Using a Support Vector Machine to Analyze a DNA Microarray*

R. Morelli
Department of Computer Science, Trinity College
Hartford, CT 06106, USA
ralph.morelli@trincoll.edu

January 15, 2008

1 Introduction

A *DNA microarray* is a small silicon chip that is covered with thousands of spots of DNA of known sequence. Biologists use microarrays to study gene expression patterns in a wide range of medical and scientific applications. Analysis of microarrays has led to discovery of genomic patterns that distinguish cancerous from non-cancerous cells. Other researchers have used microarrays to identify the genes in the brains of fish that are associated with certain kinds of mating behavior.

Microarray data sets are very large and can only be analyzed with the help of computers. A *support vector machine* (SVM) is a powerful machine learning technique that is used in a variety of *data mining* applications, including the analysis of DNA microarrays.

2 Project Overview

In this project we will learn how to use an SVM to recognize patterns in microarray data. Using an open source software package named *libsvm* and downloaded data from a cancer research project, we will to train an SVM to distinguish between gene expression patterns that are associated with two different forms of leukemia.

Although this project will focus on SVMs and machine learning techniques, we will cover basic concepts in biology and genetics that underlie DNA analysis. We will gain experience using SVM software and emerge from this project with an improved understanding of how machine learning can be used to recognize important patterns in vast amounts of data.

The specific objectives of the project are to learn:

- The basic concepts and techniques of supervised machine learning.
- Some of the issues involved in the implementation of a learning system.
- The vector space model for representing microarray (and other) data.

*This module is based on work that was done in collaboration with Saroj Aryal, Trinity class of 2010 and Jean-Pierre Haeberly, Ph.D., Trinity Director of Academic Computing. The work was supported in part from a Trinity College Howard Hughes Medical Institute grant.

- How to design a simple learning machine experiment using our own data set.
- To appreciate some of the challenges involved in machine learning in general and microarray analysis in particular.

Because microarray data are so esoteric, especially for those without training in genetics, some of these objectives will be addressed through the analysis of a large set of baseball statistics.

3 Support Vector Machines

In this section, we provide a brief summary of support vector machines and how they are used to analyze microarray data. Our summary is based on the very nice primer by Nobel[8].

A Support Vector Machine (SVM) is a machine learning tool that uses supervised learning to classify data into two or more classes. As shown in Figure 1, given a set containing two classes of objects (represented by the red and green dots), it is possible to create a boundary that separates the two classes. An SVM can be trained to find the *decision boundary*, as it is called.

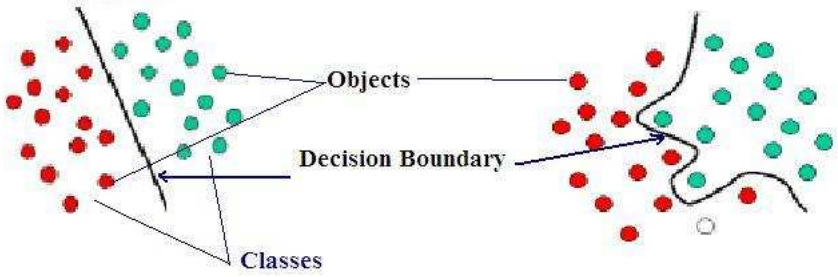

Figure 1: An SVM finds the boundary between two classes of objects.

SVMs are known as *linear classifiers* because they can always find a linear boundary between two classes of objects, where a *linear boundary* is a straight line in two-dimensions, a plane in three dimensions, and, more generally, a *n-dimensional hyperplane* in a $n+1$ dimensional space.

3.1 Kernel Functions

Often, in order to do find a suitable boundary between two classes, the SVM has to *map* the data from the *input space* into a *higher-dimensional feature space*, as shown in Figure 2. Mathematically, it is known that the input data can always be mapped into a higher-dimensional space where the classes can then be separated by a hyperplane. The function that performs this mapping is called a *kernel function*. Software implementations of SVMs usually provide several choices of built-in kernel functions. The choice of a kernel function and its parameter settings are important elements of designing an SVM experiment.

To illustrate the concept behind the kernel mapping, consider graph i in Figure 3. In this example, the input data (points on a line) cannot be separated into two classes by picking a boundary point on the line.[1] However, as shown in graph j, these data can be mapped into a 2-dimensional space by squaring each point ($y = x^2$). This gives us a parabola in 2-dimensions,

[1] In a 1-dimensional space, a single point would be a linear separator.

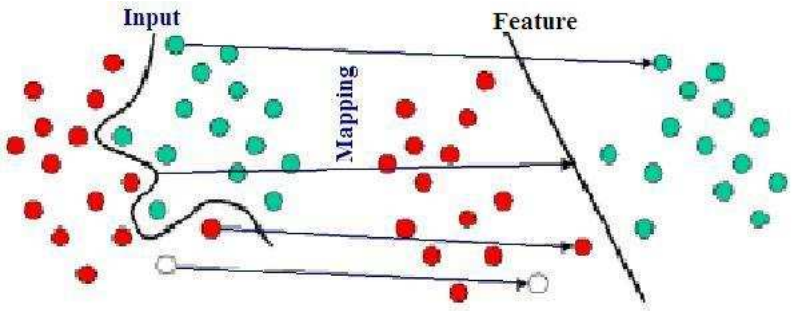

Figure 2: Mapping data from the input space to the feature space where it is linearly separable.

with the result that the two classes can be separated by a straight line (shown in black). Thus, by mapping the input data into a higher dimensional space, we are able to identify a linear separator for the two target classes.

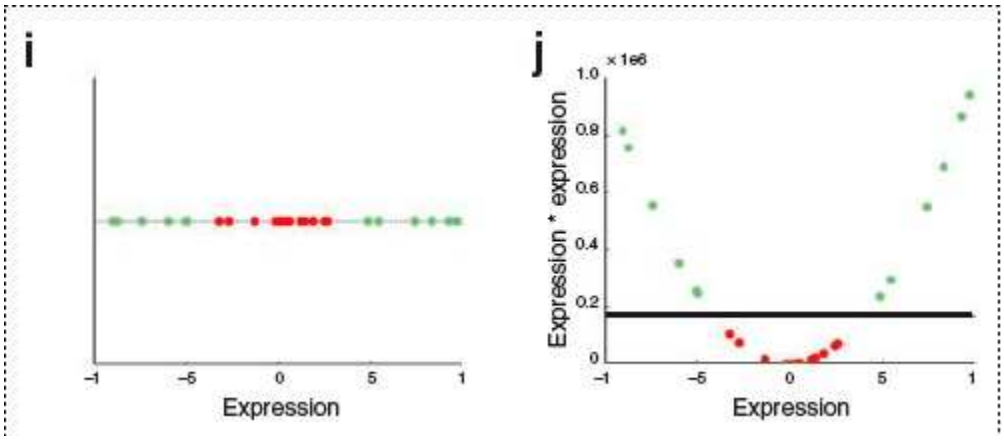

Figure 3: Mapping data from 1-dimension into 2-dimension by applying the function $y = x^2$. *Source: [8]*

3.2 Maximum-margin Hyperplane

Another important characteristic of an SVM classifier is that given any set of data it finds the hyperplane that *maximizes* the separation between the two (or more) classes. This is illustrated in Figure 4, where on the left we see that there are many hyperplanes that separate the data but only one that maximizes the separation. Finding the maximum-margin hyperplane (MMH) results in a classifier that is better able to make correct predictions when given new data.

Without going into the mathematical details, the identification of the MMH is an optimization problem that can be solved by a type of mathematical optimization known as *quadratic programming*.

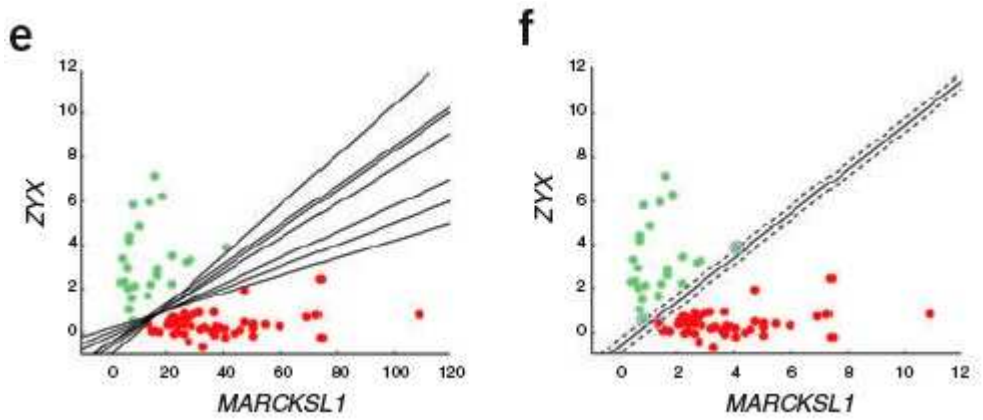

Figure 4: The SVM finds the hyperplane that maximizes the separation between the two classes.*Source: [8]*

3.3 Data Representation

In order to be processed by an SVM, input data must be represented as *labeled feature vectors*. To take an example, suppose we have a set of DNA microarrays prepared from biopsy samples of cancer patients, one microarray per biopsy sample. Each microarray contains thousands of spots, each of which represents the degree to which a certain gene is expressed in that biopsy sample. The goal is to distinguish one type of cancer from another type.

The *features* in this case are the thousands of genes and for each feature, we would have a real number value representing the gene's expression level. The *labels* in this case would be real numbers {+1.0, -1.0} that represent one or the other form of cancer. If we number the genes from 1 to n, then the following would be a snapshot of a portion of five feature vectors:

```
-1.0 1:0.76 2:0.672 3:0.230 ...
+1.0 1:0.97 2:0.735 3:0.279 ...
-1.0 1:0.84 2:0.592 3:0.255 ...
-1.0 1:0.90 2:0.772 3:0.270 ...
+1.0 1:0.94 2:0.770 3:0.274 ..
```

In this case we have three samples of one type of cancer and two samples of the other.

Note that feature vector representation is very general and can be used with virtually any kind of data. Thus, if we were analyzing baseball statistics, the *features* would be specific statistical categories (such as batting average). The key point here is that an SVM can analyze any data set provided it can be represented in the form of labeled feature vectors.

3.4 Training

Figure 5 illustrates how data are handled in an SVM microarray experiment. First, the raw data are extracted from a microarray image file. As you will learn when you do the microarray tutorial in Section 4.1, phosphorescent dyes are used to represent gene expression levels on the microarray. Geneticists are able to perform a variety of analyses just using the images themselves.

To perform SVM analysis, the image data must be converted into labeled feature vectors, which will be used to train and test an SVM model. As shown in Figure 5, in a machine learning experiment it is customary to divide the data into two portions, a *training set* and a *testing set*. The training data are used to train the SVM, the result of which is a *model*. The model can then be tested on the testing data. As you might expect, it is important that the data that make up the training and testing sets be representative of the overall data that you are analyzing. Therefore, care must be taken when creating the two sets to avoid biasing the data in some way and thereby invalidating the results of the experiment.

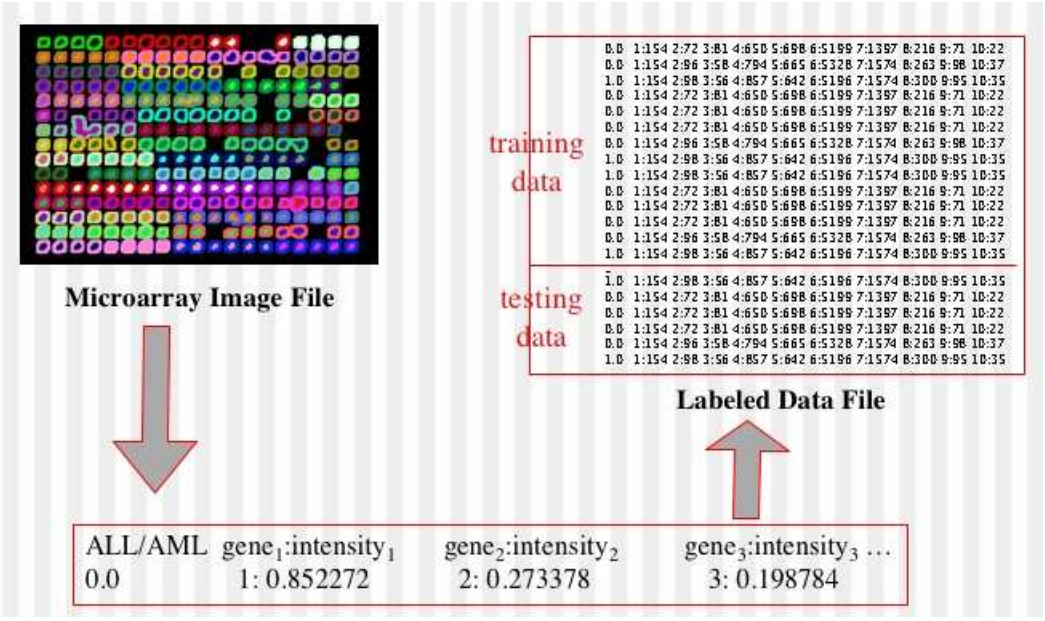

Figure 5: The labeled data are divided into two sets, *training* and *testing* data sets.

4 Project Details

This section provides details on all four phases of the project. Phases 1 and 2 are tutorial in nature. In Phase 1 we will work through an online, interactive tutorial on microarrays. The tutorial is self-contained and provides numerous questions and answers to help you verify that you are understanding the key concepts.

Phase 2 provides an overview of the *libsvm* software library. It explains and illustrates how the various commands are used in an SVM experiment. Phases 1 and 2 may be done in either order. However, it is important that both phases be completed before beginning Phase 3.

In Phase 3 we conduct an SVM experiment using cancer data taken from leukemia patients. The approach here is to compare the results of our analysis with those obtained in prior studies involving these data.

In Phase 4, we conduct a more open-ended SVM experiment using baseball statistics, with the goal of trying to discover a model that serves as a successful classifier for a winning team. By *successful* here, we mean that the model should perform significantly better than simple guessing.

4.1 Phase 1: Microarray Tutorial

In order to gain an introduction to DNA microarrays we will work through the award winning multimedia tutorial, the *Microarray Mediabook*[1]. Created by Davidson biology professor Malcolm Campbell, in collaboration with educators and University of North Carolina at Chapel Hill, the tutorial illustrates how microarrays are created and analyzed. The tutorial is broken into several section and each section has questions that help students test their understanding of the material. It comes with its own set of built-in tools, such as a calculator and glossary and presents a completely self-contained introduction to microarrays.

The tutorial contains four sections:

- The *Method* section teaches the basics of DNA microarrays, by working through an experiment involving yeast cells.
- In the *Exploration* section you will design and perform your own DNA microarray experiment to investigate changes in gene expression in mice with asthma-like symptoms.
- In the *Data Interpretation* you will learn about quantitative analysis of whole-genome DNA microarrays.

The tutorial is available at: `http://www.csie.ntu.edu.tw/~cjlin/libsvm/`.
Work through all sections of the tutorial and complete the self-evaluation quiz at the end of each section. For each quiz in the tutorial, write down the question and the correct answer (and the explanation, if you find it helpful).

4.1.1 Phase 1 Deliverable

The deliverable for Phase 1 is a word-processed document containing the questions and correct answers to each of the quizzes in the three sections of the tutorial.

4.2 Phase 2: Using LIBSVM

For this project we will be using *libsvm*, an open source SVM package[2]. It contains all the software tools needed to carry out the project. It also contains several data sets that we can use. Before beginning this section, you should review the libsvm guide ([6]). The LIBSVM package supports several basic kernel functions, but we will make use of only two, the *linear* and the *radial basis function* (RBF):

- linear: $K(\vec{x_i}, \vec{x_j}) = x_i^T \cdot x_j$.
- radial basis function (RBF): $K(\vec{x_i}, \vec{x_j}) = exp(-\gamma||\vec{x_i} - \vec{x_j}||^2), \gamma > 0$.

where $\vec{x_i}$ and $\vec{x_j}$ are vectors and γ is a kernel parameter. The linear kernel is for data that can be linearly separated in the input space. The RBF kernel is for input that cannot be linearly separated and must be mapped to a higher dimensional feature space.

The procedure we will use in our SVM experiments will go roughly as follows:

- Transform the data into labeled feature vectors and separate it into training and testing data sets.

- Scale the data.
- Use the RBF kernel, $K(\vec{x_i}, \vec{x_j}) = e^{-\gamma||\vec{x_i}-\vec{x_j}||^2}$
- Use cross-validation to find the best values for the C and γ parameters.
- Train the model on the training data.
- Test the model on the test data.

In the following sections, we will learn how to perform each of these steps.

4.2.1 Installing LIBSVM

Download and install LIBSVM by following the instructions at: `http://www.csie.ntu.edu.tw/~cjlin/libsvm/`. Note that to use LIBSVM you must have Java and/or Python and GnuPlot installed on your system. GnuPlot should be installed in the path specified in the grid.py and easy.py files. See Appendix A.3 for additional details on downloading and installing LIBSVM.

4.2.2 LIBSVM Commands

LIBSVM is a command-line tool. In this section we describe the commands that we will be using in this project. Recall that in a typical machine learning experiment a file of labeled feature vectors is divided into two parts, a *training set* and a *test set*. For the examples in this section, we will use files containing data from an astroparticle experiment. Each feature vector contains four features. The training set, named `svmguide1`, consists of 3089 cases and the test set, named `svmguide1.t` consists of 4000 cases.

The `svm-train` and `svm-predict` commands are used from the command line to create a training model from a file containing labeled feature vectors. The trained model is then used to make predictions on a file of labeled test data:

```
svm-train [options] trainingfile [model_file]
svm-predict [options] test_file model_file output_file
```

The bracketed options list are various options, such as kernel type (default RBF) and various parameter settings. To see a complete list of the options, type the command with no arguments at the command line. In the following examples we use default the default kernel and settings.

The `svm-train` command takes an input file of training data and builds a *model*, which is stored in a file named *X.model*, where *X* is the name of the training file. The `svm-predict` command takes as its input the test data file, the model file, and creates an output file that contains the results of its predictions, one label for each test case. Here is an example that uses the astroparticle training file named `svmguide1`:

```
$ svm-train svmguide1
  .....*
  optimization finished, #iter = 5329
  ..... ignoring output ....

 $ svm-predict svmguide1.t svmguide1.model svmguide1.out
  Accuracy = 66.925% (2677/4000) (classification)
```

As you see from the displayed output, the optimization algorithm required 5329 iterations before it converged on a solution. The model in this case had a predictive accuracy of only 66.925%, meaning that the model successfully predicted the values of 2677 out of the 4000 samples in the test data.

In this next example, we illustrate that a better model can be produced if we *scale* the training and testing data–i.e., if we convert each feature value to a number within a specified range. Scaling is a way to insure that features with values in large numeric ranges (e.g., thousands or millions) don't dominate features with values in small ranges (fractions). The general form of the `svm-scale` command, where l and u are the lower and upper values for the feature values and *y_lower* and *y_upper* are the low and high label values:

```
svm-scale [-l lower] [-u upper] [-y y_lower y_upper] [datafile] > [scaledfile]
```

In this example the `svmcale` command scales the feature data to the default range [-1.0, 1.0]:

```
$ svm-scale svmguide1 > svmguide1.scale
$ svm-scale svmguide1.t > svmguide1.t.scale
$ svm-train svmguide1.scale
*optimization finished, #iter = 496
..... ignoring output ....

$ svm-predict svmguide1.t.scale svmguide1.scale.model svmguide1.out
Accuracy = 95.6% (3824/4000) (classification)
```

With scaled data the optimization required only 496 iterations and the resulting model has a performance that is much improved, giving a classification accuracy of 95.6%.

4.2.3 Using Parameters During Training

SVM training is controlled by one or more parameters. The parameter C, sometimes called the *slack parameter*, controls how many of the training samples are allowed to be on the "wrong" side of the decision boundary. Allowing some training examples to leak through the boundary can help reduce *over fitting* the data. Over fitting may occur when the training model picks a boundary that classifies 100% of the training samples. An overfit model will not be a good classifier because it so narrowly defines the classes that it will be unable to make a good prediction for data that don't exactly match the training vectors. By avoiding over fitting, the slack parameter allows for a tradeoff between training accuracy–i.e., less than 100% training accuracy– and predictive accuracy.

In the previous examples, we used the default value for C. In this example we will search for the best (optimal) value of C as well as other parameters. In addition to the slack parameter, there are parameters associated with the various kernel functions. In this example, we use the Radial Basis Function (RBF) kernel, which requires that a second parameter, γ, be specified.

How does one know the best values to use for these parameters? The answer is that one doesn't know in advance. So the best way to find optimal values for (C, γ) is to perform a systematic search, known as a *grid search*, for the best values using LIBSVM's Python program, *grid.py*.

Imagine a two dimensional grid, with C values in a fixed range along the horizontal axis and γ values in a fixed range along the vertical axis. The idea is to sample values of (C, γ) for fixed increments of both C and γ within their respective ranges and to pick the values that give the best

training results. If there are five values for C, say 1, 2, 3, 4, and 5, and four values for γ, say 2, 4, 6, and 8, then the search would look at all 20 possible combinations of these values and pick the best combination.

4.2.4 Cross Validation

Another important training concept is the idea of *cross validation*, which is used to evaluate how well the model is likely to perform as a classifier. To cross validate our model, we choose a subset of the data to train the model and then test it on the remaining data–i.e., the data that was not used in training.

In *k-fold* cross validation, we divide the training set into k subsets and use each subset as the test set. The other $k-1$ subsets are combined and used as the training set. Training is repeated k times. In this way, we can evaluate how well the model derived from training is likely to behave as a predictor.

4.2.5 Performing a Grid Search

In this example we use the *grid.py* program to perform grid search with cross validation. The general form of the `grid.py` command is:

```
grid.py [-log2c begin,end,step] [-log2g begin,end,step] [-v fold] \
    [-svmtrain pathname] [-gnuplot pathname] [-out pathname] \
    [-png pathname] [svm-train parameters] [train filename]
```

The beginning, ending, and step values for both C and γ can be specified, along with the number of folds, v. By default, the values are {-5, 15, 2} for C and {3, -15, -2} for γ and 5-fold cross validation. Note that both C and γ values are given as log_2 values, which are easier to specify. Here's an example of using `grid.py` with the default parameters on the same data set as in the previous examples:

```
$ grid.py smvguide1.scale
[local] 5 -7 95.5973  (best c=32.0, g=0.0078125, rate=95.5973)
[local] -1 -7 85.2056  (best c=32.0, g=0.0078125, rate=95.5973)
[local] 5 -1 96.8598  (best c=32.0, g=0.5, rate=96.8598)
..... ignoring output ....
8.0 2.0 96.9246
```

The output lines show the parameter values and the training accuracy for each set of parameters. The rightmost column on each line, the parenthesized values, show the best-so-far values of C and γ (listed as g) by libsvm. Note that a $C = 32.0$ and $g = 0.5$ are the actual values of the parameters, equivalent to 2^5 and 2^{-1} respectively. The last output line shows the optimal parameter values: $C = 8.0$, $\gamma = 2.0$, and the training accuracy for those parameters (96.9246%).

Now we can use these parameters when we train the model, as follows:

```
$svm-train -c 8 -g 2 svmguide1.scale
.*
optimization finished, #iter = 1102
..... ignoring output ....
```

```
$svm-predict svmguide1.t.scale svmguide1.scale.model svmguide1.out
Accuracy = 96.075% (3843/4000) (classification)
```

In this case we get a classification accuracy (96.1%), which is slightly better than the result we got using the default parameters.

4.2.6 Phase 2 Deliverable

For Phase 2, hand in a word-processed document that gives complete answers to the following questions.

1. Download and install *libsvm*. To test your installation, Run the *svm-train* and *svm-predict* commands using default options on the `heart.scale` data set that is provided. Use the same file for both training and prediction. Explain why the prediction accuracy is not 100%.

2. Download and install the astroparticle data sets, called *svmguide1*, from `http://www.csie.ntu.edu.tw/~cjl`

   You need to download two data sets, *svmguide1* and *svmguide1.t*. (It may be necessary to save these as text files.) Perform each of the training and testing examples shown in this section and record your results. They should compare exactly with the results shown here.

3. Train a model for the scaled test data using the *linear* kernel and the default parameters (command: svm-train -t 0) and compare its classification accuracy with the results obtained using the RBF kernel. Recall that the linear kernel will find a linear boundary for the input data if they are linearly separable. What do your results suggest to you about the separability of these data sets?

4.3 Phase 3: Microarray Analysis

In this phase of the project we will analyze microarray data that have been derived from cancer tissue samples of Leukemia patients. These data have been analyzed with similar results in several classification studies[4, 3, 5]. The goal of this phase of the project is to conduct an analysis using LIBSVM and compare it with the results obtained in these previous studies.

Before beginning this phase of the project, you should work through the very nice tutorial.

Leukemia is a type of cancer that results in a significant increase in immature or dysfunctional white blood cells (leukocytes). This type of cancer starts in blood-forming tissues such as bone marrow, and causes a large number of white blood cells to be produced and enter the blood stream. This usually leads to displacement of the normal bone marrow cells with higher numbers of immature white blood cells, hence damaging the bone marrow.

Clinically, Leukemia can be categorized into two types: acute and chronic. In this experiment, we will be looking at the acute leukemias and trying to classify two types: acute lymphoblastic leukemia (ALL) acute myeloid leukemia (AML).

One method for classifying ALL and AML leukemias is to use DNA microarrays prepared from tissue samples of ALL and AML patients and to measure the relative amounts of gene expression in each. The goal is to identify sets of genes that express (or repress) in one or the other type of cancer cell. Using an SVM or some other machine learning or statistical modeling approach, a

Table 1: Summary of analyses in prior studies.

Approach	Training/Testing Details	Training Accuracy	Testing Accuracy
Weighted Voting ([4])	Fifty informative genes cast weighted votes Hold-out-one cross validation	36/38 (94.7%)	29/34 (85.3%)
Weighted Voting ([9])	Fifty gene predictor Cross validation with prediction strength	36/38 (94.7%)	29/34 (85.3%)
SVM ([3])	Hold-out-one cross validation Top ranked 25, 250, 500, 100 genes Linear kernel plus diagonal factor	38/38 (100%)	30/34 - 32/34 (88% - 94%)

model is built by analyzing the microarrays and the model is then used to predict ALL or AML in unclassified samples.

The training data for this experiment consist of 38 known bone marrow samples (27 ALL and 11 AML) obtained from acute leukemia patients at the time of diagnosis. RNA prepared from bone marrow cells was hybridized (mixed together) on high-density oligonucleotide microarrays produced by Affymetrix. The microarrays contain probes for 7129 human genes. For each gene, a quantitative expression level is obtained. For ALL samples, some genes were highly induced, some were highly repressed, and others underwent no change. Similarly, for AML samples, some genes (not necessarily the same ones) were induced, some repressed, and others underwent no change. The goal of the microarray analysis is to learn the pattern of gene expression–which genes are induced, repressed, or unchanged–in association with one or the other form of leukemia and to use this information as a classifier. Once a classification model is learned, it can be tested on an independent data set consisting of 34 samples (24 ALL and 10 AML).

We will conduct an SVM experiment on the leukemia data and compare our results with three previous studies. The first two studies used what is known as a *weighted voting* technique. In this approach, *weights* based on a gene's expression level is associated with each of the 7129 genes or some subset of them. For example, in the first two studies, a set of 50 *informative* genes were identified according to their high correlation with the two forms of cancer[4, 9]. For each patient sample, each informative gene then casts a weighted vote, with the magnitude of each vote dependent on the gene's expression level. The sum of the votes is used to determine the classification. In the third study, an SVM approach was used on subsets of 25, 250, 500, and 1000 genes([3]).

Table 1 summarizes the results obtained in previous studies of the leukemia data. As you can see there, all three studies achieve good predictive accuracy (85study is how our LIBSVM performs in comparison with these results.

4.3.1 Experimental Design

Our experiment will use LIBSVM to analyze the same microarray data as in the previous studies. Our approach will differ from that taken in the the third study ([3]) in the following ways:

- We will not make any attempt to identify *informative* genes. Instead our model will be based on expression levels from all 7129 genes.
- We will use both the RBF kernel and the linear kernel, whereas the SVM in the previous study used only the linear kernel.

4.3.2 Data Representation

Download the testing and training data for the leukemia experiment from the following web site:

`http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary.html#leukemia`

As described in the previous section, the leukemia dataset consists of 38 bone marrow samples (27 ALL and 11 AML) obtained from acute leukemia patients at the time of diagnosis. The data are from 38 Affymetrix microarrays, each containing probes for 7129 human genes. For each gene, a quantitative expression level was obtained. Similarly, the test data set consists of 34 independent microarrays (20 ALL and 14 AML).

Both data sets have been labeled, with ALL samples labeled 1.0 and AML labeled -1.0. However, in the raw data sets the gene expression values have not been scaled.

4.3.3 Experimental Procedure

Using LIBSVM software, design and conduct a number of experiments with the leukemia data to see how your results compare with results obtained in previous studies. Your experiments should answer the following questions:

- Are better results achieved for predictive accuracy with scaled or unscaled data? What are the exact quantitative results obtained in each case?
- Using RBF kernel, what are optimal parameter settings for C and γ, as recommended by a grid search? How do the results obtained with the optimal parameters compare to the results obtained with the default parameters?
- Using the linear kernel, how do the results obtained with the optimal value for the parameter C compare with those obtained using the default value?
- Doing everything you can to optimize your results, what are the best results you can obtain using LIBSVM on the leukemia data sets?

4.3.4 Phase 3 Deliverable

For Phase 3, hand in a word-processed document giving a complete description of the experiments you performed and the results you obtained. The results should be summarized in a table, similar to Table 1. Your document should include a brief "Discussion" section in which you discuss how your results differ from those in the previous studies and possible explanations for why.

Your document should have a "Procedures" section in which you record the exact LIBSVM commands you used for each step in your procedure so that the experiments can be repeated.

4.4 Phase 4 – The Baseball Study

One disadvantage of working with microarray data–at least for non-biologists–is that it is difficult to get a strong sense of how well our models are performing because the data are so esoteric. In this section we will perform SVM analysis of baseball statistics, with the goal of trying to develop a model that will successfully distinguish winning teams from losing teams.

Table 2: Summary of baseball statistics.

G - number of games	W - wins	L - losses
PCT - winning percentage	GB - games behind	R - runs scored
OR - opponents runs	AB - official at bats	H - hits
2B - doubles	3B - triples	HR - home runs
BB - walks	SO - strike outs	AVG -batting average
OBP - on base percentage	SLG - slugging percentage	SB - number of stolen bases
CS - caught stealing	ERA - earned run average	CG -complete games
SHO - shut outs	SV - saves	IP - innings pitched
H-P - hits per pitcher	HR-P	BB-P
SO-P	BPF - batters park factor	PPF - pitchers park factor

The data for this phase of the project come from [7]. The original data set contains final standings and team statistics for each team for baseball seasons from 1871 through 2000. Seasons that contained missing values or features were removed from the data set. Thus, we have edited the original file to eliminate the years between 1871-1920. This reduces the total number of instances to 1592.

In its edited form each entry contains 30 different features (attributes) for each team for each season. Table 2 gives a summary of some of the attributes.

4.4.1 Data Representation

The data set is available in spreadsheet format. To convert it to SVM data format, the attributes must be numbered and class labels must assigned to each entry. Because we are interested in separating winners and losers, the teams that finished in first place for a particular season in a particular league (National League, American League, Eastern Division American League, and so on) should be assigned the label 1.0 and the rest of the teams should be assigned 0.0.

The features in Table 2 must be numbered from 1 through 30 in the given order. The following are three examples of SVM-formatted data:

```
0.0 1:154 2:72 3:81 4:650 5:698 6:5199 7:1397 8:216 9:71 10:22 11:533
12:429 13:0.26 14:0.34 15:0.35 16:98 17:111 18:3.82 19:92 20:11 21:6
22:1395 23:1481 24:39 25:461 26:481 27:96 28:96

0.0 1:154 2:96 3:58 4:794 5:665 6:5328 7:1574 8:263 9:98 10:37 11:471
12:353 13:0.29 14:0.35 15:0.4 16:112 17:96 18:3.59 19:109 20:9 21:10
22:1386 23:1467 24:45 25:405 26:438 27:100 28:99

1.0 1:154 2:98 3:56 4:857 5:642 6:5196 7:1574 8:300 9:95 10:35 11:576
12:379 13:0.3 14:0.37 15:0.41 16:73 17:92 18:3.41 19:94 20:11 21:7
22:1377 23:1448 24:31 25:401 26:466 27:103 28:100
```

4.4.2 Training and Testing Data

As noted above, the modified data set contains 1592 entries. Note that among these 1592 entries, there approximately 85% of the entries are losers and 15% are winners.

These entries must be separated into training and testing data sets. In performing this step, you should take care to use a random procedure to separate the data into two sets. It would also be useful to separate the two sets so that each set has approximately 15% winners.

4.4.3 Control Data Set

In order to assist with the analysis, it will be useful to create a data set consisting of a random assignment of labels to the feature vectors. For example, one way to do this might be to rearrange the 1592 labels in some random fashion while leaving the feature vectors unchanged.

4.4.4 Experiments

Using LIBSVM software, design and conduct a number of experiments with the baseball data with the goal of trying to build a classifier that can predict winners in the test data set.

Your experiments should answer the following questions:

- Which form of data (scaled or unscaled), which kernel function (RBF or linear), and which parameter settings produced the best model?
- It is obvious that certain features, e.g., *games behind*, should be strongly predictive of winning teams. That is, teams that are 0 games behind are always the winners. Devise one or more models that confirm that the SVM will identify this feature as strongly predictive of winning teams. Are there other features that are strongly predictive?
- Design a data set and an experiment that causes the SVM to perform no better than random guessing. For example, a *random* model would perform no better than flipping a coin to predict the result of each entry in the test data set.
- Design one or more experiments to confirm (or reject) the hypothesis that the SVM can identify a model that performs better than a random model when trivially predictive features (such as games behind) are removed from the data set.
- Try to identify a model that uses a small number of *informative* features to make better than random predictions.

4.4.5 Phase 4 Deliverable

For Phase 4, hand in a word-processed document giving a complete description of the experiments you performed and the results you obtained. The results should be summarized in an easy-to-understand table Your document should include a brief "Discussion" section in which you discuss the significance of your results.

Your document should have a "Procedures" section in which you describe the procedures you used to obtain your results.

References

[1] M. Campbell. Micrarrays mediabook. Website, 2006. http://gcat.davidson.edu/Pirelli/index.htm.

[2] C.-C. Chang and C.-J. Lin. *LIBSVM: A library for support vector machines*, 2001. Software available at http://www.csie.ntu.edu.tw/c̃jlin/libsvm.

[3] T. S. Furey, N. Cristianini, N. Duffy, D. W. Bednarski, M. Schummer, and D. Haussler. Support vector machine classification and validation of cancer tissue samples using microarray expression data. *Bioinformatics*, 16(10):906–914, 2000.

[4] T. R. Golub, D. K. Slonim, P. Tamayo, C. Huard, M. Gaasenbeek, J. P. Mesirov, H. Collier, M. L. Loh, J. R. Downing, M. A. Caligiuri, C. D. Bloomfield, and E. S. Lander. Molecular classification of cancer: Class discovery and class prediction by gene expression monitoring. *Science*, 286(5439):531–537, 1999.

[5] I. Guyon, J. Weston, S. Barnhill, and V. Vapnik. Gene selection for cancer classification using support vector machines. *Machine Learning*, 46(1-3):389–422, 2000.

[6] C.-W. Hsu, C.-C. Chang, and C.-J. Lin. A practical guide to support vector classification. Website, July 2007. http://www.csie.ntu.edu.tw/ cjlin/papers/guide/guide.pdf.

[7] S. Lahman. Sean lahman's baseball archive. Website, 2007. http://baseball1.com/content/view/57/82/.

[8] W. S. Nobel. What is a support vector machine. *Nature Biotechnology*, 24(12), December 2006.

[9] D. K. Slonim, P. Tamayo, J. P. Mesirov, T. R. Golub, and E. S. Lander. Class prediction and discover using gene expression data. In *Proceedings of the 4th Annual International Conference on Computational Molecular Biology (RECOMB)*, pages 263–272. RECOMB, 2000.

APPENDICES

A Downloading and Installing LIBSVM

This appendix gives addition guidance on downloading and installing LIBSVM. LIBSVM may be downloaded from the authors' web site at:

`http://www.csie.ntu.edu.tw/~cjlin/libsvm/`

The current release (Version 2.84, April 2007) of LIBSVM can be obtained by downloading the zip file or tar.gz file.

The package includes the source code of the library in C++ and Java, and a simple program for scaling training data. A README file with detailed explanation is provided.

For MS Windows users, there is a subdirectory in the zip file containing binary executable files. A pre-compiled Java class archive is also included. For Unix or Linux users, it will be necessary to compile the *c*-source code using the *make* command. Complete instructions on how to do this are provided in the *README* file contained in the main *libsvm* directory.

A.1 Installation on Windows XP

1. Download LIBSVM from `http://www.csie.ntu.edu/~cjlin/libsvm`.
2. Download Python from `http://www.python.org`.
3. Install LIBSVM to the default directory, usually `C:\libsvm\libsvm-2.84`.
4. Within the default directory create a folder named *Files* or something you can easily recognize as a place to store the various data sets you will download.
5. Install Python to the default directory, usually `C:\Python25`.
6. To run the executables you find in the windows subfolder of the LIBSVM directory you must point to the path of the file, so your compiler knows where to locate it. To do this open the control panel, select system, advanced environment variables, then the system variables sub-box, scroll down until you find the *Path* variable and press edit. A box should open with a string of paths separated by delimiters. At the very start of the string you should add the following line:

   ```
   C:\libsvm\libsvm-2.84\windows;C:\libsvm\libsvm-2.84\tools;C:\python25;C:\python25\scripts
   ```

   This will tell the compiler where to find all executable files you will need to run while using LIBSVM.
7. It should be noted that on a windows machine the commands to run the executables omit the "-" shown in the examples in 4.2.2. Thus, *svm-train trainingfile* becomes *svmtrain training file*. You can look within the windows subdirectory of the libsvm directory to see the names of the files. Additionally, it should not be necessary to call *phython grid.py filename*. Just use *grid.py filename*.

A.2 Installation on Mac OS

1. Download LIBSVM from `http://www.csie.ntu.edu/~cjlin/libsvm`.
2. Extract the folder to a directory of your choosing.
3. Download and install python `http://www.python.org`.
4. Use the terminal to go the the `libsvm` directory and execute the *make* command to build svm-train and svm-predict.
5. To run either program type `./svm-train` or `./svm-predict`.
6. To use `grid.py` and `easy.py`, use the terminal to go to the `tools` directory and type `python` and the name of the program you want to run.

A.3 Using grid.py and easy.py

In order to use the *grid.py* and *easy.py* programs you must also install Python and GNUPlot. Python is an interpreted programming language that runs on most platforms. GNUPlot is a graphing tool that is used by both of these programs. Links to both of these packages are provided on the LIBSVM download page.

An alternative to downloading GNUPlot would be to modify the *grid.py* and *easy.py* source code to remove the plotting feature from these programs. Of course, the downside of this is that you will be unable to see the *parameter space* during the grid search.

grid.py will attempt to locate GNUPlot on your system, using the pathname that is supplied in the program and will generate an error message if it cannot find it or if it cannot start the program. To disable these error messages, comment out the following lines (which are line 93 and 95 on my version):

```
#    assert os.path.exists(gnuplot_exe),''gnuplot executable not found''
#    gnuplot = os.popen(gnuplot_exe,'w')
```

Finally, to stop *grid.py* from attempting to draw the graph, it is necessary to comment out the following lines, which are lines 347-348 of my current version:

```
    #        redraw(db)
    #        redraw(db,1)
```

This will suppress drawing of the graph while the program is running.

Once these changes are made to *grid.py*, then To use the *easy.py* tool without generating error messages, it is only necessary to comment out the *assert* statement on line 37:

```
#    assert os.path.exists(gnuplot_exe),''gnuplot executable not found''
```