American University of Armenia

College of Science and Engineering

Numerical Analysis Project

# Microarray Data Classification using Support Vector Machine (SVM)

Team: Elen Vardanyan, Lilit Janjughazyan

Fall, 2016

# Abstract

Microarray gene expression data are usually highly dimensional, ranging from a few to tens of thousands of genes, in contrast to the number of samples, e.g., a few tens of patients. In this project, we use the support vector machine (SVM) for blood cancer classification with microarray data. For this, we will implement the algorithm for SVM classification, train an SVM model and report its efficiency on a new data.

**Keywords:** Microarray; Gene expression data; Cancer classification; Cancer diagnosis mathematical model; Support vector machine (SVM); Prediction; Supervised Learning.

# Contents

# List of Figures

# Chapter 1

# Introduction

## 1.1 Problem Setting and Description

Over the last few years the widespread use of DNA microarrays has made possible to create large datasets of molecular information that can characterize complex biological systems very precisely. More and more machine learning algorithms are being applied to DNA microarray data to do molecular classification, and they have shown to have statistical and clinical relevance for a variety of cancer types and their classification.

A widely used machine learning algorithm, Support Vector Machines (SVMs), has shown satisfying performance and results on biological classification tasks, including gene expression microarrays.

The aim of this Project is to:

***Classify two types of blood cancer, Acute Myeloid Leukemia (AML) and Acute Lymphocytic Leukemia (ALL), using the Support Vector Machines (SVMs).***

## 1.2 History and Background of the Problem

Over the past decades, a persistent development related to cancer research has been accomplished. Screening in an early stage is one of the methods applied to discover types of cancer long before symptoms appear. Moreover, scientists have developed new methods for the early prediction of cancer treatment outcome. New technologies in the field of medicine have contributed to the collection of large amounts of cancer data, as well as gave medical researchers an opportunity

to easily access those data. Still, the accurate prediction of a disease outcome is one of the most interesting and challenging tasks for doctors. As a result, machine learning (ML) methods have become a popular tool for medical research community. Due to these techniques it is now possible to find and identify patterns from complex databases, as well as to analyze the relationships between those patterns and effectively predict future outcomes of a cancer type.

Discrimination of acute myeloid leukemia (AML) from acute lymphoblastic leukemia (ALL) [TG99] was one of the first cancer classification studies ever conducted. In this problem, a total of 38 training samples belong to the two classes, 27 ALL instances and 11 AML instances. The accuracy of the trained classifier was evaluated using 35 test samples. The expression levels of 7,129 genes were given for each sample. A linear SVM trained on this data accurately classified 34 out of 35 test samples.

Nowadays, scientists and doctors are in the constant search of new methods to differentiate between types and subtypes of various diseases. Certainly, its very first advantage is the importance of personalized medication and treatment. The idea of personalized medicine is that when people have precise diagnosis on their diseases, they can be treated differently, which intuitively suggests maximum results. It is worth noting that in some cases, taking into account the different genetic backgrounds of different patients, their cancer types and other relevant biological data, the treatment that is good for someone may be inefficient, or, in the worst case, even hazardous for another person. Thus, applications of different Machine Learning methods in disease classification are a growing trend that have already proven to be very useful. In this paper, the usage of SVM classification in prediction of ALL and AML cancer types is going to be explained. In addition, the overall performance of our implementation of SVM is going to be discussed.

## 1.3 The Structure of the Project Paper

The next chapter explains the idea behind microarray analysis, gives the theoretical background of the SVMs, and serves as an introduction to their use in analysis of DNA microarray data. An informal theoretical motivation of SVMs both from a geometric and algorithmic perspective followed by an application to leukemia classification is described in Section 2. The numerical solution, as well as the algorithm and its implementation are provided in Section 3. Lastly, the results and conclusions are presented in Section 4.

# Chapter 2

# Theoretical Background

## 2.1 DNA Microarrays

Microarray, also known as DNA chip, is a collection of microscopic DNA spots attached to a solid surface. [Wik16a] Microarrays provide a convenient way of obtaining gene expression levels for a large number of genes simultaneously. Each spot on a microarray chip contains the clone of a gene from a tissue sample. Some mRNA samples are labeled with two different kinds of dyes. After mRNA interaction with the genes, i.e., hybridization, the color of each spot on the chip will change. The resulted image reflects the characteristics of the tissue at the molecular level.

Microarrays can, therefore, be used to help classify and predict different types of cancers. Traditional methods for diagnosis of cancers are mainly based on the structural (morphological) appearances of the cancers. However, sometimes it is almost impossible to see clear distinctions between some types of cancers according to their appearances. Thus, the microarray technology stands to provide a more quantitative and trustworthy means for cancer diagnosis. For instance, gene expression data have been used to obtain good results in the classifications of lymphoma, leukemia, breast cancer, and liver cancer. [FC05]

SVM has become an increasingly popular tool for supervised learning tasks involving classification and regression. Training an SVM requires the solution of a large quadratic programming problem, depending on the feature and sample quantities. Due to memory restrictions traditional optimization methods cannot be directly applied. Up to now, several approaches exist for circumventing the above shortcomings and work well. The obtained results by these methods are tested on a leukemia microarray dataset and compared with the exact solution model problem. [NS05]

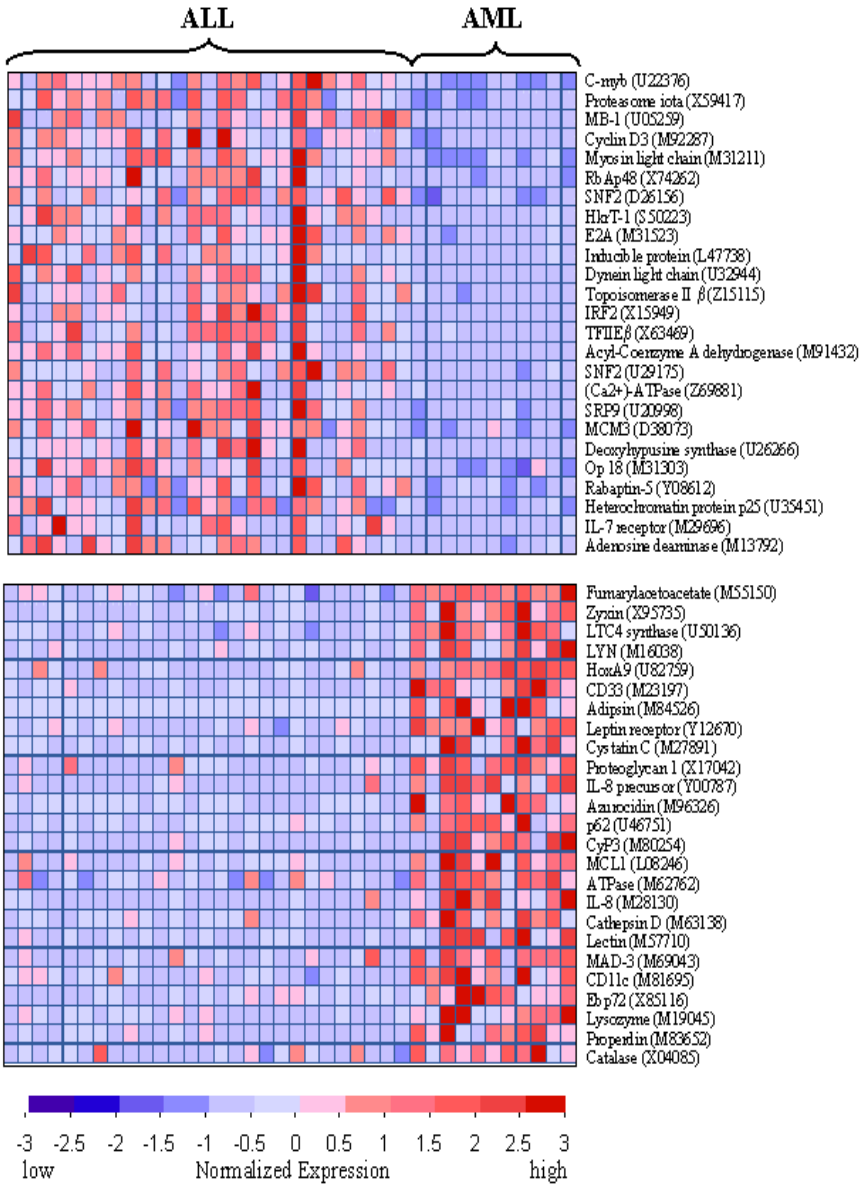


Figure 2.1: Genes distinguishing ALL from AML. The 50 genes most highly correlated with the ALL/AML class distinction are shown. Each row corresponds to a gene, with the columns corresponding to expression levels in different samples.

## 2.2 Machine Learning

Machine learning algorithms can be divided into two main types, supervised and unsupervised learning. Machine learning that makes use of labeled training data is referred to as supervised learning. The problem of unsupervised learning is to try to find hidden structure in unlabeled data.

All the data points in the training set contain class labels, called "target values", and several features or observed variables, called "attributes". The usage of SVM

aims to create a training set based model, that will predict the target values of the test data having only the test data attributes.

Given:

- A data universe X
- A sample set S, where $S \subset X$
- A hypothesis function (labeling process) $f : X \mapsto \{\text{true, false}\}$
- A labeled training set D, we $\{ D = ( x,y ) \mid x \in S \text{ and } y = f(x) \}$

Compute a function $\hat{f} : X \mapsto \{\text{true, false}\}$ using D such that

$$\hat{f}(x) \cong f(x).$$

Let us proceed to more detailed explanation of the definition above. The data universe $X$ is the set of objects that are in the scope of interest. For example, this might be a set of proteins that have distinct functions; or a set of people who visited a particular web page. As indicated above, the sample set S is a subset of the data universe $X$. In general, most of the collections of data are very large (sometimes, infinitely large) and building models for them may be a highly time consuming task (impossible for infinite data universes) and that is where the sample set becomes a necessity. Thus, the sample set $S$ is there to make the process of model construction as tractable as possible, by representing the data universe.

The hypothesis function $f$ is the process that is responsible for providing the observed labels. $f$ should be able to differentiate the values true, false for all the elements $X$ when that element is observed. As an aside, please note, that the names of the labels true, false are arbitrary; instead of true, false we could have used 0, 1 or healthy, unhealthy etc. [Ham09]

The only limitation is to have a set with two distinct labels: one for the first class membership and one for the second class membership. We can also consider classification problems with more than two possibilities. The only difference from our definition of machine learning above would be that the codomain of the original labeling process $f$ and its model $\hat{f}$ is a set that includes an appropriate number of distinct labels. Summing it up, it is possible to get an information about the labels assigned to the elements in the data universe, without having a direct access

to the constructed model itself. For example, when we have a set of proteins, a hypothesis function $f$ might label the proteins by observing whether or not they are involved in the creation of hormones. This property of the hypothesis function is generally used to construct the training set $D$ by observing the labels of objects in $S$. Formally speaking, learning can be viewed as computing the function $\hat{f}$ as an approximation to or a model of the original process $f$ based on the training examples in $D$. The elements in the training set are input$-$output pairs of the original labeling function, $(x, y) \in f$ with $x \in S$ and y = $f$(x), and this means that modeling the function $f$ and modeling the training data $D$ are one and the same.

So, as soon as there is a model of the original labeling process, two useful things can be accomplished. First, the model can be used to predict the label of an element in the data universe $X$ without having to observe that element. Second, the model can provide some insight into the original labeling process.

### 2.2.1 Describing Data Mathematically

We are interested in constructing models of processes that label objects in a data universe according to which class they belong. Please also note, that each of the objects that may be labeled belong to either one of the classes, i.e. this is a binary classification problem. To make sure that the effectiveness is maximal, there is a need to describe the objects far beyond than by just defining an object as an element of the cross product of its attributes. In particular, a more mathematical approach will be brought in the definition of the similarities and differences of the objects in the data universe. Also a few more questions may arise: is it possible to separate two groups easily or is it a more challenging task than it may seem? Generally speaking, the idea of neatly describing the similarities and differences between objects is central in the development of classifiers, in particular for SVMs.

It is very important to understand the connection between some foundational basics of linear algebra, such as dot products, planes, vector spaces, and the input dataset. Understanding this relationship will facilitate the description of objects mathematically, as well as the construction of effective classifiers. For the beginning, note that objects in a data universe are described by a common set of attributes. Due to huge data universes, a subset of objects is usually taken into consideration for knowledge discovery tasks. Those subsets are referred to as training sets. Also, we assume that all the description attributes of the objects are over the real numbers $\mathbb{R}$. In that case, it becomes possible to visualize (interpret geometrically) each object as a point in n-dimensional coordinate system $\mathbb{R}_n$ by treating

each attribute as a distinct dimension.

Position vectors are the best, in case if we are interested in a point that best describes all the objects labeled, for instance true. Position vectors, technically speaking, are given as column vectors of the coordinate values of the objects of interest. A more geometric interpretation to position vectors is that of them being arrows rooted at the origin of the coordinate system that point to the location of the object in the n-dimensional space. Thus, the easiest way to find the above-mentioned descriptive point is by computing a position vector for the average object of all the objects labeled true. The position vector of an average object is computed by adding the position vectors of every objects labeled true, then multiplying the sum obtained by 1/n (in n-dimensional case there are n such objects).

Performing the above-mentioned operations will result in having the objects viewed as members of the data universe, as well as effectively converting our data universe into a vector space, i.e. each of the objects from the data universe can be viewed as vectors and new objects can be computed by using algebraic vector operations. Moreover, using dot products one can measure similarities between objects. Thus, it becomes evident that geometric interpretation of vector can help out in this kind of situations very well.

### 2.2.2 From Data Sets to Decision Functions

Let us start with a binary classification problem where the objects of interest are labeled +1 and -1. We take these numbers as labels because they are mathematically convenient, and as already known from the discussion in Chapter 2, Section 2.2 the names of labels do not really matter (are arbitrary) as long as there are appropriate number of distinct labels; in this case, one label for each class. We also assume that all attributes of our objects range over the real numbers; that is, we can view each object in our data universe as a position vector in the n-dimensional dot product space $\mathbb{R}^n$, where n is the number of attributes. Let us cast our classification problem into the machine learning framework developed in Section 2.2:

Given:

- Let the dot product space $\mathbb{R}^n$ be our data universe with vectors x $\in \mathbb{R}^n$ as objects.
- Let S be a sample set such that S $\subset \mathbb{R}^n$
- Let f : X $\mapsto$ {-1, 1} be the hypothesis.
- Let D = $\{(\bar{x}, y) \mid \bar{x} \in S$ and y = f($\bar{x}$)$\}$ be the training set $\forall \bar{x} \in \mathbb{R}^n$.

Compute a function $\hat{f} : \mathbb{R}^n \mapsto$ {-1, 1} using D such that

$$\hat{f}(\bar{x}) \cong f(\bar{x}).$$

The framework above is virtually identical to the previous Definition, except that we replaced our general notion of a data universe with the dot product space $\mathbb{R}^n$, and we use the label set {-1, 1} instead of true and false. Our aim is to construct a model $\hat{f}$ that approximates the original labeling function $f$.

## 2.3 Support Vector Machine (SVM)

As mentioned in the previous section, the SVMs are very useful for multi-dimensional data. The numeric input variables in our data (the column vectors) form an n-dimensional input space. In our test dataset, there are 38 samples, each with more than 7000 features. To understand better how the SVM copes with high-dimensional data, let us first start with the definition of a Maximum Margin Classifier that will give the intuition of the algorithm.

### 2.3.1 Maximum Margin Classifier

Two definitions that we will need starting from now are the following.

***A margin*** is the distance from a data point to the decision boundary.

***A hyperplane*** is a line that splits the input variable space, formally defined by the function

$$f(x) = \beta_0 + \beta^T x, \tag{2.1}$$

where $\beta$ is known as the weight vector and $\beta_0$ as the bias.

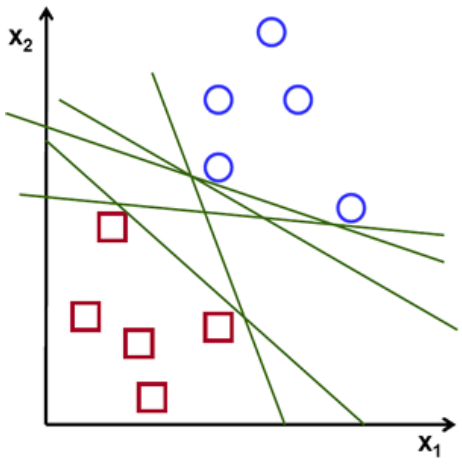


Figure 2.2: A binary classifier, separating two classes.

To better visualize the margin, we will consider a two-dimensional example (Figure 2.2). Imagine a problem of binary classification of circles and triangles. And, for the sake of simplicity, let's assume that our data is linearly separable, that is, all points belonging to one class lie on one side of the hyperplane (line), and all points of the other class lie on the other side.

It can be easily seen that the hyperplane is not unique. In fact, there are infinitely many hyperplanes that separate the training data. As a matter of convention, among all the possible representations of the hyperplane, the one chosen is

$$|\beta_0 + \beta^T x| = 1 \tag{2.2}$$

where x symbolizes the training examples closest to the hyperplane. In general, the training examples that are closest to the hyperplane are called *support vectors*. This representation is known as the canonical hyperplane and is unique based on any linearly separable training set. The maximum margin linear boundary also has the interesting property that the solution depends only on a subset of the examples, the support vectors, which appear exactly on the margin.

Now, we use the result of geometry that gives the distance between a point x and a hyperplane ($\beta$, $\beta_0$):

$$\text{distance} = \frac{|\beta_0 + \beta^T x|}{||\beta||}. \tag{2.3}$$

In particular, for the canonical hyperplane, the numerator is equal to one and the distance to the support vectors is

$$\text{distance}_{\text{ support vectors}} = \frac{|\beta_0 + \beta^T x|}{||\beta||} = \frac{1}{||\beta||}. \tag{2.4}$$

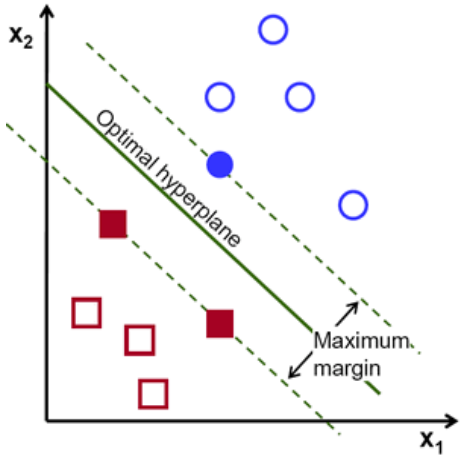


Figure 2.3: The most optimal binary classifier with the maximum margin.

The operation of the SVM algorithm is based on finding the hyperplane that gives the largest minimum distance to the training examples. Twice, this distance receives the important name of margin within SVM's theory.

$$M = \frac{2}{||\beta||} \tag{2.5}$$

Therefore, the optimal separating hyperplane maximizes the margin of the training data (See Figure 2.3). Finally, the problem of maximizing M is equivalent to the problem of minimizing a function L($\beta$) subject to some constraints. The constraints model the requirement for the hyperplane to classify correctly all the training examples $x_i$. Formally,

$$\min_{\beta,\beta_0} L(\beta) = \frac{1}{2}||\beta||^2 \text{ subject to } y_i(\beta^T x_i + \beta_0) \geq 1 \ \forall i = 1, ..., n, \tag{2.6}$$

where $y_i$ represents each of the labels of the training examples.

One major problem that will be addressed in the next Chapter is th misclassification error. Logically, even a single training example, if mislabeled, can radically change the maximum margin linear classifier. [Jaa]

# Chapter 3

# Numerical Solution

## 3.1 Numerical Background

In the previous chapter we described the geometric interpretation and the theory behind the maximum margin classifier. One major drawback of a real-life big data is that most of the times, they have much noise and, thus, are linearly non-separable. This means that it is not possible to find a binary linear classifier that can do the classification correctly.

Fortunately, SVMs are somewhat flexible concerning misclassifications, meaning that it is possible to obtain a hyperplane which misclassifies some of the samples. Thus, it makes sense to extend the SVM optimization problem for this case as well. The above mentioned misclassification is a new variable in the optimization problem that needs to be taken into account. Although there are many ways to modify this model to control the misclassification errors, in this paper only one of those ways will be discussed. That is the minimization of the same quantity plus a constant times the distance of the misclassified data points to the regions they belong.

$$\min_{\beta,\beta_0} ||\beta||^2 + C(\text{distance of the misclassified data points to their correct regions}), \tag{3.1}$$

where the parameter C is the SVM misclassification tolerance parameter. The costs of violating constraints are minimized together with the norm of the parameter vector $\beta$.

For every data point of the training set a new parameter $\xi_i$ is defined. Each of them contain the distance from their corresponding data points to their decision region. The following picture shows two classes of linearly non-separable data points, the separating hyperplane and the distances for the misclassified elements to their correct decision regions.

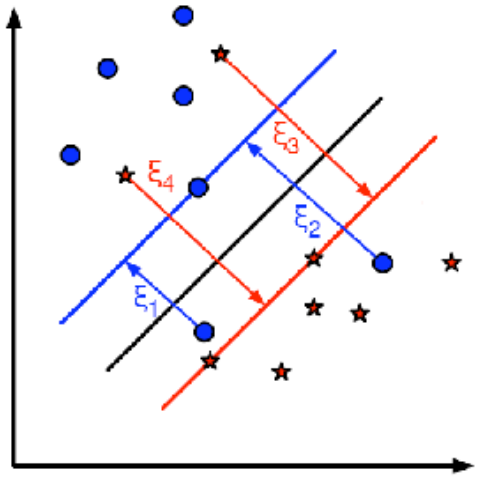


Figure 3.1: Only the distances of the samples that are misclassified are shown in the picture. The distances of the rest of the samples are zero since they lay already in their correct decision region.

The choice of the parameter C is done in order to control the trade-off between the training and the test sets. It can be considered as a way to regulate the overfitting[1]. There is no general way to choose the parameter C at all times, but here are some "rules" that can be taken into account before choosing that parameter:

- Large values of C result in *less misclassification errors* but a relatively *smaller margin*.
- Small values of C result in *more misclassification errors* but a relatively *bigger margin*.

Finally, the optimization problem 2.6 obtained in the previous chapter can be mathematically reformulated in the following way:

$$\min_{\beta,\beta_0} L(\beta) = \frac{1}{2}||\beta||^2 + C\sum_i \xi_i \text{ subject to } y_i(\beta^T x_i + \beta_0) \geq 1 - \xi_i \text{ and } \xi_i \geq 0, \forall i. \tag{3.2}$$

As the dimensions of our data are very big, the traditional optimization techniques will not work well. That is why one of the most used and efficient ways of tackling this problem is Quadratic Programming Optimization (QP) method. [Men] QP is a special type of mathematical problem of optimizing a quadratic function of several variables subject to linear constraints on them.

[1] making poor predictions due to overreacting to minor fluctuations in the training dataset

The quadratic programming problem with n variables and m constraints can be formulated as follows:

- a real-valued, n-dimensional vector **c**,
- an nxn-dimensional real symmetric matrix Q,
- an mxn-dimensional real matrix A, and
- an m-dimensional real vector **b**,

the objective of quadratic programming is to find an n-dimensional vector x, that will

$$\begin{aligned} \text{minimize } & \frac{1}{2}\mathbf{x}^T Q\mathbf{x} + \mathbf{c}^T\mathbf{x} \\ \text{subject to } & A\mathbf{x} \leq \mathbf{b}. \end{aligned}$$

The notation A$\mathbf{x} \leq \mathbf{b}$ means that every entry of the vector A$\mathbf{x}$ is less than or equal to the corresponding entry of the vector **b**.

Quadratic programming is especially simple when Q is positive definite and there are only equality constraints; specifically, the solution process is linear. By using Lagrange multipliers and seeking the extremum of the Lagrangian, it may be readily shown that the solution to the equality constrained problem

$$\begin{aligned} \text{minimize } & \frac{1}{2}\mathbf{x}^T Q\mathbf{x} + \mathbf{c}^T\mathbf{x} \\ \text{subject to } & E\mathbf{x} = \mathbf{d} \end{aligned}$$

is given by the linear system

$$\begin{bmatrix} Q & E^T \\ E & 0 \end{bmatrix} \begin{bmatrix} \mathbf{x} \\ \lambda \end{bmatrix} = \begin{bmatrix} -\mathbf{c} \\ \mathbf{d} \end{bmatrix}$$

where $\lambda$ is the set of Lagrange Multipliers which come out of the solution alongside **x**. [Wik16b]

### 3.1.1 Feature Space: Kernel Functions

This subsection discusses the method that can be used to construct a non-linear mapping $\phi : \mathcal{X} \rightarrow \mathcal{F}$ into a high dimensional feature space by the use of reproducing kernels. The idea of the kernel function is to enable operations to be performed in the input space $\mathcal{X}$ rather than the potentially high dimensional feature space $\mathcal{F}$.

Hence the inner product does not need to be evaluated in the feature space. This provides a way of addressing the high-dimensionality. However, the computation still critically depends upon the number of training instance and to provide a good data distribution for a high dimensional problem will generally require a large training set. As we measure the expression levels of approximately 7000 genes, the feature space that the second order polynomial would map into would have approximately 50 million elements, so it is more advantageous not to explicitly construct this map.

The following theory is based upon Reproducing Kernel Hilbert Spaces (RKHS). The basic idea here is to non-linearly map the data to a feature space of high or possibly infinite dimensions. We then apply the linear SVM algorithm in this feature space. A linear separating hyperplane in the feature space corresponds to a nonlinear surface in the original space.

So, rather than applying SVMs using the input attributes x, we can instead want to learn using some features $\phi(x)$. To do so, we merely need to replace x with $\phi(x)$ everywhere in our algorithm. Since the algorithm can be written entirely in terms of the inner products $\langle x, y \rangle$, this means that we can replace all those inner products with $\langle \phi(x), \phi(y) \rangle$. Specifically, given a feature mapping $\phi$, we define the corresponding Kernel to be

$$K(x, y) = \phi(x)^T \phi(y).$$

Then, everywhere where we previously had $\langle x, y \rangle$ in our algorithm, we could simply replace it with K(x,y), and our algorithm would now be learning using the features $\phi$. Now, given $\phi$, we could easily compute K(x,y) by finding $\phi(x)$ and $\phi(y)$ and taking their inner product. It is interesting to note that even though $\phi(x)$ itself may be very expensive to calculate, K(x,y) may be inexpensive to calculate. In such settings, by using in our algorithm an efficient way to calculate K(x,y), we can get SVMs to learn in the high dimensional feature space space given by $\phi$, but without ever having to explicitly find or represent vectors $\phi(x)$.

If K is a valid kernel, i.e., if it corresponds to some feature mapping $\phi$, then the corresponding Kernel matrix $K \in \mathbb{R}^{mxm}$ is *symmetric positive semi-definite*. This is, actually, not only a necessary, but also a sufficient condition for K to be a valid kernel (also called a Mercer kernel). [Ng]

**Theorem (Mercer).** Let K : $\mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$ be given. Then for K to be a valid (Mercer) kernel, it is necessary and sufficient that for any {x(1), ..., x(m)}, (m < $\infty$), the corresponding kernel matrix is symmetric positive semi-definite.

The most widely used Kernel functions, that satisfy Mercer's conditions, are given and described below:

- **Linear Kernel**

  The dot-product is called the kernel and can be re-written as:

  $$K(x, x_i) = x \cdot x_i$$

  The kernel defines the similarity or a distance measure between new data and the support vectors. The dot product is the similarity measure used for linear SVM or a linear kernel because the distance is a linear combination of the inputs.

- **Polynomial Kernel** Instead of the dot-product, we can use a polynomial kernel

  $$K(x, x_i) = (1 + x \cdot x_i)^d,$$

  where the degree of the polynomial must be specified by hand to the learning algorithm. When d=1, polynomial kernel is the same as the linear one. The polynomial kernel allows curved lines in the input space.

- **Gaussian Kernel** Finally, we can also have a more complex Gaussian kernel

  $$K(x, xi) = e^{-\frac{||x-x_i||^2}{2\sigma^2}},$$

  where sigma is a parameter that must be specified to the learning algorithm. Sigma is often 0 < sigma < 1 and a good default value for it is 0.1. The Gaussian (radial) kernel is very local and can create complex regions within the feature space, like closed polygons in two-dimensional space.

There are other types of Kernel functions (e.g. logistic kernel, sigmoid kernel) as well, that are used depending on the problem.

## 3.2 The Algorithm

The usage of SVM suggests the following steps. [Muk02]

1. **Setting up and normalizing the datasets**[2]: the training set used for this project consists of two sets of multidimensional vectors that are labeled either +1 (ALL) or -1 (AML). Generally speaking, it is a good practice to normalize all entries of the microarray.

```
training = xlsread('training_set_ALL_AML.xlsx');
test = xlsread('test_set_ALL_AML.xlsx');
training_labels = xlsread('training_labels.xlsx');
test_labels = xlsread('test_labels.xlsx');

training = training(:,3:40); % taking only number values
test = test(:,3:37);

training = manorm(transpose(training)); % microarray normalization
test = manorm(transpose(test));
```

Listing 3.1: MATLAB code for Data Loading and Normalization

2. **The choice of the tolerance parameter C**: if the data is properly normalized, the parameter C will not affect the algorithm much. Usually, it is set to be somewhere between 1-100.

```
function tol = svmtol(C)
    if C==Inf
        tol = 1e-5;
    else
        tol = C*1e-6;
    end
end
```

Listing 3.2: MATLAB function for tolerance for Support Vector Detection

3. **The choice of the kernel**: a linear kernel is enough for microarray applications, although if there is a need of examination of the correlations between genes, polynomial kernels may come to help out. In case if the linear kernel gives a bad performance, Gaussian kernel can be used instead.

```
function k = svmkernel(kernel,xi,xj)
    global p;
    switch lower(num2str(kernel))
    case 'linear'
        k = xi*xj';
    case 'poly'
        k = (xi*xj'+1)^p;
    case 'gauss'
        k = exp(-(xi-xj)*(xi-xj)'/(2*p^2));
```

[2] The dataset that we used was retrieved from Broad Institute Portals

```
        otherwise
            k = xi*xj'; % linear kernel (if no valid kernel is specified)
        end
    end
```

Listing 3.3: MATLAB function for SVM Kernel

4. **Training the Model**: the function *svmfit* is implemented to fit the SVM Model to the training dataset. It takes as arguments the training inputs, training targets (labels), kernel function of our choice and the regularization parameter C. The optimization here is done by MATLAB built-in function *quadprog*.

```
function [num_sv, beta, b0] = svmfit(training_set, training_labels, kernel, C)
    % checking if the # of arguments is correct
    if (nargin<2 || nargin>4)
        help svmfit
    else
        fprintf('Classification using SVM\n')
        n = size(training_set,1);

    if (nargin<4)
        C=Inf;
    end

    if (nargin<3)
        kernel='linear';
    end

    % Constructing the kernel matrix
    H = zeros(n,n);
    for i=1:n
        for j=1:n
            H(i,j) = training_labels(i)*training_labels(j)*svmkernel(kernel,training_set(i,:),training_set(j,:));
        end
    end
    c = -ones(n,1);

    % Add a very small number in order to avoid problems when Hessian is badly conditioned.
    H = H+1e-10*eye(size(H));

    % Setting up the parameters for the Optimization problem
    lbound = zeros(n,1);  % betas >= 0
    ubound = C*ones(n,1); % betas <= C

    % Setting up the constraint Ax = b
    A = training_labels';
    b = 0;

    % Solving the Optimization Objective via Quadratic Programming
    [beta, lambda, exitflag] = quadprog(H,c,[],[],A,b,lbound,ubound);
    fprintf('Status : %s\n', exitflag);

```

```
        w_length2 = beta'*H*beta;

        fprintf('|w0|^2 : %f\n',w_length2);
        fprintf('Margin : %f\n',2/sqrt(w_length2));
        fprintf('Sum beta : %f\n',sum(beta));

       % Computing the # of support vectors
        epsilon = svmtol(beta);
        svi = find(beta > epsilon);
       num_sv = length(svi);

        fprintf('Support Vectors : %d (%3.1f%%)\n',num_sv,100*num_sv/n);

        svii = find(beta > epsilon & beta < (C - epsilon));

        if length(svii) > 0
            b0 = (1/length(svii))*sum(training_labels(svii) - H(svii,svi)
    *beta(svi).*training_labels(svii));
        else

            fprintf('There are no SVs on margin. Cannot compute b0.\n');
        end
        end
    end
```

Listing 3.4: MATLAB function for SVM Train

5. **Prediction**: the test data is aimed to evaluate the performance of the SVM classifier, i.e. to know how efficient the model works on dataset that has not been seen earlier. For this reason, a function *svmpredict* is implemented. It takes as arguments training inputs, training targets, test inputs, kernel function, Lagrange Multipliers and bias. Lagrange Multipliers and the bias have already been obtained in the fitting step.

```
function predictions = svmpredict(training_set,training_label,
    test_set,kernel,beta,b0)

    n = size(training_set,1);
    m = size(test_set,1);
    H = zeros(m,n);

    for i=1:m
        for j=1:n
            H(i,j) = training_label(j)*svmkernel(kernel,test_set(i,:),
    training_set(j,:));
        end
    end

    predictions = sign(H*beta + b0); % Maximum (Hard) Margin

    end
```

Listing 3.5: MATLAB function for Computing Model Accuracy

5. **Computing the Accuracy**: To know how efficient the model works on test set, a simple *accuracy* function is implemented, which compares the 35 actual labels of test set to their respective predictions.

```
function acc = accuracy(test_labels,predictions)
acc=0;
n=length(test_labels);
for i=1:n
    if test_labels(i)==predictions(i) % correct predictions
       acc=acc+1;
    end
end
acc=acc/n*100;
end
```

Listing 3.6: MATLAB function for Computing Model Accuracy

# Chapter 4

# The Results and Conclusion

## 4.1 Results

Summing the whole algorithm up, we firstly loaded all datasets and their respective labels. After that, we did a little preprocessing on them using MATLAB Bioinformatics Toolbox function *manorm*, which is specifically used for microarray data normalization.

We trained the SVM on the training dataset with a *linear* kernel and chose the *parameter C* to be 15. The results are as follows:

$$Accuracy = \frac{TN + TP}{All} = \frac{25}{35} \approx 71.5\%$$

$$Precision = \frac{TP}{TP + FP} = \frac{17}{23} \approx 74\%$$

$$Sensitivity = \frac{TP}{TP + FN} = \frac{17}{21} \approx 80\%$$

$$Specificity = \frac{TN}{TN + FP} = \frac{8}{14} \approx 57\%$$

See Figure 4.1 for the corresponding Confusion Matrix and Table 4.1 for the actual labels of the test data with the corresponding predictions aligned in front of each.

| Actual | Predicted |
|---|---|
| ALL | ALL |
| ALL | ALL |
| ALL | ALL |
| ALL | ALL |
| ALL | ALL |
| ALL | ALL |
| ALL | ALL |
| ALL | ALL |
| ALL | ALL |
| ALL | ALL |
| ALL | AML |
| ALL | AML |
| ALL | ALL |
| ALL | ALL |
| ALL | ALL |
| ALL | AML |
| ALL | ALL |
| ALL | ALL |
| ALL | ALL |
| ALL | AML |
| ALL | ALL |
| AML | ALL |
| AML | ALL |
| AML | ALL |
| AML | AML |
| AML | ALL |
| AML | ALL |
| AML | AML |
| AML | AML |
| AML | AML |
| AML | AML |
| AML | AML |
| AML | ALL |
| AML | AML |
| AML | AML |

Table 4.1: Actual and Predicted Values with 71.5% accuracy.

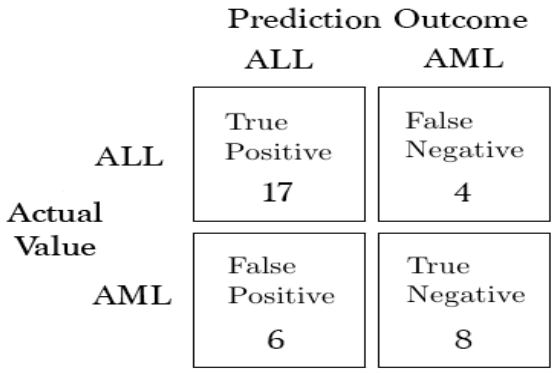


Figure 4.1: Confusion Matrix for the trained SVM.

## 4.2 Conclusion

The method we used for cancer type classification was one of the most efficient binary classifiers of modern Machine Learning. Although we did not get the best possible result, the algorithm can be further improved and a more satisfying result can be achieved.

## 4.3 Further Work That Can Be Done

As already mentioned, the Support Vector Machine is one of the best Machine Learning techniques for tackling problems with very high dimensions. However, it is not the unique one, thus we also suggest another, an indirect solution to the problem. Firstly, the data can be reduced using one of the most efficient dimensionality reduction techniques, Principal Component Analysis (PCA). There are several ways to conduct PCA such as Singular Value Decomposition, Eigendecomposition (Spectral Decomposition), etc. After that, in general, the traditional optimization problems will work very well. The constrained optimization problem can be brought to an unconstrained one with the help of Penalty Method and, then, a Gradient (Steepest) Descent Method can find the minimum of the function. [VV02]

# Bibliography


[TG99] P. Tamayo C. Huard M. Gaasenbeek J.P. Mesirov H. Coller M.L. Loh J.R. Downing M.A. Caligiuri C.D. Bloomfield E.S. Lander T.R. Golub D.K. Slonim. "Molecular Classification of Cancer: Class Discovery and Class Prediction by Gene Expression Monitoring". In: *Science* (1999).

[Muk02] S. Mukherjee. "Classifying Microarray Data Using Support Vector Machines". In: (2002).

[VV02] J. Weston S. Barnhill V. Vapnik I. Guyon. "Gene Selection for Cancer Classification using Support Vector Machines". In: *Kluwer Academic Publishers* (2002).

[FC05] L. Wang F. Chu. "Applications of Support Vector Machines to Cancer Classification with Microarray Data". In: *International Journal of Neural Systems* (2005).

[NS05] N.K. Abdel Moniem N.H. Sweilama A.A. Tharwatb. "Support vector machine for diagnosis cancer disease: A comparative study". In: *Egyptian Informatics Journal* (2005).

[Ham09] L.H. Hamel. *Knowledge Discovery with Support Vector Machines*. 2009.

[Wik16a] Wikipedia. *DNA microarray — Wikipedia, The Free Encyclopedia*. 2016.

[Wik16b] Wikipedia. *Quadratic programming — Wikipedia, The Free Encyclopedia*. 2016.

[Jaa] T. Jaakkola. *Course materials for 6.867 Machine Learning*.

[Men] A.K. Menon. *Large-Scale Support Vector Machines: Algorithms and Theory*.

[Ng] Andrew Ng. *Support Vector Machines (CS229 Lecture notes, Part V)*.