Hybrid Gene Selection Framework using Adaptive Wrapper and Filtering Techniques

Authors

Md Anisul Islam	104411
Md Mozaharul Mottalib	104438

Supervisor

Prof. Dr. M. A. Mottalib
Head of the Department
Department of Computer Science & Engineering (CSE)
Islamic University of Technology (IUT)

Co-Supervisor

Shaikh Jeeshan Kabeer
Lecturer
Department of Computer Science & Engineering (CSE)
Islamic University of Technology (IUT)

A Thesis submitted to the Department of Computer Science & Engineering (CSE) in Partial Fulfillment of the requirements for the degree of Bachelor of Science in Computer Science & Engineering (CSE)

Department of Computer Science & Engineering (CSE)
Islamic University of Technology (IUT)
Organization of the Islamic Cooperation (OIC)
Gazipur, Bangladesh

September, 2014

CERTIFICATE OF RESEARCH

This is to certify that the work presented in this thesis paper is the outcome of the research carried out by the candidates under the supervision of Prof. Dr. M. A. Mottalib, Head of the Department, Department of Computer Science and Information Technology, IUT and co-supervision of Shaikh Jeeshan Kabeer, Lecturer, Department of Computer Science and Information Technology, IUT, Gazipur. It is also declared that neither this thesis nor any part thereof has been submitted anywhere else for the award of any degree or any judgment.

Authors

Md Anisul Islam

Md Mozaharul Mottalib

Signature of Co-Supervisor

Shaikh Jeeshan Kabeer
Lecturer, Department of CSE, IUT

Signature of Supervisor

Prof. Dr. M. A. Mottalib
Head, Department of CSE, IUT

Signature of the Head of the Department

Prof. Dr. M. A. Mottalib
Head, Department of CSE, IUT

Abstract

Analysing the thousands of gene expression values is a difficult task due to the curse of dimensionality of data produced by Microarray chips. Primary role of an effective feature selection model is to simplify this task. To simplify the task of disease classification and predicting cancer, feature selection plays a vital role through removing less informative genes. In this study, we propose a hybrid approach to gene selection using adaptive filter and adaptive wrapper approach. As filter method exhibits some limitations, an adaptive form of filtering has been employed that iteratively selects genes in each iteration and emphasizes on the misclassified samples and in subsequent iteration tries to find out the effective genes for misclassified samples. This approach performs better than traditional filter methods as it focuses on its weaknesses. In gene selection, Artificial Neural Networks (ANN) are mostly used as a classifier. In this study, adaptive ANN has been used as an internal wrapper. This helps to generate a better subset of genes. The proposed hybrid approach is applied on leukaemia, colon and lung cancer benchmarked datasets. Better result has been found compared to other well-known approaches.

Table of Contents

List of Tables

Chapter 1

Introduction

1.1 Overview

Researches are going on to get rid of curse of dimensionality of data. Several methodologies have been applied and still there are scope of improvement. The main focus in recent days is to minimize the space and time complexity to get a better solution in critical research areas. Many machine learning algorithms are being applied to process huge amount of data within short period of time. Feature selection is the technique of selecting a subset of features for building learning models [1]. It attempts to identify and highlight the most informative genes from microarray dataset which have significant effect on the biological states of leaving organism. Our main focus is to select the most informative genes from the microarray dataset to reduce the dimensionality of the problem.

1.2 Problem Statement

One of the main problems of high dimensional data is the inclusion of noisy and irrelevant data in the information set[2]. Space and time complexity increases due to large number of noisy, redundant and uninformative gene expression [3]. Reducing the dimensionality is the goal in feature selection.

1.3 Research Challenges

Execution of a brute force exhaustive search is not encouraged due to high dimensional feature space. Therefore an optimal method is to be devised to achieve an accurate and efficient outcome. The desired outcome of the method is minimizing the number of features and increasing the predictive power of the classifiers. To add more intensity to the problem domain this field of bioinformatics produces inadequate testing and training samples. With the removal of noisy, irrelevant and redundant information the proposed method must be able to handle the correlation factor existing between the features and thus utilize the combined predictive power. Our proposed method encompasses all these factors and theoretically expects to bring about better results handling noisy, redundant and correlated data.

1.4 Motivation

This study aims at deriving a better method for feature selection using a hybrid approach. This approach has an upper hand on other approaches as it does consider the collective measure of genes and not only focus to individuals. The approach inherits both the merits from filter and wrapper approaches. The adaptive wrapper approach gives better result taking less runtime than the other traditional wrapper approaches.

1.5 Scopes

In case of research, to improve the search criteria for feature selection, two probable approaches can be taken. Firstly, the improvements can come from existing approaches. Secondly, it may come from generating new approaches. In this study, we have worked with a hybrid approach which is the combination of two algorithms. This approach tends to reduce some shortcomings of the existing methods which we will discuss later. Computer vision, Pattern recognition, Artificial intelligence etc. are the fields where feature selection can be of great use.

1.6 Research Contribution

In this study, the limitation of the existing filtering approach has been improved. An adaptive filter and adaptive wrapper method has been combined inheriting merits from both of the approach. The original dataset is reduced to about half by using adaptive Wilcoxon method. The adaptive Wilcoxon method cannot handle with the noise present in the dataset and also cannot fully utilize the collective predictive power of genes. To overcome this shortcomings we used an adaptive wrapper method. It includes the use of Artificial Neural Network working as an internal wrapper.

1.7 Thesis Outline

In chapter 1 we have talked about the introduction to our accomplished work. Chapter 2 will be dealing with the basics of feature selection, different existing feature selection techniques. Chapter 3 will be on highlighting our proposed work, giving an overall concept of our work. The experimental analysis along

with result comparison with different methods are chalked out in Chapter 4. Here the implementation code is also given. Chapter 5 focuses on the scopes of future work to be done on our study.

Chapter 2

Literature Review

2.1 Feature Selection

Feature selection is the technique of selecting a subset of relevant features (genes) for building robust learning models [2][14]. With the advancement of technology, processing speed of computer has increased and also a lot of data collection technologies have been improved whereas the generated data is really enormous that cannot be dealt with efficiently in short time. Use better feature selection method we can reduce the huge dataset into a smaller version that meets the need improving time and space complexity both. In machine learning and statistics, feature selection techniques are widely used. We are mainly focused on the first part. Different machine learning algorithms are there for feature selection. They are categorized below:

- **Supervised Learning:** generates a function mapping input to the desired output. Output is predetermined here.
- **Unsupervised Learning:** models a set of input but there is no mapping to desired output.
- **Semi-supervised Learning:** combines both labeled and unlabeled examples to generate an appropriate function or classifier.
- **Reinforcement Learning:** an observation of real world is given. Every action has some impact on the environment. Environment provides feedback that guides the learning algorithm.
- **Transduction:** predicts new output based on training input, training output and test input.
- **Learning to Learn:** learns its own inductive bias based on previous experience.

Most of the feature selection methods focuses on the supervised learning and in this study it's not different. An effective learning model is constructed based on supervised learning.

2.2 Feature Selection Techniques

Feature selection can be applied on a set of features which can be a better solution than choosing all possible subsets of features [3] - 14. It is impractical if a large number of sub-sets are available. Feature selection can be classified into two broad categories:

1. Feature Ranking*:* This is also known as feature weighing which assesses individual features and as-signs them weights according to their degree of relevance. Many researches have been done with feature ranking as the base method (for example Bekkerman et al., 2003, Caruana and de SA, 2003, Weston et al., 2003).

2. Feature Subset Selection (FSS): This technique measures the goodness of each found feature subset[3]. A great deal of work has also been done Feature subset selection (for example Guyon et al., 2004, Ma & Huang, 2005, Ooi & Tan 2003). Feature Subset Selection techniques are more effective than FR techniques. In our study we have emphasized on FSS. FSS follows three basic methodologies:

    - Filters
      Filter techniques takes in account the relevance of features by looking only into the intrinsic properties (mean, variance, standard deviations etc.) of data. A feature relevance score is calculated for scoring genes. From the scored set of genes the low scoring features are removed and the remaining subset features with the higher scores are presented to the classifier algorithm for classification. Similarly in information gain which is another scoring method is used to score the genes from which subsequent removal of low scoring genes is done.[4]

    - Wrapper
      In wrapper techniques the entire feature space is considered to generate a subset of features which are evaluated through a classifier.[5]

    - Embedded Methods
      The entire feature space to search in order to find optimal subsets of features. The search procedure is

built into the classifier in this variation. [Duda et al. 2001] used Bayesian theory to guide the search process based on the probability of selection. In SVM classifier was over hauled to weigh features and adjust them for making selection in the process of finding the solution.

In order to perform feature selection the feature space needs to be traversed i.e. feature searching. Feature searching involves going through the feature space to select features to be used for classification. Many approaches to feature selection exist which can be broadly classified into the following:

Exhaustive

Exhaustive search or brute-force is a general problem-solving technique that traverses all the possibilities for the solution checking whether each of the candidates satisfies the solution criteria. Exhaustive search is easy to implement and will guarantee a solution if it exists. The downside of this method is its cost. Cost is proportional to the number of candidate solutions which tends to grow very rapidly as the size of the problem increases. Thus this approach should avoided when sample size is very large.

Best first

Best first is a heuristic searching technique. It traverses through the candidate solution and selects that appears to be the best choice under the current situation and moves forward. But this approach does not ensure an optimum solution. An evaluation function defines the selection of a particular candidate.

Simulated Annealing

In simulated annealing instead of choosing the best move it picks a random one and if the move improves the situation then it is accepted otherwise the probability of the move is decreased to ensure that such moves are not chosen in the future.

2.3 Artificial Neural Network

Artificial Neural Networks (ANNs) are computational models extensively used in machine learning, pattern recognition and related fields. It is inspired by animal's central nervous system, in particular brain. It has different learning

paradigm like: Supervised, Unsupervised, Reinforcement. ANNs consist of sets of adaptive weights that are tuned by these learning algorithms. These adaptive weights are logical connection strengths between neurons. These connections are active during training and prediction. One common criticism regarding ANN is that it requires a large diversity of training for operation. ANNs are excellent for extracting information from a noisy dataset. In traditional feature selection techniques, ANNs are mostly used as classifiers. However it can also be used to assess the quality of the selected feature[19].

Chapter 3

Proposed Method

3.1 Overall Concept

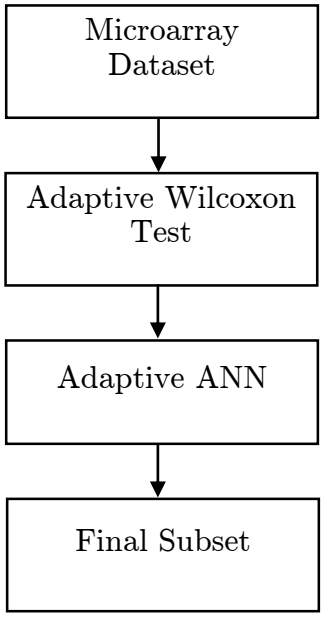


Fig. Outline of proposed approach.

The basic steps of our proposed approach is like:

1. Microarray Dataset represents the gene expression values in the form of matrix.
2. We are using Wilcoxon Rank Sum test to score genes in an iterative manner to minimize the dimensionality of the feature space.
3. Then we are using the adaptive Artificial Neural Network as an internal wrapper to get the final best gene subset.

3.2 Adaptive Wilcoxon Rank Sum Test

In comparison with single gene based feature selection, adaptive filter approach gives much better result. This process identifies the weak performers in a particular iteration by calculating the score through Wilcoxon Rank Sum Test and tries to find out the features that would perform well for those weak performers. It can be compared to giving a boost to weak performers in a particular iteration. At first all the genes are assigned equal probabilities to be selected. Then Wilcoxon Rank Sum Test is performed to rank all the genes. As all genes are assigned equal probabilities at the beginning, during first iteration every gene has equal probabilities to be selected.

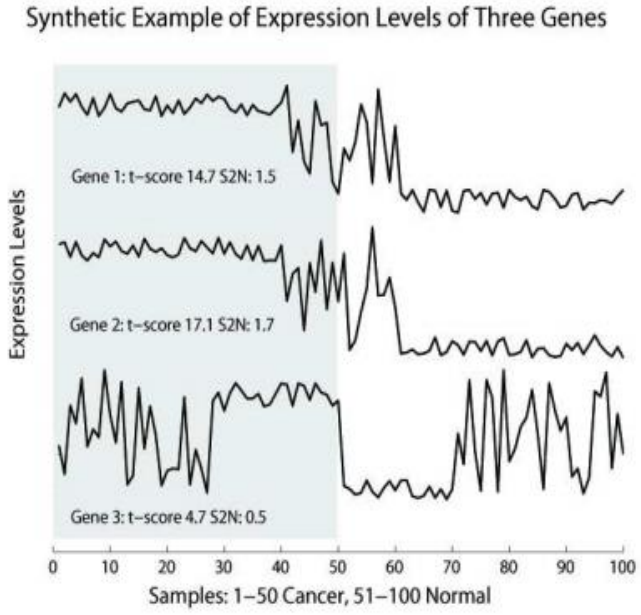


Fig. Example of redundancy in selected gene set

The first two gene, gene1 and gene2 acts similarly. In almost 80% of the samples the expression level of gene1 and gene2 can be used to identify the class label of the particular sample efficiently. The expression levels of these two genes are higher cancer samples than in normal ones.

3.2.1 Terminologies

First we discuss some basic terminologies of the adaptive Wilcoxon Rank Sum Test:

- A bootstrap sample set S^S_t is a multi-set of samples randomly drawn with replacement from the

original set of samples S. The sampling probability of each sample in S is determined by a probability table p(s) where s∈ S.

- The worst set of samples S_{worst} with respect to bootstrap dataset B and a single-gene based scoring function F is defined as a multi set:

$$argmax\ (F\ (E\ (g,\ S^{S}{}_{t} - S)))$$

$$S \subseteq S^{S}{}_{t} \text{ and } |S| = \sigma$$

Here $S^{S}{}_{t}$ - S means a set by removing S from $S^{S}{}_{t}$. We also call $S^{S}{}_{t}$ - S worst the best set of samples. This algorithm starts off by generating a set of samples called a bootstrap sample set $S^{S}{}_{t}$ which is a multi-set of samples obtained by random sampling from the pool of all samples S. The probability of a sample being selected is equal to p(s) where s ∈ S and initially all samples have a probability of $\frac{1}{S}$.

3.2.1 Adaptive Wilcoxon Steps

Steps of the adaptive Wilcoxon Rank Sum Test is given bellow:

1. Score all the genes in the dataset using Wilcoxon Rank Sum Test, sort them and select the top score gene.

2. Worst set of sample is identified.

3. Probability of these worst samples for selection in next iteration is increased.

4. Next iteration selects gene performing well for the worst sample set.

5. This algorithm would run until the number of selected genes (BG) has been found which depends on the dataset being evaluated.

3.2.2 Algorithm

Algorithm 1: Adaptive Wilcoxon Test

N is the number of genes to be selected. F is a single gene based discriminative score.

1. Initialize P(s) to be $1/s_t$ (s_t is the total number of samples in the dataset).
2. G˙s as an empty set
3. For $|G_S|$ n do
4. Generate the bootstrap sample set S^S_t
5. Calculate the Wilcoxon Rank Sum test on bootstrap
6. Add top ranked gene g based on Wilcoxon score to G_S
7. Find worst ∂ samples S_{worst} based on gene g and S^S_t using algorithm2
8. Reduce the probability for the best set of samples (those samples which are classified accurately by the gene g)
9. Remove g from dataset
11. End for
10. Return G˙s

Algorithm 2: Worst Sample Set Determination

1. S_1, S_0 to be empty sets.
2. For all s in S do
3. $S_1 \leftarrow S - \{s\}$
4. Calculate Wilcoxon Rank Sum for the gene g, add score to S_0
5. End for
6. Sort S_0, add samples S corresponding to top ∂ scores in S_0 to S_1
7. Return S_1

3.3 Adaptive Artificial Neural Network

Wrapper-based feature selection is attractive because wrapper methods are able to optimize the features they select to the specific learning algorithm [22]. Standard wrapper methods are expensive to use with neural nets in terms of space and time. We present an internal wrapper feature selection method that

gives better result than the traditional external wrapper feature selection approaches. This internal wrapper feature selection method selects features at the same time hidden units are being added to the growing neural network architecture.

3.3.1 Terminologies

The network contains INS nodes for the inputs, and OUTS nodes for the outputs. The edge weights are trained using standard back propagation of errors. Back propagation is done on the training set. Each successful iteration adds a hidden unit to network. RMSE error is calculated for all possible net and the best one is considered. We select some features for each of the sample on basis of error calculation. Finally on basis of threshold value (V_{TH}), final subset of gene is selected.

3.3.2 Algorithm

Algorithm 3: Adaptive Artificial Neural Network

```
1. Initialize network net.
2. For k = 0 to Max_Hidden
3.      Network best = NULL
4.      For each input i
5.              Network new = net with an additional hidden unit added
6.              add edge from node i to the new hidden unit
7.              add edges from each hidden unit in net to new one
8.              add edges from the new hidden unit to each output node
9.              train(new)
10.             if(best == NULL or RMSE(new) < RMSE(best))
11.                     best = new
12.             End
13.     End
14.     net = best
15. End
```

After getting selected features from each of the samples, frequency of occurrence of the genes is calculated. Using the threshold value (V_{TH}), final subset is selected.

$$X = \{\sum_{i=0}^{n} A_{i1},\ \sum_{i=0}^{n} A_{i2},\ \ldots,\ \sum_{i=0}^{n} A_{im}\}$$

$$S = \{x: x \in X \cap x \geq V_{TH}\}$$

Here, n and m corresponds to total number of samples and total number of features.

3.4 Our Proposed Approach

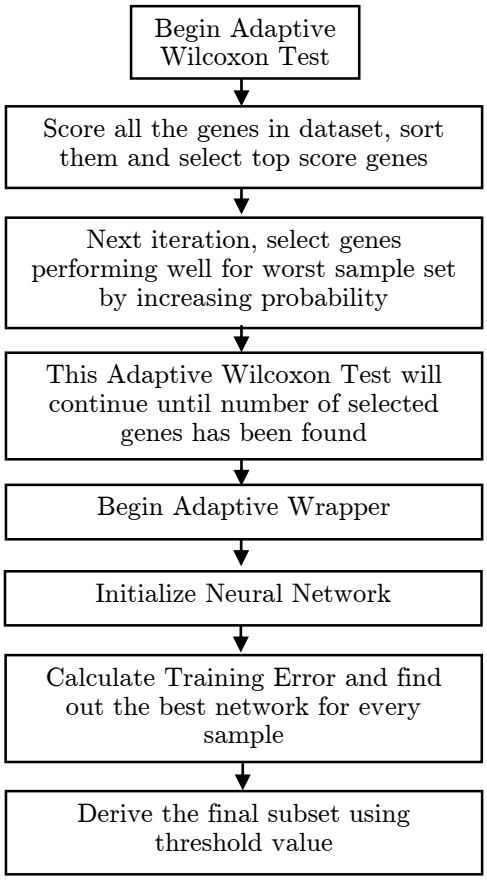


Fig. Proposed adaptive wrapper and filter approach

Chapter 4

Experimental Analysis & Result Comparison

We have used Matlab R2013 and JetBrains Intellij IDEA 13.0 as main platform of our work. Java Encog Neural Network Framework has been used for working with Artificial Neural Networks.

4.1 Dataset Details

We have applied our proposed approach on three state of the art databases: Acute Lymphoblastic leukemia cancer (ALL), Lung cancer and colon cancer. Table 1 summarizes the data sets. In the ALL dataset there are 72 tissue samples (47 B-cell and 25 T-cell). In the lung dataset there are 181 tissue samples (47 MPM and 134 ADCA). The training set contains 32 of them, 16 MPM and 16 ADCA. The rest 149 samples are used for testing. Each sample is described by 7130 genes. Colon dataset contains 62 samples collected from colon-cancer patients. Among them, 40 tumor biopsies are from tumors (labeled as "negative") and 22 normal (labeled as "positive") biopsies are from healthy parts of the colons of the same patients. 2000 genes out of around 6500 genes were selected based on the confidence in the measured expression levels.

Dataset	Number of Classes	Number of Samples in the Dataset	Number of Genes
ALL (Leukemia)	2 (B-cell ALL and T-cell ALL)	72 (47 B-cell and 25 T-cell ALL)	7130
LUNG	2 (MPM and ADCA)	181 (47 MPM and 134 ADCA)	12533
COLON	2 (Normal and tumor)	62 (22 normal and 40 tumor)	2000

Table 1: Summery of Microarray Datasets.

Table 1 shows that dataset for Colon cancer contains the lowest number of genes comparing to other two datasets. This low number of genes exposes to higher possibilities of misclassifications and over fitting. More the number of samples allows us to train classifiers so that we can correctly classify test samples.

4.2 Performance Analysis

Our implementation work begins with adaptive Wilcoxon Test where we select a particular number of genes using Wilcoxon Rank Sum Scoring method. The main objective to perform this step is to provide our adaptive wrapper approach a better initial population.

While implementing adaptive filter we have used Wilcoxon Rank Sum Score to calculate the scoring for each gene within a sample. Then we got all the scores of the genes for retrieving the best scored gene.

Then we applied adaptive filter on the three important available microarray datasets and got the reduced number of genes to apply as the input for adaptive wrapper approach. These are shown in the table below-

Datasets	Original Number Of Genes	Adaptive Wilcoxon output
Leukemia (ALL) Cancer	7130	2139
Lung Cancer	12533	3760
Colon	2000	1300

Table 2: Reduced number of genes by Boost Feature Subset Selection

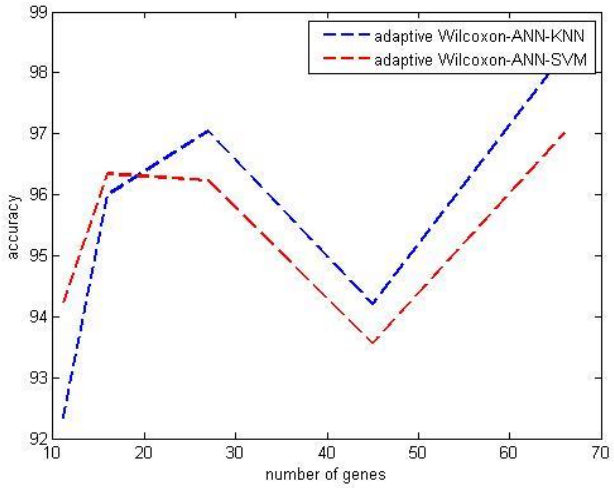
We have roughly taken 30% of genes from Leukemia and Lung cancer datasets. These genes are then considered in the adaptive wrapper approach. As the number of genes in Colon cancer dataset is only 2000, the output of adaptive Wilcoxon was set to 65% of total number of genes which is 1300.

Result for Leukemia Dataset:

Threshold value (Vth)	Number of Gene selected	Number of runs	Adaptive Wilcoxon - ANN - KNN	Adaptive Wilcoxon - ANN - SVM
2	27	1	97.39	98.43
		2	95.26	95.72
		3	96.04	96.37
		4	98.18	95.84
		5	98.34	94.79
3	16	1	94.44	94.73
		2	97.22	98.44
		3	93.87	96.38
		4	98.27	95.39
		5	96.18	96.77
4	11	1	91.66	94.11
		2	92.36	92.98
		3	89.95	95.17
		4	91.15	94.52
		5	92.19	93.97
		Average ± S.D.	95.10 ± 2.65	95.57 ± 1.54

Table 3: Result for Leukemia Dataset

Graphical Comparison:

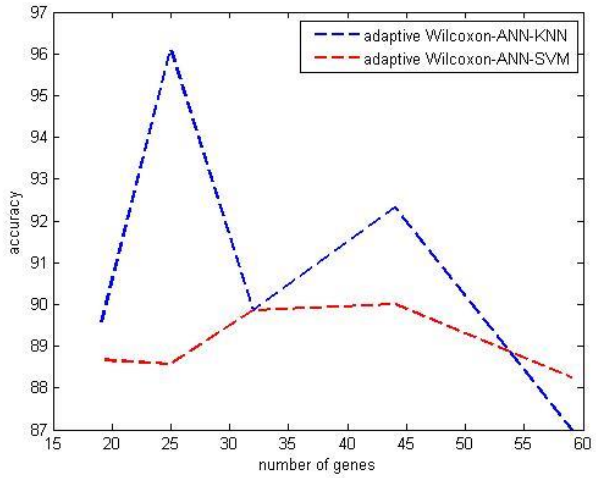


Result for Lung Cancer:

Threshold value (Vth)	Number of Gene selected	Number of runs	Adaptive Wilcoxon - ANN - KNN	Adaptive Wilcoxon - ANN - SVM
2	32	1	93.75	92.86
		2	88.19	89.28
		3	86.17	85.72
		4	91.37	89.57
		5	89.80	91.85
3	25	1	96.88	92.88
		2	96.15	85.71
		3	96.21	89.19
		4	96.40	88.92
		5	94.89	86.12
4	19	1	90.62	85.71
		2	87.50	92.52
		3	89.83	86.47
		4	90.19	85.78
		5	88.28	89.93
		Average ± S.D.	91.82 ± 3.59	89.03 ± 2.65

Table 4: Result for Lung Cancer Dataset

Graphical Comparison:

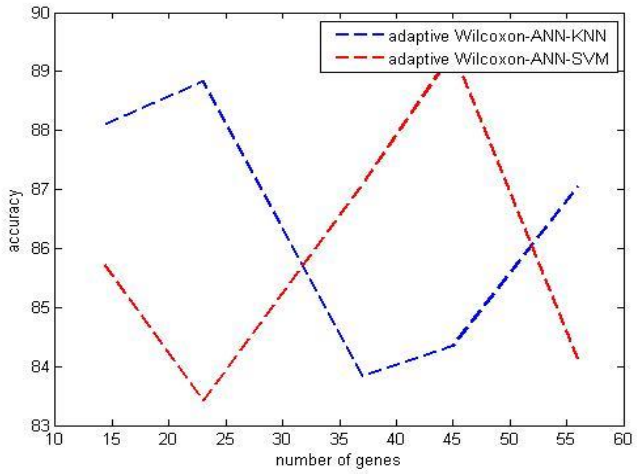


Result for Colon cancer:

Threshold value (Vth)	Number of Gene selected	Number of runs	Adaptive Wilcoxon - ANN - KNN	Adaptive Wilcoxon - ANN - SVM
40	37	1	89.9736	85.7143
		2	86.3433	86.0465
		3	85.4839	88.3721
		4	88.3212	86.0465
		5	90.3226	89.1892
42	23	1	88.7097	81.6327
		2	87.0968	83.6735
		3	88.7097	85.2933
		4	89.2072	81.2763
		5	88.9912	85.2311
44	14	1	83.8710	85.7143
		2	82.2581	83.6735
		3	85.9902	84.2239
		4	84.2322	87.2101
		5	82.8710	88.2131
		Average ± S.D.	86.8220 ± 2.6	85.4303 ± 2.3

Table 5: Result for Colon Cancer Dataset

Graphical Comparison:



4.3 Comparative Analysis

Dataset	ACO	PSO	GA-SVM	CGA-SVM	BCGA-KNN	**Adaptive Wilcoxon - ANN – KNN**	**Adaptive Wilcoxon - ANN - SVM**
Leukemia (Average ±SD; The Best)	83.89 -- 	84.22 -- 	83.80±2.14 88.24 	91.53±2.48 96.12 	96.08±1.28 98.19 	95.10±2.65 98.34 	95.57±1.54 98.44
Colon (Average ±SD; The Best)	76.87 -- 	79.69 -- 	83.48±1.77 86.27 	86.36±1.46 89.74 	87.70±1.34 90.43 	86.82±2.6 90.32 	85.43±2.3 89.18
Lung (Average ±SD; The Best)	79.76 -- 	80.09 -- 	84.85±1.54 87.92 	86.89±1.54 88.32 	88.33±1.30 89.97 	91.82±3.59 96.88 	89.03±2.65 92.88

Table 6: Comparative analysis between results from three dataset

Comparative analysis shows that both "Adaptive Wilcoxon-ANN-KNN" and "Adaptive Wilcoxon-ANN-SVM" approach give slightly better result than the other approaches.

Chapter 5

Conclusion

There are various techniques for Gene Selection. Researchers have done many works on this field. Day by day the techniques are evolving. A relatively new way is using Artificial Neural network. This method has been used rarely in this field.

The main problem of Gene Selection from microarray dataset is the large amount of data obtained in a microarray. The redundant and noisy data misdirects classification. So, a filter method can help to reduce the number of genes to take into account. Our proposal includes adaptive Wilcoxon test to select a fixed but relatively smaller number of genes from the large dataset.

Neural networks are prone to large input numbers. But Adaptive ANN method reduces the required number of genes to very minute scale, thus initiating training and testing by the neural network. It is evident that our proposed method works better than previous works in many of the cases, it can contribute in not only Gene Selection, but also in several other fields like Geology, Archeology, Geography, Climate study, Image processing etc.

Appendix
Implementation Code

Matlab Implementation Code:

```
load grp.mat
% Loading dataset
filename ='LungCancerDataset.xls';
data = xlsread(filename);
[r, c] = size(data);
c = c - 1;

%indices of samples
x = 1:c;

% Number of genes to be selected, n ( 30% of total number of features )
n = round( r * 0.3 );
bestIndex = zeros( n, 1 );

% Assigning equal probability to each sample...
p = repmat(1/c, 1, c);
cdf = cumsum( p );

% Creating an empty set to store top ranked genes...
f = zeros(1,n);
sizeOfF = 1;
reducedData = zeros( n, c + 1 );

% Initializing the bootstrap sample set...

% Loop will continue until n genes are selected...
while (sizeOfF <= n)
    bootstrapSet = zeros(r - sizeOfF + 1, ( 2 * c ) + 1 );
    bootGrp = zeros( 1, 2 * c );
    randomIndex = zeros( 1, 2 * c );
    selection = rand( 1, 2 * c );
    for i = 1 : 2 * c
        randomIndex( 1, i ) = sum( cdf <= selection( i ));
        while( randomIndex( i ) == 0 )
            v = rand;
            randomIndex( 1, i ) = sum( cdf <= v );
        end
    end
    [rndRow, rndCol] = size( randomIndex );
```

```
    for i  = 1 : rndCol
       bootstrapSet(:,i) = data(:,randomIndex(1,i) );
       bootGrp(:,i) = grp(:,randomIndex(1,i));
    end

    if sum( bootGrp ) == rndCol || sum( bootGrp ) == 0
       continue;
    end

    [IDX, Z] = rankfeatures(bootstrapSet( :, 1: (2 * c ) ), bootGrp,
'CRITERION','wilcoxon');

    % Copying the index from data to bootstrap
    bootstrapSet(:,rndCol + 1) = data(:,c + 1 );

    % Best gene containing the expression values for selected samples
    best = zeros( 1, rndCol );

    for i  = 1 : rndCol
       best( 1, i ) = data( IDX(1), randomIndex( 1, i ));
    end

    % Index of best stored
    bestIndex( sizeOfF, 1 ) = bootstrapSet( IDX( 1 ), rndCol + 1 );

    [ p ] = elimination( best, bootGrp, p, randomIndex );

    % Removing top ranked gene....
    reducedData( sizeOfF, : ) = data( IDX( 1 ), : );
    data( IDX( 1 ), : ) = [];

    sizeOfF = sizeOfF + 1;

end

reducedData = sortrows( reducedData, c + 1 );
[ r, c ] = size( reducedData );

for j = 1 : c - 1
    minimum = min( reducedData( :, j ));
    maximum = max( reducedData( :, j ));
    for i = 1 : r
        value = reducedData( i, j );
        value = ( ( maximum - value ) / ( maximum - minimum ) );
        reducedData( i, j ) = value;
    end
end
```

```
%not considering the augmented column symboling index in reducedData data set (
reducedData( :, 1 : c - 1 ) )
input = reducedData( :, 1 : c - 1 );

index = reducedData( :, c );

%in input.txt file, rows indicates samples and column indicates features
dlmwrite( 'input.txt', input', 'delimiter', '\t', 'precision', 4, 'newline', 'pc' );

%in index.txt file values signifies the index values of the selected genes
dlmwrite( 'index.txt', index', 'delimiter', '\t', 'precision', 4, 'newline', 'pc' );
```

```
function [ p ] = elimination( best, bootGrp, p, randomIndex )

    [ ~, rndCol ] = size( randomIndex );
    w = zeros( rndCol, rndCol );

    for i = 1 : rndCol
        w( i, : ) = best( 1, : );
        w( i, i ) = 0;
    end

    global eIDX;
    [ eIDX ] = rankfeatures( w, bootGrp, 'CRITERION','ttest');

    threshold = 0.005;

    for i = 1 : ( rndCol * 0.75 )
        p( 1, randomIndex( 1, eIDX( i,1 ) ) ) = p( 1, randomIndex( 1, eIDX( i,1 ) ) )
+ threshold;
    end;

    p = p./norm(p,2);
    norm(p,2);

end
```

KNN Validation:

```
% load( 'grp.mat' );

% Loading dataset
filename = 'LungCancerDataset.xls';
data = xlsread( filename );
[ r, c ] = size( data );

X = data';
```

```
X( c, : ) = [];
Y = grp';

% Number of neighbors
n = 3;

classifier = ClassificationKNN.fit( X, Y );
classifier.NumNeighbors = n;
```

SVM Validation:

```
filename = 'LungCancerDataset.xls';
data1 = xlsread(filename);
[ r, c ] = size(data1);
data = data1';
load grp.mat;
cp = classperf( grp );

geneindex = zeros( 1 );
iteration = 1;

% threshold frequency
threshold = 2;

output = dlmread( 'output.txt' );

for i = 1 : 2237
    if output( i ) > threshold
        geneindex( iteration ) = index( i );
        iteration = iteration + 1;
    end
end

data = data( :, geneIndex );

% testTrainRatio tuning
testTrainRatio = .9;
[ train, test ] = crossvalind( 'holdout', grp, testTrainRatio );

svmStruct = svmtrain(data(train,:),grp(train));
class = svmclassify(svmStruct,data(test,:));

classperf(cp,class,test);
cp.CorrectRate
```

Java Implementation Code:

Main.java

```
1    package com.company;
2
3    import com.data.Data;
4
5    import org.encog.engine.network.activation.ActivationSigmoid;
6    import org.encog.ml.data.MLData;
7    import org.encog.ml.data.MLDataPair;
8    import org.encog.ml.data.MLDataSet;
9    import org.encog.ml.data.basic.BasicMLDataSet;
10   import org.encog.neural.error.ATanErrorFunction;
11   import org.encog.neural.networks.BasicNetwork;
12   import org.encog.neural.networks.layers.BasicLayer;
13   import org.encog.neural.networks.training.propagation.resilient.ResilientPropagation;
14
15   import java.util.Arrays;
16   import java.util.HashMap;
17   import java.util.Map;
18
19   public class Main {
20       //taking 10% of the input features as prominent, it's around 200....
21       public static int numOfProminentFeature = 200;
22       public static Map<Integer,Integer> histogram = new HashMap<Integer, Integer>();
23
24       public static void main(String[] args) {
25           Data data = new Data();
26           data.loadData();
27           data.printData();
28
29           data.loadClassLabel();
30           int tempCount = 0;
31
32           for(int i = 0; i < data.featureCount; i++){
33               histogram.put(i,0);
34           }
35
36           for(int i = 0; i < data.sampleCount; i++){
37               int[] prominentArray = new int[numOfProminentFeature] ;
38               Arrays.fill(prominentArray,-1);
39
40               for(int j = 0; j<numOfProminentFeature; j++){
41
42                   int minIndex = -1;
43                   double minError = 12345;
44                   double[][] inputArray = new double[1][j+1];
45
46                   int n = 0;
47                   for(int m = 0; m < prominentArray.length; m++){
48
49                       if(prominentArray[m] >= 0){
50                           double tempValue = data.dataArray[i][prominentArray[m]];
51                           inputArray[0][n] = tempValue;
52                           n++;
53                       }
54                   }
55
56                   for(int k = 0; k < data.featureCount; k++){
57                       int flag = 0;
58                       for(int m = 0; m < prominentArray.length; m++){
59                           if(prominentArray[m] == k){
60                               flag = 1;
61                               break;
62                           }
```

```
63                      }
64
65                      if(flag == 1){
66                          continue;
67                      }
68
69                      inputArray[0][n] = data.dataArray[i][k];
70                      //creating network
71                      double[][] tempTarget = {{data.classLabel[i][0]}};
72                      MLDataSet trainingSet = new BasicMLDataSet(inputArray,tempTarget);
73
74                      BasicNetwork network = new BasicNetwork();
75                      network.addLayer(new BasicLayer(new
ActivationSigmoid(),true,j+1));
76                      network.addLayer(new BasicLayer(new
ActivationSigmoid(),true,j+1));
77                      network.addLayer(new BasicLayer(new ActivationSigmoid(),true,1));
78
79
80                      network.getStructure().finalizeStructure();
81                      network.reset();
82
83                      final ResilientPropagation train = new
ResilientPropagation(network, trainingSet);
84
85                      int epoch = 1;
86                      double tempError = 0;
87
88                      while(true){
89                          train.iteration();
90
91                          tempError = train.getError();
92                          if(tempError == Double.NaN || Double.isNaN(tempError)) {
93                              continue;
94                          }
95
96                          epoch++;
97                          if(train.getError() < 0.01 ){
98                              break;
99                          }
100                     }
101
102                     if (tempError < minError ) {
103                         minError = tempError;
104
105                         minIndex = k;
106                     }
107                 }
108
109                 prominentArray[j] = minIndex;
110                 histogram.put(minIndex,histogram.get(minIndex) + 1);
111             }
112
113             for(int t = 0; t < prominentArray.length; t++){
114                 System.out.print(prominentArray[t]+ " ");
115             }
116             System.out.println("");
117         }
118
119         for(int i = 0; i < data.featureCount; i++){
120             System.out.print(histogram.get(i)+ " ");
121         }
122     }
123
124
125 }
```

Data.java

```
1    package com.data;
2
3    import java.io.BufferedReader;
4    import java.io.FileInputStream;
5    import java.io.InputStream;
6    import java.io.InputStreamReader;
7
8
9     public class Data {
10       public String test = "";
11       public String dataFileName = "input.txt";
12       public String classLabelFileName = "classLabel.txt";
13       public String [] tempDataArray;
14       public String [] tempClassLabelArray;
15       public double minDataValue;
16       public double maxDataValue;
17       public double [][] dataArray;
18       public double [][] classLabel;
19       public int sampleCount = 72;
20       public int featureCount = 2139; //30% of the base data set
21
22
23    /*loads comma separated feature values from the input text file */
24       public void loadData(){
25           //reading
26           try{
27               InputStream ips=new FileInputStream(this.dataFileName);
28               InputStreamReader ipsr=new InputStreamReader(ips);
29               BufferedReader br=new BufferedReader(ipsr);
30               String line;
31               while ((line=br.readLine())!=null){
32                   test+=line+"\n";
33               }
34               br.close();
35
36           }
37           catch (Exception e){
38               System.out.println(e.toString());
39           }
40
41           tempDataArray =test.split("\\s+");
42           formatData();
43       }
44
45
46    /*loads comma separated class label value from input text file*/
47       public void loadClassLabel(){
48           test = "";
49           try{
50               InputStream ips=new FileInputStream(this.classLabelFileName);
51               InputStreamReader ipsr=new InputStreamReader(ips);
52               BufferedReader br=new BufferedReader(ipsr);
53               String line;
54               while ((line=br.readLine())!=null){
55                   test+=line+"\n";
56               }
57               br.close();
58           } catch (Exception e){
59               System.out.println(e.toString());
60           }
61           tempClassLabelArray = test.split("\\s+");
62           formatClassLabel();
63       }
64
```

```
65      public void printTempData(){
66          System.out.println("Array :"+ tempDataArray.length);
67          for(int i=0;i< tempDataArray.length;i++)
68          {
69              System.out.println("array"+i+"  :"+ tempDataArray[i]);
70          }
71      }
72
73      public void formatData(){
74          int flag = 0;
75          dataArray = new double[this.sampleCount][this.featureCount];
76
77          for(int i = 0; i < this.sampleCount; i++) {
78              for(int j = 0; j < this.featureCount; j++) {
79                  dataArray[i][j] = Double.parseDouble(tempDataArray[flag]);
80                  flag++;
81              }
82          }
83      }
84
85      public void formatClassLabel(){
86          int flag = 0;
87          classLabel = new double[this.sampleCount][1];
88
89          for(int i = 0; i < this.sampleCount; i++) {
90              classLabel[i][0] = Double.parseDouble(tempClassLabelArray[flag]);
91              flag++;
92          }
93      }
94
95      public void printData(){
96          for(int i = 0; i < this.sampleCount; i++) {
97              System.out.println("Sample No# " + (i+1)  );
98              for(int j = 0; j < this.featureCount; j++) {
99                  System.out.print(dataArray[i][j] + " ");
100             }
101             System.out.println();
102         }
103     }
104
105
106
107     public void setMinDataValue(){
108         minDataValue=1111;
109         for(int i = 0; i < this.sampleCount; i++) {
110             for(int j = 0; j < this.featureCount; j++) {
111                 if (dataArray[i][j] < minDataValue) {
112                     minDataValue = dataArray[i][j];
113                 }
114             }
115         }
116     }
117
118     public void setMaxDataValue(){
119         maxDataValue=1111;
120         for(int i = 0; i < this.sampleCount; i++) {
121             for(int j = 0; j < this.featureCount; j++) {
122                 if (dataArray[i][j] < maxDataValue) {
123                     maxDataValue = dataArray[i][j];
124                 }
125             }
126         }
127     }
128
129     public double getMaxDataValue(){
130         return this.maxDataValue;
131     }
```

```
132
133     public double getMinDataValue(){
134         return this.minDataValue;
135     }
136
137     public int getSampleCount() {
138         return sampleCount;
139     }
140
141     public void setSampleCount(int sampleCount) {
142         this.sampleCount = sampleCount;
143     }
144
145     public int getFeatureCount() {
146         return featureCount;
147     }
148
149     public void setFeatureCount(int featureCount) {
150         this.featureCount = featureCount;
151     }
152
153
154 }
155
```

References

[1] Li-ye-chuang et al.(2012) A hybrid BPSO-CGA Approach for Gene Selection and Classification of Microarray Data JOURNAL OF COMPUTATIONAL BIOLOGY Volume 19, Number 1,2012

[2] Juana Canul-Reich et al. (2008) Feature Selection for Microarray Data by AUC

[3] John Quackenbush (2001) Computational Genetics: Computational analysis of microarray data

[4] Cosmin Lazar (2012) A Survey on Filter Techniques for Feature Selection in Gene Expression Microarray Analysis

[5] RuichuCai et al (2010) A New Hybrid Method for Gene Selection

[6] Wei Zhao et al. (2011) A Novel Framework for Gene Selection

[7] Kelly Fleetwood et al. (2010) an Introduction to Differential Evolution

[8] Wei-Neng Chen et al. (2010) A Novel Set-Based Particle Swarm Optimization Method for Discrete Optimization Problems

[9] L.-Y. Chuang et al. (2010) Correlation-based Gene Selection and Classification Using Taguchi-BPSO

[10] Mohd Saberi Mohamad et al. (2009) Particle swarm optimization for gene selection in classifying cancer classes. In Artif Life Robotics (2009) 14:16–19

[11] Sheng Ding-(2009) Feature Selection based F-score and ACO Algorithmin Support Vector Machine

[12] Shutao Li et al. (2008) Gene selection using hybrid particle swarm optimization and genetic algorithm

[13] Mojtaba Ahmadieh Khanesar et al. (2007) A Novel Binary Particle Swarm Optimization. In 15th Mediterranean Conference on Control and Automation, Athens, Greece.

[14] Xueming Yang et al. (2007) A modified particle swarm optimizer with dynamic adaptation

[15] R. K. Agrawal, et al. (2007) a Hybrid Approach for Selection of Relevant Features for Microarray Datasets

[16] Qi Shen et al. (2006) a combination of modified particle swarm optimization algorithm and support vector machine for gene selection and tumor classification

[17] Chung-Jui Tu et al. (2006) Feature Selection using PSO-SVM

[18] Xian Xu et al. (2006) Boost Feature Subset Selection: A New Gene Selection Algorithm for Microarray Dataset

[19] Sergio Ledesma et al. (2008) Feature Selection Using Artificial Neural Networks

[20] Akbar Rahideh et al. (2011) Cancer Classification Using Clustering Based Gene Selection and Artificial Neural Networks

[21] Md. Monirul Kabir et al (2010) a new wrapper feature selection approach using neural network

[22] Backstorm, L. et al. (2006) C2FS: An Algorithm for Feature Selection in Cascade Neural Networks

[23] Vitaly Schetinin et al. (2003) A Learning Algorithm for Evolving Cascade Neural Networks