2006

Clustering High Dimensional Data Using SVM

Tam P. Ngo

San Jose State University

Clustering High Dimensional Data Using SVM

A Writing Project

Presented to

The Faculty of the Department of Computer Science

San José State University

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

by

Tam P. Ngo

December 2006

APPROVED FOR THE DEPARTMENT OF COMPUTER SCIENCE

Dr. Tsau Young Lin

Dr. Christopher Pollett

Dr. H. Chris Tseng

APPROVED FOR THE UNIVERSITY

Abstract

The Web contains massive amount of documents from across the globe to the point where it has become impossible to classify them manually. This project's goal is to find a new method for clustering documents that are as close to humans' classification as possible and at the same time to reduce the size of the documents. This project uses a combination of Latent Semantic Indexing (LSI) with Singular Value Decomposition (SVD) calculation as well as Support Vector Machine (SVM) classification. With SVD, data sets are decomposed and can be truncated to reduce the data sets size. The reduced data set will then be used to cluster. With SVM, clustered data set is used for training to allow new data to be classified based on SVM's prediction. The project's result show that the method of combining SVD and SVM is able to reduce data size and classifies documents reasonably compared to humans' classification.

Table of Contents

List of Tables and Figures

1. Introduction

Ever since the World Wide Web has become popular, document clustering has become increasingly more important. With billions of documents on the Web, it is impossible to classify all these documents by humans. The challenge is to find a way to organize this massive data in some meaningful structure. This project proposes a method that can cluster documents reasonably.

The project deals with clustering high dimensional data. The data used are processed documents organized in a text file that contains category labels and term frequency–inverse document frequency (tf–idf) values. Data sets used in the research are classified by humans and have been processed into tf-idf values. By using human-classified data set, we can compare our clustering method with humans' classification.

The first few sections of the report discuss and analyze Support Vector Machine (SVM) and Latent Semantic Indexing (LSI). This will allow the reader to understand how these methods are applied to the project. The last few sections discuss the algorithms used and the analysis of the results after applying methods from previous sections.

2. Support Vector Machine

Vladimir Vapnik and his colleagues first introduced SVM in 1963. Support Vector Machine (SVM) is a learning machine that uses supervised learning to perform data classification and regression (“Support vector machine,” 2006). The meaning of supervised learning is learning from examples or from a teacher. For instance, children learn to tell the difference between objects (e.g. dogs from cats, women from men, fruits from vegetables) by having those objects pointed out to them (Cristianini, N., & Shawe-Taylor, J., 2000). Every time they see a new object, they can determine what it is by recognizing similarities from what they already know. They are in fact putting the new object in a category. Supervised learning is the same. In SVM, each line within the data set is given a label and SVM learns the data and puts the new data in the group that is closest to the learned data.

2.1 What SVM is Used For

SVM is primarily used for categorization. Some examples of SVM usage include bioinformatics, signature/hand writing recognition, image and text classification, pattern recognition (Cristianini, N., & Shawe-Taylor, J., 2000), and e-mail spam categorization (Drucker, H., Wu, D., & Vapnik, V. N., 1999). Many research documents such as the ones mentioned above have shown that SVM can classify reasonably well. In this project, SVM is used for text classification.

Text classification is a method used to put text into meaningful groups. Beside SVM, there are many other methods for text classification such as Bayes and k-Nearest Neighbor. Based on many research papers (Joachims, T., 1998), SVM outperforms many, if not all, popular methods for text classification. The studies also show that SVM is effective, accurate, and can work well with small amount of training data.

2.2 Motivation for SVM

The concept of SVM is quite amazing once the reader understands the math behind it. For motivational purpose, the following images show the classification problem. Each dot on the images represents a document that has been grouped in a two-dimensional space. The goal is to find the best boundary that will separate these documents. Figure 1(a) looks quite simple; one needs to only find a straight line between the groups. Naturally, the line that lies exactly in the middle of the groups is chosen. However, Figure 1(b) shows a data set behavior that is much more complex. Drawing a straight line to separate the two groups is impossible and much harder in hundreds or even thousands dimensional space. Nevertheless, SVM can do it!

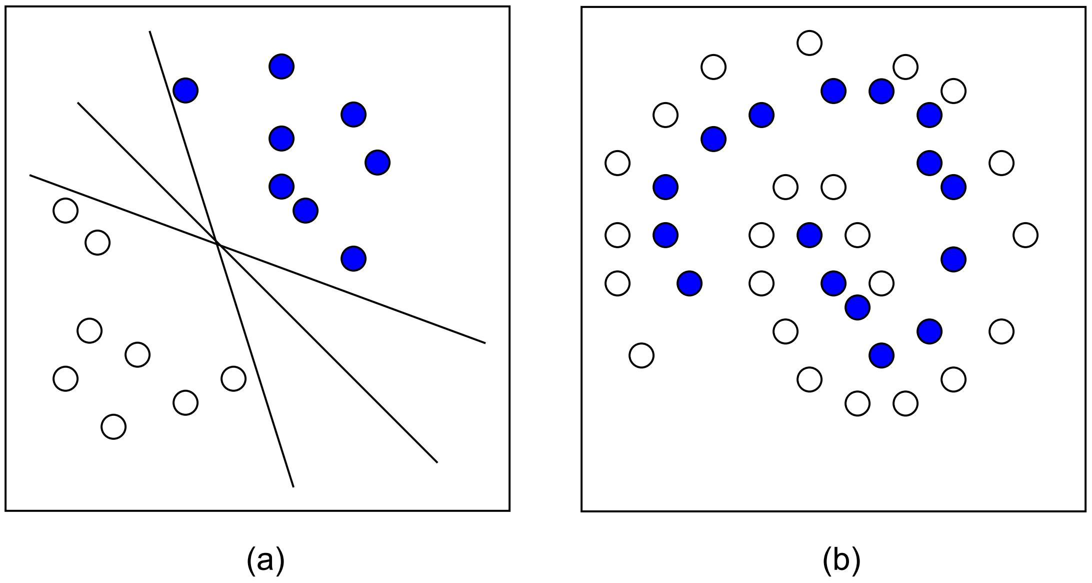


Figure 1. The Separating Problem
Source: Author's Research

2.3 How SVM Works

The idea for SVM is to find a boundary (known as a hyperplane) or boundaries that separate clusters of data. SVM does this by taking a set of points and separating those points using mathematical formulas. The following figure illustrates the data flow of SVM.

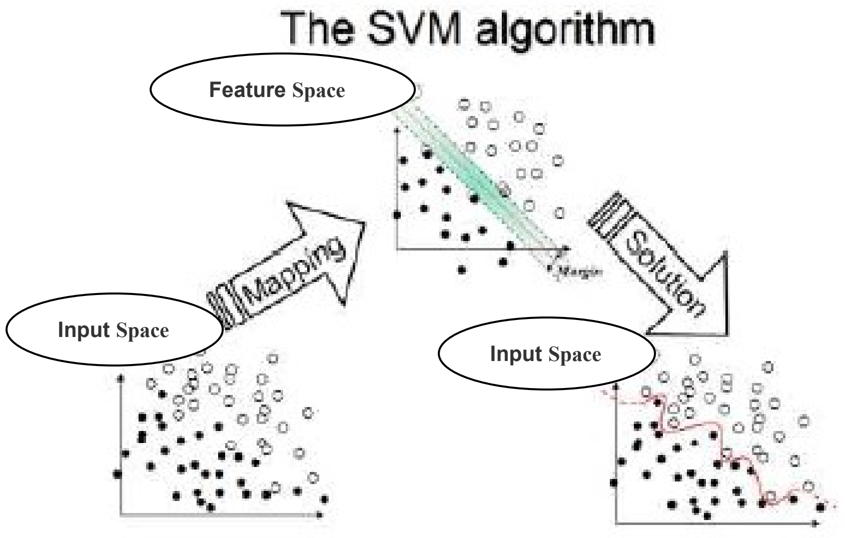


Figure 2. SVM Process Flow
Source: DTREG

In Figure 2, data are input in an input space that cannot be separated with a linear hyperplane. To separate the data linearly, points are map to a feature space using a kernel method. Once the data in the feature space are separate, the linear hyperplane gets map back to the input space and it is shown as a curvy non-linear hyperplane. This process is what makes SVM amazing.

The SVM's algorithm first starts learning from data that has already been classified, which is represented in numerical labels (e.g. 1, 2, 3, etc.) with each number representing a category. SVM then groups the data with the same label in each convex hull. From there, it determines where the hyperplane is by calculating the closest points between the convex hulls (Bennett, K. P., & Campbell, C., 2000). The figure below illustrates this. Once SVM determines the points that are closest to each other, it calculates the hyperplane, which is a plane that separates the labels.

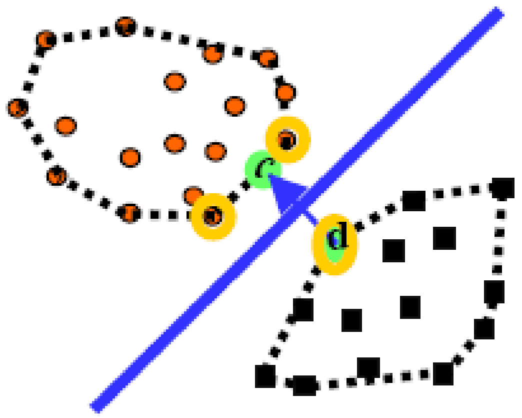


Figure 3. SVM Convex Hulls
Source: Bennett, K. P., & Campbell, C., 2000

2.3.1 Simple SVM Example

Let us use a few simple points to illustrate the concept of SVM. The following example is similar to Dr. Guestrin's lecture (Guestrin, C., 2006). Given the following points with corresponding classes (labels) in Figure 4, find a hyperplane that separated the points.

Table 1. Simple Data in 1-Dimension

Class	**X_1**
+1	0
-1	1
-1	2
+1	3

Source: Author's Research

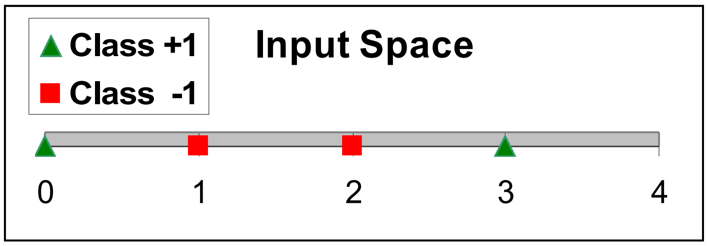


Figure 4. Simple Data in an Input Space
Source: Author's Research

As Figure 4 shows, these points lay on a 1-dimensional plane and cannot be separated by a linear hyperplane. The first step is to find a kernel that maps the points into the feature space, then within the feature space, find a hyperplane that separates the points. A simple kernel that would do the trick is $\Phi(X_1) = (X_1, X_1^2)$. This kernel is actually a polynomial type. As the reader sees, this kernel will map the points to a 2-dimensional feature space by multiplying the points to the power of 2. From calculating the kernels, we get (0, 0, +1), (1, 1, -1), (2, 4, -1), (3, 9, +1)

Table 2. Simple Data in 2-Dimension

Class	**X_1**	**X_1^2**
+1	0	0
-1	1	1
-1	2	4
+1	3	9

Source: Author's Research

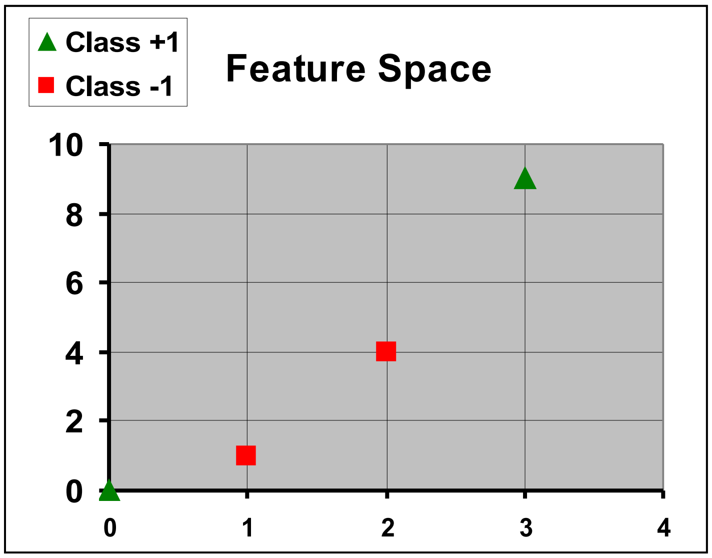


Figure 5. Simple Data in a Feature Space
Source: Author's Research

The next step is finding a hyperplane. This can be done by using the following equations, which are introduced by Vapnik, V. N. in his book, "The Nature of Statistical Learning Theory" in chapter 5.

$\langle \mathbf{w} \bullet \mathbf{x} \rangle + b = +1$ (positive labels) (1)
$\langle \mathbf{w} \bullet \mathbf{x} \rangle + b = -1$ (negative labels) (2)
$\langle \mathbf{w} \bullet \mathbf{x} \rangle + b = 0$ (hyperplane) (3)

From these equations, find the unknowns, **w** and b. Expanding the equations for the SVM problem will get:

$w_1x_1 + w_2x_2 + b = +1$
$w_1x_1 + w_2x_2 + b = -1$
$w_1x_1 + w_2x_2 + b = 0$

Solve **w** and b for the positive labels using equation, $w_1x_1 + w_2x_2 + b = +1$.

$w_1x_1 + w_2x_2 + b = +1$

$\rightarrow w_10 + w_20 + b = +1$
$\rightarrow w_13 + w_29 + b = +1$

Solve **w** and b for the negative labels using equation, $w_1x_1 + w_2x_2 + b = -1$.

$w_1x_1 + w_2x_2 + b = -1$

$\rightarrow w_1 1 + w_2 1 + b = -1$
$\rightarrow w_1 2 + w_2 4 + b = -1$

By using linear algebra, we find that the solution is $w_1 = -3$, $w_2 = 1$, $b = 1$, which satisfies the above equations. Many times, there are more than one solution or there may be no solution, but SVM can find the optimal solution that returns a hyperplane with the largest margin.

With the solutions: $w_1 = -3$, $w_2 = 1$, $b = 1$, positive plane, negative plane, and hyperplane can be calculated.

Table 3. Calculation Results of Positive, Negative, and Hyperplane

Positive Plane:	**Negative Plane:**	**Hyperplane:**
$\langle \mathbf{w} \bullet \mathbf{x} \rangle + b = +1$	$\langle \mathbf{w} \bullet \mathbf{x} \rangle + b = -1$	$\langle \mathbf{w} \bullet \mathbf{x} \rangle + b = 0$
$w_1x_1 + w_2x_2 + b = +1$	$w_1x_1 + w_2x_2 + b = -1$	$w_1x_1 + w_2x_2 + b = 0$
$\rightarrow -3x_1 + 1x_2 + 1 = +1$	$\rightarrow -3x_1 + 1x_2 + 1 = -1$	$\rightarrow -3x_1 + 1x_2 + 1 = 0$
$\rightarrow x_2 = 3x_1$	$\rightarrow x_2 = -2 + 3x_1$	$\rightarrow x_2 = -1 + 3x_1$

X_1	**X_2**	**X_1**	**X_2**	**X_1**	**X_2**
0	0	0	-2	0	-1
1	3	1	1	1	2
2	6	2	4	2	5
3	9	3	7	3	8

Source: Author's Research

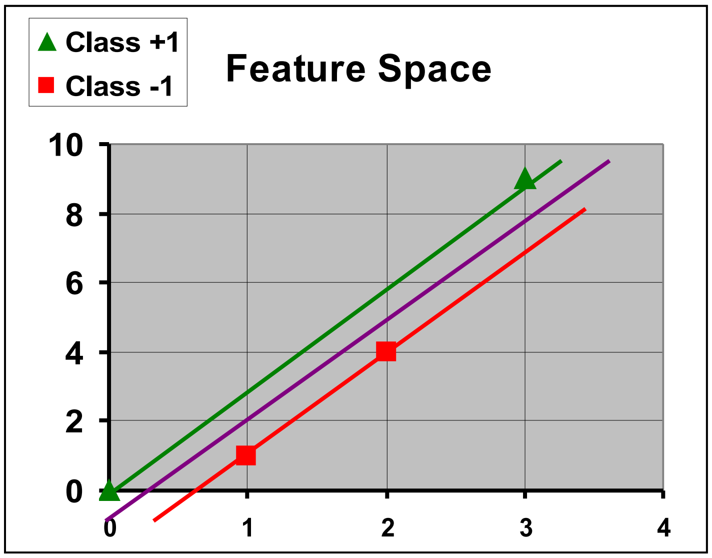


Figure 6. Simple Data in a Feature Space Separated by a Hyperplane
Source: Author's Research

Thus, we have the model that contains the solution for **w** and b and with margin $2/\sqrt{(\mathbf{w} \bullet \mathbf{w})}$. The margin is calculated as follow.

$$2/\sqrt{(\mathbf{w} \bullet \mathbf{w})} \qquad (4)$$

$\Rightarrow 2/\sqrt{(-3^2 + 1^2)} \Rightarrow$ margin = 0.632456

In SVM, this model is used to classify new data. With the solutions, new data can be classified into category. For example, if the result is less than or equal -1, the new data belongs to the -1 class and if the result is greater than or equal to +1, the new data belongs to the +1 class.

2.3.2 SVM is Not That Simple

In reality, most data set, if not all data set, are not as clean and well behaved as the example on the previous section. There will be some points that are on the wrong side of the class, points that are far off from the classes, or points that are mixed together in a spiral or checkered pattern. Fortunately, researchers have looked into those problems and tackled them.

To solve the few points that are in the wrong class, SVM minimized the following equation to create what is called a soft-margin hyperplane.

$$\Phi(w,\xi) = \frac{1}{2}(w \cdot w) + C\left(\sum_{i=1}^{\ell} \xi_i\right) \quad (5)$$

s.t. $y_i (\langle \mathbf{w} \bullet \mathbf{x}_i \rangle - b) \geq 1 - \xi_i$
and $i = 1, 2, 3, \ldots \ell$

Figure 7. Equation to Determine the Soft-Margin Hyperplane
Source: Vapnik, V. N., 2000

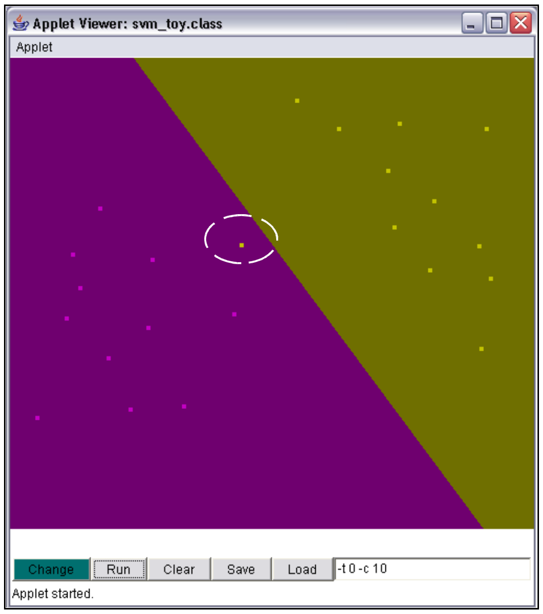
Here, C is a given value. This value is important to train data using SVM for the project. The parameter C will be explained in the next section. If the reader would like to understand more in depth of how SVM works refer to Vapnik's book, "The Nature of Statistical Learning Theory".

2.4 LIBSVM: A Java Library for SVM

A good SVM library can take years to develop. LIBSVM is a well-known library for SVM that is developed by Chih-Chung Chang and Chih-Jen Lin. This project will use LIBSVM to train and predict data. LIBSVM has 5 SVM types, 4 kernel methods, and many functions to help prepare and process data.

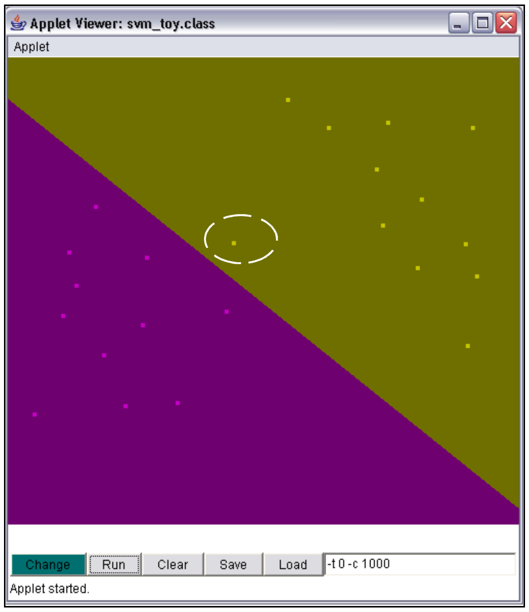
2.4.1 Choosing Parameter C

The parameter C in LIBSVM determines the soft-margin. When C is very small (Figure 8(a)), SVM only considers about maximizing the margin and the points can be on the wrong side of the plane. When the C value is very large (Figure 8(b)), SVM will want very small slack penalties to make sure that all data points in each group are separated correctly. Figure 8 shows this.



-t 0 -c 10
Applet started.

(a)



-t 0 -c 1000
Applet started.

(b)

Figure 8. Parameter C Example
Source: LIBSVM

In Figure 8(a), C is set to the value of 10 and the reader can see that SVM purposely classifies one of the points with the incorrect data group in favor of having a larger margin. However, in Figure 8(b), when C is set to 1000, SVM is in favor of getting all the labeled data points in the correct group, thus having a smaller margin. In the project, most of the parameters used are the default parameters recommended by Chih-Chung Chang and Chih-Jen Lin, with the exception of parameter C and sometimes the selection of kernel methods.

2.4.2 4 Basic Kernel Types

The kernel functions developed for SVM are still an on-going research. One of the aspects of SVM is that one can develop his or her own kernel to fit the data type used. Fortunately, LIBSVM has implemented 4 basic kernel types: linear, polynomial, radial basis function, and sigmoid as follow.

```
-t kernel_type : set type of kernel function (default 2)
        0 -- linear: u'*v
        1 -- polynomial: (gamma*u'*v + coef0)^degree
        2 -- radial basis function: exp(-gamma*|u-v|^2)
        3 -- sigmoid: tanh(gamma*u'*v + coef0)
```

The reader can experiment with each kernel to determine which one works best with the data set. In the article “A Practical Guide to Support Vector Classification”, Chih-Wei Hsu, Chih-Chung Chang, and Chih-Jen Lin recommend the readers to begin with radial basis function (RBF) kernel. In this project, RBF will be used.

3. Data Preparation Using SVD

Originally, this project’s goal was to use SVM to separate data from LSI matrix into categories and to reduce its size. However, in order to separate the data, SVM requires training data to be in categories. This project’s intention is to cluster these data, however, SVM does not cluster the data, it can only classify data. The author has spent many hours researching for an unsupervised SVM; unfortunately, there is not such proven working method. Thus, the author uses a different approach, clustering data using Singular Value Decomposition (SVD) and then predicting the category of the new data using SVM based on the clustered data.

3.1 Singular Value Decomposition (SVD)

Singular Value Decomposition (SVD) is a method that separates a matrix into three parts; left eigenvectors, singular values, and right eigenvectors (Garcia, E., 2006). It can be used to decompose data such as images and text. Since SVM requires supervised learning, SVD is chosen to cluster the data and give the data its label. The following example used is from Grossman and Frieder's textbook and a tutorial by Dr. E. Garcia to show how LSI is constructed and from there how data are being used as training data for SVM. This allow us to create an unsupervised learning SVM.

3.1.1 SVD Example

Given a matrix **A**, we can factor it into three parts: **U**, **S**, and **V**^{T}.

A =

	D1	D2	D3	D4
a	1	1	1	0
arrived	0	1	1	0
damaged	1	0	0	0
delivery	0	1	0	0
fire	1	0	0	0
gold	1	0	1	1
in	1	1	1	0
of	1	1	1	0
shipment	1	0	1	0
silver	0	2	0	1
truck	0	1	1	1

Where the documents contain the following terms:

D1: Shipment of gold damaged in a fire
D2: Delivery of silver arrived in a silver truck
D3: Shipment of gold arrived in a truck
D4: Gold Silver Truck

Note that in Grossman and Frieder's textbook and Dr. E. Garcia's tutorial, the fourth document is a search query however for our program, we included that query as part of the data. Aside from LIBSVM, the project will use a basic java matrix package (JAMA) developed by a team at the MathWorks and the National Institute of Standards and Technology (NIST) (Hicklin, J., Moler, C., & Webb, P., 2005). The following SVD calculation is done using the JAMA java package.

Doing SVD to **A** will give **U**, **S**, and $\mathbf{V}^T$.

U =

```
 0.3966  -0.1282  -0.2349   0.0941
 0.2860   0.1507  -0.0700   0.5212
 0.1106  -0.2790  -0.1649  -0.4271
 0.1523   0.2650  -0.2984  -0.0565
 0.1106  -0.2790  -0.1649  -0.4271
 0.3012  -0.2918   0.6468  -0.2252
 0.3966  -0.1282  -0.2349   0.0941
 0.3966  -0.1282  -0.2349   0.0941
 0.2443  -0.3932   0.0635   0.1507
 0.3615   0.6315  -0.0134  -0.4890
 0.3428   0.2522   0.5134   0.1453
```

S =

```
 4.2055 0.0000 0.0000 0.0000
 0.0000 2.4155 0.0000 0.0000
 0.0000 0.0000 1.4021 0.0000
 0.0000 0.0000 0.0000 1.2302
```

V =

```
 0.4652 -0.6738 -0.2312 -0.5254
 0.6406  0.6401 -0.4184 -0.0696
 0.5622 -0.2760  0.3202  0.7108
 0.2391  0.2450  0.8179 -0.4624
```

V^T =

```
  0.4652  0.6406  0.5622  0.2391
 -0.6738  0.6401 -0.2760  0.2450
 -0.2312 -0.4184  0.3202  0.8179
 -0.5254 -0.0696  0.7108 -0.4624
```

If the matrices, **U**, **S**, and **V**^{T} are multiplied together, the original matrix **A** is reconstructed. One of the nice properties of SVD is that after the matrix is decomposed its dimension could be reduced by choosing to keep only the largest singular values in the **S** matrix. In this example, singular values 4.2055 and 2.4155 are kept. This is also called a rank 2 approximation. To accomplish this, the last 2 columns of **U** and **V**, and the last 2 columns and rows of **S** are dropped. Thus, the following values of the matrices are left.

U' =

```
 0.3966  -0.1282
 0.2860   0.1507
 0.1106  -0.2790
 0.1523   0.2650
 0.1106  -0.2790
 0.3012  -0.2918
 0.3966  -0.1282
 0.3966  -0.1282
 0.2443  -0.3932
 0.3615   0.6315
 0.3428   0.2522
```

S’ =

```
 4.2055  0.0000
 0.0000  2.4155
```

V’ =

```
 0.4652  -0.6738
 0.6406   0.6401
 0.5622  -0.2760
 0.2391   0.2450
```

With **V’** containing the document vectors, as follows:

D1’ (0.4652, -0.6738)
D2’ (0.6406, 0.6401)
D3’ (0.5622, -0.2760)
D4’ (0.2391, 0.2450)

The goal is to use SVD to cluster data. This is done by calculating cosine similarities between each document. This will return the distance between the vector documents.

$$\text{sim}(\mathbf{D'}, \mathbf{D'}) = (\mathbf{D'} \bullet \mathbf{D'}) / (|\mathbf{D'}|\ |\mathbf{D'}|) \qquad (6)$$

Calculate for **D1’**:

sim(**D1’, D2’**) = (**D1’• D2’**) / (**|D1’| |D2’|**)
sim(**D1’, D3’**) = (**D1’• D3’**) / (**|D1’| |D3’|**)
sim(**D1’, D4’**) = (**D1’• D4’**) / (**|D1’| |D4’|**)
….
until **D4’** is calculated and compared

Example result for **D1’**:

$$\text{sim}(\mathbf{D1'}, \mathbf{D2'}) = \frac{((0.4652 * 0.6406) + (-0.6738 * 0.6401))}{\sqrt{(0.4652)^2 + (-0.6738)^2} * \sqrt{(0.6406)^2 + (0.6401)^2}} = -0.1797$$

$$\text{sim}(\mathbf{D1'}, \mathbf{D3'}) = \frac{((0.4652 * 0.5622) + (-0.6738 * -0.2760))}{\sqrt{(0.4652)^2 + (-0.6738)^2} * \sqrt{(0.5622)^2 + (-0.2760)^2}} = 0.8727$$

$$\text{sim}(\mathbf{D1'}, \mathbf{D4'}) = \frac{((0.4652 * 0.2391) + (-0.6738 * 0.2450))}{\sqrt{((0.4652)^2 + (-0.6738)^2)} * \sqrt{((0.2391)^2 + (0.2450)^2)}} = -0.1921$$

From the result, the reader can see that the first document, D1 is most similar to D3, since it returns the highest value. Doing this procedure for each document, the following results return.

D1: 3
D2: 4
D3: 1
D4: 2

Each document is paired with another document that it is closest. Then they are grouped into clusters; D1 and D3 in one cluster (label as 1) and D2 and D4 in another cluster (label as 2).

Result:

label 1: 1 3
label 2: 2 4

This process of preparing the data is very costly. One good thing is this process does not need to be done on the fly. For future work, this process can be improved to make it much more efficient. For example, since finding the closest document can be done independently, it can be calculated in parallel.

3.1.2 Checking the Results using SVM

SVM is a method that only learns from what is given to it. This could be data collected and putted into categories by humans or it could be data that is clustered by an application. How accurate the data is depends on the training inputs. SVM will try to predict the label of the data based on the training input. In this section, we want to know how well SVM predicts the clustered data.

The following data is inputted with the corresponding labels based on the previous section's results. To run the data on SVM, we will use the radial basis function kernel and a C value of 10,000 to ensure that all labels are in the correct group.

SVM input format:

```
1 1:1.00 2:0.00 3:1.00 4:0.00 5:1.00 6:1.00 7:1.00 8:1.00 9:1.00 10:0.00 11:0.00
2 1:1.00 2:1.00 3:0.00 4:1.00 5:0.00 6:0.00 7:1.00 8:1.00 9:0.00 10:2.00 11:1.00
1 1:1.00 2:1.00 3:0.00 4:0.00 5:0.00 6:1.00 7:1.00 8:1.00 9:1.00 10:0.00 11:1.00
2 1:0.00 2:0.00 3:0.00 4:0.00 5:0.00 6:1.00 7:0.00 8:0.00 9:0.00 10:1.00 11:1.00
```

Table 4. Results from SVM's Prediction on Original Data

Documents use for Training	Predict the Following Document	SVM Prediction Result	SVD Cluster Result
D1, D2, D3	D4	1.0	2
D1, D2, D4	D3	1.0	1
D1, D3, D4	D2	2.0	2
D2, D3, D4	D1	1.0	1

Source: Author's Research

The result shows that SVM is 75% correct in its prediction. In the result, using D1, D2, and D3 as training data, SVM predicts that D4 belongs to label 1; however SVD calculation result shows that D4 is closest to D2, which is label 2. Using the original data set is not a good way to predict the labels since the data contain noise terms such as "of", "in", and "a". One of the reasons SVD was chosen to cluster data is due to its ability to reduce the matrices, hence, the truncated **V'** matrix is used instead.

SVM input format (truncated **V** matrix):

```
1 1:0.4652 2:-0.6738
2 1:0.6406 2:0.6401
1 1:0.5622 2:-0.2760
2 1:0.2391 2:0.2450
```

Table 5. Results from SVM's Prediction on Reduced Data

Documents use for Training	Predict the Following Document	SVM Prediction Result	SVD Cluster Result
D1, D2, D3	D4	2.0	2
D1, D2, D4	D3	1.0	1
D1, D3, D4	D2	2.0	2
D2, D3, D4	D1	1.0	1

Source: Author's Research

With the same settings on SVM, but using the truncated **V'** matrix, the result shows that SVM prediction is 100% accurate. Thus, SVM predicted the same as SVD on all of the documents. Since **V'** has only 2 columns, we can analysis the data graphically.

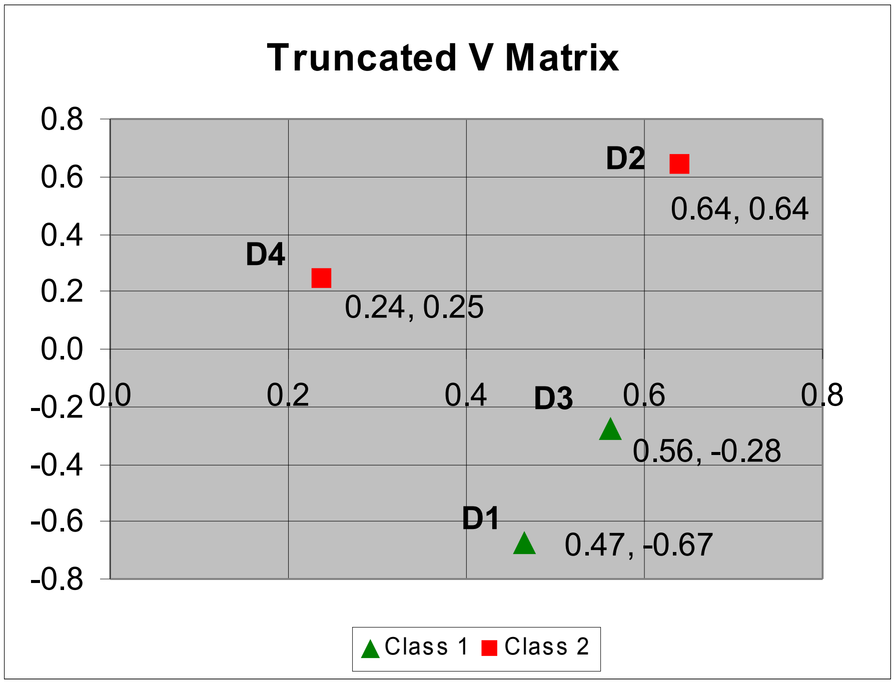


Figure 9. Truncated V Matrix on a Graph
Source: Author's Research

On the graph, the reader can see that D1 and D3 are obviously closer to each other than the other documents and D4 is slightly closer to D2 than the others. By analyzing the terms for each documents, it is shown that "D1: Shipment of gold damaged in a fire" and "D3: Shipment of gold arrived in a truck" are clustered together since they share the words "Shipment of gold". "D2: Delivery of silver arrived in a silver truck" and "D4: Gold Silver Truck" are clustered together since they share "silver truck" and "silver" appears in D2 two times. This shows that LSI is based on co-occurrence of the terms (Garcia, E., 2006).

3.2 Analysis of the Rank Approximation

Ranking approximation in SVD is important since it reduces the data noise and size. There is no right answer to which approximation value is the best. In the below section, the ranking approximation is analyzed.

Table 6. Cluster Results from Different Ranking Approximation

Rank 1	Rank 2	Rank 3	Rank 4
D1: 4 D2: 4 D3: 4 D4: 3	D1: 3 D2: 4 D3: 1 D4: 2	D1: 3 D2: 3 D3: 1 D4: 3	D1: 2 D2: 3 D3: 2 D4: 2
label 1: 1 4 2 3	label 1: 1 3 label 2: 2 4	label 1: 1 3 2 4	label 1: 1 2 3 4

Source: Author's Research

D1: Shipment of gold damaged in a fire.
D2: Delivery of silver arrived in a silver truck.
D3: Shipment of gold arrived in a truck.
D4: Gold Silver Truck

By using the rank approximation, essential values in each document are kept. In the result for rank 1 approximation, matrices are truncated leaving only one column. Although the results show that there are some pairing between documents, this is not so. This is because the algorithm forces each document to pick one other document that is closest to itself and in the end, 4 is selected since it is the last document the algorithm use to compare. This is also the same with pairing D4 with 3 (3 being the last document that is compared). All the calculations for cosine similarities result in 1.0, except for comparing D1 with D2, which show results as 0.9877. This result makes sense since between D1 and D2, the only common terms are "of", "in", and "a", which are also common with other documents. With rank 1, the documents are too close to each other.

Rank 2 approximation’s results have already been looked at in the previous section. In this data set, it seems that rank 2 is the prefer value. Rank 3 now included importance on additional values. It is easy to see why D1 and D3 are paired and so as for D2 and D3, since they have more terms that are in common with each other. For the pair D4 and D3, both shared terms: “gold” and “truck”, but D2 also shares terms: “silver” and “truck”. However, D2 has “silver” twice. In rank 4, the data set used is “as-is” with no truncation. As one can see, most documents are paired with D2. It is believed that the reason for this is because D2 has the most terms. This assumption is also pointed out by Dr. E. Garcia’s tutorial. For this project, small ranking approximation will be used since it seems to yield the best results.

4. The Project

Now that a simple example has been shown in the previous sections, the project will use the same process to cluster larger data set. The purpose is to see how well the data clusters using SVD and running the clustered data using SVM to predict new data. One might wonder why use SVM when SVD can do the same job. Based on the algorithms, SVM is faster and it has the ability to separate the data nicely. With SVM, new data is classified without having to process cosine similarities.

4.1 Tf-idf

In order to use SVM for this project, the documents need to be represented in numerical values. A way to do this is to calculate the term frequency–inverse document frequency (tf–idf) values.

$$\mathrm{tf} = \frac{n_i}{\sum_k n_k} \tag{7}$$

$$\mathrm{tfidf} = \mathrm{tf} \cdot \log\left(\frac{|D|}{|(d_j \supset t_i)|}\right) \tag{8}$$

Equation (7) and (8) show one way of calculating tf-idf. Tf stands for term frequency with n_i as the number of occurrences of a term in a document and $\sum_k n_k$ as the number of occurrences of all terms in the same document. The tf equation is then multiplied by the inverse document frequency (idf) equation. Idf, in equation (8), is the log of |D|, which is the total number of all considered documents, divided by $| d_j \supset t_i |$, which is the number of documents that a term appears ("Tf–idf," 2006). Table 7 shows an example of the structure of the matrix.

Table 7. LSI Matrix

	Term_1	Term_2	Term_3	Term_4	Term_5	…	Term_n
Doc_1	tf-idf_1	tf-idf_2	tf-idf_3	tf-idf_4	tf-idf_5	…	tf-idf_n
Doc_2	…	…	tf-idf				…
Doc_3	…		…	tf-idf			…
Doc_4	…			…	tf-idf		…
…	…				…	…	…
Doc_m	tf-idf	tf-idf	tf-idf	tf-idf	tf-idf	tf-idf	tf-idf

Source: Author's Research

Fortunately, there are many data sets in tf-idf format that have already been human-classified for the public to use to compare their results such as (Fan, R., 2006) and (Reuters-21578). Therefore, it is not necessary to compute the tf-idf values for the project.

Once a matrix of tf-idf values has been obtained, it needs to be decomposed using SVD. Both data sets that are used for training and predicting need to be truncated with SVD by the same ranking approximation value. This way the same data properties are used. There are two ways to do this. One way is to calculate the new data that needs to be predicted using SVD with the same ranking as the training data and taking the truncated **V** matrix as the new data. Another way is to multiply the new data with the **U'** and **S'**$^{-1}$ matrix of the training data.

$$\textbf{SVM Prediction Data = NewDataMatrix * trainingU' * trainingS'}^{-1} \qquad (9)$$

Based on experience, the later method (equation 9) yields better results. Figure 10, shows the data process flow.

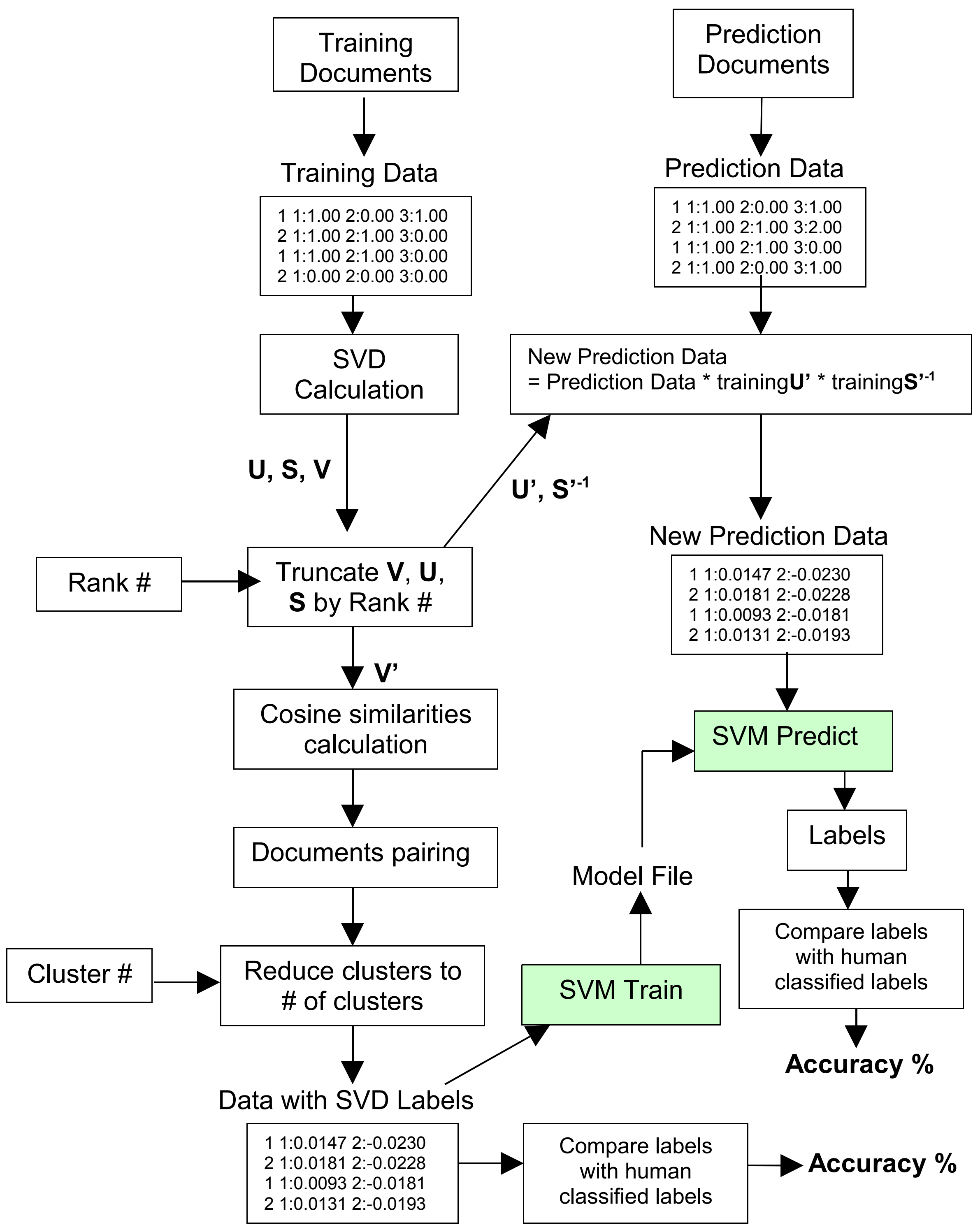


Figure 10. Data Process Flow of the Project
Source: Author's Research

4.2 Using Larger Data Set

The previous sections give background research on the approach used to cluster a data set. Now we would like to use a larger data set to test the method further. The data set that is used is Reuters-21578, which is the most widely used data set for text categorization. Reuters-21578 is a collection of newswire articles that have been human-classified by Carnegie Group, Inc. and Reuters, Ltd. The data that is used for this project is part of the already processed Reuters-21578 by (Joachims, T., 2004). Due to the expensive calculation of SVD, the data is further separated into 200 lines (rows) and 9928 terms (columns) per data set. In Table 8, "SVD Cluster Accuracy" will measure how close our SVD clustering method compares to humans classification and the "SVM Prediction Accuracy" will measure how accurate it is to use the SVD clustered data for training and then afterwards, use it to predict new data. A different set of Reuters-21578 that is 200 lines by 9928 terms is used at the new data for SVM prediction.

4.3 Result Analysis

Table 8. Results: Clustering with SVD vs. Humans Classification First Data Set

	First Data Set from Reuters-21578 (200 x 9928)		
	# of Natural Cluster	SVD Cluster Accuracy	SVM Prediction Accuracy
Rank 002	80	75.0%	65.0%
Rank 005	66	81.5%	82.0%
Rank 010	66	60.5%	54.0%
Rank 015	64	52.0%	51.5%
Rank 020	67	38.0%	46.5%
Rank 030	72	60.0%	54.0%
Rank 040	72	62.5%	58.5%
Rank 050	73	54.5%	51.5%
Rank 100	75	45.5%	58.5%

Source: Author's Research

Table 9. Results: Clustering with SVD vs. Humans Classification Second Data Set

	Second Data Set from Reuters-21578 (200 x 9928)		
	# of Natural Cluster	SVD Cluster Accuracy	SVM Prediction Accuracy
Rank 002	76	67.0%	84.5%
Rank 005	73	67.0%	84.5%
Rank 010	64	70.0%	85.5%
Rank 015	64	63.0%	81.0%
Rank 020	67	59.5%	50.0%
Rank 030	69	68.5%	83.5%
Rank 040	69	59.0%	79.0%
Rank 050	76	44.5%	25.5%
Rank 100	71	52.0%	47.0%

Source: Author's Research

Based on the results, the highest percentage accuracy for SVD clustering is 81.5% for rank 5 approximation. This accuracy percentage is reasonably good. Based on observation, the lower ranking approximation values do better than the higher approximation values. This supports many researchers' claim that truncated SVD gives better results. As for SVM prediction, the results are not surprising, since SVM can only predict what is given it to train. Therefore, its prediction percentage is about the same as SVD.

There are several reasons why the highest accuracy is 81.5%. When calculating SVD and using cosine similarities calculation to cluster, the documents form small clusters naturally. Having too many small clusters is a bit of a problem; therefore, a new algorithm is needed on top of the clustering algorithm to reduce the cluster size to a desirable number. Briefly, what the algorithm does is for each small cluster, it calculates the average of the vector documents within that cluster and compare it, using cosine similarities, to another cluster. The cluster that yields the highest value will be combined with the selected cluster. For more detail on the algorithm, refer to **Appendix B**. As the reader can see, reducing the number of clusters from about 64-80 to just two clusters will reduce the accuracy. Because the data used to test in Table 8 and 9 are classified in only 2 categories, the algorithm needs to reduce the clusters to 2 clusters so that it is possible to compare the results. Also, humans' classification is more subjective than a program so the methods used to classify are different from each other.

5. Conclusion

The project's goal is to find a method that can cluster high dimensional data. After many months of research, the chosen method is to use a combination of SVD and SVM. In section 2, the concept of SVM is explained through a small set of data in a 2-dimenional feature space. With the use of kernel methods, SVM can classify data in high dimensional space. Although SVM is an excellent method for data classification, it cannot cluster the data. Because of this, the project goes further into researching a method that can cluster and reduce the data. In section 3, SVD is used to accomplish this task. The section starts with clustering small data set. Using small data set allows the reader to understand and analyze SVD. The experiment shows that SVD can cluster and reduce the data's size greatly. In section 4, SVD is used with SVM on much larger data sets. The method is then compared with data that are classified by humans. From the experiment and analysis, the results show that the method proposed is able to cluster documents reasonably. However, there are plenty of rooms to improve this method such as making the algorithms more efficient. Overall, the result of the project is satisfactory.

5.1 Future Work

As mentioned previously, there are still a lot more work that could be done to improve this project. One way is to create a method that stores the data sets into a database. This way accessing the data each time will be much faster. In addition, a database can store massive amount of data. Another way is when calculating the distance between vectors using cosine similarities, parallel processing can be used to speed up the time. Also, the libraries, LIBSVM and JAMA, used in this project is excellent for small size data set, however, they need modification to accommodate larger data processing. For example, JAMA cannot process matrices that have m rows less than n columns ($m < n$) and it uses a double matrix array, which limits the size one can use. We can also look for more efficient kernels to use on SVM. Lastly, a nice graphical user interface for a user-friendly environment would be good.

References

Bennett, K. P., & Campbell, C. (2000). Support Vector Machines: Hype or Hellelujah?. ACM SIGKDD Explorations. VOl. 2, No. 2, 1-13

Chang, C & Lin, C. (2006). LIBSVM: a library for support vector machines, Retrieved November 29, 2006, from http://www.csie.ntu.edu.tw/~cjlin/libsvm

Cristianini, N. (2001). *Support Vector and Kernel Machines*. Retrieved November 29, 2005, from http://www.support-vector.net/icml-tutorial.pdf

Cristianini, N., & Shawe-Taylor, J. (2000). *An Introduction to Support Vector Machines*. Cambridge UK: Cambridge University Press

Drucker, H., Wu, D., & Vapnik, V. N. (1999). Support Vector Machines for Spam Categorization. *IEEE Transactions on Neural Networks, Vol. 10, No. 5*, 1048-1054.

Fan, R.(2006). LIBSVM Data: Classification, Regression, and Multi-label. Retrieved November 28, 2006, from http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/

Garcia, E. (2006). SVD and LSI Tutorial 4: Latent Semantic Indexing (LSI) How-to Calculations. Retrieved November 28, 2006, from http://www.miislita.com/information-retrieval-tutorial/svd-lsi-tutorial-4-lsi-how-to-calculations.html

Guestrin, C. (2006). Machine Learning. Retrieved November 8, 2006, from http://www.cs.cmu.edu/~guestrin/Class/10701/

Hicklin, J., Moler, C., & Webb, P. (2005). JAMA : A Java Matrix Package. Retrieved November 28, 2006, from http://math.nist.gov/javanumerics/jama/

Hsu, C., Chang, C., & Lin, C. (2006). A Practical Guide to Support Vector Classification. Retrieved November 28, 2006, from http://www.csie.ntu.edu.tw/~cjlin/papers/guide/guide.pdf

Joachims, T. (1998). Text Categorization with Support Vector Machines: Learning with Many Relevant Features. http://www.cs.cornell.edu/People/tj/publications/joachims_98a.pdf

Joachims, T. (2004). Support Vector Machines. Retrieved November 28, 2006, from http://svmlight.joachims.org/

Latent semantic analysis. (2006). Wikipedia. Retrieved December 8, 2005, from http://en.wikipedia.org/wiki/Latent_Semantic_Indexing

Lin, T. Y. (2005). Granulating the Semantics Space of Web Documents.

Reuters-21578 Text Categorization Test Collection.
Retrieved November 28, 2006, from
http://www.daviddlewis.com/resources/testcollections/reuters21578/

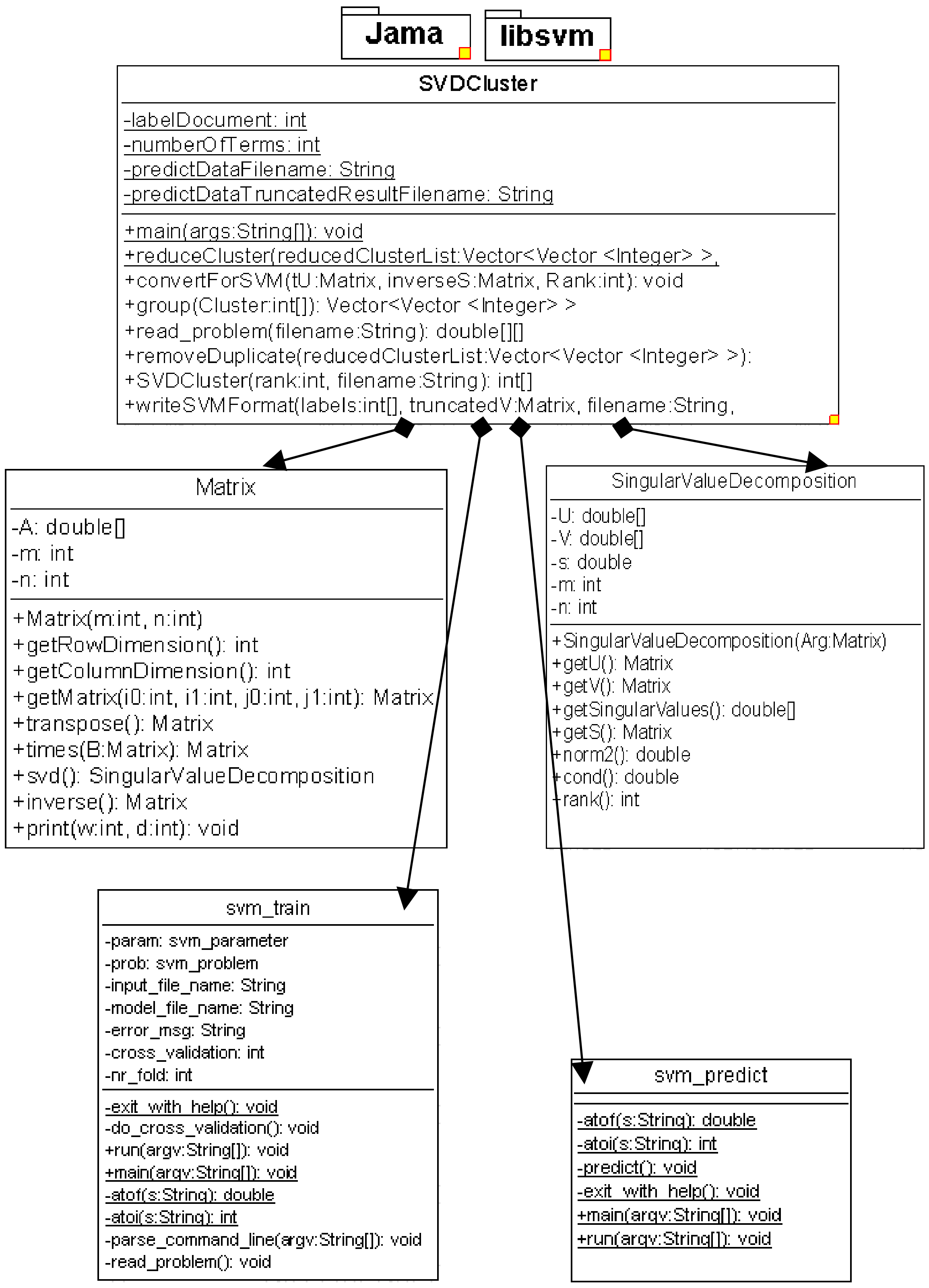
Support vector machine. (2006). Wikipedia. Retrieved December 8, 2005, from
http://en.wikipedia.org/wiki/Support_vector_machine

Tf–idf. (2006). Wikipedia. Retrieved December 8, 2005, from
http://en.wikipedia.org/wiki/Tf-idf

SVM - Support Vector Machines. DTREG. Retrieved November 28, 2006, from
http://www.dtreg.com/svm.htm

Vapnik, V. N. (2000, 1995). The Nature of Statistical Learning Theory.
Springer-Verlag New York, Inc.

Appendix A: Program UML Diagram


Jama
libsvm
SVDCluster
-labelDocument: int
-numberOfTerms: int
-predictDataFilename: String
-predictDataTruncatedResultFilename: String
+main(args:String[]): void
+reduceCluster(reducedClusterList:Vector<Vector <Integer> >,
+convertForSVM(tU:Matrix, inverseS:Matrix, Rank:int): void
+group(Cluster:int[]): Vector<Vector <Integer> >
+read_problem(filename:String): double[][]
+removeDuplicate(reducedClusterList:Vector<Vector <Integer> >):
+SVDCluster(rank:int, filename:String): int[]
+writeSVMFormat(labels:int[], truncatedV:Matrix, filename:String,
Matrix
-A: double[]
-m: int
-n: int
+Matrix(m:int, n:int)
+getRowDimension(): int
+getColumnDimension(): int
+getMatrix(i0:int, i1:int, j0:int, j1:int): Matrix
+transpose(): Matrix
+times(B:Matrix): Matrix
+svd(): SingularValueDecomposition
+inverse(): Matrix
+print(w:int, d:int): void
SingularValueDecomposition
-U: double[]
-V: double[]
-s: double
-m: int
-n: int
+SingularValueDecomposition(Arg:Matrix)
+getU(): Matrix
+getV(): Matrix
+getSingularValues(): double[]
+getS(): Matrix
+norm2(): double
+cond(): double
+rank(): int
svm_train
-param: svm_parameter
-prob: svm_problem
-input_file_name: String
-model_file_name: String
-error_msg: String
-cross_validation: int
-nr_fold: int
-exit_with_help(): void
-do_cross_validation(): void
+run(argv:String[]): void
+main(argv:String[]): void
-atof(s:String): double
-atoi(s:String): int
-parse_command_line(argv:String[]): void
-read_problem(): void
svm_predict
-atof(s:String): double
-atoi(s:String): int
-predict(): void
-exit_with_help(): void
+main(argv:String[]): void
+run(argv:String[]): void

Appendix B: Program Algorithms

In the project, two algorithms are used to do most of the processing for data clustering. The first one is **SVDCluster**, which calculates the cosine similarities and puts the documents into clusters. The second one is **reduceCluster**, which reduces the clusters to the user's inputted cluster value by taking the average of each cluster and computing each cluster's cosine similarities. Clusters with the highest cosine similarities are merged together until the total number of clusters is the same as the user's inputted number of clusters.

SVDCluster:

```
process SVD to get U, S, and V Matrix
truncate V matrix

for(int i=0; i < tV.getRowDimension(); i++) {

 Matrix Q1 = tV.getMatrix(i, i, 0, tV.getColumnDimension()-1);

          for(int k=0; k < tV.getRowDimension(); k++) {

                    if( (i != k) || ((k == 0)&&(i == 0)) ) {
                              for(int j=0; j < tV.getColumnDimension(); j++) {

                                        top = top + (Q1.get(0,j) * tV.get(k,j));
                                        lenQ1 = lenQ1 + Math.pow(Q1.get(0,j), 2);
                                        lenV = lenV + Math.pow(tV.get(k,j), 2);
                              }

                              similaritiesQ1 = top/(Math.sqrt(lenQ1)*Math.sqrt(lenV));

                              if(currentSimilarities <= similaritiesQ1) {
                                        currentSimilarities = similaritiesQ1;
                                        closestDoc = k;
                              }

                              Set similaritiesQ1, top, lenQ1, and lenV to 0.
                              }
                         }
                         Cluster[i] = closestDoc;
}
return Cluster;
```

Appendix B: Program Algorithms (cont'd)

reduceCluster:

```
while(reducedClusterList.size() > numberOfCluster) {

        //get the i cluster
        for(int l = 0; l < tV.getColumnDimension(); l++) {
                for(int j = 0; j < reducedClusterList.get(currentLabel).size(); j++) {
                average = average + tV.get(reducedClusterList.get(currentLabel).get(j), l);
                }

                selectLine[l] = average/ reducedClusterList.get(currentLabel).size();
                average = 0;
        }


        //compare all other clusters
        for(int i = 0; i < reducedClusterList.size(); i++) {

        if(i != currentLabel)
        {
                for(int l = 0; l < tV.getColumnDimension(); l++){
                        for(int j = 0; j < reducedClusterList.get(i).size(); j++){
                        average = average + tV.get(reducedClusterList.get(i).get(j), l);
                        }
                        compareLine[l] = average/reducedClusterList.get(i).size();
                        average = 0;
                }

                //calculate consine similarities
                for(int j=0; j < selectLine.length; j++){

                        top = top + (selectLine[j] * compareLine[j]);
                        lenS = lenS + Math.pow(selectLine[j], 2);
                        lenC = lenC + Math.pow(compareLine[j], 2);
                }
                sim = top/(Math.sqrt(lenS)*Math.sqrt(lenC));

                Set top, lens, and lenC to 0;

                if(currentLabel==0 && i == 1) {
                        currSim = sim;
                        deleteLine = i;
                }
                else if(i == 0){
                  currSim = sim;
                  deleteLine = i;
                }
                if(currSim <= sim){
                currSim = sim;
                deleteLine = i;
                }
        }
        }

        reducedClusterList.get(currentLabel).addAll(reducedClusterList.get(deleteLine));
        reducedClusterList.remove(deleteLine);

        currentLabel++;
        if(reducedClusterList.size() <= currentLabel) {
                currentLabel = 0;
        }
}
return reducedClusterList;
```

Appendix C: Java Documentations

Static Public Member Functions

Vector< Vector< Integer > >	**reduceCluster** (Vector< Vector< Integer > > reducedClusterList, int numberOfCluster)
Vector< Vector< Integer > >	**removeDuplicate** (Vector< Vector< Integer > > reducedClusterList)
void	**writeSVMFormat** (int[] labels, Matrix truncatedV, String filename, String originalFile) throws IOException
Vector< Vector< Integer > >	**group** (int[] Cluster)
void	**convertForSVM** (Matrix tU, Matrix inverseS, int Rank) throws IOException
double[][]	**read_problem** (String filename) throws IOException
int[]	**SVDCluster** (int rank, String filename) throws IOException

Detailed Description

COPYRIGHT (C) 2006 Tam Ngo. All Rights Reserved.

Purpose : This program uses the JAMA library to cluster

data set and output the labels and document values to a text file.

Author:
  Tam Ngo
Version:
  1.0 11/29/2006

Constructor & Destructor

SVDCluster: calculate SVM and cosine similarities and put them in an array

Parameters:
  filename: String file containing the training data
  rank: int the rank approximation value
Returns:
  int[] an array contain the paired documents and labels

Member Function

void SVDCluster.convertForSVM (Matrix tU, Matrix inverseS, int Rank) throws IOException [static]

convertForSVM: convert the new data into the truncated V' format. Use this data for SVM predict.

Parameters:

tU : Matrix truncated U from the training data
inverses : Matrix truncated S inverse from the training data
Rank : int rank value to name the data with it's rank

Returns:

void

Vector<Vector <Integer> > SVDCluster.group(nt[ ] Cluster) [static]

group: group pairs of document and labels to other pair of document and labels to form a cluster.

Parameters:

Cluster : gets an array of documents and cluster pairs

Returns:

Vector<Vector <integer> > the list of grouped cluster

Appendix C: Java Documentations (cont.)

double [][] SVDCluster.read_problem (String filename) throws IOException [static]

read_problem: load the training data from a filename

Parameters:

Filename : file containing the training data

Returns:

double[][] a matrix of the training data

Vector<Vector <Integer> > SVDCluster.reduceCluster (Vector< Vector< Integer > > reducedClusterList, int numberOfCluster) [static]

reduceCluster: method reduces the number of cluster based on the numberOfCluster parameter value. It will calculate the average vector document for between clusters and find the consine similiarties of the two. Cluster that returns the highest result will be combined with the selected cluster.

Parameters:

numberOfCluster : reduce the number clusters to the this number, numberOfCluster

reducedClusterList : gets the vector list that contains the clusters

Returns:

Vector<Vector <integer> > returns a new vector list that contains the reduced cluster

Vector<Vector <Integer> > SVDCluster.removeDuplicate (Vector< Vector< Integer > > reducedClusterList) [static]

removeDuplicate: remove any duplicate values from the cluster list

Parameters:

reducedClusterList : gets the vector list that contains the clusters

Returns:

Vector<Vector <integer> > returns a new vector list that cluster list without any duplicate.

Appendix C: Java Documentations (cont'd)

```
void SVDCluster.writeSVMFormat  (        int[]  labels,
                                         Matrix  truncatedV,
                                         String  filename,
                                         String  originalFile
                                      )  throws IOException [static]
```

writeSVMFormat: write the results of the cluster list to a SVM format for training and calcuate the accuracy by comparing with the labels on the original file

Parameters:

labels : gets the list of labels; document lines paired with a label value
filename : output file in SVM format
originalFile : the original file contain the data set

Returns:

void

Appendix D: Content of Deliverables

Deliverables are contained in a CD with the following contents:

Directories Map:

ClusterData

- JAMA package
- LIBSVM package
- User Manuel
- ClusterData
  - Data Clustering Program
  - Data
    - Training and Predicting Data Sets

Report

- report in .doc and .pdf format

Presentation

- PowerPoint project presentation