Oncogene Classification Analysis and Prediction of Gene Expression Data using a Support Vector Machine

Justin Hess
Northeastern University, Boston, MA
Email: hess.ju@husky.neu.edu

Abstract

The aim of this project was to accomplish classification analysis of large biological data sets that involve oncogene expression levels of cancer types by implementing a Support Vector Machine. A number of features were selected to represent the various genes from a data set that contributed to protein expression levels in a set of patients, and these feature vectors were run through the Support Vector Machine using both linear and non-linear kernels. Methods such as Principal Component Analysis were used to significantly reduce the dimensions of a large input data set from many different patients and their respective gene expression levels for many different genes that could potentially be labeled as oncogenes to drive cancer. These datasets also entailed labels of two different cancer types, which aided classification design into different cancer types, or class labels. The Support Vector Machine was able to more properly classify the data using a non-linear, gaussian kernel with a training function that utilized Sequential Minimal Optimization, as opposed to the linear kernel which, while easier to train, could not always find a maximally separating hyperplane.

Introduction

DNA microarray-based gene expression profiling has become a widespread approach for identifying biomarkers associated with cancer. Expression levels of genes used as a quantitative measure for gene activity have become an important factor for statistical methods (Yang 2014). The uses of machine learning, and artificial intelligence have only recently begun to exploit the field of genomics to aid in the massive explosion of biological data from years of genomic profiling. Despite recent advances in machine learning, there is still massive efforts to apply more sophisticated methods to genomic data in attempts to better classify different diseases, given the ever more growing supply of data as well as the emerging markets of disease and cancer diagnosis. As the complexity and heterogeneity of cancer is being increasingly appreciated through genomic analyses, microarray-based cancer classification comprising multiple discriminatory molecular markers is an emerging trend.

Many researchers have contributed towards tackling the challenging problem of classifying different cancer types based on the expression levels of different genes from a patient. One group of researchers in the late 1990s spearheaded to develop a generic approach to cancer classification based on gene expression monitoring by DNA microarrays applied to human acute leukemias. Todd Golub and his team of researchers published their attempts of cancer class prediction and their findings in Science in 1999 (Golub et. al 1999). His researchers obtained the gene expression levels of 38 different cancer patients across 7129 genes. Each patient either had either acute lymphocytic leukemia (ALL) or acute myeloid leukemia (AML). The data from this paper was further classified in this project using a Support Vector Machine (SVM) after being preprocessed and specific genes of high importance were filtered using data dimensionality reduction.

The main challenge with large genomic datasets, in that before classifying different patients based on the weights from each tested gene, is that it would have to be determined which genes contribute to the highest variance and will require a significant amount of data cleaning and preprocessing. It would be almost intractable to use a classifier algorithm on hundreds of samples of data sub vectors, each with a size of the thousands of genes from the patient, where each patient would have a feature vector with a column for each gene. The input for classifier algorithms in these scenarios usually would be both the training and testing datasets, where the size of the vectors would be considered the number of features, in this case the number of filtered/desired genes. Classification and prediction algorithms, such as SVMs, have tremendous potential to be used in clinical applications to possibly diagnose cancer patients or to detect genes used for targeting specific drug development for

possible cancer prevention. This paper evaluates the use of SVMs on aiding the classification difficulties of disease types based on gene expression levels. All experiments use the data mentioned from the Golub paper that will be further explained in the next sections.

Background

This paper evaluates different types of classifiers in SVMs, both linear and non-linear, and their training and classification methods. Before we discuss the background behind SVMs, we will look at the dimensionality reduction techniques mentioned that were used in the significant amount of data preprocessing that was necessary before the filtered data could be input into an SVM classifier.

Principal Component Analysis

One of the first steps in analyzing a large data set with thousands of different features is to drastically reduce the dimensionality and to understand th**e** general direction of the variance among the different contributing feature values, in this case the gene expression levels. To accomplish this, Principal Component Analysis (PCA) was utilized using the Python library NumPy's linear algebra functions and the Scikit-learn library to simplify the PCA analysis. The Scikit-learn toolkit greatly expedites the process of performing dimensionality reduction without having to perform each linear algebra step-by-step procedure. These methods are commonly used for data preprocessing of large data sets to reduce their dimensionality to find the components with the highest variation. Covariance is a calculation used primarily in data reduction, while PCA analysis is a common linear algebra technique for also reducing the dimensionality of large data sets with a significant amount of noise. The goal of PCA analysis is to detect the correlation between variables which in turn identifies patterns in the data. It attempts to reduce the dimensionality between variables while retaining the majority of the data by finding the directions of data with the highest variance and projecting them onto a smaller subspace with fewer dimensions. During analysis, the data is first standardized because PCA yields a feature subspace that maximizes variance along the axes. Once that is done, the covariance matrix is computed from the data using Scikit-learn's toolbox methods and computed the eigen decomposition from that for both the training and testing data sets. The covariance between two variables can be calculated as follows:

$$\sigma_{jk} = \frac{1}{n-1}\sum_{i=1}^{n}\left(x_{ij} - \bar{x_j}\right)\left(x_{ik} - \bar{x}_k\right)$$

This calculation is repeated for every variable combination and in turn, the covariance matrix can be computed with the following matrix equation:

$$\sum = \frac{1}{n-1}\left((X - \bar{x})^T(X - \bar{x})\right)$$

$\bar{x}$ represents the mean vector of the data. Once the covariance matrix was computed, the eigenvectors can be sorted based on the largest eigenvalues corresponding to those eigenvectors. With the eigenvectors sorted in descending order of largest eigenvalues, the data can be further devolved by plotting the cumulative variance and explained variances of the test and training data to show the principal components that attributed the most variance to the data.

Support Vector Machine

The main algorithm developed for this project was a Support Vector Machine (SVM). It was developed to try to further classify the patient gene data through training and prediction/classification steps. SVMs are a type of supervised learning classification algorithms which first train by taking a training dataset and then optimizes certain parameters that are used to later classify new data in the testing/classification method. SVMs are very commonly used for regressions or classifications, in other words labeling testing/training data to be classified from a class with a domain of two binary classes, in which each class can be labeled based on unique variations of a given set of features, or variables. Generally, training SVMs are the more difficult part as it involves convex optimization, which usually when dealing with non-linear data requires Quadratic Programming (QP) to solve (Platt 1998). It is common solution for constraint satisfaction problems when you need to classify data of usually two, or sometimes more, different class labels in which you have multiple features that are used to classify data points as belonging to one class or the other. SVMs are particularly useful where the boundary that divides different classes in a given data cluster that may not necessarily be linear, and instead could be quadratic and more complex to calculate. Both the linear and non-linear classifiers will be discussed more in the next two sections, as they involve slightly different methods of optimization.

Linear Classifier

Linear classifier SVMs are very similar to Perceptron algorithms, however they can prove to be more useful in finding the maximally separating width between different classes, or the largest margin around the separating hyperplane. In a linear SVM, the goal is to generate the decision boundary hyperplane which classifies data to be from either class, in this case binary classification. The vector *w* is known as the norm to the hyperplane line and is also the weight vector.

The scalar *b* is the bias, or the y-intercept of the hyperplane. The discriminant function is calculated as the general equation for the hyperplane:

$f(x) = w^T \cdot x + b$

The discriminant function is used to classify a data point based on its sign of *f(x)*. Support vectors are the data points that lie closest to the decision boundary hyperplane and are the most difficult data points to classify that have direct bearing on the optimum location of the decision surface. They will lie on either of the separating hyperplanes for each class, and thus their *f(x)* value will be very close to either 1 or -1. The *f(x)* value will be 0 at the middle hyperplane, or decision boundary. For linearly separable data, the algorithm is most concerned with finding the hyperplane, defined as, that best separates the training data points of different classes. The goal is to maximize the margin, $2/\|w\|$, and minimize $\|w\|^2$ subject to the constraints:

$$f(x) = \begin{cases} w^T \cdot x + b > 1, & y_i = 1 \\ w^T \cdot x + b \leq -1, & y_i = -1 \end{cases}$$

Multiplying the discriminant function by the class label, y_i, rewrites the general constraint as:

$y_i * (w^T \cdot x + b) \geq 1$

This primary constraint states that the discrimination boundary is obeyed. It becomes a constrained convex optimization problem, where we can think of the surface as a paraboloid, and we need to find the global minimum, or optimal *w*, that satisfies our two main constraints. The training consists of optimizing the weight vector, *w*, and the bias, *b*, parameters. Once we have these parameters optimized, the classification step becomes trivial by just plugging in an input feature vector for *x*, and the class is assigned based on the sign of *f(x)*.

Non-Linear Classifier

For linear classification, we have observed that discriminant function, or *f(x)*, is used to classify a data point based on its sign of *f(x)*. That exists as the primal form of the classifier function, *f(x)*, but it can also be represented as the dual version for non-linearly separable data classification:

$$f(x) = \sum_i^N \alpha_i y_i (x_i^T \cdot x) + b$$

This is another way to represent the classifier discriminant function when using kernels. α represents the Lagrange multiplier for the solution, and we still have the *b* (bias) parameter we want to optimize. α_i is the contribution of the i-th training example, with a total of *N* training examples, to the final solution of the weight vector in dual form. The weight vector is the sum of each training example's feature vector with its respective α, which is where the dual form of the discriminant function derives from. To classify a new data point using a dual, non-linear classifier, we simply perform the kernel against the input data point with each training example support vector, where α_i is non-zero, multiplied by the respective training example's y_i and α_i values, and take the cumulative sum of that which serves as our prediction for that data point. The training of the non-linear classifier involves using the Sequential Minimal Optimization (SMO) algorithm, which improves upon the linear classifier by optimizing the *a* and *b* parameters given a set of non-linearly separable data (Platt 1998). During training, the *a* and *b* parameters are optimized using the equation below:

$$\max_{\alpha_i \geq 0} \sum_i \alpha_i - \frac{1}{2} \sum_{jk} a_j \alpha_k y_i y_k \, k(x_i, x_k)$$

In the equation above from the training step during SMO, it is observed the $k(x_i, x_k)$ represents the kernel trick being applied to each combination of feature vectors while optimizing the α value. The kernel used can be any kind of polynomial kernel or gaussian, which serves to transform the input vector such that it is projected into a higher n-dimensional feature space. The beauty of the kernel trick is that it simplifies projecting a data point, or vector, onto a higher dimension in which it becomes possible to find a linearly-separable hyperplane. For the purposes of this project, it was chosen to use a gaussian kernel:

$$k(x, x') = e^{\left(\frac{-\|x - x'\|^2}{2\sigma^2}\right)}$$

Similarly, the non-linear classifier function becomes:

$$f(x) = \sum_i^N \alpha_i y_i k(x_i \cdot x) + b$$

In the classification step using the above equation, the kernel is applied to the input vector x against each support vector, x_i, and that result is multiplied by each support vector's respective α_i and y_i values. The output, or classification label, then becomes the sign of the cumulative sum of each support vector's α_i and y_i multiplied by the kernel function of itself and the input feature vector x.

Related Work

Rather than always training SVMs using linear classifiers with the general discriminant function for the hyperplane, others have used QP to classify non-linearly separable data. QP attempts to drastically speed up training SVM parameters. In particular, others have expressed the maximum margin classifier in terms of the minimization of an error function, with a simple quadratic regularizer (Bishop 2006). This becomes solvable using a soft margin classifier . Many common QP solvers exist to solve this problem, such as the Python library cvxopt. Unless a given set of data is perfectly linearly separable, training an SVM requires the solution of

a very large QP optimization problem (Platt 1998). SMO breaks this large QP problem into a series of smaller QP problems. The SMO algorithm was developed by John Platt at Microsoft Research in the late 1990s as an alternative method for training SVMs. His primary motivations and reasoning for developing this algorithm was to exploit the growing interest among the machine learning research community at the time by contributing a faster and more efficient alternative for training SVM parameters. SMO can be used to train both linearly and non-linearly separable data, which is what made it a significant breakthrough at the time and is still widely used today to train SVMs. His SMO algorithm was used in this project to train the non-linear classifier for the SVM developed in Python without requiring any QP libraries.

Project Description

After preprocessing the data using principal component analysis with the help of Scikit-learn and determining the most important genes to classify as features, the SVM was developed using the NumPy library with both linear and non-linear classifiers. The SVM was implemented as a separate class with training and classify methods, and when an instance is created it requires to know the kernel to use as input, as well as an optional sigma, σ, parameter for the gaussian kernel. It uses either the linear or gaussian kernel. Depending on if the kernel is linear or gaussian, the training and classify methods will perform different operations by doing a conditional check of what the kernel type is. Plotting the hyperplane graph results was implemented in a separate method that utilizes Matplotlib. The SVM's fit method takes in the training data as input and optimizes the parameters *w* (if linear), α (if non-linear), and *b*, which get set to its class's properties for that instance. This is so that when calling the classify method on some testing data to be classified, the SVM has already performed training and has optimized its parameters, so it does not need to recompute these as they are stored as instance variables. This was a logical design choice as an instance of an SVM should train once using all the training data but may want to classify different sets of testing data when running experiments.

The implementation of the linear classifier SVM involves starting with a predefined weight vector, based on the largest values in the training data feature vectors. We step down several iterations by the size of *w* multiplied by a factor, either 0.1, 0.01, or 0.001. We initialize *b* to be small. We apply several transformations to the weight vector, and each step iteration we check that the data set's feature vector, *x*, and that:

$y_i * (w^T \cdot x + b) \geq 1$

We then store this value of ||*w*|| for the current iteration and put it in a dictionary with its respective *(w, b)* values. At the end of each iteration we check if *w* is not less than or equal to 0, and if so we decrease our weight vector by the current step size. We keep repeating until the step iteration loop until we increase the step by a very small factor of *w*, and we grab the *(w, b)* tuple that has the corresponding minimum ||*w*|| from our dictionary.

The non-linear classifier's training method was implemented using SMO, which the pseudocode is provided in figure 1. The inputs are the training data feature vectors, the

1: $C \leftarrow$ regularization parameter
2: $tol \leftarrow$ tolerance
3: $maxPasses \leftarrow 50$
1: $\alpha_i \leftarrow 0, \forall i, b \leftarrow 0$
2: $passes \leftarrow 0$
3: **while** $passes < maxPasses$ **do**
4: $numChangedAlphas \leftarrow 0$
5: **for** $i = 1\ to\ i = m$ **do**
6: $E_i \leftarrow f(x_i) - y_j$
7: **if** $(y_j E_i < -tol$ **and** $\alpha_i < C)$ **or** $y_j E_i > tol$ **and** $\alpha_i > 0)$ **then**
8: j $\leftarrow$ random value $\neq i$
9: $E_j \leftarrow f(x^{(j)}) - y^{(j)}$
10: $\alpha_i^{old} \leftarrow \alpha_i, \alpha_j^{old} \leftarrow \alpha_j$
11: **if** $y^{(i)} \neq y^{(j)}$ **then**
12: $L \leftarrow max(0, \alpha_j - \alpha_i), H \leftarrow min(C, C + \alpha_j - \alpha_i)$
13: **else if** $y^{(i)} = y^{(j)}$ **then**
14: $L \leftarrow max(0, \alpha_j + \alpha_i - C), H \leftarrow min(C, \alpha_j + \alpha_i)$
15: **end if**
16: **if** $L == H$ **then**
17: **continue**
18: **end if**
19: $\eta \leftarrow 2\langle x_i, x_j \rangle - \langle x_i, x_i \rangle - \langle x_j, x_i \rangle$
20: **if** $\eta >= 0$ **then**
21: **continue**
22: **end if**
23: $\alpha_j \leftarrow \alpha_j - (y_j (E_i - E_j))/\eta$
24: **if** $|\alpha_j - \alpha_j^{old}| < 10^{-5}$ **then**
25: **continue**
26: **end if**
27: $\alpha_i \leftarrow \alpha_i + y_j y_j (\alpha_j^{old} - \alpha_j)$
28: $b_1 \leftarrow b - E_i - y_i(\alpha_i - \alpha_i^{old})\langle x_i, x_i \rangle - y_j(\alpha_j - \alpha_j^{old})\langle x_i, x_j \rangle$
29: $b_2 \leftarrow b - E_j - y_i(\alpha_i - \alpha_i^{old})\langle x_i, x_j \rangle - y_j(\alpha_j - \alpha_j^{old})\langle x_j, x_j \rangle$
30: **if** $\alpha_i > 0$ **and** $\alpha_i < C$ **then**
31: $b \leftarrow b_1$
32: **end if**
30: **else if** $\alpha_j > 0$ **and** $\alpha_j < C$ **then**
31: $b \leftarrow b_2$
32: **end if**
33: **else**
34: $b \leftarrow (b_1 + b_2)/2$
35: **end if**
36: $numChangedAlphas \leftarrow numChangedAlphas + 1$

```
37:     end if
38:   end for
39:   if numChangedAlphas == 0 then
40:     passes ← passes + 1
41:   else
42:     passes ← 0
41:   end if
42: end while
```

Figure 1: Pseudocode for SMO

training labels, and a σ value for the kernel. Lines 5-6 show a loop through each of the sample feature vectors, where the error is calculated between the predicted label and the true label, y_i. Lines 8-18 we compute the error and α value for index, j, and find α_j so as to maximize the objective function. If this value ends up lying outside the bounds L and H, we clip the value of α_j to lie within this range. Otherwise in line 27 we solve for α_i now that we have optimized α_j, and we apply the kernel to x_i and x_j to then calculate the b parameter. The SMO training method outputs the LaGrange multipliers and the bias parameter as outputs α and b, respectively. The α parameter is outputted as a NumPy array with the LaGrange multiplier values for each training feature vector. The support vectors are then initialized as a NumPy array of N arrays of zeros, where n is the number of samples, or patients from our original data. The support vectors are then created as each index of the training feature vector, x, corresponding to the non-zero LaGrange multipliers from our α array. Now that we have the α, y and support vectors for each index of x, we can create our classifier function to predict new input test feature vectors for classification. The nonlinear classifier mentioned earlier is implemented as its own method:

$$f(x) = \sum_{i}^{N} \alpha_i y_i k(x_i \cdot x) + b$$

The for loop in the code iterates up to N samples from the size of the training data. Each support vector for that corresponding index of the training data, x_i, is applied with the kernel against the input feature vector to be classified, x. The kernel is implemented as its own method as well, which in this project a gaussian kernel was used.

$$k(x, x') = e^{\left(\frac{-\|x-x'\|^2}{2\sigma^2}\right)}$$

Various values for sigma, σ, were tested on the classifier method's performance. The performance was measured as the number of correct predictions of the class labels, y_i, where $y_i \in \{-1, 1\}$. The classifier outputted the predicted labels as the *y_predict* array compared to the testing labels stored in another array called *y_test*. This will be described in more detail in the next sections.

Experiments

Implementation

The main tool used for this project was using the Python programming language. The following toolkits were used for data preprocessing and PCA, as well as create some of the visualizations: Matplotlib, NumPy, Pandas, and SciPy. The Scitkit-learn toolkit was used for covariance matrix calculations and generated the correlation heatmap grid visualizations. The SVM class was not created with any major libraries other than NumPy for the necessary array and vector manipulations as well as Matplotlib for the plotting of the hyperplane and decision boundary.

Data

The training and testing data sets were obtained from segments of the dataset by Golub et. al. The data set comprised of labels for these two cancer types to distinguish to users of the data of which cancer type a patient had, in the form of 0 or 1. A label of 0 represented the patient having ALL, while 1 represented the patient having AML. The training dataset contains the first of the gene expression data for the first 38 patients, and the testing dataset with the remaining patients 39 through 72 gene expression levels. Each patient has 7129 genes in the dataset, and a different expression level for each gene. The actual.csv file contained the labels for each patient in a column. The class labels for ALL and AML were converted so that the domain is $\in \{-1, 1\}$, where -1 represented ALL, and 1 still represented AML. A subset of the genes was used as features after PCA filtered out the genes that contributed most to the variation in the dataset to ultimately 44 genes. A subset of these final candidates was then used as features for the training and classification experiments of the SVM.

Results

Before the large data set could be analyzed, a significant amount of preprocessing was required to prepare it for dimensionality reduction and further classification analysis using the SVM. In the training and testing datasets, each row represents a gene and each column represents the patient. Columns 1 and 2 provide descriptions for that gene, the Gene Description and the Gene Accession number for each gene. These datasets were inputted into Pandas data frame objects. A matrix transpose operation was performed

so that the genes became the columns as they are the features.

Once the data was cleaned and adjusted, the first step in analyzing this large data set was to drastically reduce the dimensionality and to understand the general direction of the variance among the gene expression levels. PCA was applied using NumPy's linear algebra functions and Scikit-learn to simplify the PCA analysis. The covariance matrix was computed using Scikit-learn, and the eigenvectors were then sorted based on the largest eigenvalues corresponding to those eigenvectors. The cumulative variance and explained variances were plotted of the test and training data to show the principal components that attributed the most variance to the data. It became clear that the first twenty principal components were needed to calculate roughly 80% of the entire data variance. The important discovery is however that the first three principal components account for approximately 33% of the entire variance. Because components after Principal Component (PC) 8 account for only about 1-2% of the variance, it was decided to plot the first ten PCs.

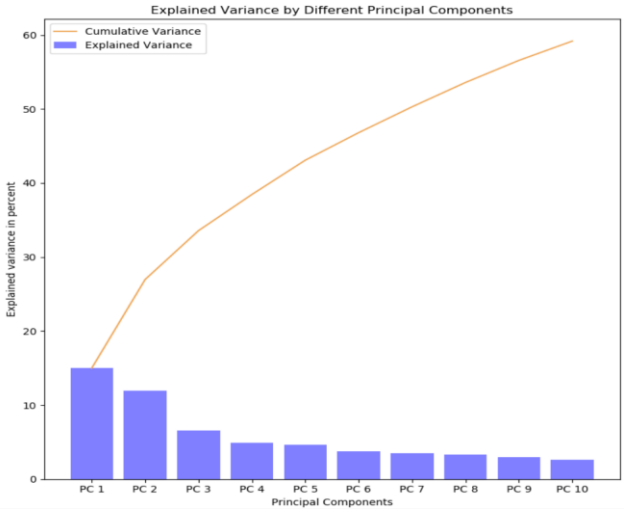

Figure 2: Explained and Cumulative Variances versus Principal Components

From Figure 2, it became pertinently clear that the first 2-3 PCs contribute the majority of the entire data set. After about the eighth principal component, the contributed variance drops off significantly. This implies that at least the first two or three PCs should be plotted in their own feature space to correctly reduce the dimensionality of the entire data set, while keeping the vast majority of the variance intact.

Now that we understand the variance as it relates to the two different cancer classification groups, each patient was classified as having either AML or ALL by attempting to correlate the expression values to each patient. Scikit-learn was used to extract the main features from the data following PCA. The most important features were the genes with the highest correlations to be classified as either AML or ALL contributors. The main Python script geneExpressionClassification.py has a method, called *extract_features()*, which created a stratified shuffle split object and extracted the features that contributed the most to the data's variance and their labels. These final features were then plotted onto a heat map using another method, *visualize_correlation_grid()*, which displayed the correlations to be cancerous or not. The most important features, or in this case genes, were the ones outputted by the stratified shuffle split object with the highest correlation scores and plotted onto the heat map shown below for the testing dataset. The correlation heatmap grid can be seen below in Figure 3 for the testing dataset.

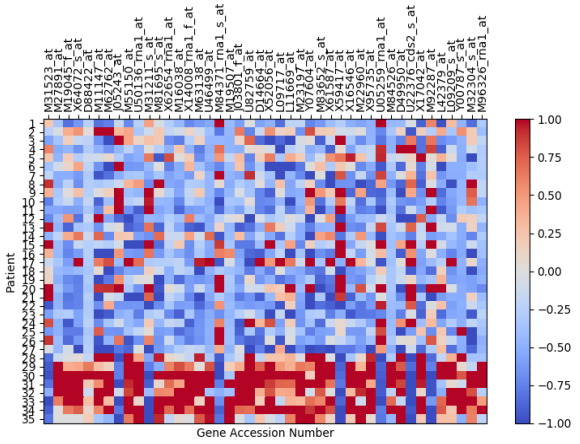

Figure 3: Gene Expression Correlation Heatmap for Test Data

The correlation heatmap grid looks very similar to the original figure in the original paper by Golub et. al. One of the goals for this project was to try to visualize the data in a way that looks like a DNA microarray and also strives to be similar to the one in the original paper, which visualizes the data best by classifying each patient if they have either AML or ALL based on color gradients. The most important genes relevant to the data with the highest scores were plotted, and they are deemed to be the best indicators to classify the patients as either having AML or ALL. Expression levels greater than the mean are shaded in red, and the expression levels less than the mean are in blue. The scale of the bar indicates standard deviations above or below the mean. The redder the block is in the grid for a patient's (y-axis) particular gene (x-axis), the more highly correlated that gene expression level is to AML. The bluer the block is in the grid, the more negatively correlated that gene expression is to AML, and hence more positively correlated to ALL. Our original dictionary has ALL labeled as 0, and AML as 1.

Once the genes of most importance were filtered into the final candidates of 44 genes, the classification of the patient

gene expression level vectors using the SVM could commence. A subset of the 44 genes were trained and tested on the SVM. The linear classifier was first tested using the linear kernel on the transpose of the weight vector, w, and each input feature vector from the data, x_i. After attempting to classify the data from different genes against each other, it became clearly evident that the training method was unable to find a linearly-separable hyperplane that satisfied the discriminant function constraint using any combination of different transformations of the weight vector and each set of feature vectors. This is where the linear classifier was shown to not perform as well as the non-linear classifier on gene expression data, which clearly is non-linearly separable in most instances, especially if using multiple genes as features from many patients. Two genes in particular, M22960 and X62654, when used as features did successfully generate a linear hyperplane, which can be seen in the following figure.

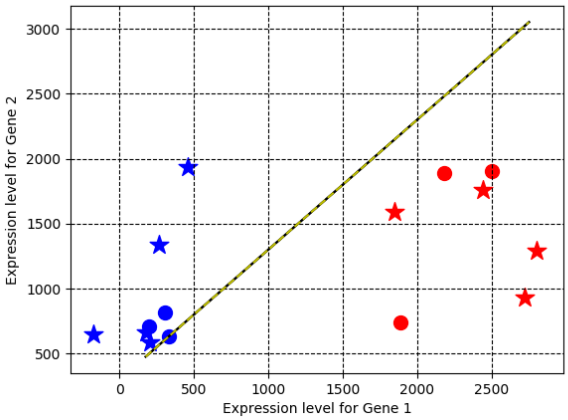

Figure 4: Linear SVM Hyperplane Decision Boundary using genes M22960 and X62654 as features

In figure 4, the genes were plotted where each axis represents a different feature, in this case the expression level from a specific gene. The genes plotted were M22960 and X62654. The hyperplane is clearly plotted as separating the decision boundary between the two classes, where anything to the right of the hyperplane are colored red and get classified as $y_i = 1$, and new data points to the left are colored blue and are classified as $y_i = -1$. As mentioned earlier, the original labels from the Golub et. al data had 0 representing a patient having ALL, while 1 represented the patient having AML. The 0 label was converted to -1, so the blue points indicate patient samples, or feature vectors, that are classified as the patient having ALL, and feature vectors classified as 1 are colored red and indicate the patient having AML. The training points were plotted as regular circles, and the new classified vectors from the testing data were plotted as stars. Despite a sample size with two features and samples from 16 different patients, the linear classifier was 100% accurate in predicting the new testing points' class labels.

After it was discovered that the majority of the expression level data was non-linearly separable, the data was sampled and tested by the non-linear classifier. To do this, when initializing an instance of the SVM class, an extra input parameter for the kernel was inputted as a string as "*gaussian*", along with several σ values to be tested. The training for the non-linear classifier using SMO outputted the α and b parameters used for classification each training run. The experiments consisted of training the SVM using SMO with all the training data from patients 1-38 with different sets of genes. Different sampling of genes were tested, so that it could be determined how well the classifier predicts the testing labels from the patients based on the number of features used to train the SVM. Additionally, the varying of the sigma, σ, parameter was tested on the gaussian kernel and how it would affect classification results. The testing results are displayed in the following table figures.

σ	Test Accuracy
10.0	0.57
8.0	0.70
7.0	0.76
5.0	0.78
3.0	0.57
1.5	0.54
0.5	0.54

Table 1: Performance of varying σ using 3 genes as features

σ	Test Accuracy
10.0	0.54
8.0	0.54
7.0	0.92
5.0	0.97
3.0	0.89
1.5	0.68
0.5	0.54

Table 2: Performance of varying σ using first 10 genes as features

It is clear from the tables that varying σ in the gaussian kernel has a rather significant effect on the accuracy of classifying the test labels. The ideal value for σ seemed to be around 5.0 when classifying and training samples with 3 of the filtered genes as features. Having a σ too low or too high outside this range would overclassify points from either class, and as a result one class would be more favored over the other. The number of features, or genes used, also greatly affected the training and classification accuracy. Table 2 shows testing using the first 10 genes from the training and testing data, compared to the sample of 3 genes used and their results in table 1. It seemed that the more features used

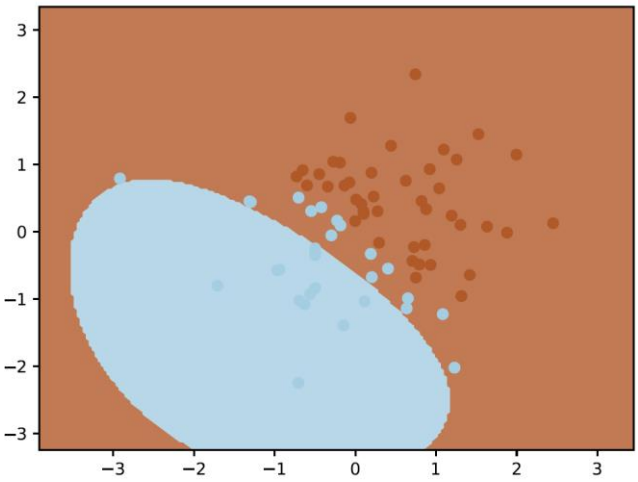

Figure 5: Non-linear SVM Decision Boundary with gaussian kernel using sample of genes as features

to train the parameters, the more optimal the parameters were calculated and thus the more accurate the predictions were of the test labels for each patient during classification. The use of varying the σ value on the gaussian kernel is further shown in figure 5. The test and train values were plotted from a sample of genes after being normalized. The decision boundary plane for the non-linear classifier was plotted using the gaussian kernel with a σ value of 1.5. It can be clearly seen that the classifier only correctly predicts roughly half of the patient samples that have ALL, or a y_i value of -1. When increasing the σ value closer to 5.0, the classifier's accuracy improves, as shown in tables 1 and 2. If we increase σ, more points are classified as ALL and thus the blue-colored blob enlarges, thus encompassing more points. The test results from table 2 however reiterates that the test accuracy depends on both the σ value and the total number of features, or genes, used. To get the highest accuracy of predicting the patient samples' disease class, one should optimize σ to around 5.0 and use at least 10 genes as features.

Conclusion

With a non-linear classifier using a gaussian kernel with a σ value of 5.0, the SVM was able to correctly classify 36 out of the 37 patients from the testing data as having either AML or ALL using the first 10 genes as features for training. PCA applied to the original data from Golub et. al filtered out the 44 genes that contributed to the majority of the variance of the expression levels across all 7129 genes. Although this is possibly a simpler version of an SVM compared to some of the more sophisticated machine learning packages available to the average Python user such as Scikit-learn, the SVM was able to be implemented without any libraries other than NumPy and successfully classified a majority of the patients of their disease types. These results show that it is possible to classify a patient as having a disease type based on binary classification using a number of genes' expression levels as features. It further shows that machine learning, specifically SVMs, are a useful tool for analyzing DNA microarray data and can greatly assist with today's challenge of analyzing and making sense of the vast amount of sequencing data from patients that is becoming more widely available to researchers. It may even be possible to achieve much better performance with an SVM than what was accomplished with this project. As it was seen in the optimization of the parameters during training and the features used, the performance of the SVM can be incredibly dependent on the design of these aspects. The linear classifier can be improved in that having to satisfy the discriminant function's general constraint for the hyperplane can be very restrictive on what combination of feature vectors can be used with a weight vector during training. SMO was shown to be able to fix this by being an effective method for training an SVM with the non-linear classifier. Another alternative to using SMO could be implementing a soft margin classifier. Although the vast majority of gene expression data is non-linearly-separable, the linear classifier can be improved to provide an alternative kernel when small samples of microarray data is tested that may be linearly-separable. The implementation of the SVM in this project may be sufficient, but it can certainly be further optimized. Therefore, further work should include improving the linear fitting function, or training, and explore other non-linear training methods aside from SMO such as soft margin classifiers in order to make this an even more robust SVM that can classify different sets of DNA sequencing and microarray data in order to thoroughly classify many patients' disease types.

References

Bishop, C. 2006. *Pattern Recognition and Machine Learning (Information Science and Statistics).* Springer-Verlag.

Golub, T. R.; Slonim, D. K.; Tamayo, P.; Huard1, C.; Gaasenbeek, M.; Mesirov, J. P.; Coller, H.; Loh, M. L.; Downing, J. R.; Caligiuri, M. A.; Bloomfield, C. D.; Lander, E. S. 1999. Molecular Classification of Cancer: Class Discovery and Class Prediction by Gene Expression Monitoring. *Science* 286:531-537.

Platt, J.; Scholkopf, B.; Burges, C.; Smola, A. eds. 1998. Fast Training of Support Vector Machines using Sequential Minimal Optimization. *Advances in Kernel Methods – Support Vector Learning*. Cambridge, Mass.: MIT Press.

Yang, S., and Naiman, D. Q. eds. 2014. Multiclass cancer classification based on gene expression comparison. *Statistical Applications in Genetics and Molecular Biology* 13(4): 477-496.