DOKUZ EYLÜL UNIVERSITY

GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES

MACHINE LEARNING MODELS FOR AUTOVERIFICATION OF MEDICAL LABORATORY TEST RESULTS

by

Velid Ali

December, 2015

İZMİR

MACHINE LEARNING MODELS FOR AUTOVERIFICATION OF MEDICAL LABORATORY TEST RESULTS

A Thesis Submitted to the
Graduate School of Natural and Applied Sciences of Dokuz Eylül University
In Partial Fulfillment of the Requirements for the Degree of Master of Sciences
in Computer Engineering

by

Velid Ali

December, 2015

İZMİR

M. Sc THESIS EXAMINATION RESULT FORM

We have read the thesis entitled **"MACHINE LEARNING MODELS FOR AUTOVERIFICATION OF MEDICAL LABORATORY TEST RESULTS"** completed by **VELID ALI** under supervision of **PROF. DR SÜLEYMAN SEVİNÇ** and we certify that in our opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science

[illegible]

Prof. Dr Süleyman SEVİNÇ

Supervisor

Prof. Dr. Alp Kut	Prof. Dr. Efendi Nasiboğlu
[illegible]	[illegible]
(Jury Member)	(Jury Member)

[illegible]

Prof.Dr. Ayşe OKUR

Director

Graduate School of Natural and Applied Sciences

ACKNOWLEDGEMENTS

Huge appreciation, in first place, I owe to my supervisor Prof. Dr. Süleyman SEVİNÇ who allowed me to work in this project, and orient me with his advices in the preparation of this project, same to, Assoc. Prof. Dr Pınar AKAN, Dr. Ferhat DEMİRCİ for their great support. And finally thanks to my family for their great support.

Velid Ali

MACHINE LEARNING MODELS FOR AUTOVERIFICATION OF MEDICAL LABORATORY TEST RESULTS

ABSTRACT

Understanding the job of biochemistry specialists while accepting and checking blood test results during in a day where they spend a lot of time, we decided to develop a learning system that can help them.

In our experiment we used machine learning models, LibSVM and ANN for classifying the data sets. Because our experiment was started with Artificial Neural Networks (ANN) datasets were obtained from the prior experiment and that approach was tested in a view of Support Vector Machines. We used Replace Missing Values filter for cleaning up the data instances of null values, and correlation of attributes was done with Correlation Attribute Evaluator of WEKA software.

We trained and tested our datasets which were taken from Beckmann blood analyzing machine, where machine learning models classified with high average rate.

Keywords : Support vector machines, LibSVM, artificial neural networks, machine learning, SMO, Spegasos

TIBBİ LABORATUVAR TETKİK SONUÇLARININ OTOVERİFİKASYONU İÇİN MAKİNA ÖĞRENME MODELLERİ

ÖZ

Biyokimya uzmanlarının gün içinde kan pıhtılaşması test sonuçlarının onaylanmasında, harcadıkları zamanı azaltmak amacı ile yardım edebilecek öğrenim sistemi geliştirilmiştir.

Uyguladığımız çalışmada verilerin sınıflandırılması için LibSVM ve yapay sinir ağları öğrenme modelleri kullanılmıştır. Veriler önceki çalışmadan elde edilmiştir, bir önceki çalışmada yapay sinir ağları burada o yaklaşım destekçi vektör makinesi üzerinde test edilmiştir. Verilerin null değerlerinden temizlenmesi için Replace Missing Values filtrelemesi kullanılmıştır, atribütlerin bağıntısı ise Correlation Attribute Evaluator ile gerçekleşmiştir.

Beckmann kan analiz makinasından alınan veri setlerimiz eğitilmiş ve test edilmiştir. Test edilen bu veri setleri yüksek doğruluk oranı elde edilmiştir.

Anahtar kelimler : Destekçi vektör makinesi**,** LibSVM**,** yapay sinir ağları, makina öğrenme, SMO, Spegasos

CONTENTS

Pages

LIST OF FIGURES

Page

LIST OF TABLES

Page

CHAPTER ONE
INTRODUCTION

The aim and the basic idea of this project is to integrate a learning system with a view to reduce time where biochemistry specialists are spending during the process of checking patient blood results, blood results also have to be accepted by the biochemistry specialist. To reduce the time where biochemistry specialists spends during the processes of accepting and checking patient blood results we decided to develop a learning system that can help them. Our learning system is able to make decisions like a biochemistry specialist. For facilitating the work of biochemistry specialists and assistants we dealt with Machine Learning models. We will experimentally evaluate Artificial Neural Networks (ANN), LibSVM, SPegasos, SMO models. All four models are based on a prepared data set where data cooperation was done with biochemistry specialists.

In order to solve practical problems learning algorithms are considered intelligent if they have the ability to give conclusions on the basis of certain facts. Since ANN and SVM algorithms characterize a high degree of error tolerance we have reached a satisfactory result. The performance of ANN with back-propagation is slower then Lib SVM, results shows that Lib SVM works with better accuracy also statistically better. If we have to compare two algorithms understanding and solving the way of ANN is easier then Lib SVM model.

How the project is divided into two parts in the first part is destined ANN, and for the second part where to improve learning system is developed SVM models. The models were trained and tested by Weka 3.7 tools.

1.1 Neural Networks (NN)

Neural network simulates the operation of the human brain while performing a given task or functions. Neural network is parallelized distributed processor with a natural ability for storing knowledge and ensuring it for using.

Artificial Neural Networks reminds the process of working the human brain in two respects:

1. Neural networks saves knowledge through the training process.
2. Weights between neural connections (the strength of synaptic connections) are used to memorizing the knowledge.

The procedure for performing the training neural network is same as training the algorithm. In algorithmic way through this procedure synaptic weights are changing in order to have desired network performance.
Performance of Neural Networks:

1. Nonlinearity, which is basically distributed.
2. Input-Output mapping, which can restore through the process of training
3. Adaptability- Ability of changing the strength of synaptic connections.
4. Response – As output neural networks can produce and degree certificates of a given decision.

1.2 Introduction to Machine Learning Algorithms

Machine Learning is a natural outgrowth of the intersection of Computer Science and Statistics. Computer Science has focused primarily on how to manually program computers, on the other side Machine Learning focuses on how to get computers to program themselves whereas Statistics has focused primarily on what conclusions can be inferred from data (Mitchel, 2006). Mathematical models for adaptation in these fields are somewhat different from those commonly used in machine learning suggesting significant potential for cross-fertilization of models and theories.

1.2.1 Place of Machine Learning within Computer Science

The question is “What is the role of machine learning in the field of computer applications?” First answer can be to imagine the world of all software applications, and to recognize the applications suggest within this space where machine learning has a special role for manipulating. Actually, machine learning methods and algorithms are the best methods available for developing particular types of software, in applications where:

- The application is too complex for people to manually design the algorithm. Here machine learning algorithm is the software development method of choice simply because it is relatively easy to collect labeled training data, and relatively ineffective to try writing down a successful algorithm.

- The application requires that the software customize to its operational environment after it is fielded. Machine learning provides the mechanism for adaption. Software applications that customize to users are growing rapidly within machine learning algorithms (Mitchell, 2006).

If we assume how much machine learning methods play a key role in the world of computer science, there will remain software applications for demand grows of self-customizing software, as computer gain access to more data, as we use effective machine learning algorithms.

Machine learning will help reshape the field of Statistics, by bringing a computational perspective to the fore, and raising issues such as never-ending learning, both Computer Science and Statistics will also help shape Machine Learning as we progress and provide new ideas to change the way we view learning (Mitchell, 2006).

For understanding the relationship between different learning algorithms, and which algorithms should be used and when it should be used is experimentally in different application domains. One part of understanding the relationships between algorithms and which one to use is to develop a theoretical understanding of the relationships among these algorithms, and of when it is appropriate to use each. More generally, the theoretical characterization of learning algorithms, their convergence properties, and their relative strengths and weaknesses remains a major research.

1.3 Support Vector Machines (SVM)

Support vector machines are supervised learning methods used for regression and classification. At the same time, is a regression and classification prediction tool that uses machine learning theory to maximize predictive accuracy. Support vector machines use hypothesis space of a linear function in a high dimensional feature space and implements a learning bias derived from statistical learning theory (Support Vector Machines, 2011).

With most machine learning tasks, the aim is usually to classify something into a group that you can then inspect later. When it is a couple of class types that you are trying to classify, then it is a fairly trivial matter to perform the classification. When we are dealing with many types of classes, the process becomes more of a challenge. Support vector machines helps us to work through the challenging classification.

Medical science has long used support vector machines for protein classification. The National Institute of Health has even developed a support vector machine protein software library. It is a web-based tool that classifies a protein into its functional family.

Some people criticize the support vector machine because it can be difficult to understand, unless you are blessed with a very good mathematician who can guide and explain to you what is going on. In some cases you are left with a black box implementation of a support vector machine that is taking in input data and producing output data, but you have little knowledge in between (Bell, 2013).

1.3.1 Linear Classifiers

To determine in which group an object belongs, we use a linear classifier to establish the locations of the objects and see if there is a neat dividing line-called a *hyperplane* – in place; there should be a group of objects clearly on one side of the line and another group of objects clearly on one side of the line and another group of objects just as clearly on the opposite side.

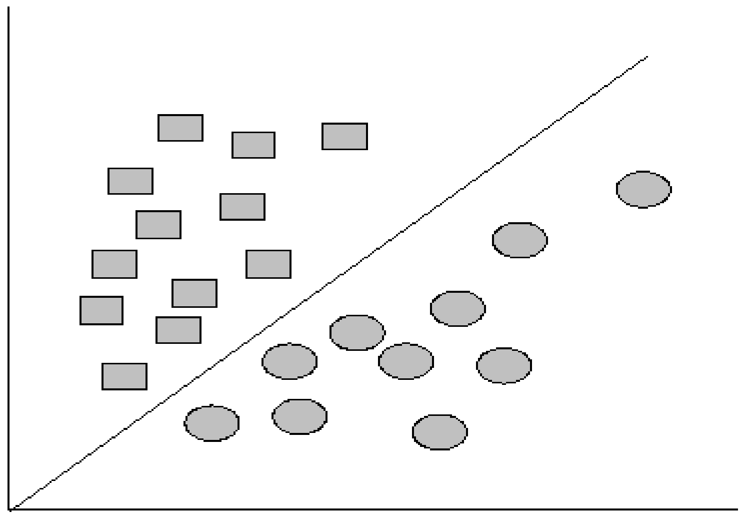


Figure 1.1 Illustration of Linear Classification

Figure 1.1 visually illustrates linear classification, it looks straightforward, but we need to compute it mathematically. Every object that we classify is called a point, and every point has a set of features.

For each point in the figure 1.1, we know there is an x-axis value and there is a y-axis value. The classification point is calculated as

$$sign(ax + by + c) \tag{1.1}$$

The values for a, b, and c are the values that define the line; these values are ones that you choose, and we will need to tweak them along the way until we get good fit (clear separation). What we are interested in, though, is the result; we want a function that returns +1 if the result of the function is positive, signifying the point is in one category, and returns -1 when the points is in the other category. The function is resulting value (+1 or -1) must be correct for every point that we are trying to classify.

If we look at figure 1.1 objects again. We know how a hyperplane divides the objects into either +1 or -1 on the plane. Extending that notion further, support vector machines define the maximum margin, assuming that the hyperplane is separated in a linear function. We can see in figure 1.2 with the main hyperplane line giving the written notation of

$$w \bullet x - b = 0 \tag{1.2}$$

This dot product shows the normal vector, and x is the point of the object. There is an offset of the hyperplane that goes from the origin to the normal vector. As the objects are linearly separable, we can create another two hyperplanes – edge hyperplanes – that define the offset on either side of the main hyperplane. There are no objects within the region that spans between the main hyperplane and the edge hyperplanes.

On one side, there is the equation

$$w \bullet x - b = 1 \tag{1.3}$$

and on the other side there is

$$w \bullet x - b = -1v \tag{1.4}$$

The objects that lie on the edge hyperplanes are the support vectors.

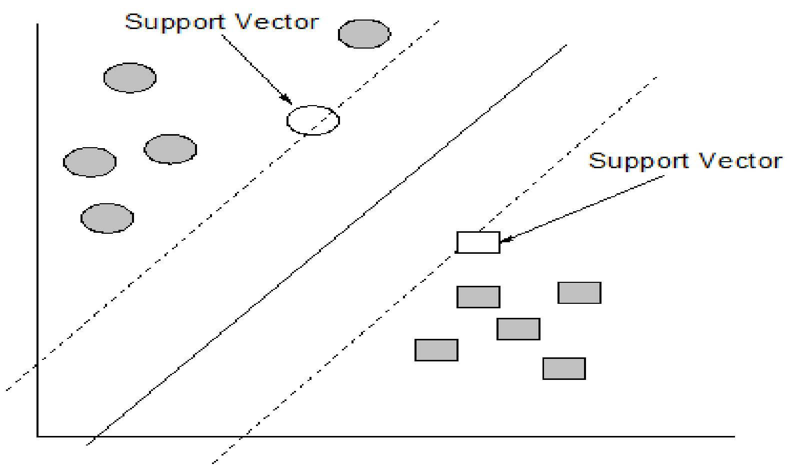


Figure 1.2 The Support Vectors on the hyperplane edges

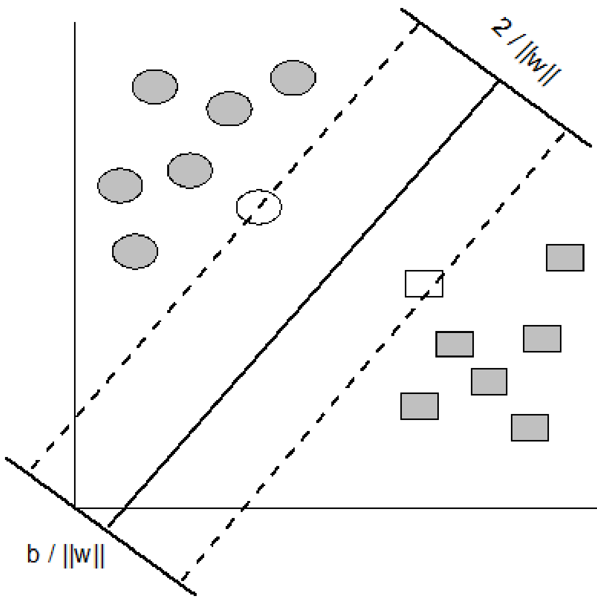


Figure 1.3 Support vector machines max margin hyperplane

When new objects are added to the classification, then the hyperplane and its edges might move. The key objective is to ensure a maximum margin between the +1 edge hyperplane and the -1 edge hyperplane. If we can manage to keep a big gap between the categories, then there is an increase in confidence in your predictions. Knowing the values of the hyperplane edges gives you a feel for how well your categories are separated.

After minimizing the value *w* (called ||w|| in mathematical notation), we can look at optimizing *w* by applying the following equation:

$$\frac{1}{2}\ ||w||^2 \qquad (1.5)$$

Basically, we are taking half of ||w|| squared instead of using the square root of ||w||. Based on Lagrange multipliers, to find the maxima and minima in the function, we can now look for a saddle point and discount other points that do not match zero. We are shaping the graph into a multidimensional space and seeing where the vectors lie in order to make the category distinctions as big as possible. With standard quadratic programming, we then apply the function expressing the training vectors as a linear combination

$$w = \sum_{i=1}^{n} \alpha_i\, y_i x_i \qquad (1.6)$$

Where α_i is greater than zero, the x_i value is a support vector (Bell, 2013)

1.3.2 Non-Linear Classification

In an ideal world, the objects would lie on one side of the hyperplane or the other. Life, unfortunately, is rarely like that. Instead, we see objects straying from the hyperplane, as shown in Figure 1.4

By applying the kernel function, we can apply an algorithm to fit the hyperplane’s maximum margin in a feature space. The method is very similar to the dot products discussed in the linear methods, but this replaces the dot product with a kernel function. With a radial basis function, we have a few kernel types to choose from: the hyperbolic tangent, Gaussian radial basis function (or RBF, which is supported in Weka), and two polynomial functions-one homogenous and the other inhomogeneous.

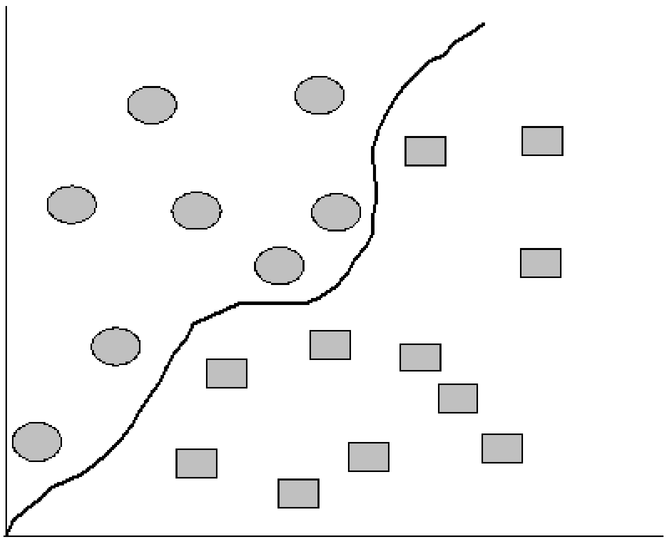


Figure 1.4 Linearly inseparable and linearly separable inputs

1.3.3 Maximizing and Minimizing to Find the Line

Using a log function that is always increasing maximizes values that are above the equation. So, we end up with a function written as

$$\sum_{i=1}^{N} -\log(1 + \exp(-l_i(ax_i + by_i + c))) \qquad (1.7)$$

To achieve minimization, we just multiply the equation by -1. It then becomes a “cost” or “lost” function. The goal is to find line parameters that minimize this function (Bell, 2013).

1.3.4 SVM for Classification

The goal of SVM is to produce a model which predicts target value of data instance in the testing set which are given only the attributes. Classification in SVM is an example of Supervised Learning. Known labels help indicate whether the system is performing in a right way or not. This information points to a desired response, validating the accuracy of the system, or be used to help the system learn to act correctly. A step in SVM classification involves identification as which are intimately connected to the known classes. This is called feature selection or feature extraction. Feature selection and SVM classification together have a use even when prediction of unknown samples is not necessary. They can be used to identify key sets which are involved in whatever processes distinguish the classes (Cristianini and Shawe-Taylor, 2000).

1.3.5 SVM for Regression

SVM's can also be applied to regression problems by the introduction of an alternative loss function. The loss function must be modified to include a distance measure. The regression can be linear and non linear. Linear models mainly consist of the following loss functions, e-intensive loss function, quadratic and Huber loss function. Similarly to classification problems, a non linear model is usually required to adequately model data. In the same manner as the non linear SVC approach, a non linear mapping can be used to map the data into a high dimensional feature space where linear regression is performed. The kernel approach is again employed to address the curse of dimensionality. In the regression method there are considerations based on prior knowledge of the problem and the distribution of the noise. In the absence of such information Huber's robust loss function, has been shown to be a good alternative (Corina et al., 1995)

1.3.6 Applications of SVM

There are standard choices such as a Gaussian or polynomial kernel that are the default options, but if these prove ineffective or if the inputs are discrete structures more elaborate kernels will be needed. By implicitly defining feature space, the kernel provides the description language used by the machine for viewing the data. Once the choice of kernel and optimization criterion has been made the key components of the system are in place (Nello et al., 2000). The first real world task on which Support Vector Machines were tested was the problem hand written character recognition. Furthermore, multi-class SVM's have been tested on these data. It is interesting not only to compare SVM's with other classifiers, but also to compare different SVM's amongst themselves (Mark et al., 1996)

CHAPTER TWO
METHODOLOGY

2.1 Preprocess

2.1.1 Preparing the Training and Test Data

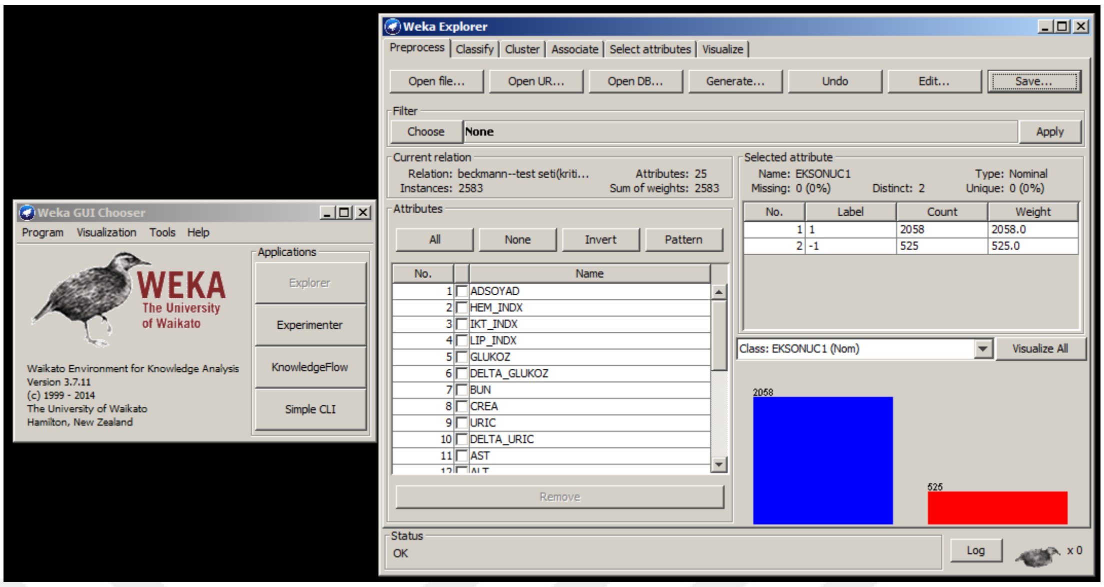
The raw data has the blood tests of 4550 patients which were made between 01.01.2013-30.06.2013 and 500 attributes (Name, Age, Sex, Glucose, K… etc.). Then the raw data is evaluated by FileMaker 8.5Pro (US) software (Nil et al., 2014).

2.2 Weka Software

Weka is open source software and set of tools for pre-processing, classification, regression, clustering, association rules, and for visualization data files. System is written in Java programming language which enables to work with different kind of filters and classifiers, also allows to experiment the data sets with data mining and machine learning algorithms.

Includes a very vide area of preprocessing tools. Users can very efficiently learn how to use methods and algorithms, other good side of the software is that users can experiment more than one classifier and different type of methods in one data set for identifying the most appropriate classifier. Clustering algorithms in weka are specified with no class attribute but classifiers like SVM algorithms are specified with class attributes.

The main GUI (graphical user interface) in weka is the explorer, with a six different panels. The most used part of weka's GUI the explorer is shown in the following Figure 2.1.


Weka GUI Chooser
Program Visualization Tools Help
WEKA
The University of Waikato
Applications
Explorer
Experimenter
KnowledgeFlow
Simple CLI
Weka Explorer
Preprocess Classify Cluster Associate Select attributes Visualize
Open file... Open URL... Open DB... Generate... Undo Edit... Save...
Filter
Choose
Apply
All None Invert Pattern
Remove
Visualize All
Log
x 0

Waikato Environment for Knowledge Analysis
Version 3.7.11
(c) 1999 - 2014
The University of Waikato
Hamilton, New Zealand

Filter: None

Current relation
Relation: beckmann--test seti(kriti... Attributes: 25
Instances: 2583 Sum of weights: 2583

Selected attribute
Name: EKSONUC1 Type: Nominal
Missing: 0 (0%) Distinct: 2 Unique: 0 (0%)

No.	Label	Count	Weight
1	1	2058	2058.0
2	-1	525	525.0

Attributes

No.	Name
1	ADSOYAD
2	HEM_INDX
3	IKT_INDX
4	LIP_INDX
5	GLUKOZ
6	DELTA_GLUKOZ
7	BUN
8	CREA
9	URIC
10	DELTA_URIC
11	AST
12	ALT

Class: EKSONUC1 (Nom)

2058

525

Status
OK

Figure 2.1 Shows Weka's GUI

2.2.1 Preparing ARFF Data File

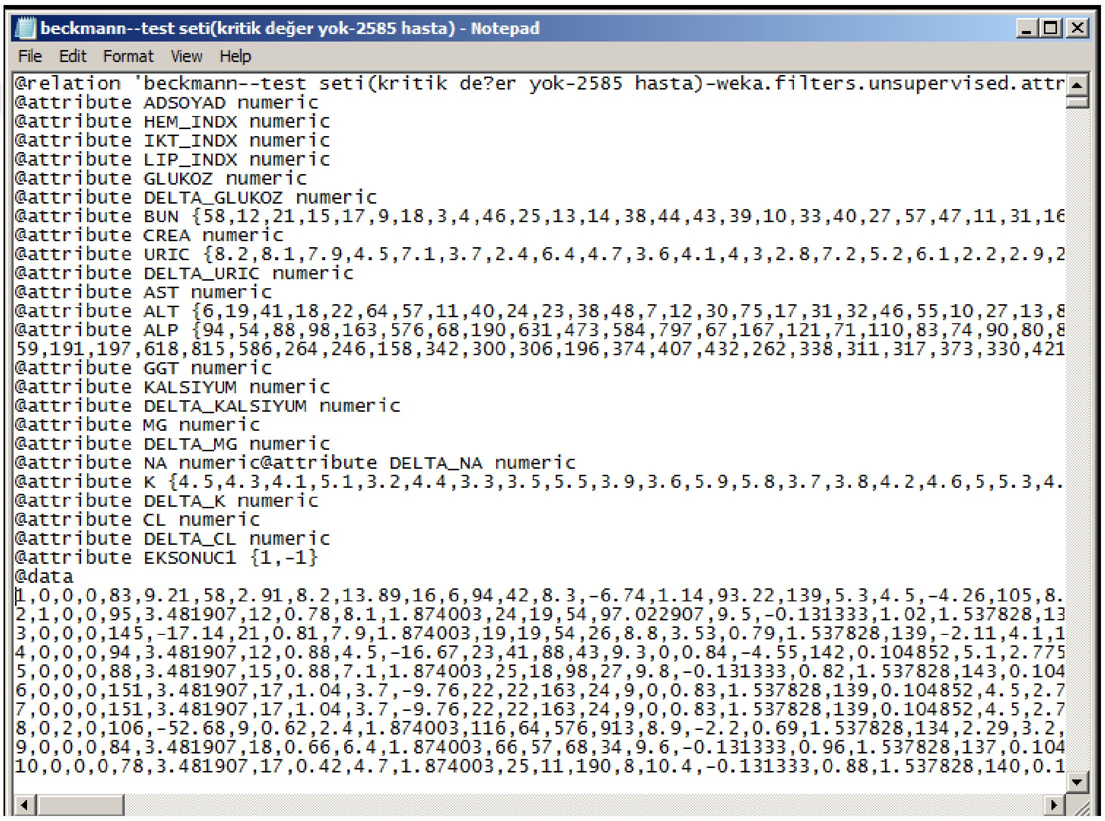
In ARFF (Attribute Relation File Format) file there is three sections. The first section is called the Relation section and every line begins with "@" sign this is how weka deals with an ARFF file in order to understand the sections. After "@Relation" for naming our data file name we can use any other name in order to define our data file relation, it does not have to be same as data file name.

The second section is an attribute section, this section actually is holding the data attributes also starts with "@" sign. We can see it in the second line we have to write "@attribute" for every feature whereas we have to define every attribute with data type.

Our class attribute is called Onay/Red (eksonuc1) where have values negative one (-1) and positive one (1) for classifying our patients in order to make decision which patients are for examination. In general features section or attributes section in our data file we can have either nominal types of data type or numeric types of features.

The third section of data file is the data section, also starts with “@” sign written as “@data”. Weka starts to populate the algorithm inputs from third section, later on we will explain our SVM algorithms how data results are dealing with each section or part of the algorithm.

We have to maintain order of the features in our data section just like the way we have ordered our instances in attribute section. The first attribute is ADSOYAD, so in data section our first result has to be compatible with our attribute.


beckmann--test seti(kritik değer yok-2585 hasta) - Notepad
File Edit Format View Help

```
@relation 'beckmann--test seti(kritik de?er yok-2585 hasta)-weka.filters.unsupervised.attr
@attribute ADSOYAD numeric
@attribute HEM_INDX numeric
@attribute IKT_INDX numeric
@attribute LIP_INDX numeric
@attribute GLUKOZ numeric
@attribute DELTA_GLUKOZ numeric
@attribute BUN {58,12,21,15,17,9,18,3,4,46,25,13,14,38,44,43,39,10,33,40,27,57,47,11,31,16
@attribute CREA numeric
@attribute URIC {8.2,8.1,7.9,4.5,7.1,3.7,2.4,6.4,4.7,3.6,4.1,4,3,2.8,7.2,5.2,6.1,2.2,2.9,2
@attribute DELTA_URIC numeric
@attribute AST numeric
@attribute ALT {6,19,41,18,22,64,57,11,40,24,23,38,48,7,12,30,75,17,31,32,46,55,10,27,13,8
@attribute ALP {94,54,88,98,163,576,68,190,631,473,584,797,67,167,121,71,110,83,74,90,80,8
59,191,197,618,815,586,264,246,158,342,300,306,196,374,407,432,262,338,311,317,373,330,421
@attribute GGT numeric
@attribute KALSIYUM numeric
@attribute DELTA_KALSIYUM numeric
@attribute MG numeric
@attribute DELTA_MG numeric
@attribute NA numeric@attribute DELTA_NA numeric
@attribute K {4.5,4.3,4.1,5.1,3.2,4.4,3.3,3.5,5.5,3.9,3.6,5.9,5.8,3.7,3.8,4.2,4.6,5,5.3,4.
@attribute DELTA_K numeric
@attribute CL numeric
@attribute DELTA_CL numeric
@attribute EKSONUC1 {1,-1}
@data
1,0,0,0,83,9.21,58,2.91,8.2,13.89,16,6,94,42,8.3,-6.74,1.14,93.22,139,5.3,4.5,-4.26,105,8.
2,1,0,0,95,3.481907,12,0.78,8.1,1.874003,24,19,54,97.022907,9.5,-0.131333,1.02,1.537828,13
3,0,0,0,145,-17.14,21,0.81,7.9,1.874003,19,19,54,26,8.8,3.53,0.79,1.537828,139,-2.11,4.1,1
4,0,0,0,94,3.481907,12,0.88,4.5,-16.67,23,41,88,43,9.3,0,0.84,-4.55,142,0.104852,5.1,2.775
5,0,0,0,88,3.481907,15,0.88,7.1,1.874003,25,18,98,27,9.8,-0.131333,0.82,1.537828,143,0.104
6,0,0,0,151,3.481907,17,1.04,3.7,-9.76,22,22,163,24,9,0,0.83,1.537828,139,0.104852,4.5,2.7
7,0,0,0,151,3.481907,17,1.04,3.7,-9.76,22,22,163,24,9,0,0.83,1.537828,139,0.104852,4.5,2.7
8,0,2,0,106,-52.68,9,0.62,2.4,1.874003,116,64,576,913,8.9,-2.2,0.69,1.537828,134,2.29,3.2,
9,0,0,0,84,3.481907,18,0.66,6.4,1.874003,66,57,68,34,9.6,-0.131333,0.96,1.537828,137,0.104
10,0,0,0,78,3.481907,17,0.42,4.7,1.874003,25,11,190,8,10.4,-0.131333,0.88,1.537828,140,0.1
```

Figure 2.2 Shows Arff data file

2.2.2 *Correlation Attribute Evaluator*

Correlation Attribute Evaluator calculates the correlation coefficient what we refer to as our 25 attributes, 1635 for training, 2585 for testing the instances for correlation of the coefficients.

$$r = \frac{\frac{1}{N-1}(\Sigma XY - NXY)}{S_x S_y} \tag{2.1}$$

In this case we have 2585 patients so N is equal to 2585, X and Y‘s are our attributes, this X and Y‘s were calculated by laboratory machines. We also needed to know the mean squared error and standard deviation for our attributes.

Within the parenthesis we can see that, there is two major calculations, first one is the sum of X and Y this means that we take the sum of all the X and Y’s multiplied together and add them together , after that algorithm will subtract the summed values. Second one in the parenthesis N multiplies mean of X times mean of Y.

In the denominator on the bottom of the equation we can see S of x and S of y which means that the standard deviation of all attributes multiplied together. Results shown in table one are correlation attribute evaluator with 12 Cross-validation for training the dataset.

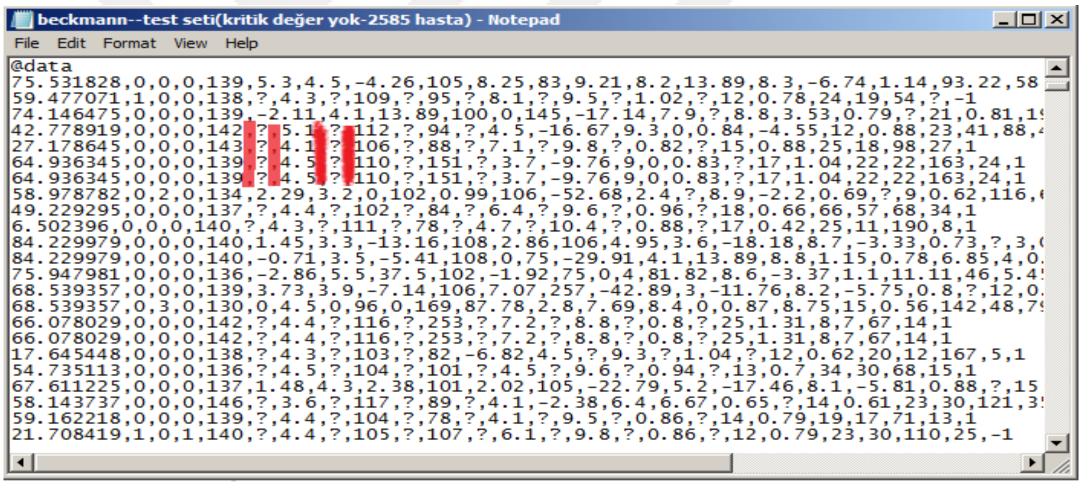
Table 2.1 Shows the result of correlation attribute evaluator

Average	**Merit**	**Average**	**Rank**	**Attribute**
0.746	0.007	1	0	1 HEM_INDX
0.315	0.009	2	0	10 GLUKOZ
0.168	0.006	3	0	3 LIP_INDX
0.135	0.003	4	0	12 URIC
0.116	0.002	5.2	0.37	14 KALSIYUM
0.106	0.011	6.8	1.53	11 DELTA_GLUKOZ
0.106	0.008	6.8	1.14	2 IKT_INDX
0.096	0.004	8	0.82	23 GGT
0.09	0.008	9.1	0.95	5 DELTA_NA
0.082	0.006	10.4	1.11	17 DELTA_MG
0.079	0.009	11.3	1.88	7 DELTA_K
0.07	0.002	12.3	0.75	20 AST
0.072	0.009	12.3	2.36	22 ALP
0.066	0.002	13.8	0.72	4 NA
0.063	0.002	15	0.91	21 ALT
0.057	0.009	15.9	2.02	15
0.054	0.001	16.8	0.69	DELTA_KALSIYUM
0.047	0.006	18	1.22	8 CL
0.044	0.001	18.8	0.37	13 DELTA_URIC
0.036	0.002	20	0	6 K
0.025	0.001	21.1	0.28	18 BUN
0.015	0.015	21.6	2.02	19 CREA
0.007	0.005	22.8	0.37	9 DELTA_CL
				16 MG
Total Instances:	**1635**			

2.2.3 Replace Missing Values Filter

When the blood test machine outputs the results for each instance and when that results are replaced in excel form (in our situation we worked with excel files) there can be seen in some colons missing values for instances. Handling that problem which is very significant for accuracy of our learning system we could replace it with just replacing the question marks with nothing in notepad, furthermore we used filtering to handle this problem.
Where the explanation for replacing the missing values is:

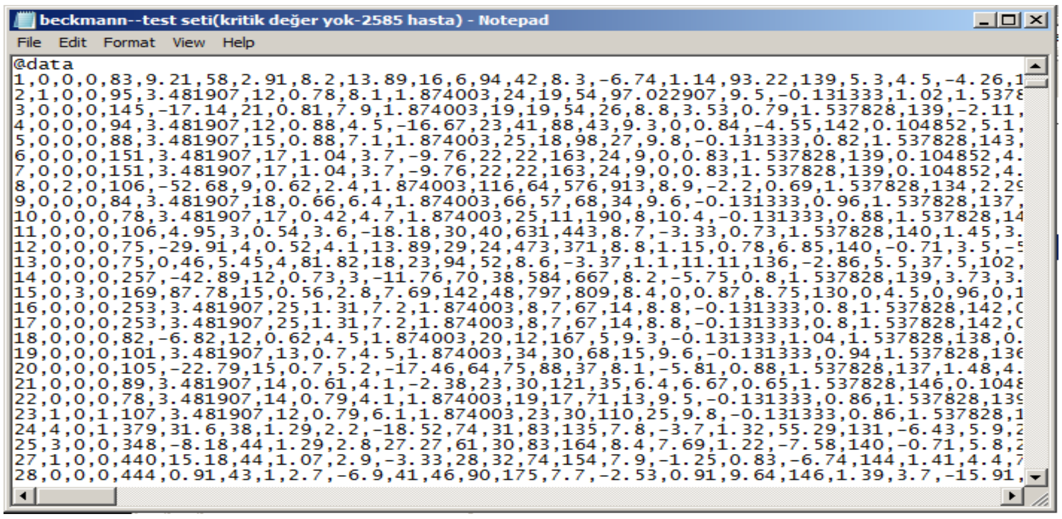
- Replaces all missing values for nominal and numeric attributes in a dataset with the modes and means from the training data.



beckmann--test seti(kritik değer yok-2585 hasta) - Notepad
File Edit Format View Help

```
@data
75.531828,0,0,0,139,5.3,4.5,-4.26,105,8.25,83,9.21,8.2,13.89,8.3,-6.74,1.14,93.22,58
59.477071,1,0,0,138,?,4.3,?,109,?,95,?,8.1,?,9.5,?,1.02,?,12,0.78,24,19,54,?,-1
74.146475,0,0,0,139,-2.11,4.1,13.89,100,0,145,-17.14,7.9,?,8.8,3.53,0.79,?,21,0.81,1
42.778919,0,0,0,142,?,5.1,?,112,?,94,?,4.5,-16.67,9.3,0,0.84,-4.55,12,0.88,23,41,88,
27.178645,0,0,0,143,?,4.1,?,106,?,88,?,7.1,?,9.8,?,0.82,?,15,0.88,25,18,98,27,1
64.936345,0,0,0,139,?,4.5,?,110,?,151,?,3.7,-9.76,9,0,0.83,?,17,1.04,22,22,163,24,1
64.936345,0,0,0,139,?,4.5,?,110,?,151,?,3.7,-9.76,9,0,0.83,?,17,1.04,22,22,163,24,1
58.978782,0,2,0,134,2.29,3.2,0,102,0.99,106,-52.68,2.4,?,8.9,-2.2,0.69,?,9,0.62,116,
49.229295,0,0,0,137,?,4.4,?,102,?,84,?,6.4,?,9.6,?,0.96,?,18,0.66,66,57,68,34,1
6.502396,0,0,0,140,?,4.3,?,111,?,78,?,4.7,?,10.4,?,0.88,?,17,0.42,25,11,190,8,1
84.229979,0,0,0,140,1.45,3.3,-13.16,108,2.86,106,4.95,3.6,-18.18,8.7,-3.33,0.73,?,3,
84.229979,0,0,0,140,-0.71,3.5,-5.41,108,0,75,-29.91,4.1,13.89,8.8,1.15,0.78,6.85,4,0.
75.947981,0,0,0,136,-2.86,5.5,37.5,102,-1.92,75,0,4,81.82,8.6,-3.37,1.1,11.11,46,5.4
68.539357,0,0,0,139,3.73,3.9,-7.14,106,7.07,257,-42.89,3,-11.76,8.2,-5.75,0.8,?,12,0.
68.539357,0,3,0,130,0,4.5,0,96,0,169,87.78,2.8,7.69,8.4,0,0.87,8.75,15,0.56,142,48,7
66.078029,0,0,0,142,?,4.4,?,116,?,253,?,7.2,?,8.8,?,0.8,?,25,1.31,8,7,67,14,1
66.078029,0,0,0,142,?,4.4,?,116,?,253,?,7.2,?,8.8,?,0.8,?,25,1.31,8,7,67,14,1
17.645448,0,0,0,138,?,4.3,?,103,?,82,-6.82,4.5,?,9.3,?,1.04,?,12,0.62,20,12,167,5,1
54.735113,0,0,0,136,?,4.5,?,104,?,101,?,4.5,?,9.6,?,0.94,?,13,0.7,34,30,68,15,1
67.611225,0,0,0,137,1.48,4.3,2.38,101,2.02,105,-22.79,5.2,-17.46,8.1,-5.81,0.88,?,15
58.143737,0,0,0,146,?,3.6,?,117,?,89,?,4.1,-2.38,6.4,6.67,0.65,?,14,0.61,23,30,121,3
59.162218,0,0,0,139,?,4.4,?,104,?,78,?,4.1,?,9.5,?,0.86,?,14,0.79,19,17,71,13,1
21.708419,1,0,1,140,?,4.4,?,105,?,107,?,6.1,?,9.8,?,0.86,?,12,0.79,23,30,110,25,-1
```

Figure 2.3 Shows dataset with question marks.


beckmann--test seti(kritik değer yok-2585 hasta) - Notepad
File Edit Format View Help

```
@data
1,0,0,0,83,9.21,58,2.91,8.2,13.89,16,6,94,42,8.3,-6.74,1.14,93.22,139,5.3,4.5,-4.26,1
2,1,0,0,95,3.481907,12,0.78,8.1,1.874003,24,19,54,97.022907,9.5,-0.131333,1.02,1.537
3,0,0,0,145,-17.14,21,0.81,7.9,1.874003,19,19,54,26,8.8,3.53,0.79,1.537828,139,-2.11,
4,0,0,0,94,3.481907,12,0.88,4.5,-16.67,23,41,88,43,9.3,0,0.84,-4.55,142,0.104852,5.1,
5,0,0,0,88,3.481907,15,0.88,7.1,1.874003,25,18,98,27,9.8,-0.131333,0.82,1.537828,143,
6,0,0,0,151,3.481907,17,1.04,3.7,-9.76,22,22,163,24,9,0,0.83,1.537828,139,0.104852,4.
7,0,0,0,151,3.481907,17,1.04,3.7,-9.76,22,22,163,24,9,0,0.83,1.537828,139,0.104852,4.
8,0,2,0,106,-52.68,9,0.62,2.4,1.874003,116,64,576,913,8.9,-2.2,0.69,1.537828,134,2.2
9,0,0,0,84,3.481907,18,0.66,6.4,1.874003,66,57,68,34,9.6,-0.131333,0.96,1.537828,137,
10,0,0,0,78,3.481907,17,0.42,4.7,1.874003,25,11,190,8,10.4,-0.131333,0.88,1.537828,14
11,0,0,0,106,4.95,3,0.54,3.6,-18.18,30,40,631,443,8.7,-3.33,0.73,1.537828,140,1.45,3.
12,0,0,0,75,-29.91,4,0.52,4.1,13.89,29,24,473,371,8.8,1.15,0.78,6.85,140,-0.71,3.5,-
13,0,0,0,75,0,46,5.45,4,81.82,18,23,94,52,8.6,-3.37,1.1,11.11,136,-2.86,5.5,37.5,102,
14,0,0,0,257,-42.89,12,0.73,3,-11.76,70,38,584,667,8.2,-5.75,0.8,1.537828,139,3.73,3.
15,0,3,0,169,87.78,15,0.56,2.8,7.69,142,48,797,809,8.4,0,0.87,8.75,130,0,4.5,0,96,0,1
16,0,0,0,253,3.481907,25,1.31,7.2,1.874003,8,7,67,14,8.8,-0.131333,0.8,1.537828,142,
17,0,0,0,253,3.481907,25,1.31,7.2,1.874003,8,7,67,14,8.8,-0.131333,0.8,1.537828,142,
18,0,0,0,82,-6.82,12,0.62,4.5,1.874003,20,12,167,5,9.3,-0.131333,1.04,1.537828,138,0.
19,0,0,0,101,3.481907,13,0.7,4.5,1.874003,34,30,68,15,9.6,-0.131333,0.94,1.537828,13
20,0,0,0,105,-22.79,15,0.7,5.2,-17.46,64,75,88,37,8.1,-5.81,0.88,1.537828,137,1.48,4.
21,0,0,0,89,3.481907,14,0.61,4.1,-2.38,23,30,121,35,6.4,6.67,0.65,1.537828,146,0.104
22,0,0,0,78,3.481907,14,0.79,4.1,1.874003,19,17,71,13,9.5,-0.131333,0.86,1.537828,13
23,1,0,1,107,3.481907,12,0.79,6.1,1.874003,23,30,110,25,9.8,-0.131333,0.86,1.537828,1
24,4,0,1,379,31.6,38,1.29,2.2,-18.52,74,31,83,135,7.8,-3.7,1.32,55.29,131,-6.43,5.9,
25,3,0,0,348,-8.18,44,1.29,2.8,27.27,61,30,83,164,8.4,7.69,1.22,-7.58,140,-0.71,5.8,
27,1,0,0,440,15.18,44,1.07,2.9,-3.33,28,32,74,154,7.9,-1.25,0.83,-6.74,144,1.41,4.4,
28,0,0,0,444,0.91,43,1,2.7,-6.9,41,46,90,175,7.7,-2.53,0.91,9.64,146,1.39,3.7,-15.91,
```

Figure 2.4 Shows the dataset after using replace missing values filter.

2.3 Predictions of Performance Measures

Data accuracy will be shown from the experimental tables with correctly and incorrectly classified instances. For defining the accuracy as we said before was used two classes, which is often defined like positive and negative class. Class numbering represents two different categories in which classifier distributes a set of instances.

So that the instances classified as positive can be really positive and false positive. Samples marked as negative can be really classified as true negatives and false negatives.

One of the measures that is using for evaluating the performance prediction is accuracy. The accuracy of classification can be defined as the ratio of correctly classified samples and the total number of samples.

$$\text{Accuracy} = \frac{(\text{TP}+\text{TN})}{(\text{TP}+\text{FN}+\text{FP}+\text{TN})} \tag{2.2}$$

Recall is defined as the ratio of correctly classified positive instances and the instances that the classifier declared positive.

$$\text{recall} = \frac{TP}{(\mathrm{TP+FN})} \tag{2.3}$$

Precision is defined as the ratio of correctly classified positive instances and the instances that the classifier declared as positive.

$$\text{Precission} = \frac{TP}{(TP+FP)} \tag{2.4}$$

F-Measure is a weighted harmonic mean of precision and recall.

$$\mathrm{F-measure} = 2 * \frac{(precision*recall)}{(precision+recall)} \tag{2.5}$$

2.3.1 Roc (Receiver Operating Characteristics)

Roc is a visualization tool by which we can tell that our classifier is really appropriate or not. Roc curve is a two dimensional performance classifier in which on the Y-axis displays true positives (TP) and on the X-axis false positives (FP).This two axis are showing us the relationship between the true positives (TP) and false positives (FP).

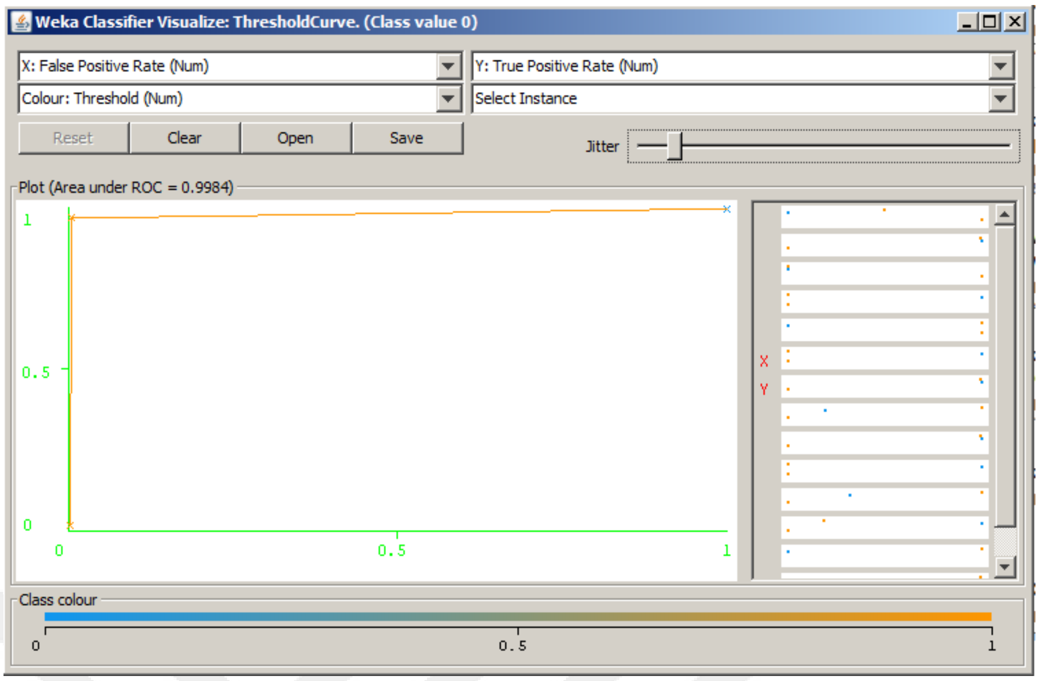

Weka Classifier Visualize: ThresholdCurve. (Class value 0)
X: False Positive Rate (Num)
Y: True Positive Rate (Num)
Colour: Threshold (Num)
Select Instance
Reset
Clear
Open
Save
Jitter
Plot (Area under ROC = 0.9984)
1
0.5
0
0
0.5
1
X
Y
Class colour
0
0.5
1

Figure 2.5 Shows the representation of roc area

2.3.2 Precision Recall

It is using in the case of asymmetry distribution in the class and it is corresponding curve is often used as an alternative to the Receiver Operating Characteristics (ROC) curves. An important difference between Precision recall and ROC space is a visual representation of the curve.

So in ROC curve the main idea is to have the curve much more upper left on the corner, where is the top of the Y-axis which is one. It means that we are using the best classifier for our dataset.

To produce it in a different way we can also see Area under ROC which has 0.9984 percent of accuracy, it has really high percentage for using that classifier.

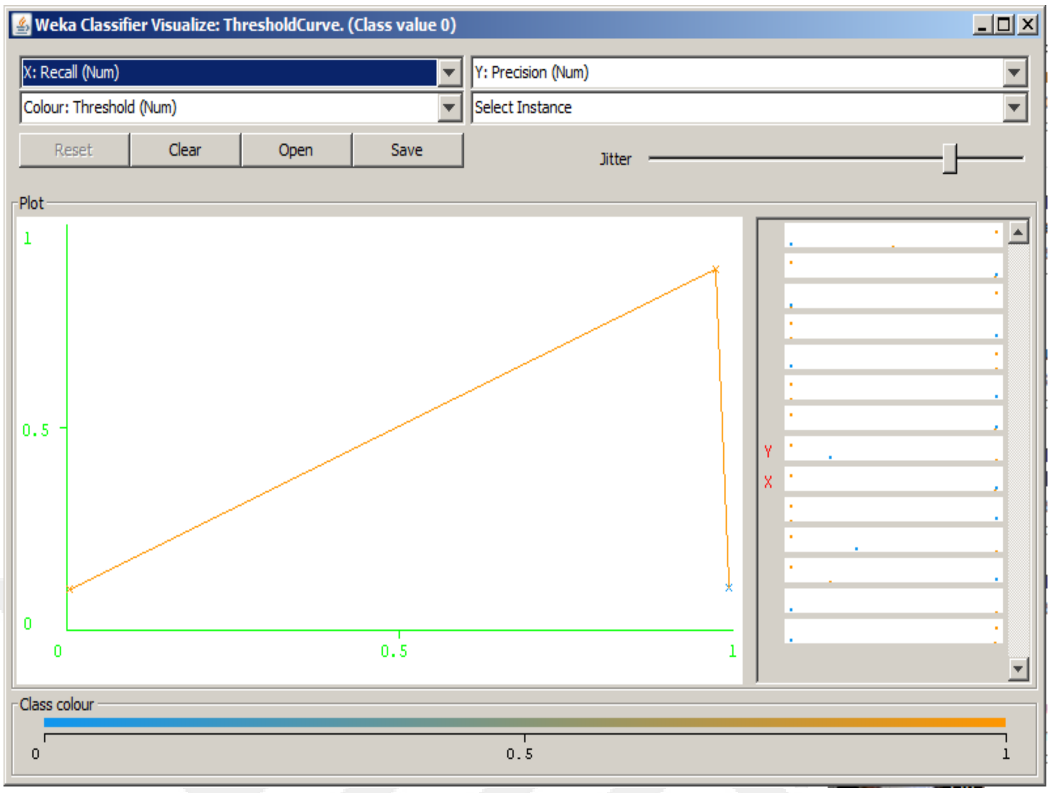


Figure 2.6 Shows the representation curve of precision recall

2.4 Choosing Model for Classification

Our experiment was separated into two parts. First part was experimented by my colleagues with Artificial Neuron Networks (ANN), we had good results also with back propagation algorithm where the ANN results were up to 95 percent of accuracy with well experimented learning rates, momentum and cross validation.

2.4.1 Artificial Neural Network Model

Polynomial classifiers can model decision surfaces of any shape; and yet their practical utility is limited because of the easiness with which they overfit noisy training data, and because of the sometimes impractically high number of trainable parameters. Much more popular are artificial neural networks where many simple units, called neurons, are interconnected by weighted links into larger structures of remarkably high performance (Kubat, 2015).

Feed-forward architecture with sigmoid function often have one or more hidden layer neurons, whereupon is located output layers with a linear function. Multiple layers of neurons with nonlinear transfer function enable the network to learn nonlinear relationships between input and output vectors. Linear output layer is most commonly used for troubleshooting adjustment function. ANN uses back-propagation algorithm for backward the obtained weights and for minimizing the errors uses gradient descent algorithm. For optimizing the architecture network it is used Levenberg-Marquardt optimization technique. It is hard for determine for selecting the neurons in hidden layers although there is formula (Figure 2.9). If the number of the neurons in the hidden layer are not enough then network begins to decline, when the number of neurons are enough for the hidden layer learning process increases and the network becomes ineffective.
Structure of ANN is shown in Figure 2.7

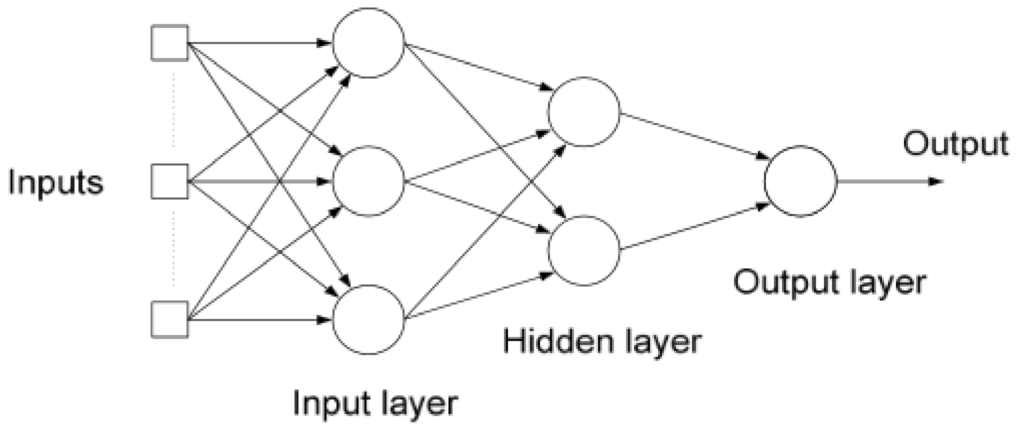


Figure 2.7 Shows Structure of ANN

2.4.1.1 Multilayer Perceptrons as Classifiers

Neurons: The function of a neuron, the basic unit of a multilayer perceptron, is quite simple. A weighted sum of signals arriving at the input is subjected to a transfer function. Several different transfer functions can be used; the one that is preferred is sigmoid, defining by the following formula where $\sum$ is the weighted sum of inputs.

$$f(\sum) = \frac{1}{1+ e^{-\Sigma}} \tag{2.6}$$

Figure 2.8 shows the curve representing the transfer function. The reader can see that $f(\sum)$ grows monotonically with the increasing value of $\sum$ but is destined never to leave the open interval (0, 1) because $f(-\infty)$ = 0, and $f(\infty)$ = 1. The vertical axis is interested at $f(0)$ = 0.5. We will assume that each neuron has the same transfer function.

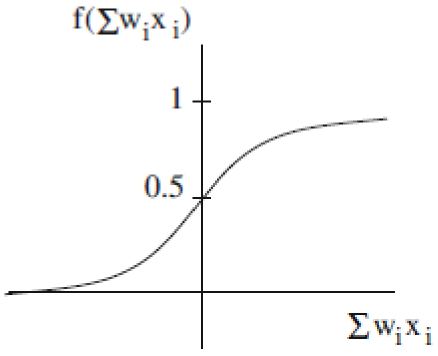


Figure 2.8 Sigmoid transfer function (Kubat, 2015)

Multilayer perceptron. The neural network in figure 2.8 is known as the multilayer perceptron. The neurons, represented by circles, are arranged in the output layer and the hidden layer while remembering that it is quite common to employ two such -layers, even three, though rarely more than that.

While there is no communication between neurons of the same layer, adjacent layers are fully interconnected. Importantly, each neuron-to-neuron link is associated with a weight. The weight of the link from the j-th hidden neuron to the i-th output neuron is denoted as w_{ji}, and the weight of the link from the k-th attribute to the j-th hidden neuron as w_{kj}. First index always refers to the link's "beginning"; the second, to its "end". (Kubat, 2015)

2.4.1.2 Gradient Descent

$$b = a - \gamma\nabla F(a) \qquad (2.7)$$

Equation describes the first steps of Gradient Descent, but before explaining the equation, we will illustrate how gradient descent works.

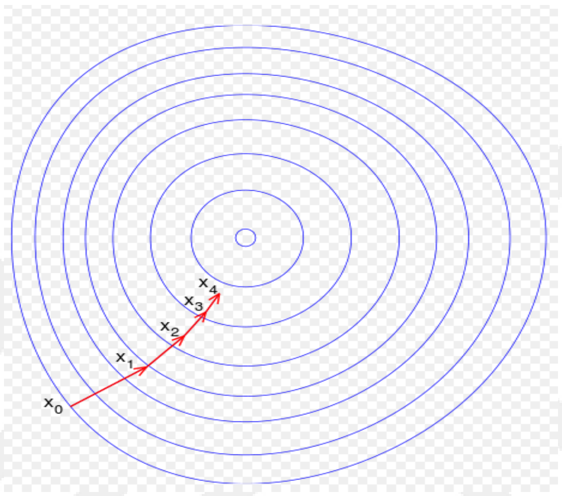


Figure 2.9 Shows starting point of gradient descent

Figure 2.8 shows how gradient vector starts from point local minimum X_0 to the ending point X_4. To arrive in the middle of the gradient it depends exactly of length of the vector. We want to minimize the function before we get overshoot.

From the expression (2.6) b is our current position b is going to be our next position, gradient term tells us the direction of steepest ascent, and gamma is a factor. With this formula is taking the position of gradient descent.
After this step we have to minimize our function.

2.4.1.3 Learning Rate

Learning rate is a parameter for controlling the step size and for updating the weights of the learning system. Where with several experimentations we can comprehend that if we use high learning rate it is obviously that we will have faster learning but it will have big risk of overshooting.

In the other side if we take slow learning rate of values we will have slower learning, but overshooting will be avoided.

In the below table the learning rates of ANNs were experimented.

Table 2.1 Learning rate results (Nil et al., 2014).

Learning Rate	Kappa Statistic	Mean Absolute error	Root Mean squared error	Root absolute error	Root relative sq. error
0.001	0.9329	0.0884	0.1467	28.1944 %	37.067 %
0.005	0.9782	0.0243	0.0821	7.7481 %	20.752 %
0.007	0.9782	0.0199	0.0758	6.3434 %	19.145 %
0.01	0.9803	0.0158	0.0717	5.0532 %	18.802 %
0.05	0.9783	0.0089	0.0734	2.8259 %	18.541 %
0.07	0.9803	0.0082	0.0733	2.6173 %	18.518 %
0.1	0.9783	0.008	0.0747	2.5569 %	18.872 %
0.5	0.9803	0.0078	0.0726	2.4857 %	18.342 %
0.7	0.9764	0.0085	0.0779	2.7015 %	19.682 %
1	0.9783	0.0087	0.0803	2.7759 %	20.282 %

2.4.1.4 Momentum

With respect to proven experiments for momentum term that improves the speed of back-propagation algorithm and at the same time can improve the range of a learning rate. We experimented momentum value to back-propagation algorithm for proving the best accuracy for the learning system.

There are many different varieties of learning algorithms, the majority of them including the popular back-propagation learning algorithm are of the gradient descent type. For a given network architecture, one usually starts with an error function which is parameterized by the weights (the connection strengths between units) in the network. The gradient of the error function with respect to each weight is then computed and the weights are modified along the downhill direction of the gradient in order to reduce the error. Let $E(\mathbf{w})$ be the error function, where **w** is a vector representing all the weights in the network, the simplest gradient descent algorithm known as the steepest descent, modifies the weights at time step *t* according to:

$$\Delta w_t = -\epsilon \nabla_w E(w_t) \tag{2.8}$$

where ∇_w represents the gradient operator with respect to the weights and e is a small positive number known as the learning rate.

It is well known that such a learning scheme can be very slow. The inclusion of a momentum term has been found to increase the rate of convergence dramatically (Rumelhart et al., 1986).

$$\Delta w_t = -\epsilon \nabla_w \mathrm{E(w)} + \rho \Delta \mathrm{w}_{t-1} \tag{2.9}$$

Where p is the momentum parameter. That is, the modification of the weight vector at the current time step depends on both current gradient and the weight change of the previous step. Intuitively, the rationale for the use of the momentum term is that the steepest descent is particularly slow when there is a long narrow valley in the error function surface. In this situation, the direction of the gradient is almost perpendicular to the long axis of the valley. The system thus oscillates back and forth in the direction of the short axis, and only moves very slowly along the long axis of the valley. The momentum term helps average out the oscillation along the short axis while at the same time adds up contributions along the long axis (Rumelhart et al., 1986).

Table 2.2 Momentum results (Nil et al., 2014).

Momentum	Kappa Statistic	Mean Absolute error	Root mean squared error	Relative absolute error	Root relative sq. Error
0.01	**0.9961**	**0.0053**	**0.0341**	**1.6836 %**	**8.6234 %**
0.03	**0.9961**	**0.0052**	**0.0341**	**1.6577 %**	**8.6058 %**
0.04	**0.9961**	**0.0052**	**0.0341**	**1.6447 %**	**8.5971 %**
0.05	**0.9961**	**0.0051**	**0.0341**	**1.6317 %**	**8.5801 %**
0.06	**0.9961**	**0.0051**	**0.0341**	**1.6188 %**	**8.5886 %**
0.07	**0.9961**	**0.005**	**0.0339**	**1.6058 %**	**8.5917 %**
0.08	**0.9961**	**0.005**	**0.0339**	**1.5928 %**	**8.6035 %**
0.09	**0.9961**	**0.005**	**0.0339**	**1.5798 %**	**8.6053 %**
0.1	**0.9961**	**0.0049**	**0.0338**	**1.5668 %**	**8.6152 %**
0.5	**0.9961**	**0.0032**	**0.0333**	**1.0361 %**	**8.6195 %**
1	**0**	**0.1945**	**0.441**	**62.0277 %**	**111.4207 %**

2.4.1.5 Hidden Layer

Neural network consists of three layers with neurons. Layer which are referred as inputs, hidden layer neurons and output.

Each neuron in the entrance layer is connected with all neurons in the output layer over the weight coefficients. With input and hidden neurons ANN architecture also includes a bias nodes with given value size +1 in the input and hidden layer.

On the other side, the number of hidden neurons subject to adjustment depending on the number of elements, such as desired approximation and the possibility of generalization of the model.

For deciding the number of hidden layers and nodes in a hidden layer, we need to use cross-validation for testing the accuracy on the training set. Because there is no rule or formula for deciding the number of hidden layers like number of neurons in the hidden layer.

$$\text{number of neurons in the hidden layer} = \frac{\text{number of inputs} + \text{number of classes}}{2} \qquad (3.0)$$

We decide to experiment different number of hidden layers with different number of neurons in each hidden layer (results will be explained in result chapter). We were aware that our ANN architecture needs more than one hidden layer and actually results proved it.

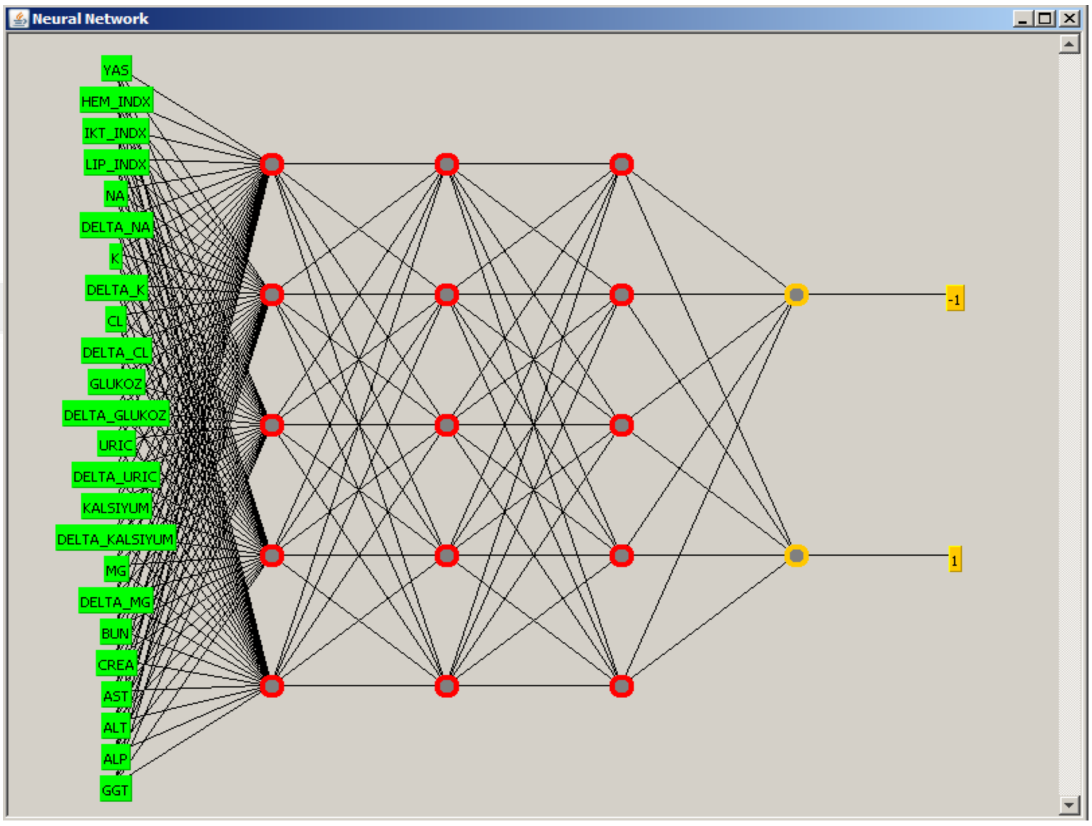


Figure 2.9 Shows neural network with 15 neurons in 3 hidden layers

2.4.2 K-Fold Cross Validation

K-fold cross-validation is a statistical technique used to estimate the generalization and prediction of the model. Actually, how will the results of the model predict the new generalized set of data.

The input data is divided into subsets complementary with the data set to train one subset to build a model to predict, and the subset of model is a test data for validation in k-fold cross validation. Input data set is dividing by the principle of random selection in the subset to a variable parameter and the most commonly used testing is 10-fold cross validation, but sometimes depends on number of instances in related dataset. For modeling the subsets, one subset is using for model validation that is test data set and the remaining k-1 subset is using to build the model for data set for training.

Process for cross validation is repeating k-times, where in each of the subsets is using only once for model validation. Results of k-fold cross validation is the average value of individual k-results. The advantage of this technique is that in the same time all instances in the subset are used for the construction of the model and for validation of the model.

2.4.3 LibSVM

LibSVM method is an efficient implementation of a Support vector machines for classification, regression and distribution estimation. LibSVM supports multi-class classification.

LibSVM method includes:

- Efficient multi-class classification
- Cross-validation for model selection
- Probability estimates
- Various kernels
- Weighted SVM for unbalanced data

In this experiment will be used radial basis function (RBF) because the analyzing produces good results where can be used more complex kernel type functions:

- Linear function
- Polynomial
- Radial basis function

- Sigmoid

For finding the ideal parameters coef0 and gamma is written in programming language Python (Support Vector Machines 2011). Script for finding the ideal parameters is doing a very easy process where script runs LibSVM with default values for coef0 and gamma. It performs a process of cross-validation(see section Cross-validation) where in input data selects a number of instances that are used as a test set and the rest of the instances for training set. After that it is performing cross-validation and training of the set. When the process is completed from the total set it is selecting the second test set, same size as the previous test set but with new variables.

The number of the process iterations is depending on cross-validation. After completing the process of cross-validation it is obtaining certain results. After that it is changing parameters for coef0 and gamma with and is performing again cross-validation for the new obtained results for parameters coef0 and gamma. The obtained results are compared with the old ones, if the results obtained are better than the oldest obtained results better parameters used in this process are taken as ideal parameters.

The difference is that for learning and testing with ideal parameters may not result the best obtained, but when is testing trained data set with ideal parameters and with new input variables can be obtained best results.

2.4.4 SPegasos

Implements the stochastic variant of the Pegasos (Primal Estimated sub-Gradient Solver for SVM) method of Shalev-Shwartz. This implementation globally replaces all missing values and transforms nominal attributes into binary ones. It also normalizes all attributes, so the coefficients in the output are based on the normalized data (Hall, 2011).

2.4.5 Sequential Minimal Optimization (SMO)

In this experiment we used the new SVM's implemented training algorithm called Sequential Minimal Optimization.
Differences between the standard SVM training algorithms and the SMO:

- Conceptually simple
- Faster
- Better scaling properties

As an inner loop that use standard SVM algorithms (numerical quadratic programming), SMO uses analytic quadratic programming.

SMO can very fast solve SVM's quadratic problem, where SMO optimizes the large quadratic programming (QP) problem into smaller QP problems. A very big advantage of SMO is that can solve the Lagrange multipliers to optimize then finds the optimal values and updates the SVM to reflect the new variables.

SMO solves effectively:
- Two Lagrange multipliers
- Heuristic for choosing which multipliers to optimize

Osuna's theorem states that the global training problem can be broken down into a sequence of smaller sub-problems (Platt, 1998).

As long as SMO always optimizes and alters two Lagrange multipliers at every step and at least one of the Lagrange multipliers violated the KKT conditions before the step, then each step will decrease the objective function according to Osuna's theorem (Freund, 1997).

So far, of the SVM training methods have the same flavor: since solving the underlying QP problem outright is inefficient, they break down the problem in one way or another, and then solve the QP tasks of these carefully chosen subproblems. In doing so there is a balance. The more sophisticated our decomposition algorithm (resulting in smaller subproblems), themore time is spent iterating over subproblems, and the less assurance we have of good convergence rates. The benefit, of course, is that less memory is required in solving the QP subproblems, and that we can worry less about the numerical instability of QP packages. On the other hand, the less “fiddling” we do in the way of decomposition, the more “intact” our problem remains, and the more assurance we have in the program’s outputting a global optimum. Indeed, the reliance on QP packages is bothersome. As mentioned before, numerical instability is a concern for QP solvers. And it is a little unsettling knowing that most QP routines are too complex and difficult to be implemented “on the fly” by the user. For this reason, the method of Sequential Minimal Optimization (SMO), given in (Platt, 1999), is very appealing.

Platt takes decomposition to the extreme by only allowing a working set of size 2. Solving a QP problem of size 2 can be done analytically, and so this method avoids the use of a numerical QP solver altogether. The tradeoff, naturally, is that pairs of examples optimized in this way must be iterated over many times. The claim, however, is that since the heart of the algorithm is just a simple analytic formula, the overall runtime is reduced. The biggest advantage of this method is that the derivation and implementation are astoundingly simple.

It should be noted that SMO, while introduced as a method for training SVM classifiers, has also been extended to SVM regression (Flake, Lawrence, 2000). Also, (Keerthi et al. 1999) argue that Platt’s method of computing the threshold in SMO is inefficient, and give modifications to his pseudocode that make the algorithm much faster (Bosswell, 2002).

CHAPTER THREE
RESULTS

3.1 Biochemistry Test Results

How we explained before for the training and testing data sets were achieved from the ANN experiment, which data sets were deeply analyzed by biochemistry specialists and assistants. For making best decisions analyzing and experimenting the predicted values which can directly effect our learning models were choosing critical values.

For validation of blood test results we used typical binary classification, with values -1 and +1 respectively. Determination of Gray Zone is important part of this experiment, with limit values from -0.3650 to +0.6833. Predicted values which are smaller than -0.3650 biochemistry specialists can accept as invalid test result otherwise if predicted values are greater than +0.6833 the results are acceptable.

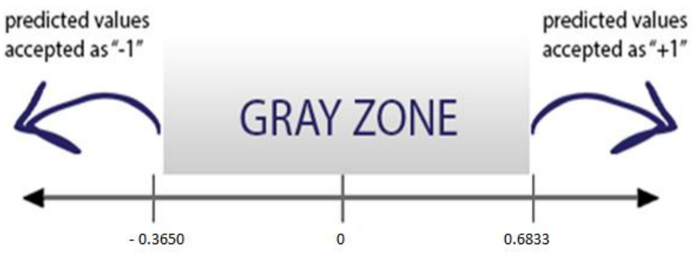

predicted values accepted as "-1"
predicted values accepted as "+1"
GRAY ZONE
- 0.3650
0
0.6833

Figure 3.1 Shows gray zone for prediction of values.

3.2 ANN for Hidden Layer Results

In this experiment we experimented the various of combination of neurons in hidden layers, more clearly we wanted to find out if we use more neurons in different hidden layers or more neurons in a single layer in to our learning system how will respond to accuracy and to statistic metrics. We were aware that if we increase the number of hidden layers and neurons, execution times of the model would be slower, where in the beginning of the experiment since the model was experimented with just five neurons and one hidden layer time taken for it was just for 7.2 seconds. Increasing the number of hidden layers and neurons up to three hidden layers and twenty-five neurons time taken for execution was 23.11 seconds for cross-validation.

Table 3.1 Results of ANN with hidden layers

Hidden Layers	Neurons	Kappa statistic	Mean absolute error	Root mean squared error	Correctly Class. Instances
1	5	0.828	0.0808	0.2104	94.0852 %
2	10	0.8771	0.0529	0.1936	95.7017 %
3	15	0.8609	0.0708	0.2008	95.1874 %
3	20	0.8682	0.0535	0.1934	95.4078 %
3	25	0.8744	0.0621	0.1897	95.3711 %

Because of in the first layer of the network input data variables are multiplied by the weights and is adding the bias, if the input data variables are large amount weights must be very small. Therefore before applying the input variables to a network it is using normalization to the input data variables. Generally normalization is applied to the input and to the target vectors. In this way the results of the network is always in normalized range. The data set is entered with 1635 instances for training and 2585 for testing with 25 attributes for both sets.

In this experiment it is used 0.05 value for learning and momentum and 12cross-validation. In the Table 4 results shows that with two hidden layers and ten neurons our learning model obtained best accuracy, where and kappa statistic is very high close to one with 0.8771 percentage.

Table 3.2 Results of ANN with hidden layers

Hidden Layers	Neurons	Kappa statistic	Mean absolute error	Root mean squared error	Correctly Class. Instances
1	5	0.0233	0.2032	0.4408	79.9458 %
2	10	0.0614	0.1953	0.4388	80.4491 %
3	15	0.0381	0.204	0.4349	80.1394 %
3	20	0.0598	0.1977	0.4387	80.3717 %
3	25	0.077	0.1983	0.4358	80.6039 %

Table 3.2 shows that with increasing the number of hidden layers and neurons the accuracy is increasing (purely not with three hidden layers and fifteen neurons network).

3.3 SVM Results

3.3.1 Cross-Validation Results

During training of SVM algorithms for optimal training is necessary to perform the process of cross-validation.
Cross Validation divides data into three parts:

- Training
- Validation
- Test sets

After dividing to three parts Cross Validation finds the optimal model on the training set. That optimal training model uses the test set for determining its optimal competence.

For estimating the model was used 10 Cross-Validation, model data was separated into 10 samples for matching the model to the training model in order to use trained model for using the test model.

Table 3.3, 3.4, 3.5 shows the results of cross-validation for SVM algorithms.

Table 3.3 Experiment results of cross-validation in LibSVM model

Correctly Classified Instances	**1367**	**83.6086 %**
Incorrectly Classified Instances	**268**	**16.3914 %**
Kappa statistic	0.2365	
Mean absolute error	0.1639	
Root mean squared error	0.4049	
Relative absolute error	52.2706 %	
Root relative squared error	102.2864 %	
Coverage of cases (0.95 level)	83.6086 %	
Mean rel. reg size (0.95 level)	50 %	
Total Number of Instances	1635	

	TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	Roc Area	PRC Area	Class
	0.164	0.002	0.963	0.164	0.280	0.359	0.581	0.320	-1
	0.998	0.836	0.832	0.998	0.908	0.359	0.581	0.832	1
Avg.	**0.836**	**0.674**	**0.857**	**0.836**	**0.785**	**0.359**	**0.581**	**0.732**	

To demonstrate the accuracy of the methods and how the results are predicted by the methods we need to take care of measuring of how close a fitted data is to data points. With (Mean Absolute Error, Root Mean Squared Error) and Root relative squared error for values that indicate a schema greater or lower than 100.

LibSVM model for measuring the average of the squares of the error (Mean Absolute Error) we got 0.1639, for differences between predicted values and the values that was observed (Root Mean Squared Error) we got 0.4049, root relative squared error is 102.2864 percent. Classification with 83.6086 percent of correctly classified 1367 instances and 16.3914 percent of incorrectly classified 268 instances.

Table 3.4 Results for SMO with 10cross-validation

Correctly Classified Instances	**1592**	**97.37 %**
Incorrectly Classified Instances	**43**	**2.63 %**
Kappa statistic	0.9131	
Mean absolute error	0.0263	
Root mean squared error	0.1622	
Relative absolute error	8.3867 %	
Root relative squared error	40.9718 %	
Coverage of cases (0.95 level)	97.37 %	
Mean rel. region size (0.95 level)	50 %	
Total Number of Instances	**1635**	

	TP Rate	**FP Rate**	**Precision**	**Recall**	**F-Measure**	**MCC**	**Roc Area**	**PRC Area**	**Class**
	0.887	0.005	0.092	0.887	0.929	0.915	0.941	0.89	-1
	0.995	0.113	0.973	0.995	0.984	0.915	0.941	0.97	1
Avg.	**0.974**	**0.092**	**0.974**	**0.974**	**0.973**	**0.915**	**0.941**	**0.96**	

Experiment based on SMO (Sequential Minimal Optimization) method we got better result in order to LibSVM and SPegasos methods, where for Mean Absolute Error we got 0.0263, for Root Mean Squared Error 0.1622 and for Root Relative squared Error 40.9718 percent. Classification with 97.37 percent of correctly classified 1592 instances and 2.63 percent of incorrectly classified 43 instances.

Experiment based on SPegasos model is quite relative with SMO experiment, Cross-Validation metrics. Where mean absolute error is 0.0294, Root mean squared error 0.1713 and Root relative squared error 43.2884 percent. Classification with 97.0642 percent of correctly classified 1587 Instances and 2.9358 percent of incorrectly classified 48 instances.

Table 3.5 Results for SPegasos with 10 cross-validation

Correctly Classified Instances	**1587**	**97.0642 %**
Incorrectly Classified Instances	**48**	**2.9358 %**
Kappa statistic	0.9026	
Mean absolute error	0.0294	
Root mean squared error	0.1713	
Relative absolute error	9.3619 %	
Root relative squared error	43.2884%	
Coverage of cases (0.95 level)	97.0642%	
Mean rel. region size (0.95 level)	50 %	
Total Number of Instances	**1635**	

	TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	Roc Area	PRC Area	Class
	0.874	0.006	0.972	0.874	0.921	0.904	0.93	0.874	-1
	0.994	0.126	0.970	0.994	0.982	0.904	0.93	0.969	1
Avg.	**0.971**	**0.103**	**0.971**	**0.971**	**0.970**	**0.904**	**0.93**	**0.951**	

3.3.2 Training Results

After a certain cross-validation to data sets obtained values can be used for a training the data sets in learning model. In this part we will focus on two classes that classifiers (LIBSVM, SMO, SPegasos) for approved and rejected results. Also rising of cross-validation metrics and prediction of performance measures.

Table 3.6 shows the experiment results of LibSVM classifier, where we can see that all the cross-validation metrics results are decreasing (Mean Absolute Error, Root Mean Squared Error, Root Relative squared error) also the percentage of correctly classified instances are increasing in the other side incorrectly classified instances are decreasing respectively. This are the main points how cross-validation effect our learning model in positive way.

Table 3.6 Shows training results for LibSVM

Correctly Classified Instances	**1381**	**84.4648 %**
Incorrectly Classified Instances	**254**	**15.5352 %**
Kappa statistic	0.2887	
Mean absolute error	0.1554	
Root mean squared error	0.3941	
Relative absolute error	49.5442 %	
Root relative squared error	99.5793 %	
Coverage of cases (0.95 level)	84.4648 %	
Mean rel. region size (0.95 level)	50 %	
Total Number of Instances	**1635**	

	TP Rate	**FP Rate**	**Precision**	**Recall**	**F-Measure**	**MCC**	**Roc Area**	**PRC Area**	**Class**
	0.201	0.000	1.000	0.201	0.335	0.411	0.60	0.357	-1
	1.000	0.799	0.838	1.000	0.912	0.411	0.60	0.838	1
Avg.	**0.845**	**0.643**	**0.870**	**0.845**	**0.800**	**0.411**	**0.60**	**0.745**	

For prediction of performance measures we will focus on results where accuracy by class is predicted. From the table 9 MCC (Measure of the quality of binary classification) is for both classes 0.411 that is no better than the random prediction where in the next classifiers we can see the better predictions. Precision where for rejected (-1) class correctly classified positive instances and the classifier that classified as positives is very high with 1.000 precision and 0.838 for approved (+1) class.

Table 3.7 Shows training results for SMO

Correctly Classified Instances	**1634**	**99.939 %**
Incorrectly Classified Instances	**1**	**0.0612 %**
Kappa statistic	0.998	
Mean absolute error	0.0006	
Root mean squared error	0.0247	
Relative absolute error	0.1951 %	
Root relative squared error	6.2482 %	
Coverage of cases (0.95 level)	99.9388 %	
Mean rel. region size (0.95 level)	50 %	
Total Number of Instances	**1635**	

	TP Rate	**FP Rate**	**Precision**	**Recall**	**F-Measure**	**MCC**	**Roc Area**	**PRC Area**	**Class**
	0.997	0.000	1.000	0.997	0.998	0.998	0.998	0.997	-1
	1.000	0.003	0.999	1.000	1.000	0.998	0.998	0.999	1
Avg.	**0.999**	**0.003**	**0.999**	**0.999**	**0.999**	**0.998**	**0.998**	**0.999**	

Sequential Minimal Optimization (SMO) known as optimization algorithms results are quite impressing with 99.939 percent of correctly classified 1634 instances and 0.0612 percent of incorrectly classified 1 instance. ROC area is also close to 1 with result of 0.998 where we explained in previous section for ROC area if the result is close to one (1) its describes that the method which we are using for the model is best implemented.

Table 3.8 Shows training results for SPegasos

Correctly Classified Instances	**1634**	**99.9388 %**
Incorrectly Classified Instances	**1**	**0.0612%**
Kappa statistic	0.998	
Mean absolute error	0.0006	
Root mean squared error	0.0247	
Relative absolute error	0.1951 %	
Root relative squared error	6.2482%	
Coverage of cases (0.95 level)	99.9388%	
Mean rel. region size (0.95 level)	50%	
Total Number of Instances	**1635**	

	TP Rate	**FP Rate**	**Precision**	**Recall**	**F-Measure**	**MCC**	**Roc Area**	**PRC Area**	**Class**
	0.997	0.000	1.000	0.997	0.998	0.998	0.998	0.997	-1
	1.000	0.003	0.999	1.000	1.000	0.998	0. 99	0.999	1
Avg.	**0.999**	**0.003**	**0.999**	**0.999**	**0.999**	**0.998**	**0.998**	**0.999**	

3.3.3 Testing Results

Testing set results was conducted on 2585 instances using 25 attributes same as in the training set. Increasing number of instances in the testing set does not affect accuracy on decreasing way on LibSVM and SPegasos models except on SMO model.

We used default parameters for all models. LibSVM works with kernel type radial basis function, SMO works with polynomial kernel, where SPegasos with epochs and lambda parameters respectively.

In the Table 3.9 we can see the LibSVM’s test results with 95.8637 percent of correctly classified 2479 instances and 4.0263 percent of incorrectly classified 104 instances with increased accuracy for 11.3989 percent and decreased 11.5089 percent

Table 3.9 Shows testing results for LibSVMI

Correctly Classified Instances	**2479**	**95.97 %**
Incorrectly Classified Instances	**104**	**4.026 %**
Kappa statistic	0.8658	
Mean absolute error	0.0403	
Root mean squared error	0.2007	
Relative absolute error	12.4263 %	
Root relative squared error	49.8628 %	
Coverage of cases (0.95 level)	95.9737%	
Mean rel. region size (0.95 level)	50 %	
Total Number of Instances	**2585**	

	TP Rate	**FP Rate**	**Precision**	**Recall**	**F-Measure**	**MCC**	**Roc Area**	**PRC Area**	**Class**
	0.802	0.000	1.000	0.802	0.890	0.874	0.901	0.842	-1
	1.000	0.198	0.952	1.000	0.975	0.874	0.901	0.952	1
Avg.	**0.960**	**0.158**	**0.962**	**0.960**	**0.958**	**0.874**	**0.901**	**0.930**	

In the Table 3.9 we can see the LibSVM’s test results with 95.8637 percent of correctly classified 2479 instances and 4.0263 percent of incorrectly classified 104 instances with increased accuracy for 11.3989 percent and decreased 11.5089 percent

Table 3.10 Shows testing results of SPegasos

Correctly Classified Instances	**2582**	**99.9613 %**
Incorrectly Classified Instances	**1**	**0.0387 %**
Kappa statistic	0.998	
Mean absolute error	0.0004	
Root mean squared error	0.0197	
Relative absolute error	0.1195 %	
Root relative squared error	4.8894 %	
Coverage of cases (0.95 level)	99.9613 %	
Mean rel. region size (0.95 level)	50 %	
Total Number of Instances	**2583**	

	TP Rate	**FP Rate**	**Precision**	**Recall**	**F-Measure**	**MCC**	**Roc Area**	**PRC Area**	**Class**
	0.998	0.000	1.000	0.998	0.999	0.999	0.999	0.998	-1
	1.000	0.002	1.000	1.000	1.000	0.999	0.999	1.000	1
Avg.	**1.000**	**0.002**	**1.000**	**1.000**	**1.000**	**0.999**	**0.999**	**0.999**	

In the table 3.10 we can see SPegasos test results where best results were obtained. With 99.9613 percent of correctly classified 2582 instances and 0.0387 percent of incorrectly classified 1 instance. If we look in the results table 13 for MCC 0.999, ROC area 0.999, F-Measure 0.999, Precision 1.000 we have the highest results that can obtain one SVM model.

CHAPTER FOUR
CONCLUSION AND FUTURE WORK

4.1 Conclusion

By completing this project it should be noted that it seems inevitable to mention as improvement of Artificial Neural Network (ANN) to Support Vector Machines (SVM) algorithm as a great success. The most deserving person for that i consider my supervisor Prof. Dr. Süleyman Sevinç, who gathered around him a group of young people assured the importance of this project and he always became interested in the problems also for the solutions. I was lucky and I was one of his students who was working in this area.

The implementation of the SVM classifier with using LibSVM library and for the training datasets SMO and SPegasos models finally is obtained qualitative measures.

I tried to make as many measurements and explain them as clearly as possible but there are still much more questions that have to be answered.

4.2 Future Work

This project focuses on detection of hemolytic laboratory patient blood test results, which is based on learning and educative system. We compared the effectiveness of Support Vector Machines (SVM) algorithms which first one is the multilayer perceptron (MLP) and the second one LibSVM model.

We got good results in both models with differences where for MLP the accuracy was 80.6 percent and for LibSVM was 95.97 percent with integrated SMO and SPegasos training models. By using experimented parameters for MLP which were momentum, learning rate, and cross-validation it is obtained 80.6 percent of accuracy, but with respect to C-SVM where we used just default parameters for

LibSVM with radial basis function (RBF) we obtained better results also our roc area was under 0.9984 in comparison with MLP.

Since in both model our learning systems accuracy was very high the system can be integrated into biochemistry laboratory safely, with developed software for validating test results. To increase the accuracy of the model one idea can be experimenting the RBF parameters, or as we know how technology and the biochemistry departments are growing and updating very fast our future work can be implementing to the new models.

REFERENCES

Platt, J. (1998). *Fast training of support vector machines using sequential minimal optimization.* In B. Schölkopf, C. Burges, and A. Smola, editors, Advances in Kernel Methods-Support Vector Learning. Cambridge: MIT Press.

Boswell, D. (2002). *Introduction to support vector machines*. Retrived May 20, 2015, from http://dustwell.com/PastWork/IntroToSVM.pdf

Cristianini, N. and Shawe-Taylor, J. (2000). *An introduction to support vector machines and other Kernel-based learning methods.* Cambridge: Cambridge University Press.

Bell, J. (2013). *Machine learning Hands-on for developers and technical professionals.* Indianapolis: Wiley Press.

Mitchel, T. (2006). *Machine learning.* Retrieved May 19, 2015 from http://www.cs.cmu.edu/~tom/pubs/MachineLearningTR.pdf

Cortes, C. and Vapnik, V. (1995). *Support vector networks*. *Machine Learning*. Retrived May 15, 2015 from http://image.diku.dk/imagecanon/material/cortes_vapnik95.pdf

Stitson, M. and Weston, J. (1996). *Implementational issues of support vector machines.* Retrived June 17, 2015, from http://users.ecs.soton.ac.uk/srg/publications/pdf/SVM.pdf

Živković, Ž. Mihajlović, I. Nikolić, Dj. (2009). *Artificial neural network method applied for nonlinear multivariate problems.* Belgrade: Serbian Journal Press.

Chang, C.C and Chih, J. (2001). *LIBSVM: A library for support vector machines.* Retrived June 10, 2015 from http://www.csie.ntu.edu.tw/~cjlin/papers/libsvm.pdf

Luca, Z. (2006). *Parallel software for training large scale support vector machines on multiprocessor systems*. Retrived April 25, 2015 from http://www.jmlr.org/papers/volume7/zanni06a/zanni06a.pdf

Replace missing values filter (2012). Retrieved May 10, 2015 from http://weka.sourceforge.net/doc.dev/weka/filters/unsupervised/attribute/ReplaceMissingValues.html

Rumelhart, D. E., Hinton, G. E., Williams, R.J., (1986). *Learning internal representation by back-propagating errors.* Retrived June 10, 2015 from http://psych.stanford.edu/~jlm/papers/PDP/Volume%201/Chap8_PDP86.pdf

Support vector machines (2011). Retrieved June 10, 2015 from http://www.cs.cornell.edu/courses/cs4700/2011fa/lectures/14_SVM.pdf

Karadag, N. and Peker, E. (2014). *Autoverification of blood analysis results with artificial neural networks.* Bachelor Thesis, Dokuz Eylül University, İzmir.

Hall, M. (2011). *Classifier.* Retrieved May 18, 2015 from http://weka.sourceforge.net/doc.stable/weka/classifiers/functions/SPegasos.html

Osuna, E. Freund, R. and Girosi, F. (1997). Improved training algorithm for support vector machines. *Proceeding of IEEE NNSP'97.* Retrieved June 16, 2015 from http://citeseerx.ist.psu.edu/viewdoc/download;jsessionid=D2E56D083E519E16CFD8EDFE37DD037C?doi=10.1.1.37.7400&rep=rep1&type=pdf

Kubat, M. (2015). *An introduction to machine learning.* Miami: Springer Presss