John Larkin
4/12/16
E91 Final Project
Biomedical Signals

E91 Final Project: Flow Cytometry

11 February 2016

John Larkin

Abstract: This lab's purpose was to focus on understanding the various mechanics and origins of flow cytometry. In addition, the particular understanding of AML disease and how flow cytometry data can be used to classify instances of this disease. The project uses R software to analyze the data uses multiple algorithmic methods such as feature detection through use of the k-means algorithm, as well as support vector machines. Various biological processes as well as the physics behind the flow cytometer were explored as well.

Note: Because this was the author's first time learning about many of the operations and mechanisms, including a lot of the biology, numerous graphics were included, not only for the reader's understanding, but also for the author's.

Table of Contents

Introduction:

Introduction to Flow Cytometry:

Once again, this lab served as an introduction into multiple different topics. As a result of it being a large learning project, the introduction will be rather lengthy. Firstly, this lab was an introduction into flow cytometry and the various techniques of collection for the data. This will be the initial focus of the Introduction section. The second focus will be the disease that is being looked at, acute myeloid leukemia. Further mathematical analyses will be discussed in the theory section.

Flow cytometry is a method of analyzing individual cells, at a rapid rate, leading to large amounts of data being able to be processed and understood. The mechanics of the operation work based on the fluid's characteristics, normally blood for biomedical purposes. As each cell flows through the flow cytometer, it is tagged with some fluorescent dye based on how the particle reacts to the laser that is being emitted by the flow cytometer. It will be more evident how exactly this process occurs by understanding how the flow cytometer is constructed. There are three main parts of the device:

- The flow cell which carries the particle to form a single stream of cells that can be appropriately tagged by the laser; this encompasses the fluid transport system
- The measuring system; in this case, most likely synonymous with the optics system. The individual cell is tagged here
- The detectors coupled with the computer system to analyze the data quickly and efficiently about what the system has just seen and how it is being separated.

More specifically, the particles are carried to get tagged and hit with the laser. Once the particle, most normally a cell for flow cytometry, is hit with the laser, it scatters the energy and light. If there are any fluorescent aspects of the cell, they are also illuminated. This information is then subsequently collected by lenses positioned in the flow cytometer.

To once again, briefly supply more detail about each one of the sections. The flow cell requires a laminar flow where the particles are all in single file line. This requires a decent amount of engineering in terms of construction of the flow cytometry. The pressure cannot be too great where there is convection and turbulence occurs, disrupting the flow of the liquid. There are several features of the experiment that are dependent on the flow rate of the fluid. For example, if solely a qualitative analysis of the data is being performed, a higher fluid flow rate can be performed so that the information is collected more coarsely. However, if precise information is needed, a more moderate speed of the fluid is going to be required.

The scattering and light tagging of each of the particles requires a further examination for those not comfortable in optics. In addition, several of the terms defined here, will be useful for the analysis section. This section can be further divided into two portions: the light scatter and the fluorescence. Before talking about these two sections, it is important to understand the independent variables that contribute to how a single particle will actually scatter. To name some of them, the internal complexity and the size are two of the largest driving factors.

There are two main types of light scatter that flow cytometry analysis normally implements and focuses on. Those two main types of scatter, are forward scattered light (FSC) and side scattered light (SSC). FSC is mainly related to the size and surface area of the particle and is detected close to the axis where the laser is being emitted from. SSC is more related to the internal complexity of the particle. The SSC light is detected orthogonal to the axis where the light is being emitted. The image below further illustrates the relationship between FSC and SSC light.

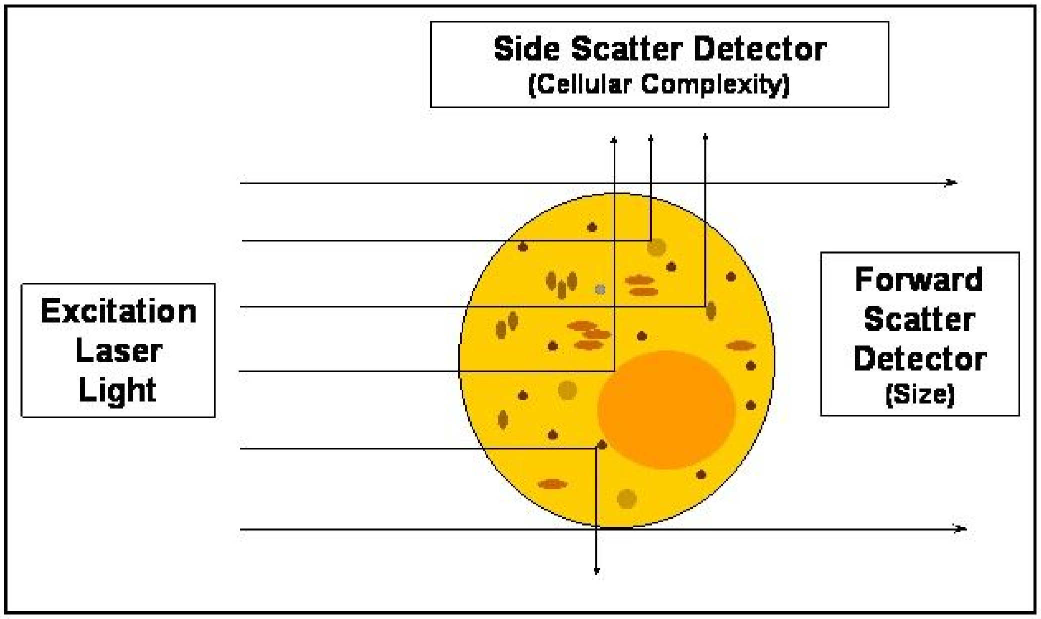


Figure 1 - Illustration of the Difference between FSC and SSC Light

The second clarification about the flow cytometer is about the fluorescence of the cell. Without going into depth about physical chemistry and the transition of states, fluorescence can be summarized as the cell's electrons being excited to a higher energy level. The electrons then relax to the ground state of the higher energy level and then fluorese down to the ground level that they originally started. A simple diagram to illustrate the cell's property at the molecular level is found below:

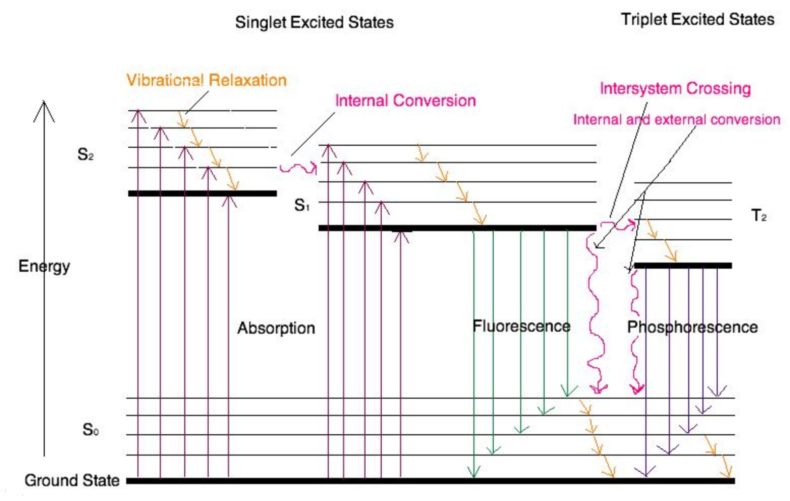


Figure 2 - Jablonski Diagram showing outline of fluorescence

Depending on what chemical compounds are in the cell, the cell will have a range of different emissions of set wavelengths from the electrons that are being fluoresced. With multiple different types of compounds in the cell, multiple different wavelengths of distinct intensities will be emitted. This is a major source of classification among the cells. A little bit of biology background is required here. Often in biological samples of flow cytometer, the fact that antibodies are being analyzed is often taken advantage of. Antibodies are large Y-shaped proteins produced by plasma cells, that are used by the immune system to neutralize pathogens like bacteria or viruses. Plasma cells are derived from a specific set of leukocytes, which is simply another name for white blood cells. The specific white blood cell is called a B cell. The only real importance of the B cells is that they differentiate into the cells that produce plasma cells, which in turn produce the antibodies. The reason to care about the antibodies is because they recognize specific characteristics of harmful agents. The specific portion that they recognize is called the *antigen*. The relationship between the antibody and the antigen is similar to a lock and key. An illustration can be shown below:

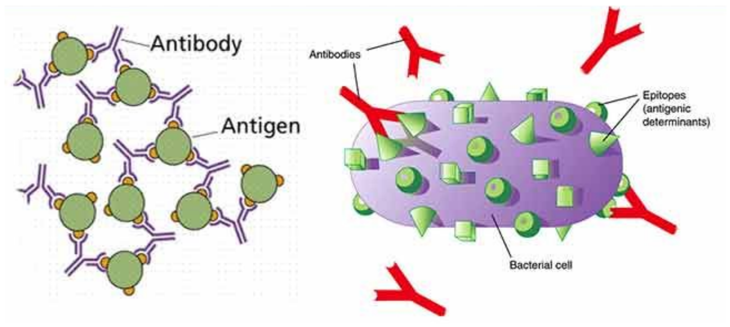


Figure 3 - Diagram of Antibody and Antigens

So then what flow cytometry does is takes advantage of the antibodies that are monoclonal. This means that the antibodies solely bind to one antigen. The simile would be the comparison to uniqueness in mathematical partial differential equation solutions. So monoclonal antibodies can only bond to one type of substance. So the specific monoclonal antibodies are conjugated with the fluorescence and then the antigens that *are on the cell's surface* are able to be identified based upon what tagged antibodies are attached to that position. Once again, another illustration is helpful in understanding part of this biological process.

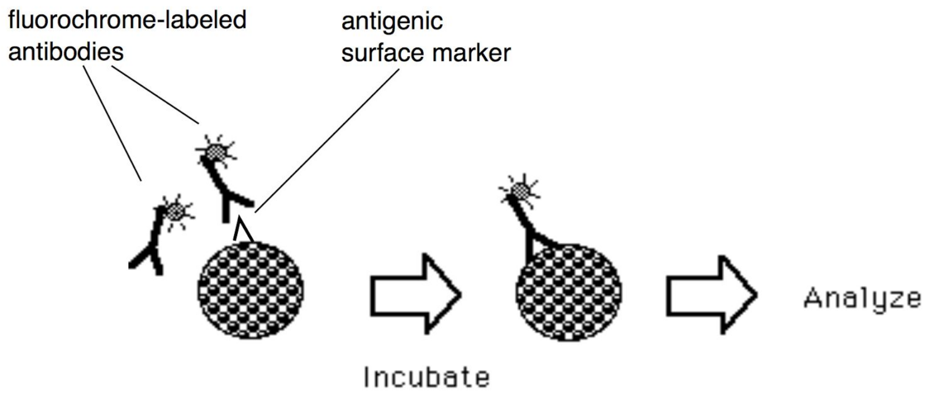


Figure 4 - Example of Flow Cytometer's Fluorescence Tagging Process

The optical system is rather clear cut; the FSC signals are collected by a simple photodiode, while the SSC and the fluorescence signals are diverted to the photomultiplier tubes (PMTs). There are a variety of standard filters in the flow cytometer, namely bandpass filters, low pass, and high pass filters are all incorporated in the machine. When the light is being measured, the PMTs are used for the less intense SSC light because they are more sensitive and costly. For the FSC light, a simple photodiode can be used to capture information about the light, such as intensity, time, and transmittance.

Introduction to Acute Myeloid Leukemia:

The flow cytometry data was taken from a competition, but concerns the classification of patients who suffer from the disease of acute myeloid leukemia, also known as AML. For the sake of entirety in understanding of this project, some of the biology behind AML will be explained.

AML has a number of different names, such as acute myelogenous leukemia, acute myeloblastic leukemia, and acute granulocytic leukemia. It is a type of cancer that is called acute leukemia, and AML is of the most common of this type of cancer. AML is a rapidly growing form of cancer that is found in bone marrow and the blood. While AML is a rare type of cancer, it accounts for 1.2% of the cancer deaths in the United States[2].

AML develops when the DNA of a developing cell in the bone marrow is damaged. Bone marrow normally produces blood stem cells. The stem cells have to be differentiated; they can become one of three things. Either they will become red blood cells, the cells that carry oxygen throughout the body, white blood cells, which fight infection, or platelets which essentially aid in the clotting process and regulate bleeding. So when there is a deformation in the stem cells that make them deformed and unable to create normal white blood cells. This is commonly called an *acquired mutation*. If one has this form of a mutation, then what happens is the mutated and deformed cell called a leukemia cell or a blast, then multiplies into billions of cells. These multiplied cells are commonly called the leukemic blasts. The blast cells perform multiple actions that impede the body's general wellness, including blocking the normal production of healthy cells. In addition, the blast cells actually survive longer than the normal cells. After multiple generations of cells, the blast cells will start to dominate. This means that the healthy blood cells (all three types - red cells, white cells, and platelets) are going to be lower than the normal expected value. As a result of this, there are numerous symptoms, such as shortness of breath, immune system being vastly weakened, easy bleeding and bruising. Having low numbers of all three types of blood cells is often called *pancytopenia*.

While curable, AML progresses rapidly and targets the undeveloped cells. It is a disease that can prove fatal to a high percentage of the people that it inhabits. Early detection of AML is crucial for prevention of further harm. The rest of the report will analyze blood samples collected from patients in order to effectively detect AML.

Theory:

This report does not involve a lot of theory. The process of flow cytometry is grounded in physics that have largely been explained in the introduction. The background for the understanding of AML has also been described in the introduction section. Any algorithms or computational complexities that are used in the analysis will be further explained here.

Instead of trying to analyze the data solely by the author's discretion, the various algorithms that were used in the competition will be parsed and one selected to explore in more depth and eventually, implement to reclassify the segment the flow cytometry data.

The algorithm that was selected was a combination between kmeans and support vector machines. As the author has previously explored the theory of support vector machines in another engineering final project, an abridged version of the theory has been attached as an abstract.

Theory of K-means Algorithm:

K-means is a type of vector quantization method. It is a type of clustering algorithm which is a unsupervised learning technique. Vector quantization allows for the modelling of of probability density functions by the distribution of prototype vectors. Essentially, a large group of points is divided so that approximately the same number of points are closest to them. A centroid point for the group is then calculated. K-means is an extension of this. K-means partitions the data points (of which there is *n*) into *k* clusters. Each observation of the k clusters, belongs to the cluster with the nearest mean. The terminology is that the mean served as the prototype. A data prototype is a parameter that the clustering is based off of. From Wikipedia, the objectives of data prototyping are to set rules so that all data is able to be clustered based upon that pertinent prototyping information. This type of partitioning is known as dividing the space into Voronoi cells. Once the space is divided, the points are able to be classified and separated.

Note: If the author's summary was not sufficient, more informaiton about the algorithm can be found here: http://www.ncbi.nlm.nih.gov/pmc/articles/PMC3400953/ [6].

Experimental Procedure:

Original Experiment Procedure:

Before user analysis could begin, the conditions and nuances of the original experiment had to be taken into account.

The flow cytometry data was used in a FlowCAP challenge for classification. The challenge is presented below:

"Challenge 2: Acute Myeloid Leukemia (AML)
The goal of this challenge was to find cell populations that can discriminate between AML positive (n = 43) and healthy donor (n = 316) patients. Peripheral blood or bone marrow aspirate samples were collected over a 1-year period using 8 tubes (tube #1 is an isotype control and #8 is unstained) with different marker combinations. In addition to raw FCS files, half of the subject labels were provided for training purposes. Algorithms were to use this data to label the rest of the samples. These labels were be used to evaluate algorithm performance."

In terms of organization of the flow cytometry data, there are 8 tubes of blood per individual. There are 359 individuals, leading to 2,872 FCS files. Overall, 43 of these patients have AML and 316 patients are totally normal patients.

Because of the large amount of data, only a subset was selected. This will be further discussed in the author's own experimental procedure section.

Analysis and Experimental Procedure:

The experimental procedure was unclear at the beginning of the project. Many issues occurred with initially reading in the data. Flow cytometry data is formatted in a low level type of file specified by the FCS extension. The AML data that was to be analyzed actually was formatted in a way that a popular MATLAB program was unable to read in. With help of Professor Moser, the error was found to be that the MATLAB file did not actually support two alternative endians to specify how the bytes should be read (i.e. the MSB vs the LSB). The primary researcher of the experiment was emailed to see if this problem could be resolved; the researcher helpfully suggested that R be used rather than analysis done in MATLAB. While this helpful suggestion was taken, the author of the published MATLAB code to read in the FCS file was emailed in order to update his FCS reader code.

With the data finally able to be imported successfully into both R and MATLAB, the author decided to continue through using R, as many packages and analysis documents exist through this platform, and less so through MATLAB.

To deal with the mass amount of data, only one tube for each individual was actually selected. For this lab, the third tube from each patient was selected, as this proves relatively good indicators for the AML classificators. The proportion to the total population of diseases vs AML was relatively retained. Some sample statistics are shown below:

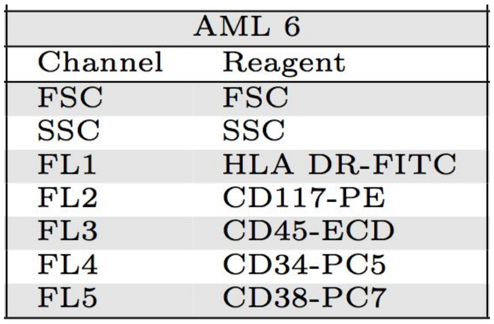
Statistics on Patients		
Selected Subset	*Number of AML Patients*	16
	Number of Normal Patients	87
	Total Population	103
	Percentage with AML	15.53%
	Percentage normal	84.47%
True Population	*Number of AML Patients*	43
	Number of Normal Patients	316
	Total Population	359
	Percentage with AML	11.98%
	Percentage normal	88.02%

Table 1 - General Breakdown of Selected Subset

Specifically, for each of these patients looked at, the sixth tube was selected. After some research into other effective antigen classifications for acute myeloid leukemia, a specific tube was selected from each individual because of the antigen that was needed. An excerpt from a paper written by Brian Webber indicates that CD38 is a strong antigen classifier for AML. A quote from the paper is found below:

> The most frequently expressed antigen was the non-specific lymphoid progenitor marker CD38 (92% of all cases).This was followed by the myeloid lineage markers CD13 (91%), CD33 (87%), and CD117 (80%). Also demonstrating relatively high prevalence rates were the hematopoietic progenitor cell markers CD34 (71%) and HLA-DR (79%). Monocytic markers were moderately frequent: including CD4 (63%), CD11b (41%), CD11c (43%), CD14 (16%), and CD36 (34%). CD7, a T cell antigen known to show aberrant expression in a subset of AML cases, was positive in 28% of all cases. Of note, another T cell antigen, CD2, was expressed in a considerable number of cases (18%). B cell markers CD10, CD19, and CD22 were present in 13%, 8%, and 2% of all cases, respectively.

Therefore, the sixth tube was selected. The following figure once again, reiterates the reagents that are found in that tube for testing, and subsequently the tubes that will be analyzed.



AML 6	
Channel	Reagent
FSC	FSC
SSC	SSC
FL1	HLA DR-FITC
FL2	CD117-PE
FL3	CD45-ECD
FL4	CD34-PC5
FL5	CD38-PC7

Figure 4 - Specific Reagents for Selected Tube

Next, an algorithm needed to be selected so that the data could be analyzed appropriately. As noted in the theory section, the algorithm that was selected was the flowPeakssvm algorithm. This algorithm specifically applied a large clustering of K-means, specifically the author of the algorithm specified 300 groupings. In order to optimize the performance, the author of the algorithm cites two specific incorporations to differentiate their algorithm. Specifically:

> " i) the seeds of the K-means algorithm are generated by the kmeans++ ;
>
> ii) the algorithm is implemented by using k-d tree to speed up computation."

For more information involving the theory behind K-means and support vector machines, please see the theory section.

Moving from the set up to the analysis, the previous portions of the lab experiment solely discuss the training set, in order to generate the appropriate classifier. The last 50 individuals' from the study also had the sixth vial of tube downloaded, in order to act as a actual classification test. The accuracy of the classification was tested after the algorithm had been applied. The six individuals who had AML out of the last 50 individuals (310-359), were patients 314, 326, 337, 340, 344, and 348.

Results:

Note: The majority of error from this code seem to came from the flowPeaks algorithm disproportionately classifying the vast majority of data points as following into one of the clusters.

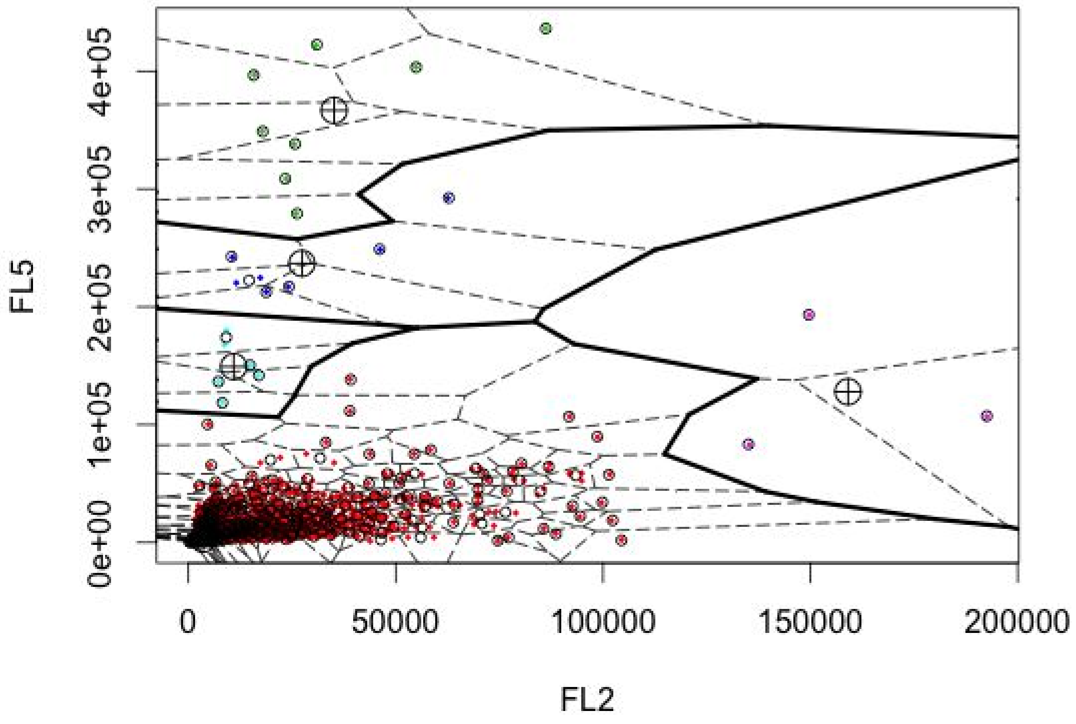


Figure 5 - Example of the flowPeaks algorithm on FL5 vs FL2

Example Output of Feature Matrix:

```
> total_feature_matrix
          [,1]         [,2]         [,3]      [,4]         [,5]         [,6]         [,7]
 [1,] 1.0000000 0.000000e+00 0.000000e+00 1.0000000 0.000000e+00 0.000000e+00 0.000000e+00
 [2,] 0.9999611 3.891656e-05 0.000000e+00 0.9999531 0.000000e+00 0.000000e+00 4.689112e-05
 [3,] 1.0000000 0.000000e+00 0.000000e+00 1.0000000 0.000000e+00 0.000000e+00 0.000000e+00
 [4,] 0.9999648 3.516422e-05 0.000000e+00 1.0000000 0.000000e+00 0.000000e+00 0.000000e+00
 [5,] 1.0000000 0.000000e+00 0.000000e+00 1.0000000 0.000000e+00 0.000000e+00 0.000000e+00
 [6,] 1.0000000 0.000000e+00 0.000000e+00 1.0000000 0.000000e+00 0.000000e+00 0.000000e+00
 [7,] 1.0000000 0.000000e+00 0.000000e+00 1.0000000 0.000000e+00 0.000000e+00 0.000000e+00
 [8,] 1.0000000 0.000000e+00 0.000000e+00 1.0000000 0.000000e+00 0.000000e+00 0.000000e+00
 [9,] 1.0000000 0.000000e+00 0.000000e+00 1.0000000 0.000000e+00 0.000000e+00 0.000000e+00
[10,] 1.0000000 0.000000e+00 0.000000e+00 1.0000000 0.000000e+00 0.000000e+00 0.000000e+00
[11,] 1.0000000 0.000000e+00 0.000000e+00 1.0000000 0.000000e+00 0.000000e+00 0.000000e+00
[12,] 1.0000000 0.000000e+00 0.000000e+00 1.0000000 0.000000e+00 0.000000e+00 0.000000e+00
[13,] 0.9999248 0.000000e+00 7.519362e-05 0.9999572 4.280272e-05 0.000000e+00 0.000000e+00
[14,] 0.9998938 1.061684e-04 0.000000e+00 0.9998885 0.000000e+00 1.114993e-04 0.000000e+00
[15,] 1.0000000 0.000000e+00 0.000000e+00 1.0000000 0.000000e+00 0.000000e+00 0.000000e+00
[16,] 0.9999617 3.828337e-05 0.000000e+00 0.9999480 5.196965e-05 0.000000e+00 0.000000e+00
[17,] 1.0000000 0.000000e+00 0.000000e+00 1.0000000 0.000000e+00 0.000000e+00 0.000000e+00
[18,] 1.0000000 0.000000e+00 0.000000e+00 1.0000000 0.000000e+00 0.000000e+00 0.000000e+00
[19,] 1.0000000 0.000000e+00 0.000000e+00 1.0000000 0.000000e+00 0.000000e+00 0.000000e+00
[20,] 1.0000000 0.000000e+00 0.000000e+00 1.0000000 0.000000e+00 0.000000e+00 0.000000e+00
[21,] 1.0000000 0.000000e+00 0.000000e+00 1.0000000 0.000000e+00 0.000000e+00 0.000000e+00
[22,] 1.0000000 0.000000e+00 0.000000e+00 1.0000000 0.000000e+00 0.000000e+00 0.000000e+00
[23,] 1.0000000 0.000000e+00 0.000000e+00 1.0000000 0.000000e+00 0.000000e+00 0.000000e+00
[24,] 1.0000000 0.000000e+00 0.000000e+00 1.0000000 0.000000e+00 0.000000e+00 0.000000e+00
[25,] 0.9999644 3.562522e-05 0.000000e+00 0.9999624 0.000000e+00 3.759964e-05 0.000000e+00
[26,] 0.9999655 3.452919e-05 0.000000e+00 1.0000000 0.000000e+00 0.000000e+00 0.000000e+00
[27,] 0.9999298 7.021979e-05 0.000000e+00 0.9999643 0.000000e+00 3.566715e-05 0.000000e+00
[28,] 0.9999281 7.185973e-05 0.000000e+00 0.9999156 4.220656e-05 4.220656e-05 0.000000e+00
[29,] 1.0000000 0.000000e+00 0.000000e+00 1.0000000 0.000000e+00 0.000000e+00 0.000000e+00
[30,] 1.0000000 0.000000e+00 0.000000e+00 1.0000000 0.000000e+00 0.000000e+00 0.000000e+00
[31,] 0.9999661 3.386616e-05 0.000000e+00 0.9999652 0.000000e+00 3.479471e-05 0.000000e+00
[32,] 1.0000000 0.000000e+00 0.000000e+00 1.0000000 0.000000e+00 0.000000e+00 0.000000e+00
```

Note, these are the proportion matrices. There are three clusterings for the AML data and four clusterings for the normal data.

The feature matrix is already shown.

```
> testing_feature_matrix
          [,1]         [,2]         [,3]      [,4]         [,5]        [,6]         [,7]
 [1,] 0.9999644 3.557833e-05 0.000000e+00 0.9999617 0.000000e+00 0.000000000 3.833621e-05
 [2,] 1.0000000 0.000000e+00 0.000000e+00 1.0000000 0.000000e+00 0.000000000 0.000000e+00
 [3,] 0.9999656 3.443645e-05 0.000000e+00 1.0000000 0.000000e+00 0.000000000 0.000000e+00
 [4,] 1.0000000 0.000000e+00 0.000000e+00 1.0000000 0.000000e+00 0.000000000 0.000000e+00
 [5,] 1.0000000 0.000000e+00 0.000000e+00 1.0000000 0.000000e+00 0.000000000 0.000000e+00
 [6,] 1.0000000 0.000000e+00 0.000000e+00 1.0000000 0.000000e+00 0.000000000 0.000000e+00
 [7,] 1.0000000 0.000000e+00 0.000000e+00 1.0000000 0.000000e+00 0.000000000 0.000000e+00
 [8,] 1.0000000 0.000000e+00 0.000000e+00 1.0000000 0.000000e+00 0.000000000 0.000000e+00
 [9,] 1.0000000 0.000000e+00 0.000000e+00 1.0000000 0.000000e+00 0.000000000 0.000000e+00
[10,] 1.0000000 0.000000e+00 0.000000e+00 0.9999598 4.022526e-05 0.000000000 0.000000e+00
[11,] 1.0000000 0.000000e+00 0.000000e+00 1.0000000 0.000000e+00 0.000000000 0.000000e+00
[12,] 1.0000000 0.000000e+00 0.000000e+00 1.0000000 0.000000e+00 0.000000000 0.000000e+00
[13,] 1.0000000 0.000000e+00 0.000000e+00 1.0000000 0.000000e+00 0.000000000 0.000000e+00
[14,] 1.0000000 0.000000e+00 0.000000e+00 1.0000000 0.000000e+00 0.000000000 0.000000e+00
[15,] 1.0000000 0.000000e+00 0.000000e+00 1.0000000 0.000000e+00 0.000000000 0.000000e+00
[16,] 1.0000000 0.000000e+00 0.000000e+00 1.0000000 0.000000e+00 0.000000000 0.000000e+00
[17,] 1.0000000 0.000000e+00 0.000000e+00 1.0000000 0.000000e+00 0.000000000 0.000000e+00
[18,] 1.0000000 0.000000e+00 0.000000e+00 1.0000000 0.000000e+00 0.000000000 0.000000e+00
[19,] 1.0000000 0.000000e+00 0.000000e+00 1.0000000 0.000000e+00 0.000000000 0.000000e+00
[20,] 0.9999314 6.862005e-05 0.000000e+00 0.9999289 3.555176e-05 0.000000000 3.555176e-05
[21,] 1.0000000 0.000000e+00 0.000000e+00 1.0000000 0.000000e+00 0.000000000 0.000000e+00
[22,] 1.0000000 0.000000e+00 0.000000e+00 1.0000000 0.000000e+00 0.000000000 0.000000e+00
[23,] 1.0000000 0.000000e+00 0.000000e+00 1.0000000 0.000000e+00 0.000000000 0.000000e+00
[24,] 0.9999650 0.000000e+00 3.497237e-05 0.9999278 7.224651e-05 0.000000000 0.000000e+00
[25,] 1.0000000 0.000000e+00 0.000000e+00 1.0000000 0.000000e+00 0.000000000 0.000000e+00
[26,] 1.0000000 0.000000e+00 0.000000e+00 1.0000000 0.000000e+00 0.000000000 0.000000e+00
[27,] 1.0000000 0.000000e+00 0.000000e+00 1.0000000 0.000000e+00 0.000000000 0.000000e+00
[28,] 0.9967909 3.209084e-03 0.000000e+00 0.9969645 2.759534e-04 0.002759534 0.000000e+00
[29,] 1.0000000 0.000000e+00 0.000000e+00 1.0000000 0.000000e+00 0.000000000 0.000000e+00
[30,] 1.0000000 0.000000e+00 0.000000e+00 1.0000000 0.000000e+00 0.000000000 0.000000e+00
[31,] 1.0000000 0.000000e+00 0.000000e+00 1.0000000 0.000000e+00 0.000000000 0.000000e+00
[32,] 1.0000000 0.000000e+00 0.000000e+00 1.0000000 0.000000e+00 0.000000000 0.000000e+00
[33,] 1.0000000 0.000000e+00 0.000000e+00 1.0000000 0.000000e+00 0.000000000 0.000000e+00
[34,] 1.0000000 0.000000e+00 0.000000e+00 1.0000000 0.000000e+00 0.000000000 0.000000e+00
[35,] 1.0000000 0.000000e+00 0.000000e+00 1.0000000 0.000000e+00 0.000000000 0.000000e+00
[36,] 1.0000000 0.000000e+00 0.000000e+00 1.0000000 0.000000e+00 0.000000000 0.000000e+00
[37,] 1.0000000 0.000000e+00 0.000000e+00 1.0000000 0.000000e+00 0.000000000 0.000000e+00
[38,] 1.0000000 0.000000e+00 0.000000e+00 1.0000000 0.000000e+00 0.000000000 0.000000e+00
[39,] 1.0000000 0.000000e+00 0.000000e+00 1.0000000 0.000000e+00 0.000000000 0.000000e+00
[40,] 1.0000000 0.000000e+00 0.000000e+00 1.0000000 0.000000e+00 0.000000000 0.000000e+00
[41,] 0.9999654 3.455306e-05 0.000000e+00 1.0000000 0.000000e+00 0.000000000 0.000000e+00
[42,] 1.0000000 0.000000e+00 0.000000e+00 1.0000000 0.000000e+00 0.000000000 0.000000e+00
[43,] 0.9999604 3.959142e-05 0.000000e+00 0.9999590 0.000000e+00 0.000000000 4.102396e-05
[44,] 0.9999660 3.402518e-05 0.000000e+00 1.0000000 0.000000e+00 0.000000000 0.000000e+00
[45,] 1.0000000 0.000000e+00 0.000000e+00 1.0000000 0.000000e+00 0.000000000 0.000000e+00
[46,] 1.0000000 0.000000e+00 0.000000e+00 1.0000000 0.000000e+00 0.000000000 0.000000e+00
[47,] 1.0000000 0.000000e+00 0.000000e+00 1.0000000 0.000000e+00 0.000000000 0.000000e+00
```

```
[48,] 1.0000000 0.000000e+00 0.000000e+00 1.0000000 0.000000e+00 0.000000000 0.000000e+00
[49,] 1.0000000 0.000000e+00 0.000000e+00 1.0000000 0.000000e+00 0.000000000 0.000000e+00
[50,] 0.9999296 7.036060e-05 0.000000e+00 0.9999612 0.000000e+00 0.000000000 3.881234e-05
```

However, the unfortunate final result, is that the data splits the groups entirely. Note, a numeric value had to be provided for the AML and normal data (1 and 0, respectively).

```
> pred
  1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18  19  20  21  22  23
24  25  26  27  28  29  30  31  32
0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5
0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5
 33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50
0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5
```

Discussion:

The data manipulation and analysis of this project was a feat in and of itself. The idea behind the feature detection as acknowledged in the paper was relatively complex, with clustering being based on the total expression matrix of all of the patients in the sample. Once we had these clusters, we could look at the mean and see the proportion of each individual who has sample data points lying in these regions. Based upon these proportion vectors (note that both the AML and NORM matrices were used to create a side by side proportion vector), a support vector machine was then properly trained. Note, the majority of the imprecision with Once this was properly subsetted, the proportion vectors for each of the samples had to be made. This involved using the two expression matrices and flowPeaks outputs to generate a clustered dataset. The proportion for each individual was then recorded. A matrix of these results was created as a feature dataset in order to classify and train our support vector machine. However, due to an error in the code, the training set feature matrix did not seem appropriate or reliable to train the data.

The reader is encouraged to observe the attached code and see the analysis followed. The code has not been perfected and the results are not ideal. It is the author's hope to continue this project in the future and correct the code to appropriately classify the flow cytometry data.

Acknowledgements:

The project would not even by close to being completed on time or in the least bit accurately, without the help of Professor Moser. He was an invaluable resource, who the author leaned on multiple times, in topics ranging from conceptual understanding to precise analytical analysis of the flow cytometry data.

Conclusion and Further Extensions:

The project was an excellent opportunity to learn more about an intersection between computationally intriguing algorithms, a sample of biology and a disease which has repercussions in today's society, and an interesting data mining challenge.

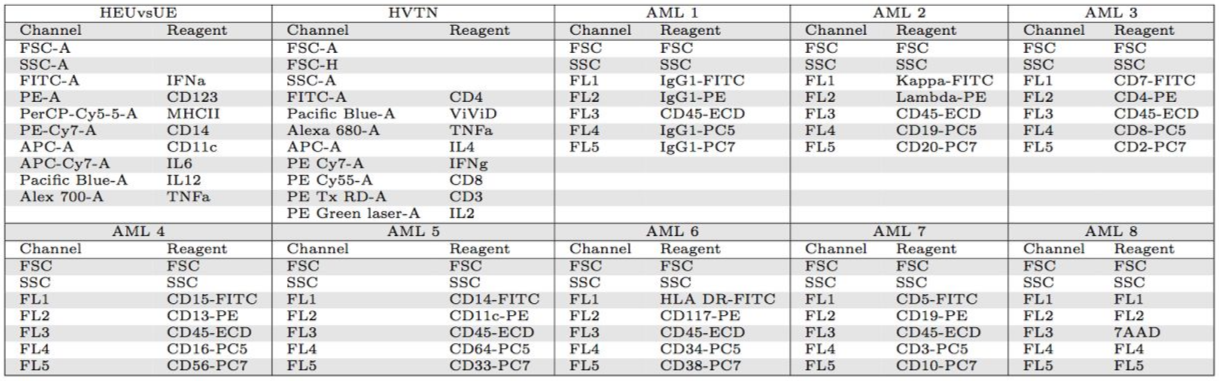
In future work for this project, the code could be totally finished and the classification could be perfected. In addition, further exploration into the theory and mathematical derivation of the k-means clustering algorithm could be used.

Overall though, the project helped the author to learn a lot, as well as increase their computational skills with R. It was thoroughly challenging and enjoyable. Again, hopefully the project can be finished at a later time.

Appendix:

Appendix A: Antigen List Used in Each Tube

Supplementary Table 5: List of reporter and analytes for the HVTN, HEUvsUE, and all nine panels of the AML datasets.



HEUvsUE		HVTN		AML 1		AML 2		AML 3	
Channel	Reagent	Channel	Reagent	Channel	Reagent	Channel	Reagent	Channel	Reagent
FSC-A		FSC-A		FSC	FSC	FSC	FSC	FSC	FSC
SSC-A		FSC-H		SSC	SSC	SSC	SSC	SSC	SSC
FITC-A	IFNa	SSC-A		FL1	IgG1-FITC	FL1	Kappa-FITC	FL1	CD7-FITC
PE-A	CD123	FITC-A	CD4	FL2	IgG1-PE	FL2	Lambda-PE	FL2	CD4-PE
PerCP-Cy5-5-A	MHCII	Pacific Blue-A	ViViD	FL3	CD45-ECD	FL3	CD45-ECD	FL3	CD45-ECD
PE-Cy7-A	CD14	Alexa 680-A	TNFa	FL4	IgG1-PC5	FL4	CD19-PC5	FL4	CD8-PC5
APC-A	CD11c	APC-A	IL4	FL5	IgG1-PC7	FL5	CD20-PC7	FL5	CD2-PC7
APC-Cy7-A	IL6	PE Cy7-A	IFNg						
Pacific Blue-A	IL12	PE Cy55-A	CD8						
Alex 700-A	TNFa	PE Tx RD-A	CD3						
		PE Green laser-A	IL2						

AML 4		AML 5		AML 6		AML 7		AML 8	
Channel	Reagent	Channel	Reagent	Channel	Reagent	Channel	Reagent	Channel	Reagent
FSC	FSC	FSC	FSC	FSC	FSC	FSC	FSC	FSC	FSC
SSC	SSC	SSC	SSC	SSC	SSC	SSC	SSC	SSC	SSC
FL1	CD15-FITC	FL1	CD14-FITC	FL1	HLA DR-FITC	FL1	CD5-FITC	FL1	FL1
FL2	CD13-PE	FL2	CD11c-PE	FL2	CD117-PE	FL2	CD19-PE	FL2	FL2
FL3	CD45-ECD	FL3	CD45-ECD	FL3	CD45-ECD	FL3	CD45-ECD	FL3	7AAD
FL4	CD16-PC5	FL4	CD64-PC5	FL4	CD34-PC5	FL4	CD3-PC5	FL4	FL4
FL5	CD56-PC7	FL5	CD33-PC7	FL5	CD38-PC7	FL5	CD10-PC7	FL5	FL5

Appendix B: Theory of Support Vector Machines adapted from E19 Final Project

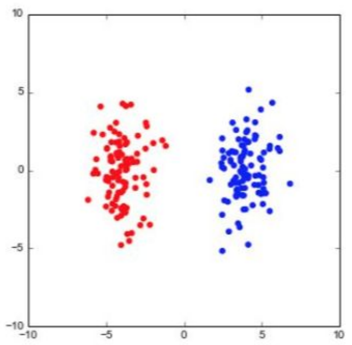


Figure 1 - Example of a basic two group classification problem

However, in the example above, there are a vast number of classifiers that could divide the group into two groups. Seemingly, the best classifier would be a vertical line dividing the group into two, but another option would be to have a somewhat sloped line that is closer to both of the groups. This raises the question of: are all decision boundaries of equal value or are some more accurate in a way?

The idea behind this small thought experiment is to maximize the margin of how far the decision boundary line is from both of the data sets. In other words, we're trying to have as much space as possible between the two groups of points. Mathematically, call this margin *m*. We're trying to maximize this margin in other words. So if we consider that we have a vector $\overline{u}$ that goes to an unknown, and we also have a vector $\overline{w}$ that goes to the margin that's grouping the first cluster and runs orthogonal to that line. In other words, a pictorial description is shown below:

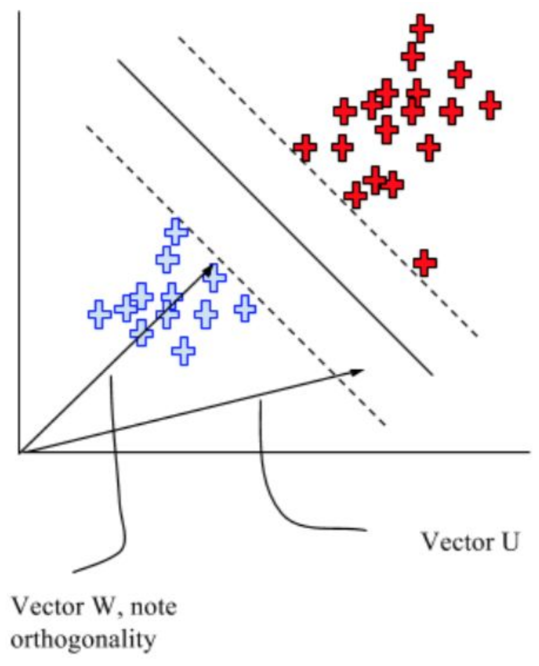


Figure 2 - Computer Drawn Visualization of the Vectors

So then, what we want to do is project the vector *u* onto the vector *w*. So then, the dot product takes the projection onto w, so we have:

$$\overline{w} \cdot \overline{u} \geq c$$

where c is a constant that classifies the unknown point as either being a blue sample, or a red sample. So then we can also rearrange this to generate our decision rule:

$$\overline{w} \cdot \overline{u} + b \geq 0 \qquad (1)$$

The equation has been given a star as it will be of critical use. Now the issue is that we don't have a *b* constant, and we also don't have our vector *w*. While we know that *w* must be orthogonal to the edge of the classifier, we don't know it's length or magnitude.

So now we can constrain our function classifier in order to simplify our equations. Let's consider a positive and a negative sample. This is going to be a binary classifier so instead of saying blue and red samples, the type of points will be assigned numerical values. For this derivation, denote the blue values in Figure 2, as having an output value of -1 and the red crosses as having values +1. So then we can show the derivation:

$$\text{For red sample: } \overline{w} \cdot \overline{x}_{red} + b \geq 1$$

For blue sample: $\overline{w} \cdot \overline{x}_{blue} + b \leq -1$

Then wrapping up the assigned values in a function:

$$y(color) = +1 \; if \; red, \; -1 \; if \; blue$$

Then it is seen that:

$$y\,(\overline{w} \cdot \overline{x} + b) \geq 1$$

Or

$$y\,(\overline{w} \cdot \overline{x} + b) - 1 \geq 0$$

Then, the additional constraint that if the data point is in the middle of the classifying region, the value assigned is actually 0. So

$$y\,(\overline{w} \cdot \overline{x} + b) - 1 = 0 \qquad (2)$$

When the data point is within the margin of the classifier.

Then let's consider the margin once again, and then also consider two points on the edge of the classifier, like in Figure 3 shown below.

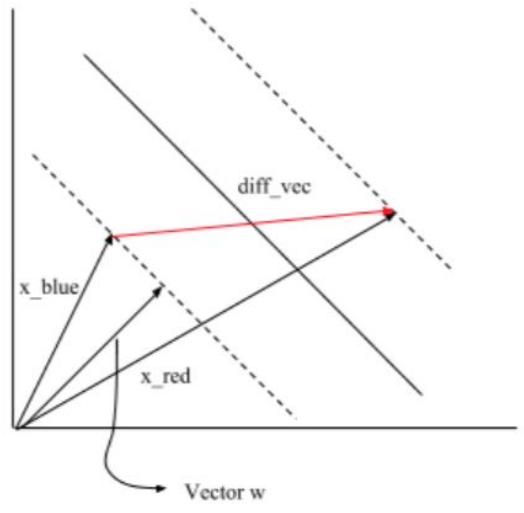


Figure 3 - Analyzing the Margin Width (drawn)

So then we need to get the magnitude of the width. So we have the vector w, so the width of the classifier region can be found by normalization of the w vector:

$$width = (\overline{x}_{red} - \overline{x}_{blue}) \cdot \frac{\overline{w}}{||w||}$$

But we can actually go farther! Because of the assertion made previously, that if the data points are on the edge of the classifier then we know that they are actually equal to from the way our classifier was set up. So distribution over vectors is allowed for the dot product.

5

$$width = (\overline{x}_{red} \cdot \overline{w} - \overline{x}_{blue} \cdot \overline{w}) \cdot \frac{1}{||w||}$$

Noting that the *x_red* and *x_blue* are points on one side of the decision classifier, so we know that

$$width = (1 - b - 1 + b) \cdot \frac{1}{||w||}$$

That substitution is totally allowed because of our equation 2. Therefore,

$$width = \frac{2}{||w||} \qquad (3)$$

Once we have our width, we're trying to maximize the width. In other words, we're trying to minimize the ||w|| so that the width is actually maximized. In fact, the focus on minimization is actually going to be: $\frac{1}{2}||w^2||$ as this will make some of the mathematics easier in later manipulations. So the area that we turn to is a constrained minimization problem, something that is easily able to be handled with Lagrange Multipliers.

Recalling that Lagrange Multipliers can be summarized by the two equations:

$$L(x, y, \lambda) = f(x, y) + \lambda \cdot g(x, y)$$

Where the minimization is performed as following :

$$\nabla_{x, y, \lambda} L(x, y, \lambda) = 0$$

So then this minimization using Lagrange Multipliers can be expressed as:

$$L = \frac{1}{2}||\overline{w}||^2 - \sum \alpha_i \left[y_i (\overline{w} \cdot \overline{x} + b) - 1 \right]$$

Then following the algorithm for Lagrange multipliers, noting that differentiation with respect to a vector follows the same form as differentiation in one dimension.

$$\frac{\delta L}{\delta w} = w_i - \sum \alpha_i y_i x_i = 0$$

$$w_i = \sum \alpha_i y_i x_i$$

This is result is exciting and powerful. This means that the vector *w* is actually a linear sum of the sample set vectors, multiplied by a coefficient and also the y value. The Lagrange multiplier method is not finished as the other partial derivative with respect to the other variable needs to be taken.

$$\frac{\delta L}{db} = -\sum \alpha_i y_i = 0$$

$$\sum \alpha_i y_i = 0$$

This comes to an important stopping point. This issue in looking at the Lagrangian Multiplier is seeing that there is a square in the equation. This is known in optimization problems as a *quadratic programming problem* (QP problem). See Appendix B for more information on quadratic programming.

To continue with the derivation however, the result for the vector *w* is going to be substituted back into the original equation for the Lagrangian. The derivation is as follows:

$$L = \tfrac{1}{2}\|\overline{w}\|^2 - \sum\alpha_i\left[y_i(\overline{w}\cdot\overline{x} + b) - 1\right]$$

$$L = \tfrac{1}{2}(\sum\alpha_i y_i \overline{x}_i)\cdot(\sum\alpha_j y_j \overline{x}_j) - \sum(\alpha_i y_i \overline{x}_i)\cdot(\sum\alpha_j y_j \overline{x}_j) - \sum(\alpha_i y_i b) + \sum\alpha_i$$

Note that there are seeming two $\sum(\alpha_i y_i \overline{x}_i)$ terms coming from the fact that there is also a *w* vector in the equation being multiplied by the other components.
Also note, that we can move b out of the third term to have:

$$L = \tfrac{1}{2}(\sum\alpha_i y_i \overline{x}_i)\cdot(\sum\alpha_j y_j \overline{x}_j) - \sum(\alpha_i y_i \overline{x}_i)\cdot(\sum\alpha_j y_j \overline{x}_j) - b\sum(\alpha_i y_i) + \sum\alpha_i$$

Here note that $\sum\alpha_i y_i = 0$. Therefore, our equation goes to

$$L = \tfrac{1}{2}(\sum\alpha_i y_i \overline{x}_i)\cdot(\sum\alpha_j y_j \overline{x}_j) - \sum(\alpha_i y_i \overline{x}_i)\cdot(\sum\alpha_j y_j \overline{x}_j) + \sum\alpha_i$$

This equation can be further simplified by combining some of the linear sums to get:

$$L = \sum\alpha_i - \tfrac{1}{2}(\sum_i\sum_j\alpha_i\alpha_j y_i y_j\ \overline{x}_i\cdot\overline{x}_j)$$

This is it! This is the equation that we are trying to find the maximum of. It is important to note that the maximization depends on the sample data itself! **It depends only on the dot product between the pairs of samples.** This will allow us to apply the Kernel trick to transform the data into a new space.

Finally, we can circle back to our original decision rule and see that the decision rule also only depends on the dot product between the sample vectors and the neighbors. The following mathematical translation is shown below:

$$\sum\alpha_i y_i * (\overline{x}_i\cdot u_i) + b \geq 0 \text{ then it's a red sample}$$

$$\sum\alpha_i y_i * (\overline{x}_i\cdot u_i) + b \leq 0 \text{ then it's a blue sample}$$

Relating back to the original pictorial representation.

That concludes the derivation for the linear classifiers using support vector machines. However, if the data cannot be separated, like in Figure 4, then perhaps the data could be transformed. See Figure 5 below.

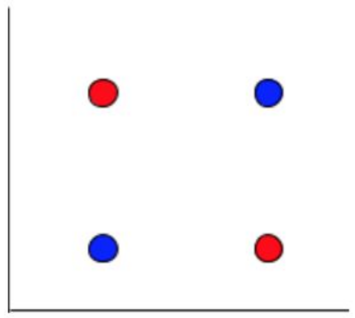


Figure 4 - Impossible Data to Separate Linearly

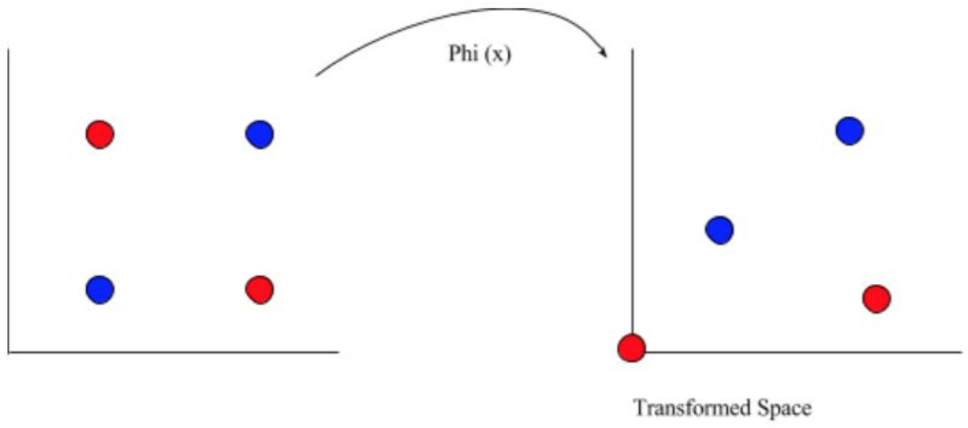


Figure 5 - Transformation of Data to Domain where data can be classified

Let us call the transformation $\phi\ (\overline{x})$. So then because the dependence is solely on the dot product between two vectors, what we are now trying to maximize in the transformed space is

$$\phi\ (x_i) \cdot \phi\ (x_j)$$

And our decision metric is then:

$$\phi\ (x_i) \cdot \phi\ (u)$$

Then for the sake of notation, if we have a function

$$K(x_i,\ x_j)\ = \phi\ (x_i) \cdot \phi\ (x_j)$$

Then we are finished! This is the **kernel function**; it provides us with the dot product in another space. The transformation into the other space does not even have to be known. The most common example of a kernel function (which is actually the default basis when a support vector classifier is declared in python's `sklearn`), is the radial basis function. The radial basis function is shown below:

$$K(x_i,\ x_j)\ = exp(\frac{-||x_i - x_j||}{2\sigma^2})$$

Appendix C: Full R Code for Analysis

```
# John Larkin. 4/13/16. Swarthmore College.
# E91 Final Project: Analyzing Flow Cytometry Data
# *************************************************************************
#
#     PROGRAM:      biomed_final_proj
#     ========      ==================
#
#     DESCRIPTION:   Analyze the flow cytometry also presented.
#     ============
#                    This program is going to be reported for the author's
#                    E91 Final Project.
#                    The purpose of this program is to demonstrate to the
#                    user (and also the writer) of how flow cytometry data
#                    can be used. In addition, the author hopes to produce
#                    interesting analyse of the presented data and to
#                    demonstrate some interesting visualizations.
#
#     COPYWRITE:     John Larkin
#     ==========     Swarthmore College
#                    500 College Avenue
#                    Swarthmore, PA 19081
#
#     DATE CREATED:  04/13/2016
#     =============
#
#     LAST CHANGED:  04/21/2016
#     =============
#
# **************************************************************************
# This project is going to observe and load in various files of the FCS
# file type. The flow cytometry analyse will also be accompanied with a
# formal write up.

# We are going to need to use SVMs to train our classifier
# Import a package where we can use SVMs
library("e1071")
# Also going to perform some data manipulation
library("dplyr")
library("flowCore")
library("flowPeaks")
library("flowViz")
# This is just going to load in the various data files from our folder.

# Only need the following two lines on the first time, we need to install the R packages
# source("http://bioconductor.org/biocLite.R")
# biocLite("flowCore")

time_init <- proc.time()

#********************************************************************************************#
#***********************            READING IN THE DATA           ***************************#
#********************************************************************************************#

# Preallocating the list to hold all of the data.
fcsdataset <- vector(mode= "list", length = 103)
# Now going to import all of the data.
# Was going to download every third tube for every individual
# Did some more research and then decided on the sixth one because better antigen classifiers

# lengthvec <- seq(3,819,8)
lengthvec <- seq(6, 822, 8)
print('Please wait... The data is being imported.')
j <- 1
for (i in lengthvec){
    # First need to create the appropriate extension where all of the data is stored.
    directory =
"~/Desktop/College/Academics/Junior/Spring/E91_Biomedical_Signals/FinalProjectFlowCytometry/Fl
owRepositoryData/"
    extension = sprintf("%04d", i)
    end = ".FCS"
    totalstring = paste(directory,extension,end, sep="")
    fcsdataset[[j]] <- flowCore::read.FCS(totalstring, transformation = FALSE, dataset= 2)
    j <- j+1
}

print('The data has been succesfully imported! 103 different FCS tubes from different patients
are being looked at.')

print('Let\'s take a basic look at some of our data visually. Generating Scatter Plot
Matrices...')
plot(fcsdataset[[1]])
plot(fcsdataset[[2]])

# We can also look at some statistics for some of the samples. Let's just look at the first
one again.
print('Let\'s also take a basic look at some statistics of the expression matrix for the first
sample. Generating statistics...')
summary(fcsdataset[[1]]@exprs)

# Ok. So now that we have everything imported. The data is divided up among the exprs and
# the parameters slots.
# Event information is accessed via exprs() and exprs<-.
# EVENT INFORMATION - this allows us to look at the individual FL data as well as what was
going on at a single event level in a tube.
# Parameter information. Stored as an AnnotatedDataFrame that contains information derived
from
# the FCS file's $P<n> keywords. DESCRIBE THE DETECTOR AND STAIN INFORMATION.
# The entire list is available via the parameter method, but more commonly accessed through
# the "names", "featureNames", and "colnames" method.
# names function returns a concatenated version of names and featureName.
# colnames return s the detector names.

# FS n- forward scatter
# SS - side scatter
# FL. - these are particular fluorescent dyes that are conjugated with the anti bodies

# The algorithm that is to be implemented is the flowPeaks and SVM algorithm used by one of
the groups in the
# competition. The competition authors cite the following steps leading to success in their
classification.
#
# Specific steps for the AML challenge:
# 1. For each of the 8 tubes, we collect a dataset that contains all cells from “normal”
patients and another
#    dataset that contains all cells from “aml” patients.
# 2. For the 16 datasets (8 tubes x 2 disease conditions), we applied flowPeaks algorithm to
do automatic
#    clustering after the FS and SS channels have been removed.
# 3. For each original CSV file, find out the corresponding tube number, and then compute the
proportions using
#    the flowPeaks clustering from step 2. Note that two proportion vectors are computed, one
for “aml” and the
#    other for “normal”, even though the CSV file itself may be associated with just “aml” or
with just “normal”.
# 4. For each patient (“aml” or “normal” or “NA”), concatenate all of the two proportion
vectors of the 8 tubes
#    together to form a long vector.
# 5. We filter out the element of the long vector (formed from step 4) where the proportion
changes less than 0.1%
```

```
#    or the p-value for the two sample t-test is greater than 0.5 between the two groups of
patients. A support
#    vector machine (SVM) with a linear kernel and scale normalization is used to build a
classifier, which is
#    used to predict the NA labeled patients. The parameters of the SVM are optimized by SMO.

# so now let's build a dataset where everryone is normal and where everyone has aml. going to
use this as preknown knowledge
aml_ident <- c(3,5,7,24,31,35,47,56,58,65,86,87,93,99,101,103)
normal_ident <- 1:103
normal_ident <- setdiff(normal_ident, aml_ident)

# Now lets actually separate the data
total_aml <- fcsdataset[aml_ident]
total_normal <- fcsdataset[normal_ident]

# Now going to jump into the flowPeaks library
library(flowPeaks)

# Let's create a list of matrices for the data and then remove the FSC and SSC and time
training_exprs_set_AML <- list()
training_exprs_set_NORM <- list()
# Creating the List of all relevant quant. data
for(i in 1:length(aml_ident)){
    training_exprs_set_AML[[i]] <- total_aml[[i]]@exprs
}
for(i in 1:length(normal_ident)) {
    training_exprs_set_NORM[[i]] <- total_normal[[i]]@exprs
}
for(i in 1:length(training_exprs_set_AML)) {
    matrix_to_look_at <- training_exprs_set_AML[[i]]
    # now we want to drop FSC and SSC and Time
    matrix_to_look_at <- matrix_to_look_at[ , -c(1,2,8)] # that comma is very important!
    # differentiates between rows and columns. don't forget that
    training_exprs_set_AML[[i]] <- matrix_to_look_at
}
for(i in 1:length(training_exprs_set_NORM)) {
    matrix_to_look_at <- training_exprs_set_NORM[[i]]
    # now we want to drop FSC and SSC and Time
    matrix_to_look_at <- matrix_to_look_at[ , -c(1,2,8)] # that comma is very important!
    # differentiates between rows and columns. don't forget that
    training_exprs_set_NORM[[i]] <- matrix_to_look_at
}


# Let's try and take a peak at some of the classification
# Let's once again just use the first one for an example
# From the user guide:
# "If a user has the data that is of class flowFrame in the flowCore package, he just needs to
use the
# transformed data of the expression slot of the flowFrame where only channles of interest are
selected"
# Let's look at channel 2 and CD38 because those seem to be relatively good predictors
one_example_AML <- flowPeaks(training_exprs_set_AML[[1]][,c(2,5)])
plot(one_example_AML)

one_example_NORM <- flowPeaks(training_exprs_set_NORM[[1]][,c(2,5)])
plot(one_example_NORM)
# Ok so that's progress... then we just need to get our classifier.
# So the first thing to do is to find the proportion vector for all of the AML patients
# and then subsequently to find the proportion vector for all of the normal patients
# Then as the paper suggests, we're going to concatenate that together and then eventually use
SVM to generate the end
# classifier. So we're going to have to run the flowPeaks algorithm quite a bit. Might be time
expesnive.
```

```
# For accurate and precise memory location

#*********************************************************************************************#
#********************         GENERATING CLASS EXPRESSION MATRIX      **********************#
#*********************************************************************************************#
AML_training_matrix <- matrix(data = NA, ncol = 5)
NORM_training_matrix <- matrix(data = NA,  ncol = 5)
AML_training_matrix <- training_exprs_set_AML[[1]]
NORM_training_matrix <- training_exprs_set_NORM[[1]]
# Now let's bind the rows
for (i in 2:length(training_exprs_set_AML)) {
    AML_training_matrix <- rbind(AML_training_matrix, training_exprs_set_AML[[i]])
}
for (i in 2:length(training_exprs_set_NORM)) {
    NORM_training_matrix <- rbind(NORM_training_matrix, training_exprs_set_NORM[[i]])
}

#*********************************************************************************************#
#*********************            flowPeaks ALGORITHM CALLED         ***********************#
#*********************************************************************************************#
# Cool! That's so great. so far, everything is at least compiling and running. not throwing
errors I should say.
# I actually can't believe that worked. I'm not 100% sure but i have some confidennce.
# So then we need to run our flow peaks clustering algorithm on both of these MASSIVE matrices
clustered_AML_train_matrix <- flowPeaks(AML_training_matrix)
clustered_NORM_train_matrix <- flowPeaks(NORM_training_matrix)

# Then we need to assign our clustered points
# So that data is stored in <name>_training_matrix$peaks.cluster
# Then we just need to count the number of things that have been classified as 1, classified
as 2, classified as 3
num_of_unique_AML <- length(unique(clustered_AML_train_matrix$peaks.cluster))
print(c('The number of distinct values in the AML data is', toString(num_of_unique_AML)))
proportion_list <- vector(mode = 'list', length = num_of_unique_AML)
for (i in 1:length(proportion_list)) {
    proportion_list[[i]] <- sum(clustered_AML_train_matrix$peaks.cluster == i)
}
proportion_list_AML <- unlist(proportion_list)
proportion_list_AML_unity <- proportion_list_AML / sum(proportion_list_AML)

# Now let's repeat this with the normal data
num_of_unique_NORM <- length(unique(clustered_NORM_train_matrix$peaks.cluster))
print(c('The number of distinct values in the NORM data is', toString(num_of_unique_NORM)))
proportion_list <- vector(mode = 'list', length = num_of_unique_NORM)
for (i in 1:length(proportion_list)) {
    proportion_list[[i]] <- sum(clustered_NORM_train_matrix$peaks.cluster == i)
}
proportion_list_NORM <- unlist(proportion_list)
proportion_list_NORM_unity <- proportion_list_NORM / sum(proportion_list_NORM)

#*********************************************************************************************#
#************************          GENERATING FEATURE MATRIX      ***************************#
#*********************************************************************************************#
# Ok dope. So now that we have our proportion vectors for the entire sample set then we can
# look at the proportion for each of individuals in those two sets.
# This will become our feature matrix!!!
# for the entire feature subset, compute the labels for every person
AML_labeled_data_set <- vector('list', length = length(total_aml))
for (i in 1:length(total_aml)) {
    AML_labeled_data_set[[i]] <- assign.flowPeaks(clustered_AML_train_matrix,
training_exprs_set_AML[[i]])
}
AML_feature_matrix <- vector('list', length = length(total_aml))
for (i in 1:length(total_aml)) {
    for (j in 1:num_of_unique_AML) {
        AML_feature_matrix[[i]][j] <- sum(AML_labeled_data_set[[i]] == j)
    }
}
AML_with_norm_labeled_data_set <- vector('list', length = length(total_aml))
for (i in 1:length(total_aml)) {
    AML_with_norm_labeled_data_set[[i]] <- assign.flowPeaks(clustered_NORM_train_matrix,
training_exprs_set_AML[[i]])
}
temp <- vector('list', length = length(total_aml))
for (i in 1:length(total_aml)) {
    for (j in 1:num_of_unique_NORM) {
        temp[[i]][j] <- sum(AML_with_norm_labeled_data_set[[i]] == j)
    }
}
# Then we just need to column bind these two together
AML_feature_matrix<-do.call(rbind, AML_feature_matrix)
temp <- do.call(rbind, temp)
AML_feature_matrix <- cbind(AML_feature_matrix, temp)

############# Now same with the normal data
NORM_labeled_data_set <- vector('list', length = length(total_normal))
for (i in 1:length(total_normal)) {
    NORM_labeled_data_set[[i]] <- assign.flowPeaks(clustered_NORM_train_matrix,
training_exprs_set_NORM[[i]])
}
NORM_feature_matrix <- vector('list', length = length(total_normal))
for (i in 1:length(total_normal)) {
    for (j in 1:num_of_unique_NORM) {
        NORM_feature_matrix[[i]][j] <- sum(NORM_labeled_data_set[[i]] == j)
    }
}
NORM_with_aml_labeled_data_set <- vector('list', length = length(total_normal))
for (i in 1:length(total_normal)) {
    NORM_with_aml_labeled_data_set[[i]] <- assign.flowPeaks(clustered_AML_train_matrix,
training_exprs_set_NORM[[i]])
}
temp <- vector('list', length = length(total_normal))
for (i in 1:length(total_normal)) {
    for (j in 1:num_of_unique_AML) {
        temp[[i]][j] <- sum(NORM_with_aml_labeled_data_set[[i]] == j)
    }
}
# Then we just need to column bind these two together
NORM_feature_matrix<-do.call(rbind, NORM_feature_matrix)
temp <- do.call(rbind, temp)
NORM_feature_matrix <- cbind(temp, NORM_feature_matrix)


total_feature_matrix <- rbind(AML_feature_matrix,
NORM_feature_matrix[1:nrow(AML_feature_matrix),])
for (i in 1:nrow(total_feature_matrix)) {
    total_feature_matrix[i,1:3] <-
total_feature_matrix[i,1:3]/sum(total_feature_matrix[i,1:3])
    total_feature_matrix[i,4:7] <-
total_feature_matrix[i,4:7]/sum(total_feature_matrix[i,4:7])
}
#
# AML_feature_matrix<-do.call(rbind, AML_feature_matrix)
# NORM_feature_matrix<-do.call(rbind, NORM_feature_matrix)
# # Then we need to bind these by column
# temp <- head(NORM_feature_matrix, nrow(AML_feature_matrix))
# total_feature_matrix <- cbind(AML_feature_matrix, temp)

# Also should generate an answer key to train our SVM on
answerpt1 <- rep(1, 16)
answerpt2 <- rep(0, 16)
answer <- c(answerpt1, answerpt2)
```

```
final_training_set <- cbind(total_feature_matrix, answer)
#*****************************************************************************************#
#************************* BUILDING OUR SUPPORT VECTOR CLASSIFIER *************************#
#*****************************************************************************************#
model <- e1071::svm(total_feature_matrix, answer, scale = FALSE)

#*****************************************************************************************#
#************************* TESTING OUR SUPPORT VECTOR CLASSIFIER **************************#
#*****************************************************************************************#
# Ok now let's load in our actual test data so that we can see how well our classifier did
fcsTestingSet <- vector(mode= "list", length = 50)
lengthvec <- seq(2478, 2870, 8)
print('Please wait... The testing set data is being imported.')
j <- 1
for (i in lengthvec){
    # First need to create the appropriate extension where all of the data is stored.
    directory =
"~/Desktop/College/Academics/Junior/Spring/E91_Biomedical_Signals/FinalProjectFlowCytometry/Fl
owRepositoryData/"
    extension = sprintf("%04d", i)
    end = ".FCS"
    totalstring = paste(directory,extension,end, sep="")
    fcsTestingSet[[j]] <- flowCore::read.FCS(totalstring, transformation = FALSE, dataset= 2)
    j <- j+1
}

testing_exprs_set <- list()
# Creating the List of all relevant quant. data
for(i in 1:length(fcsTestingSet)){
    testing_exprs_set[[i]] <- fcsTestingSet[[i]]@exprs
}
for(i in 1:length(testing_exprs_set)) {
    matrix_to_look_at <- testing_exprs_set[[i]]
    # now we want to drop FSC and SSC and Time
    matrix_to_look_at <- matrix_to_look_at[ , -c(1,2,8)] # that comma is very important!
    # differentiates between rows and columns. don't forget that
    testing_exprs_set[[i]] <- matrix_to_look_at
}

testing_labeled_data_set <- vector('list', length = length(fcsTestingSet))
for (i in 1:length(fcsTestingSet)) {
    testing_labeled_data_set[[i]] <- assign.flowPeaks(clustered_AML_train_matrix,
testing_exprs_set[[i]])
}
testing_feature_matrix <- vector('list', length = length(fcsTestingSet))
for (i in 1:length(fcsTestingSet)) {
    for (j in 1:3) {
        testing_feature_matrix[[i]][j] <- sum(testing_labeled_data_set[[i]] == j)
    }
}
testing_with_norm_labeled_data_set <- vector('list', length = length(fcsTestingSet))
for (i in 1:length(fcsTestingSet)) {
    testing_with_norm_labeled_data_set[[i]] <- assign.flowPeaks(clustered_NORM_train_matrix,
testing_exprs_set[[i]])
}
temp <- vector('list', length = length(fcsTestingSet))
for (i in 1:length(fcsTestingSet)) {
    for (j in 1:4) {
        temp[[i]][j] <- sum(testing_with_norm_labeled_data_set[[i]] == j)
    }
}
# Then we just need to column bind these two together
testing_feature_matrix<-do.call(rbind, testing_feature_matrix)
temp <- do.call(rbind, temp)
testing_feature_matrix <- cbind(testing_feature_matrix, temp)
```

```
for (i in 1:nrow(testing_feature_matrix)) {
    testing_feature_matrix[i,1:3] <-
testing_feature_matrix[i,1:3]/sum(testing_feature_matrix[i,1:3])
    testing_feature_matrix[i,4:7] <-
testing_feature_matrix[i,4:7]/sum(testing_feature_matrix[i,4:7])
}
# Now just for our reference let's have the answer key for which one's are actually right
# for our classifications
AML_ans_key <- c(314, 326, 337, 340, 344, 348)
normal_ans_key <- 310:359
normal_ans_key <- setdiff(normal_ans_key, AML_ans_key)

pred <- predict(model, testing_feature_matrix)

total_time_of_prog <- proc.time() - time_init
print('The total time (elapsed) of the program to run is (s)')
total_time_of_prog[3]
print('The user time of the program is (s)')
total_time_of_prog[1]
print('Therefore, the system time is (s)')
total_time_of_prog[2]
```

Figure Works Cited:

1. Surpatne, Nikita V. "Optical System." *Nikita V Surpatne*. Nikita V Surpatne, Jan. 2000. Web. 27 Apr. 2016.
2. ChemWik, UC Davis. "Electronic Spectroscopy: Theory." *Electronic Spectroscopy: Theory*. UC Davis, 02 Oct. 2013. Web. 27 Apr. 2016.
3. Aryal, Sagar. "Differences Between Antigen and Antibody." *Online Microbiology Notes*. Microbiology Info, 05 Oct. 2015. Web. 27 Apr. 2016.
4. Biosciences, BD. "Flow Cytometry." *Introduction to Flow Cytometry: A Learning Guide* (2000): 390-432. Print.

Literature Works Cited:

1. Biosciences, BD. "Flow Cytometry." *Introduction to Flow Cytometry: A Learning Guide* (2000): 390-432. Print.
2. Cancer Treatment Centers of America. *Acute Myeloid Leukemia: Overview & Treatment Options*. N.p., n.d. Web. 29 Apr. 2016.
3. Appelbaum, Frederick. "Acute Myeloid Leukemia." *Gknation*. Leukemia & Lymphoma Society, n.d. Web. 29 Apr. 2016.
4. Aghaeepour, Nima et al. "CRITICAL ASSESSMENT OF AUTOMATED FLOW CYTOMETRY DATA ANALYSIS TECHNIQUES." *Nature methods* 10.3 (2013): 228–238. *PMC*. Web. 3 May 2016.
5. Webber, Brian A, Melissa M Cushing, and Shiyong Li. "Prognostic Significance of Flow Cytometric Immunophenotyping in Acute Myeloid Leukemia." *International Journal of Clinical and Experimental Pathology* 1.2 (2008): 124–133. Print.

Code Works Cited:

1. Balkay, Laszlo. "FCS Data Reader - MATLAB Central." *FCS Data Reader*. MATLAB Central, 13 Jan. 2006. Web. 03 May 2016.
2.