181	0.0413	0.0414	0.0414	0.0414	0.0166	0.0018
0.122	-0.0059	-0.0059	-0.0059	-0.0059	-0.0024	-0.001
0.5278	0.0465	0.0463	0.0463	0.0463	0.035	0.0081
-0.4367	0.0096	0.0096	0.0096	0.0096	-0.0003	-0.0001
-0.007	0.0325	0.0327	0.0327	0.0327	0.0313	0.0052
0.0774	0.0778	0.0776	0.0776	0.0776	0.0673	0.0165
0.2815	0.0024	0.0024	0.0024	0.0023	-0.0047	-0.0011
0.4798	0.0807	0.0806	0.0806	0.0806	0.0562	0.0119
0.1803	-0.0089	-0.009	-0.009	-0.009	0.0079	-0.0018
-0.036	0.0077	0.0079	0.008	0.008	0.0116	-0.0007
0.0718	0.0258	0.0257	0.0258	0.0258	0.0111	0.0007
-0.1891	0.0405	0.0405	0.0405	0.0405	0.0144	0.0011
-0.1337	-0.0174	-0.0175	-0.0175	-0.0175	-0.0101	0.0021
-0.0998	-0.0337	-0.034	-0.034	-0.034	0.0071	0.0021
-0.0459	0.0352	0.0351	0.0351	0.0351	0.02	0.0089
0.0339	0.0347	0.0348	0.0349	0.0349	0.0193	0.0003
0.0924	0.0626	0.0624	0.0624	0.0624	0.0477	0.0115
0.5688	-0.0102	-0.0101	-0.0101	-0.0101	0.0043	0.0003

Appendix B: Question 3 Misclassified Points Results

-1.69658	2	2	0
5.722934	1	1	0
2.673501	1	1	0
9.691656	1	1	0
1.123529	1	1	0
-1.61933	2	2	0
-0.32854	2	2	0
5.257136	1	1	0
2.516291	1	1	0
3.445645	1	1	0
6.014707	1	1	0
-1.08373	2	2	0
-1.39258	2	2	0
0.28381	1	1	0
-1	2	2	0
4.920887	1	1	0
-1.59136	2	2	0
1	1	1	0
-1.58923	2	2	0
1.297608	1	1	0
Misclassified Points			0