5. MUSCULAR DYSTROPHY DISEASE IDENTIFICATION MODELS THROUGH SHALLOW LEARNING

This chapter details the implementation of a muscular dystrophy disease identification models using supervised learning algorithms. In this work, python is used as a programming language on the top of the scikit-learn machine-learning library. A brief description of the tools and libraries used are given at the beginning of the chapter, and the implementation of the muscular dystrophy disease identification models are followed later. Five independent experiments were carried out in this study based on various mutational features and are explained with the flow of events during training and predicting. The feature extraction process, building the model and the performance evaluation measures of the classifiers are explained in detail.

Data driven systems solves the problem by developing own models based on the examples and experiences. These methods develop intelligent systems that discover patterns from large datasets based on computational analysis that provides concrete theory and predictions.The key idea in shallow learning approach using supervised learning algorithms is to extract hand crafted discriminative descriptors from diseased gene sequences associated with all types of mutations and to provide an effective solution for predicting the type of disease. Change or mutation in the gene sequence, alters the structure of the sequence which implies the cause of disease. These structural changes are captured as features of a mutational sequence to learn the prediction model.

Four data driven supervised learning algorithms commonly used for classification task such as Decision tree, Naïve bayes, Artificial neural network and Support vector machine are employed in this research work of genetic disorder prediction.10 fold cross validation is used to test the models and results are analyzed. The autonomous models built in this research work are (i) Predicting Muscular dystrophy disease using features related to Missense and Non sense (Non-synonymous) mutations (ii) Predicting Muscular dystrophy disease using features related to silent (synonymous) mutations (iii) Predicting Muscular dystrophy disease using features related to Insertion and Deletion Mutations (iv) Predicting Muscular dystrophy disease using features related to Splicing Mutations (v) Predicting Muscular dystrophy using pooled features.

5.1 Tools and Libraries

Scikit-learn is an open source machine learning library written in python which provides a range of supervised and unsupervised learning algorithms through a consistent interface in python. It is a google summer project developed by David cournapeau. Python is a powerful scripting language now applied in machine learning and it provides efficient tools for data analysis and data mining problems. Various machine learning techniques such as classification, regression and clustering algorithms including support vector machines, random forests, gradient boost, K-means and DBSCAN are designed to work using the python with numerical and scientific libraries such as numpy and scipy.

Scikit learn is largely written in python with some core algorithms written in python to achieve performance. Python is a widely used high-level, general-purpose, interpreted, dynamic programming language. Main advantages of using python than other programming languages such as C++ or Java is its code readability and its syntax to express concepts that requires only fewer lines of code. Python is a object-oriented, imperative and functional programming or procedural styles that ensures the multiple programming par diagram facility. The advantages of the python library is its dynamic type system and automatic memory management.

Python interpreters are available for many operating systems, allowing python code to run on a wide variety of systems. Python can be used along with the third-party tools like Py2exe or Pyinstaller, where the python code is combined into stand-alone executable programs and so the python based software is used in distributed environments without a python interpreter. Machine learning in python is,

- Simple and efficient tools for data mining and data analysis.
- Accessible to everybody and reusable in various contexts.
- Built on Numpy, Scipy and matplotlib.
- Opensource, commercially usable.

Most of the algorithms are built based on the library packages such as pandas, numpy, scipy, and matplotlib.

Numpy

Numpy is one of the fundamental library in python that supports large n-dimensional arrays and matrices. This library has a collection of high level mathematical functions to operate on the arrays. It comprises of various packages and tools for integrating other programming languages code. Numpy libraries are useful for linear algebraic functions, fourier transform and random number capabilities. At the core of the NumPy package, is the ndarray object which n-dimensional arrays of homogeneous data types, that performs well with the operations performed using compiled code. It is a table of numerical elements, all of the same type, indexed by a tuple of positive integers. In numpy, the dimensions are the axes. The number of axes is rank.

Scipy

Scipy is the fundamental library for scientific computing. Scipy has bindings to low-level numerical computations implemented in Fortran, which allows scikit-learn to be both fast and easy to use at the same time. SciPy builds on the NumPy array object and is part of the NumPystack, which includes tools like Matplotlib, pandas and SymPy, and an expanding set of scientific computing libraries.

Pandas

Pandas is a data analysis toolkit based on Python, which makes it possible to access and edit tabular data in a fashion similar to that of a relational database. Pandas' two basic data structures are Series and Dataframes, where series is for one dimensional and dataframes is for two-dimensional respectively. This library supports the operations in the relational database such as selection, insertion, grouping and joining the datasets based on the column values in the table.

Matplotlib

Matplotlib is the two dimensional plotting library coded in python that produces publication quality figures. Plots, histograms, power spectra, bar charts, error charts, scatter plots can be generated using this library with a few lines of code. It has a MATLAB-style programming interface with default plot styles. This library has a deep integration with Python.

The learning problems covered in scikit-learn are Classification, Regression, Clustering, Dimensionality reduction, Model selection, Preprocessing. The pre-requisites to install python are python 2.6, numpy 1.6.1, scipy 0.9. Anaconda python library is used in this implementation.

Classification using scikit-learn

Supervised machine learning comprises of both regression and classification that refers to the problem of inferring a function from labeled training data. Scikit learn provides an object oriented interface centered on the concept of Estimator that extracts useful features from raw data. Scikit learn consists of a huge array of generalized linear models, that comprises of discriminate analysis, naive bayes, logistic regression, K-nearest neighbor, Neural networks, support vector machines and decision trees. The features such as feature extraction, feature selection, parameter tuning, dimensionality reduction, cross validation and manifold learning in scikit learn make the classification of data very flexible and more accurate.

5.2 Model I: Predicting Muscular Dystrophy Disease using Features Related to Missense and Non sense (Non-synonymous) Mutations

The first experiment aims in building the disease identification model using of Non Synonymous mutational descriptors. Point mutational features such as Structural, Annotation and Alignment descriptors are considered to be the non-synonymous mutational features.

Feature Extraction

The missense and nonsense mutational features are based on annotation, structure and alignment of the diseased gene sequences. The missense and nonsense mutational features includes GeneID, Gene symbol and Chromosome number, Length of the sequence, Alteration type, Protein changed, Reference allele, Observed allele, Mutation position, Mutation start position, Mutation end position, Position of mutation in gene sequence, amino acid change leads to stop codon, stop codon, Position of start codon in cDNA sequence, position of stop codon in DNA sequence, the nucleotide composition of A, G, C, T, AT and GC component, alignment features.

R is known to be the most powerful and specialized statistical programming language, and supports a vast library of statistics and machine learning algorithms. Most of the features are extracted from the gene sequences through R script and is given in the Appendix B.

Annotation Features: Gene sequences are identified by the attributes like gene identifier and symbol of the gene. As many to one relationship occur between gene and the disease, these descriptors are considered to differentiate the gene sequence in every disease type. The attributes of gene sequences like Gene ID, Gene symbol are identified by using the biomart package in R

and are extracted using getgenes(id). Library files such as Biostrings, seqinr are imported to perform manipulation in the sequence files. Readfasta() is used to read the fasta file from the directory. After the fasta file is read the sequence is converted into the data frame for manipulation.

Example: getGene(id=2010, type="entrezgene", mart=ens)

The Gene ID is the NCBI gene identifier for the affected phenotype. Some examples are GeneID 1746 and gene symbol DMD is for Dystrophin gene, GeneID 2010 and gene symbol EMD for Emerin gene etc.

Sequence Length: The Length of the sequence plays an important role in examining the difference in length of the sequence. When the insertion or deletion mutation occurs, the length of the sequence gets varied automatically. This feature is determined using Length() function by converting the fasta file into a data frame.

Example:

```
df<-read.fasta(file = "SH3TC2_cdna_ NM_024577.3.fasta")
ds<- df[[1]]
ln<- length(ds)
```

where, ds is the data frame of the sequence file and the length of the sequence is found out using length() function.

Alteration Type: The next descriptor alteration type denotes the type of mutation occurred such as insertion, deletion and duplications. This feature is captured by hardcoding the mutation type to its corresponding numeric values from 1 to 6 such as 1 for missense/nonsense, 2 for synonymous, 3 for insertion, 4 for duplication and 5 for deletion and 6 for splicing.

Mutation position (3): This descriptor points the position of the alteration in the diseased gene sequence. The position of mutation in the gene sequence is identified by blasting the mutated sequence against the reference gene sequence. Nucleotide blast is used to capture the position of gene sequence. Starting and ending position of the alteration in the sequence is captured as mutation start position and mutation end position.

Reference allele and observed allele: The codon or amino acid observed in the normal gene sequence constitutes the reference allele and the allele that is observed after alteration is the observed allele of the disease gene sequence. To identify the reference allele and observed allele, the position of codon is identified with the position of mutation from the mutated sequence file.

The first step in finding the observed allele is to read the fasta file and split it into codons. The required codon is acquired and altered based on the position information of codon change. Seqinr and Biostrings library are inquired for this work.

Splitting the sequence into codons

cod<-splitseq(ds)

Splitseq(ds) splits the sequence into codons and cod holds all the codon values.

cod[199]

cod[199] gives the codon in the 199th position.

ori<-cod[cod_pos]

ori

To extract the reference allele the codon is retrieved from the specified position from the reference gene sequence.

cod<-splitseq(ds2)

mut<-cod[cod_pos]

mut

To extract the observed allele the codon is retrieved from the specified position from the mutated gene sequence gene sequence.

Amino acid change leads to stop codon: This descriptor states whether the change in the amino acid leads to stop codon. It is a Boolean value descriptor that decides the type of mutation whether missense or nonsense where in missense amino acid change does not lead to stop codon where in nonsense it leads to stop codon. This descriptor is captured as same as the reference and observed allele descriptor.

The first step is to hardcode all the 20 amino acids by assigning the values. The next step is to capture the reference allele and to check whether it leads to stop codon using boolean functions.

```
if ((mut == "tag") || (mut == "taa") || (mut == "tga")){
ref<-0}
print(ref)
```

Protein changed(Y/N): This descriptor results in boolean value that states whether the amino acid is altered or not after the occurrence of the mutation.

Position of Start and Stop codon: ATG is the start codon and TAG, TAA, TGA are the stop codons. The position of the Stop codon reveals the end of the coding part in the sequence. This position may be altered with the occurrence of mutation and hence it is noted. match pattern () function is employed to identify and capture the position of stop codon. The mutated sequence is converted into a string of sequence and the position of the start and stop codons can be retrieved.

```
dstartstring<- c2s(ds)
matchPattern("tag", dstartstring)
```

where the data frame is converted into data string using c2s(). Using matchpattern() the position of the start and stop codons will be retrieved.

Nucleotide composition values: The base composition A, C, G,T are calculated to count the number of occurrences of the four different nucleotides ("A", "C", "G", and "T") in the sequence. GC content is the fraction of the sequence that consists of Gs and Cs, i.e. The GC content can be calculated as the percentage of the bases in the genome that are Gs or Cs. That is,

AT content = (number of As + number of Ts)*100/ (genome length)

GC content = (number of Gs + number of Cs)*100/ (genome length)

Therefore, six different descriptor values are calculated as the nucleotide composition values.

Alignment Scores: Alignment scores are considered as the important feature for disease prediction. The global pairwise alignment based on edit distance is done with the mutated sequence against with the reference cDNA sequence and the three alignment scores are calculated using the edit distance scoring method. PairwiseAlignment( ) in R is used to calculate the alignment scores.

```
pairwiseAlignment(s1, s2,substitutionMatrix = nucleotideSubstitutionMatrix(2, -1, TRUE),gapOpening = -2, gapExtension = -8)
```

The PhredQuality measures are calculated with the patternQuality and subjectQuality to examine the quality-based match and mismatch bit scores for DNA/RNA. By default patternQuality and subjectQuality are PhredQuality(22L). QualitySubstitutionMatrices() is used to examine the PhredQuality measures. The substitution scores are calculated by setting the error probability to 0. Table 5.1 depicts the features and its description.

Table 5.1 Features and their Descriptions

Features	Description
Gene ID	Identifier of the gene taken from NCBI
Gene Symbol	Name of the gene involved
Chromosome Number	The chromosome involved in mutation
Alteration type	Mutation type such as missense, non sense, silent, deletion and duplication
Protein changed	Whether protein altered through mutation
Observed allele	The amino acid present in normal gene
Reference allele	The observed amino acid after mutation
Mutation Position	Position of alteration in cDNA sequence
Length	Length of the mutated gene sequence
Mutation start position	The starting position of alteration in cDNA sequence
Mutation end position	The position where the mutation ends in cDNA sequence
Position	Mutation Position in gene sequence is identified through nucleotide blast against reference gene sequence
Nucleotide Composition	Composition of A, C, G, T, AT, GC in mutated sequence.
Position of stop codon	Last position of stop codon ATG
Edit distance scores	Alignment scores using edit distance method
PhredQuality measures	Calculated with patternQuality and subjectQuality
Substitution scores	Calculated with the error probability set to 0 or 1
ConsensusStart	The starting position of conserved region
ConsensusEnd	The end position of the conserved region

A feature vector of a DMD disease mutated with HGMD accession number HM080103is shown below. The codon change is CAG-TAG, amino acid change Gln35Term and the nucleotide change 103C>T. Feature vector for a sample gene sequence is given below.

GeneID: 1756	GeneSym: 1
Chr: 23	MutPosition: 103
SeqLen: 11058	AlterType: 2
CodonNum: 35	Mutstart: 103
Mutend: 105	Lenvariant:1
Proteinchanged: 1	Referalelle:6
Obseralelle: 0	Posstartcdn: 1
Posstpcdn: 11056	Aminotostpcdn:1
Aminostopcdn: 1	editdisscore: 22113
Qualityscores: 21907.28	Subscoreserr: 18816.67
A: 33.32	G: 20.75
C: 23.7	T: 22.23
GC: 44.45	AT: 55.55
Label 1	

Training dataset

The features extracted from each disease gene sequence forms a feature vector. Depending on the type of mutation the mutational features are varied and the size of the feature vector also varies here. Annotation, structure and alignment features are extracted from1000 disease gene sequences. 1000 feature vectors of dimension 26 are created and the dataset (NSM) is prepared. For each feature vector the class label is assigned a sequence number 1 to 5 according to the category of disease. The training data set with instances related to five categories of muscular dystrophy that is Duchenne muscular dystrophy, Becker's muscular dystrophy, Emery-Dreifuss, Limb-girdle muscular dystrophy and Charcot marie Tooth disease has been developed. The sample training dataset is depicted below.

1756	1	23	6678	11058	2	6678	6680	6922	1	1	18	0	1
	11056	1	3	22113	21907.28		18816.67		1	6677	33.33	20.75	
	23.69	22.22	44.45	55.55	A								
1756	1	23	6721	11058	2	6721	6723	6965	1	1	8	0	1
	11056	1	3	22113	21907.28		18816.67		1	6720	33.32	20.75	
	23.69	22.23	44.45	55.55	A								
1756	1	23	6255	11058	2	6253	6255	6986	1	1	18	0	1
	11056	0	0	22113	21907.28		18816.67		1	6254	33.34	20.73	
	23.66	22.27	44.38	55.62	B								
1756	1	23	6742	11058	2	6742	6744	7646	1	1	7	0	1
	11056	0	0	22113	21907.28		18816.67		1	6741	33.33	20.73	
	23.66	22.28	44.38	55.62	B								
2010	2	23	2	765	1	1	3	250	1	1	13	2	
	208	762	0	0	1518	1500.073		1286.257		3	765	20.16	
	31.68	24.21	23.95	55.89	44.11	C							
4000	3	1	626	1994	5	627	629	875	1	1	3	17	1
	1992	0	0	3888	3883.749		3328.282		1	625	21.82	29.54	
	33.75	14.89	63.29	36.71	C								
825	5	15	77	2466	1	76	78	383	1	1	15	11	1
	2464	0	0	4929	4879.157		4193.204		1	76	26.64	25.75	
	26.85	20.76	52.6	47.4	D								
825	5	15	133	2466	1	133	135	439	1	1	1	17	1
	2464	0	0	4929	4879.157		4193.204		1	132	26.68	25.79	
	26.8	20.72	52.6	47.4	D								
79628	49	5	680	3867	1	679	681	832	1	1	2	6	1
	3865	0	0	7731	7655.846		6577.915		1	679	22.06	27.05	
	27.98	22.91	55.03	44.97	E								
79628	49	5	920	3867	2	919	921	1072	1	1	18	0	1
	3865	1	1	7731	7655.846		6577.915		1	919	22.06	27.05	
	27.98	22.91	55.03	44.97	E								

Building the model

The supervised learning techniques namely Naïve Bayes Classifier, Decision tree induction, Support vector machine and artificial neural network have been used to learn and are built using Scikit learn. The Scikit-Learn library uses NumPy arrays in its implementation, therefore NumPy arrays should be created from *.csv files. The data frame is built with the numpy array. It is a table of elements usually numbers, all of the same type, indexed by a tuple of positive integers. The attributes extracted from the mutated gene sequences related to non-synonymous mutations are stored in .csv file which is shown in Appendix - B.

The following script imports the necessary packages and reads the feature vectors as .csv file and normalization is done. The python coding of all the classification algorithms are shown in Appendix - C.

```
import numpy as np
import pandas as pd
import io
df=pd.read_csv('C:\Users\HCL\Documents\missense_nonsense_scikit_modified.csv')
print df
from numpy import genfromtxt
my_data = genfromtxt('C:\Users\HCL\Documents\missense_nonsense_scikit_modified.csv', delimiter=',')
X = my_data[:,0:25]
y = my_data[:,26]
from sklearn import preprocessing
normalized_X = preprocessing.normalize(X)
standardized_X = preprocessing.scale(X)
```

Import the numpy library into the algorithm with the code

import numpy as np /// importing numpy array package

The dataset is stored as comma separated values and therefore genfromtxt() is used to convert it into numpy array.

from numpy import genfromtxt ////converting the .csv file into numpy array

```
my_data = genfromtxt ('C:\Users\HCL\Documents\Features_sci_G.csv', delimiter=',')
```

Separate the data from target attributes

Total number of attributes is 20(excluding Label) it is taken in y axis

```
X feature-object matrix and values of the y target variable.
X = my_data [:,0:25]
y = my_data [:,26]
```

The dataset is normalized using min max normalization by transforming the numeric values into the range between 0 and 1 which aid in scaling the input attributes for building accurate model. The dataset is split into training and testing dataset in the ratio of 80:20.

```
fromsklearn.cross_validation import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)
```

The dataset is trained and a model is fitted with the above listed supervised classification algorithms using

```
classifier.fit(X, y)
```

In case of decision tree classifier and Naïve bayes default parameter settings are used during classification whereas in Artificial neural network implementation, the hidden layers are tuned to gain a stable accuracy. In case of SVM various kernels such as linear, polynomial and RBF kernel are employed with different parameter settings for C regularization parameter. In case of polynomial and RBF kernels, the default settings for d and gamma are used. In scikit-learn, an estimator for classification is a Python object that implements the methods fit(X, y) and predict(T).

Performance evaluation

The task is to predict the disease type the samples are given and from each of the 5 possible classes on which an estimator is fitted to predict the classes to which unseen samples belong. Evaluating the generalization power of the classifiers and to estimate their predictive capabilities for unknown samples, a standard k- fold cross-validation technique is used. As dataset comprises of 1000 instances, it is appropriate to use cross validation with K=10. This 10-

fold cross validation iterates the algorithm 10 times with different groupings of training and testing datasets. The performance of trained models measured in terms of classification accuracy, precision, recall, F-score, kappa statistic, precision recall curve and ROC curve.

```
>>> print(metrics.confusion_matrix(expected, predicted))
[187   13    0    0    0]
 [ 37  163    0    0    0]
 [ 0    0   165   30    5]
 [ 0    0   20   170   10]
 [ 0    0    6    30   164]
>>> from sklearn.metrics import accuracy_score
>>> accuracy_score(expected,predicted)
0.849
>>> cohen_kappa_score(expected,predicted)
0.861
```

Various scores are retrieved, the performances of each classifier are analyzed using the function classifier. score( ) in scikit learn and the results are tabulated in Table 5.2 and 5.3 and illustrated in Fig.5.1. The predictive performance of the disease classification models shows that SVM classifier yielded a best accuracy of 84.9%.

Table 5.2 Predictive Performance of the Classifiers (Non –Synonymous Mutations)

Performance criteria	Decision Tree Classifier	Artificial Neural Network	Naïve Bayes Classifier	SVM
Correctly classified Instances	805	793	698	849
Incorrectly classified instances	195	207	302	151
Prediction accuracy	**80.5**	**79.3**	**69.8**	**84.9**

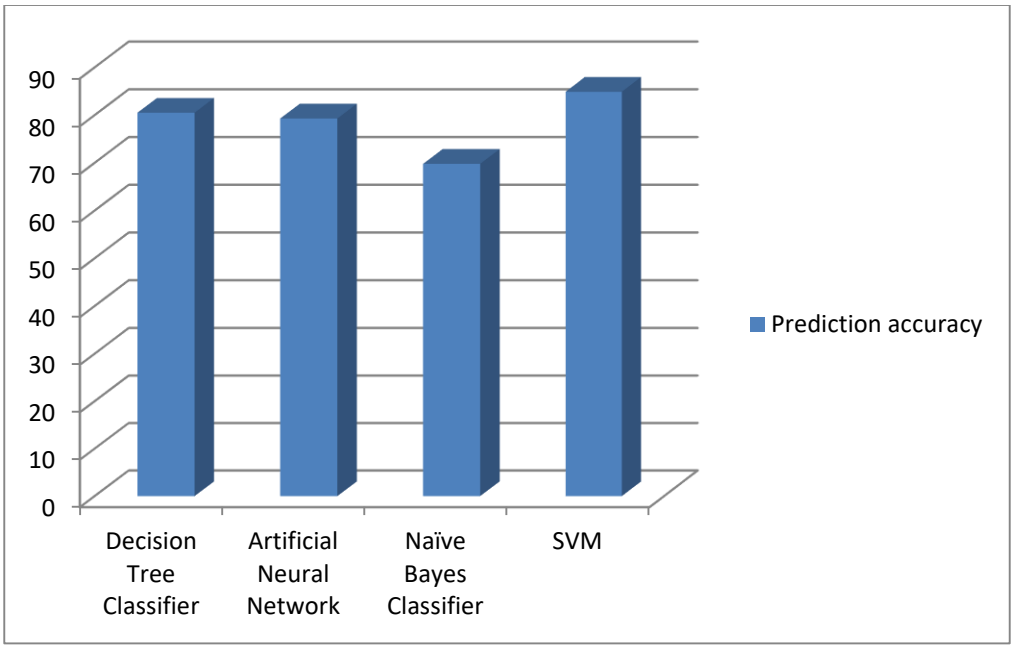

Fig.5.1 Prediction Accuracy of the Classifiers (Non – synonymous mutations)

Table 5.3 Performance Evaluation of the Classifiers (Non-synonymous mutations)

Performance criteria	Decision Tree Classifier	Artificial Neural Network	Naïve Bayes Classifier	SVM
Prediction accuracy (%)	**80.5**	**79.3**	**69.8**	**84.9**
Precision	0.8	0.803	0.689	0.859
Recall	0.825	0.805	0.678	0.866
F1 Score	81.9	81.8	69.9	86.1
Cohen's Kappa	0.82	0.813	0.692	0.861
Time taken to build the model (in sec)	8.4	10.7	9.6	7

The output of the binary classifier is typically studied with the Precision-recall curves. In this multi-class classification work, the target attribute values are binarized and the class values are shaped into 1. The precision- recall curve is computed by adding some noisy features and the micro-average ROC and ROC area is calculated. The precision- recall curve is plotted for each class based on SVM linear classifier. Fig.5.2 shows the precision-recall curve for each class in SVM classifier.

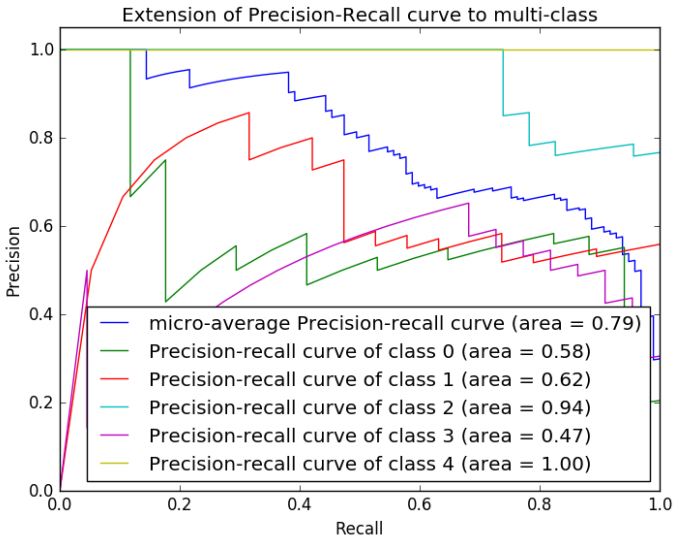

Fig.5.2 Precision-Recall Curve for SVM Classifier

The above figure insists that class 2 and class 4 have high precision of 0.94 and 1.00 and class 3 is curved low at the area 0.47. On the average the micro-average curve shows the value of 0.79. ROC curve is plotted for SVM linear classifier for each class. Fig.5.3 depicts the ROC curve based on SVM classifier in predicting muscular dystrophy.

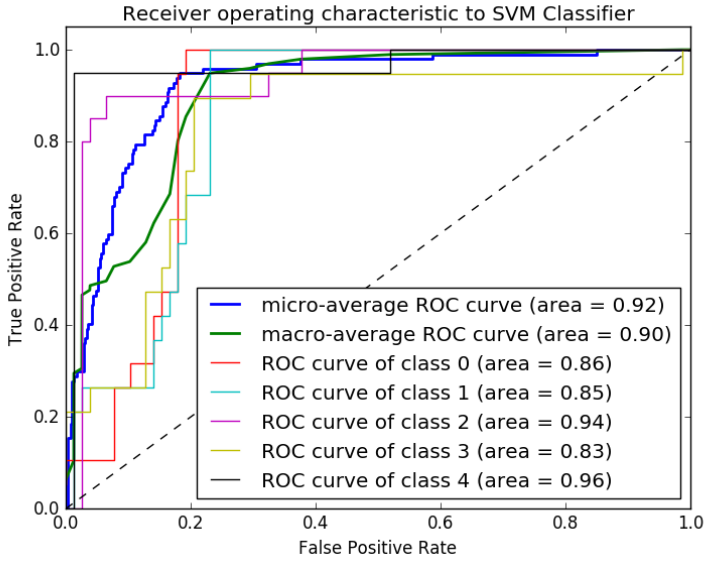

Fig.5.3 ROC Curve for SVM Classifier

Findings

From the above results, it is observed that the prediction accuracy is high for SVM than other algorithms. The precision and Recall measure for SVM is high when compared with other learning methods. Overall, the balance accuracy measure is also eminent in SVM when measured with other algorithms. The sensitivity and specificity measure for all classes is prominent in SVM when compared with other learning techniques. The cohen's kappa is also high in SVM with 0.861 score and the time taken is minimal of 7 seconds. The experiment proves that the features designed for building the classifier are more appropriate and suitable for disease identification.

5.3 Model II: Predicting Muscular Dystrophy Disease using Features Related to Synonymous Mutations

In the first experiment, only non-synonymous mutational features are taken into account to identify the disease, where the silent mutational features are required to identify the disease that is caused due to synonymous mutations. Therefore, in this second work, the codon usage patterns are considered as the contributing features for representing the mutated gene sequences. Since codon usage patterns are diverse in different gene families, this feature input is a well-chosen descriptors for specifying different gene families for all types of diseases.

Feature Extraction

A codon is the triplet of nucleotides that code for a specific amino acid. Many to one relationship occurs between the codon and amino acid. Many amino acids are coded by more than one codon because of the degeneracy of the genetic codes. A total number of codons in a DNA sequence counts to 64. Since methionine (ATG) and tryptophan (TGG) have only one corresponding codon, they are not counted and are eliminated from the analysis as their RSCU values are always equal to 1. The three stop codons (TGA, TAA, TAG) are also not included. Accordingly, the number of codons considered is 59. The RSCU features are extracted from mutated gene sequences through R script that is created using seqinr() package downloaded from www.CRAN.org.

The differences in the frequency of occurrence of synonymous codons are referred as codon usage bias. The formula for calculating RSCU can be explained as, the number of times a

particular codon is observed, relative to the number of times that the codon would be observed in the absence of any codon usage bias.

RSCU values are the number of times a particular codon is observed, relative to the number of times that the codon would be observed in the absence of any codon usage bias. RSCU is a simple measure of non-uniform usage of synonymous codons in a coding sequence. RSCU values are the number of times a particular codon is observed, relative to the number of times that the codon would be observed for a uniform synonymous codon usage where all the codons for a given amino-acid have the same probability. In the absence of any codon usage bias, the RSCU values would be 1.00. A codon that is used less frequently than expected will have an RSCU value of less than 1.00 and vice versa for a codon that is used more frequently than expected.

The RSCU carries the value 1.00 if the codon usage bias of that particular codon is absent. If the codon is used less frequently than expected, the RSCU values tend to have the negative values. Following formula is used to calculate RSCU.

$$RSCU = X_{ij} / (1/n_i * S \{X_{ij}; j=1, n_i \})$$

whereX_{ij} is the number of occurrences of the j^{th} codon for the i^{th} amino acid, and n_i is the number of alternative codons for the i^{th} amino acid.

If the synonymous codons of an amino acid are used with equal frequencies, then their RSCU values are 1.

The program adds up the total number of times that the codons for a particular amino acid are observed. It then divides this number by the number of codons for the amino acid, this gives the expected number of times that the codons should be observed. Then for each codon, the frequency of observation is divided by the expected frequency. Sometimes the observed frequency will be greater than the expected frequency (RSCU value greater than 1.00), and sometimes it will be less (RSCU value less than 1.00).

Based on the above formula the RSCU values for every codon in the mutated gene sequence is calculated with the uco( ) function in R. The calculation is as follows:

uco(seq, frame = 0, index = c("eff", "freq", "rscu"), as.data.frame = FALSE,NA.rscu = NA)

Arguments

Seq = coding sequence as a vector of chars

Frame = an integer (0, 1, 2) giving the frame of the coding sequence

Index= codon usage index choice, partial matching is allowed. eff for codon counts, freq for codon relative frequencies, and rscu the RSCU index

as.data.frame = logical. If TRUE: all indices are returned into a data frame.

NA.rscu = when an amino-acid is missing, RSCU are no more defined and reported as missing values (NA). You can force them to another value (typically 0 or 1) with this argument.

Likewise, the RSCU values of each and every codon are calculated. Table 5.4 shows an example that holds RSCU values of 59 codons for a mutated gene sequence.

Table 5.4 RSCU Values for 59 Codons

Codon	Value	Codon	Value	Codon	Value
AAA	1.05	CCC	0.97	GGC	0.92
AAC	0.812	CCG	0.12	GGG	0.75
AAG	0.948	CCT	1.64	GGT	0.64
AAT	1.18	CGA	0.87	GTA	0.81
ACA	1.52	CGC	0.54	GTC	0.93
ACC	0.76	CGG	0.66	GTG	1.40
ACG	0.24	CGT	0.63	GTT	0.85
ACT	1.48	CTA	0.73	TAC	0.61
AGA	1.84	CTC	0.87	TAT	1.38
AGC	0.99	CTG	1.41	TCA	1.23
AGG	1.42	CTT	1.03	TCC	0.91
AGT	1.36	GAA	1.22	TCG	0.14
ATA	0.52	GAC	0.81	TCT	1.33
ATC	1.10	GAG	0.77	TGC	1.16
ATT	1.36	GAT	1.18	TGT	0.833
CAA	0.87	GCA	1.23	TTA	0.71
CAC	0.86	GCC	1.18	TTC	0.64
CAG	1.13	GCG	0.15	TTG	1.23
CAT	1.14	GCT	1.42	TTT	1.63
CCA	1.25	GGA	1.67		

Training Dataset

The RSCU values are derived for 59 codons from each mutated gene sequence, which forms a feature vector with 59 elements for training dataset for classification task. Since the corpus consists of 1000 sequences of 5 types of Muscular dystrophy diseases, a training set (SYM) with 1000 feature vectors has been created and for each feature vector the class label is assigned from 1 to 5 indicating the five types of muscular dystrophy diseases. The sample training dataset is depicted below.

```
1.0511945 0.8121212 0.9488055 1.1878788 1.52 0.76 0.24 1.48
1.8484848 0.9917355 1.4242424 1.3636364 0.5263158 1.1052632
1.3684211 0.8701299 0.8636364 1.1298701 1.1363636 1.2519084
0.9770992 0.1221374 1.648855 0.8787879 0.5454545 0.6666667
0.6363636 0.7384615 0.8703297 1.410989 1.0285714 1.2248062
0.8186528 0.7751938 1.1813472 1.2394366 1.1830986 0.1502347
1.42723 1.6785714 0.9285714 0.75 0.6428571 0.8121827
0.9340102 1.4010152 0.8527919 0.6129032 1.3870968 1.2396694
0.9173554 0.1487603 1.338843 1.1666667 0.8333333 0.7120879
0.6419753 1.2395604 1.3580247 1

1.0511945 0.8121212 0.9488055 1.1878788 1.52 0.76 0.24 1.48
1.8484848 0.9876543 1.4242424 1.3580247 0.5263158 1.1052632
1.3684211 0.8701299 0.8636364 1.1298701 1.1363636 1.2519084
0.9770992 0.1221374 1.648855 0.8787879 0.5454545 0.6666667
0.6363636 0.7384615 0.8703297 1.410989 1.0285714 1.2248062
0.8229167 0.7751938 1.1770833 1.2394366 1.1830986 0.1502347
1.42723 1.6785714 0.9285714 0.75 0.6428571 0.8121827
0.9340102 1.4010152 0.8527919 0.6031746 1.3968254 1.2592593
0.9135802 0.1481481 1.3333333 1.1666667 0.8333333 0.7120879
0.6419753 1.2395604 1.3580247 2

0.2 1.2380952 1.8 0.7619048 0.1111111 2.3333333 0.5555556 1
-1 2.3478261 0.4761905 0.4347826 0.1875 2.625 0.1875 0.05
1.3846154 1.95 0.6153846 0.5 2.25 0.5 0.75 0.2857143 2.952381
1.4285714 0.8571429 0.1643836 0.8219178 4.2739726 0.4931507
0.1971831 1.3333333 1.8028169 0.6666667 0.6101695 1.9661017
0.4745763 0.9491525 0.4761905 1.5238095 1.6190476 0.3809524
0.1290323 0.9032258 2.8387097 0.1290323 1.6363636 0.3636364
0.5217391 1.3913043 0.5217391 0.7826087 2 -1 -1 1.25
0.2465753 0.75 3
```

```
0.6551724 1.6190476 1.3448276 0.3809524 1.4358974 1.7435897
0.4102564 0.4102564 1.3953488 1.4444444 0.8372093 0.8888889
0.1914894 1.8510638 0.9574468 0.3783784 1.6190476 1.6216216
0.3809524 1.025641 1.025641 0.8205128 1.1282051 0.4186047
0.9767442 1.9534884 0.4186047 0.3050847 1.9322034 2.9491525
0.6101695 0.3666667 1.0526316 1.6333333 0.9473684 1.0909091
1.8181818 0.1818182 0.9090909 1.0196078 1.1764706 1.0196078
0.7843137 0.1818182 1.0909091 1.8181818 0.9090909 1.12 0.88
0.5555556 1.4444444 0.2222222 1.4444444 1.625 0.375 -1
1.5652174 0.2033898 0.4347826 4

0.58064516 1.14285714 1.41935484 0.85714286 1.11627907 1.72093023
0.27906977 0.88372093 2.015625 1.19277108 3.46875 0.75903614
0.62790698 1.6744186 0.69767442 0.46808511 1 1.53191489 1
1.36507937 1.23809524 0.28571429 1.11111111 0.09375 0.09375
0.328125 -1 0.22222222 1.05555556 2.22222222 0.77777778
0.66666667 1.86666667 1.33333333 0.13333333 1 1.61904762
0.04761905 1.33333333 0.91566265 1.25301205 1.15662651 0.6746988
0.56 0.88 1.92 0.64 0.92307692 1.07692308 1.30120482 1.30120482
0.21686747 1.22891566 1.10344828 0.89655172 0.38888889 1.33333333
1.33333333 0.66666667 5
```

Building the Model

The second experiment is conducted by learning the above SYM dataset with the standard supervised pattern learning techniques, Naïve Bayes Classifier, Decision tree induction, Artificial neural network and Support vector machine (SVM) by tuning the parameters. As specified in first experiment the dataset is converted in numpy arrays and then it is normalized by scaling the input attributes. The below depicted script builds a decision tree classifier model in Scikit learn environment.

```
import numpy as np
import pandas as pd
df=pd.read_csv('C:\Users\HCL\Documents\RSCU.csv')
from numpy import genfromtxt
my_data = genfromtxt('C:\Users\HCL\Documents\RSCU.csv', delimiter=',')
X = my_data[:,0:58]
y = my_data[:,59]
from sklearn import preprocessing
normalized_X = preprocessing.normalize(X)
standardized_X = preprocessing.scale(X)
from sklearn import metrics
from sklearn.tree import DecisionTreeClassifier
model = DecisionTreeClassifier()
model.fit(X, y)
print(model)
expected = y
predicted = model.predict(X)
```

Performance Evaluation

The performance of trained models is evaluated using the same 10-fold cross validation technique and measured in terms of various metrics as in the terms of previous case. The predictive performance of the disease classification models shows that decision tree classifier yielded a best accuracy of 86% and the results are tabulated in Table 5.5 and predictive performance graph is shown in Fig.5.4.

```
# summarize the fit of the model
print(metrics.classification_report(expected, predicted))
print(metrics.confusion_matrix(expected, predicted))
```

Table 5.5 Predictive Performance of the Classifiers (Synonymous Mutations)

Performance criteria	Decision Tree Classifier	Artificial Neural Network	Naïve Bayes Classifier	SVM
Correctly classified Instances	860	833	840	846
Incorrectly classified instances	140	167	160	154
Prediction accuracy	**86**	**83.33**	**84**	**84.6**

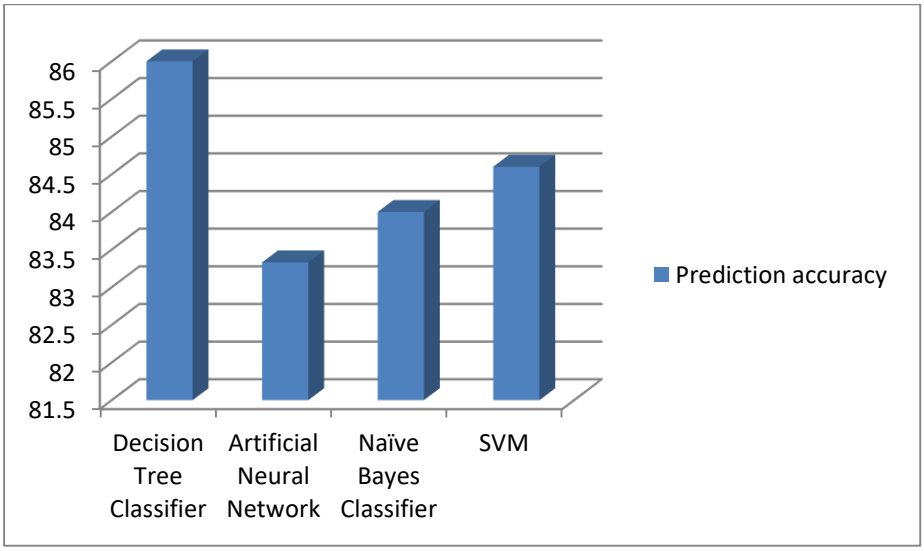

Fig.5.4 Predictive Accuracy of the Classifiers (Synonymous mutations)

The performances of the classifiers are evaluated and the measures such as prediction accuracy, precision, recall, F1- score, cohen's kappa and Time taken to build the model are calculated and tabulated in Table 5.6. The precision-recall curve is computed for each class based on the decision tree classifier. Fig.5.5 shows the precision-recall curve for each class in decision tree classifier.

Table 5.6 Performance Evaluation of the Classifiers (Synonymous mutations)

Performance criteria	Decision Tree Classifier	Artificial Neural Network	Naïve Bayes Classifier	SVM
Prediction accuracy	**86**	**83.33**	**84**	**84.6**
Precision	0.86	0.831	0.835	0.841
Recall	0.854	0.83	0.841	0.85
F1 Score	85.6	83.1	83.3	84.8
Cohen's Kappa	0.86	0.81	0.83	0.84
Time taken to build the model (in sec)	7.47	11.7	12.7	10.5

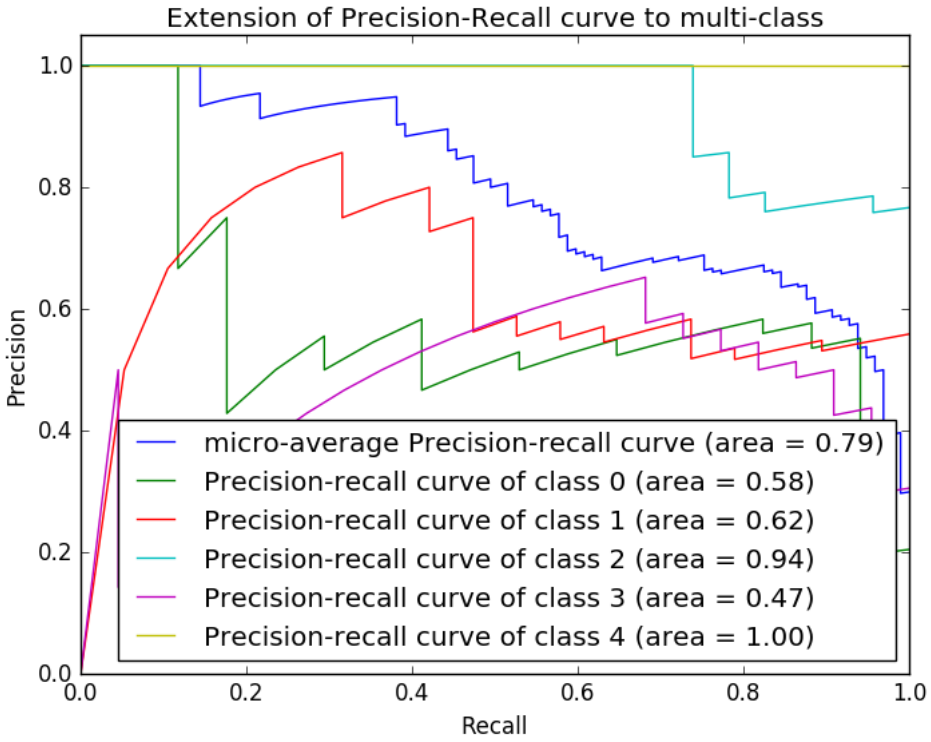

Fig.5.5 Precision- Recall Curve for Decision Tree Classifier

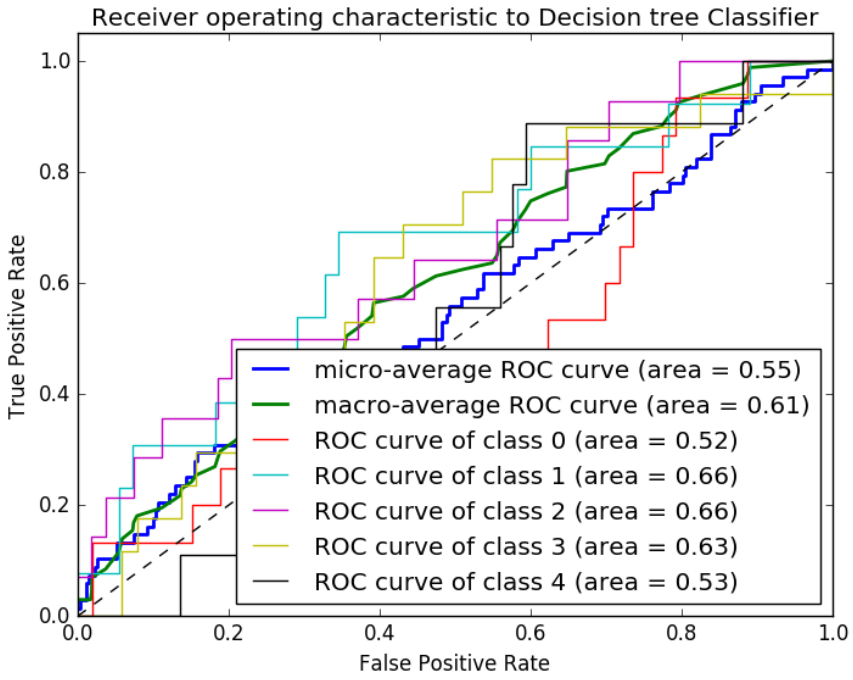

Fig.5.6 ROC Curve for Decision Tree Classifier

ROC Curve in the above figure shows that roc curve for class 0 and 4 has low range and it is elevated from 2 and have a high precision. ROC Curve is plotted for decision tree classifier. Fig.5.6 depicts the ROC curve based on decision tree classifier.

Findings

The models built in this experiment aids in classifying type of muscular dystrophy using mutated gene sequences by capturing RSCU features of silent mutations. Decision tree classifier elevated its accuracy to 86% and the precision, recall measure is also high for the decision tree classifier than other classifiers. Less time is taken to build the decision tree model. These models can facilitate in investigating the changes in protein folding and function.

5.4 Model III: Predicting Muscular Dystrophy Disease using Features Related to Insertion/Duplication, Deletion Mutations

Insertions/Duplications and Deletions alter the structure of the sequence and throws a heavy impact and therefore, in the third experiment imperative extrinsic and intrinsic descriptors are considered for learning the model using supervised classification algorithms.

Feature Extraction

The exonic and intronic features are considered from diverse gene families, to extract the well-defined descriptors related to insertion, deletion and duplication mutations in the mutated gene sequences. Code is written using R for extracting most of the descriptor values from the mutated gene sequences.

Sample Coding sequence of DMD gene

```
1          10         20         30         40         50         60         70         80         90         100        109
ATGCTTTGGTGGGAAGAAGTAGAGGACTGTTATGTTGATACCACCTATCCAGATAAGAAGTCCATCTTAATGTACATCACATCACTCTTCCAAGTTTTGCCTCAACAAGTGA
|----Exon1----------|----------Exon 2-----------------|-------Exon 3--------|----------------------Exon 4-----------------------|------Exon5-----|----Exon 6----|Exon 7-------|

When deletion occurs in exon 2 and 3
1          10         20         30         40         50         60         74
ATGCTTTGGTGGGACCAGATAAGAAGTCCATCTTAATGTACATCACATCACTCTTCCAAGTTTTGCCTCAACAAGTGA
|----Exon1----------|----------------------Exon 4-----------------------|------Exon5-----|----Exon 6----|Exon 7---------|
```

Gene Id: 1746	No. of Exons deleted: 2
Gene Symbol: DMD	Starting position of exon: 14
Sequence Length: 74	Ending position of exon: 47
Alteration Type: Internal exon	Inframe/ outframe: Inframe

Annotation Features: Gene sequences are identified by the attributes like gene identifier and symbol of the gene. As many to one relationship occur between gene and the disease, these descriptors are considered to differentiate the gene sequence in every disease type. The attributes of gene sequences like Gene ID, Gene symbol are identified by using the bio mart package in R and are extracted using get genes(id). The Gene ID is the NCBI gene identifier for the affected phenotype. Some examples are GeneID 1746 and gene symbol DMD is for Dystrophin gene, GeneID 2010 and gene symbol EMD for Emerin gene etc.

Alteration Type: The next descriptor alteration type denotes the type of mutation occurred such as insertion, deletion and duplications. This feature is captured by hardcoding the mutation type to its corresponding numeric values from 3 to 5 such as 3 for insertion, 4 for duplication and 5 for deletion.

Gene Starting position and Gene Ending position: A chromosome is comprised of several genes and every gene has its starting and ending position. If an insertion/duplication or deletion mutation occurs in a sequence, then there may be a change in the gene's starting or ending position, hence these features aids in classifying the disease type. Nucleotide blast is used to capture the starting position and ending position of gene by aligning the sequence with its reference gene sequence.

Sequence Length: The Length of the sequence plays an important role in examining the difference in length of the sequence. When the insertion or deletion mutation occurs, the length of the sequence gets varied automatically. This feature is determined using Length() function by converting the fasta file into a data frame.

Number of Exons inserted/deleted: Severe effect on the deletion of exons leads to DMD and mild deletion of exons will results in BMD. While gross insertions and gross deletions occurred, the severity of the disease is determined with number of exons inserted or deleted. This descriptor is calculated by using the deletion region information column in the HGMD.

Exon and intron boundary: Every gene sequence is comprised of coding (exonic) and non coding (intronic) regions. Boundary of exonic and intronic region gets altered when Insertion/duplication or deletion of exons occurs and so, these descriptors are captured to identify the differences in the boundary between the normal and the diseased sequences. By visualizing the sequences in geneious pro these descriptors are captured.

Deletion type: If the sequence can be still read after deletion mutation occurs, then it is considered as inframe deletion. In outframe deletion type, the sequence cannot be read after the deletion mutation occurs. Deletion type is a contributive feature in identifying the type of the disease as in some diseases like BMD, where the sequence can be still read after deletion and in some diseases like DMD, the sequence cannot be read after deletion as it is outframe. This feature is captured by translating the diseased sequences into its corresponding amino acid sequence. Splitseq (), tablecode () functions from biostrings, seqinr packages are used to capture this descriptor.

Exon type: Depending on location of the exon, the type of exon may be Initial, Internal, Terminal and Single exons. The mutation in each type of exon has its own severity which aids in classifying the disease type. This discriminative feature is captured using geneious pro tool.

Conservation score: The structure or the function of the sequence is identified by calculating the conservation score by aligning the sequence with all organisms. University of California Santa Cruz (UCSC) genome browser is employed to calculate the conservation score.

Protein Coding Region score: The score of the protein coding region is calculated with coding potential calculator based on the sequence features to distinguish protein coding from non coding regions. When a deletion occurs in an exon the protein coding region score decides the severity of deletion on the sequence.

Nucleotide composition values: The base composition A, C, G,T are calculated to count the number of occurrences of the four different nucleotides ("A", "C", "G", and "T") in the sequence. GC content is the fraction of the sequence that consists of Gs and Cs, i.e. The GC content can be calculated as the percentage of the bases in the genome that are Gs or Cs. That is,

AT content = (number of As + number of Ts)*100/ (genome length)

GC content = (number of Gs + number of Cs)*100/ (genome length)

Therefore six different descriptor values are calculated as the nucleotide composition values.

Stop codon position: The position of the Stop codon reveals the end of the coding part in the sequence. This position may be altered with the occurrence of mutation and hence it is noted. match pattern () function is employed to identify and capture the position of stop codon.

Alignment Scores: Alignment scores are considered as the important feature for disease prediction. The global pairwise alignment based on edit distance is done with the mutated sequence against with the reference cDNA sequence and the three alignment scores are calculated using the edit distance scoring method. The PhredQuality measures are calculated with the patternQuality and subjectQuality to examine the quality-based match and mismatch bit scores for DNA/RNA. The substitution scores are calculated by setting the error probability to 0. Table 5.7 depicts the IDM features and their descriptions.

Table 5.7 IDM Features and their Descriptions

Features	**Description**
Gene ID	Identifier of the gene taken from NCBI
Gene Symbol	Name of the gene involved
Chromosome Number	The chromosome involved in mutation
Alteration type	Mutation type such as missense, non sense, silent, deletion and duplication
Gene start position	The starting position of the gene
Gene end position	The starting position of the gene
Length	Length of the mutated gene sequence
Exon Boundary	The position of inserted or deleted exons boundary
Intron Boundary	The position of inserted or deleted introns boundary
Deletion Type	Deletion type whether inframe or outframe
No. of Exons Deleted	Number of exons deleted or inserted
Exon Type	Type of exon. Initial, Internal, Terminal and Single exons
Starting position of Exon	The starting position of the Exon
Ending position of Exon	The ending position of the Exon
Nucleotide Composition	Composition of A, C, G, T, AT, GC in mutated sequence.
Position of stop codon	Last position of stop codon ATG
Edit distance scores	Alignment scores using edit distance method
PhredQuality measures	Calculated with patternQuality and subjectQuality
Substitution scores	Calculated with the error probability set to 0 or 1

Feature vector for a sequence is as follows:

Gene ID	:	1746
Gene Symbol	:	DMD
Alteration type	:	2
Length	:	74
Exon Boundary	:	345
Intron Boundary	:	654
Deletion Type	:	Inframe
No. of Exons Deleted	:	2
Exon Type	:	Internal exon
Starting position of Exon	:	14
Ending position of Exon	:	47
Conservation Score	:	1.493
Protein Coding region score	:	24.9
Nucleotide Composition	:	A - 33.33 T-20.75 G-23.7 C-22.22 AT- 44.45 GC-55.55
Edit distance scores	:	22113
PhredQuality measures	:	21907.28
Substitution scores	:	18816.67

Training Dataset

The above twenty-three features are extracted from each diseased gene sequence and a dataset (IDM) with 1000 feature vectors is created. The sample training dataset is depicted below.

	1756	1	11058	2	74	345	654	0	2	14	47	
	1.493	24.9	22113	18816.67		33.33	20.75	23.7	22.22	44.45	55.55	1
	1756	1	10409	5	7	1	7	2	649	245	893	
	15624	18022.56		15110.32		33.33	20.79	23.79	22.1	44.58	55.42	1
	1756	1	10524	5	3	10	12	1	522	1205	1727	
	16774	18710.54		15766.83		33.16	21	23.65	22.19	44.65	55.35	2
	1756	1	8799	5	13	16	29	1	2258	2057	4315	-
476	8391.182		5930.44		32.8	20.98	23.87	22.35	44.85	55.15	2	
	2010	2	715	5	1	5	5	0	50	648	697	
	1028	1206.958		1006.864		19.86	32.87	23.78	23.5	56.64	43.36	3
	2010	2	503	5	2	4	5	0	261	436	697	-
1	092	-61.17883		-201.9393		17.3	35.19	23.26	24.25	58.45	41.55	3
	10585	12	1991	5	2	19	20	0	1052	2094	3146	
	1956	2923.671		2366.496		19.49	27.27	29.18	24.06	56.45	43.55	4
	10585	12	1939	5	2	18	19	0	304	1967	2271	
	1436	2612.63	2070	20.17	27.59	28.83	23.41	56.42	43.58	4		
	5376	44	504	4	1	2	2	0	20	536	556	
	796	863.1859		728.0229		17.86	30.36	24.21	27.58	54.56	45.44	5
	5376	44	966	4	4	2	5	0	483	209	691	-
	2900	-984.7863		-1119.978		18.01	30.64	23.6	27.74	54.24	45.76	5

Building the Model

The models demonstrated in section 5.2 and 5.3 concentrates on non-synonymous and synonymous mutations where as in this experiment IDM dataset is employed in building the disease classification model. Numpy arrays are generated based on the IDM related features. The dataset is normalized using min max normalization. Feature object matrix is created and target value has been set. The same classification algorithms are adopted to build the models. In case of decision tree classifier and Naïve bayes default parameter settings are used during classification whereas in Artificial neural network implementation, the hidden layers are tuned to gain a stable accuracy. In case of SVM, Radial basis kernel performed well with the cost value of 1 and gamma value 0.016.The number of support vectors created by this model is 90.

```
from sklearn import metrics
from sklearn.svm import SVC
# fit a SVM model to the data
model = SVC()
model.fit(X, y)
print(model)
# make predictions
expected = y
predicted = model.predict(X)
```

An example of an estimator is the class sklearn.svm.SVC that implements support vector classification. The constructor of an estimator takes as arguments the parameters of the model.

```
Class sklearn.svm.SVC(C=1.0, kernel='rbf', degree=3, gamma='auto', coef0=0.0, shrinking=True, probability=False, tol=0.001, cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_function_shape=None, random_state=None)
```

The implementation is based on libsvm.

```
from sklearn import metrics
from sklearn.svm import SVC
estimator = SVC(kernel = 'rbf')
```

Cross validation iterator

```
from sklearn.cross_validation import ShuffleSplit
cv = ShuffleSplit(X_train.shape[0], n_iter=10, test_size=0.2, random_state=0)
```

Applying Cross validation on the training set

The sklearn provides an object that, given data, computes the score during the fit of an estimator on a parameter grid and chooses the parameters to maximize the cross-validation score. This object invokes an estimator while construction and depicts an estimator API.

```
from sklearn.grid_search import GridSearchCV
gammas = np.logspace(-6, -1, 10)
classifier = GridSearchCV(estimator=estimator, cv=cv, param_grid=dict(gamma=gammas))
classifier.fit(X_train, y_train)
```

Evaluating on the test dataset

```
classifier.score(X_test, y_test)
Train final model on whole dataset
classifier.fit(X, y)
```

Performance Evaluation

The predictive performance of the disease classification models have been evaluated as done in earlier experiments and shows that SVM classifier yielded a best accuracy of 86.3% and the results are tabulated in Table 5.8 and Table 5.9 and drawn in figures Fig.5.7.

Table 5.8 Predictive Performance of the Classifiers (Insertion, Deletion and Duplication Mutations)

Performance criteria	**Decision Tree Classifier**	**Artificial Neural Network**	**Naïve Bayes Classifier**	**SVM**
Correctly classified Instance	853	856	831	863
Incorrectly classified instance	147	144	169	137
Prediction accuracy	**85.3**	**85.6**	**83.1**	**86.3**

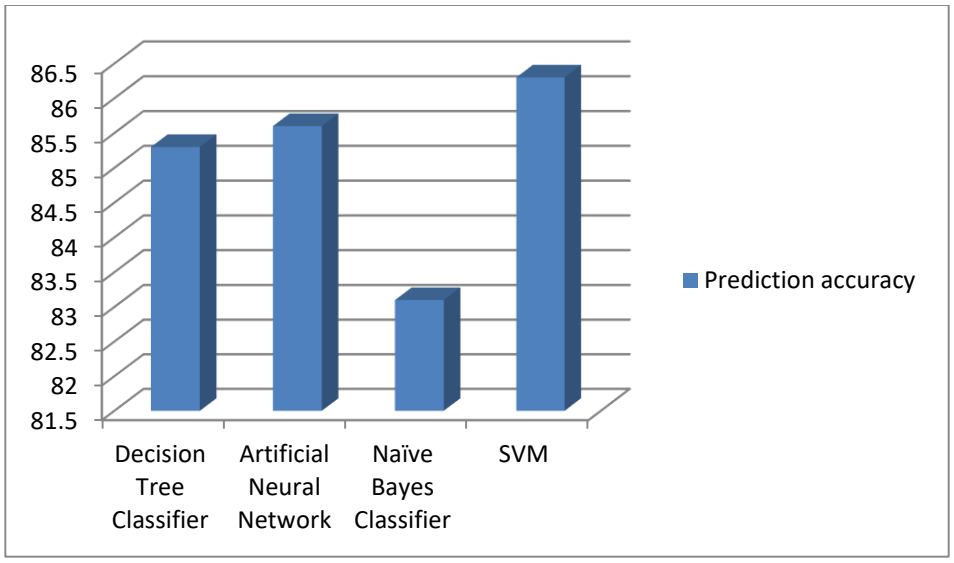

Fig.5.7 Predictive Accuracy of the Classifiers (Insertion, Deletion and Duplication Mutations)

Table 5.9 Performance Evaluation of the Classifiers (IDD mutations)

Performance criteria	Decision Tree Classifier	Artificial Neural Network	Naïve Bayes Classifier	SVM
Prediction accuracy	**85.3**	**85.6**	**83.1**	**86.3**
Precision	0.853	0.86	0.831	0.883
Recall	0.85	0.8	0.83	0.89
F1 Score	85.3	85.9	83.1	88.3
Cohen's Kappa	0.88	0.862	0.81	0.89
Time taken to build the model (in sec)	8.7	9.6	11.7	7.6

The precision- recall curve is plotted for each class based on SVM linear classifier Fig.5.8 shows the precision-recall curve for each class in SVM classifier. Fig.5.9 shows the ROC curve for each class in SVM classifier.

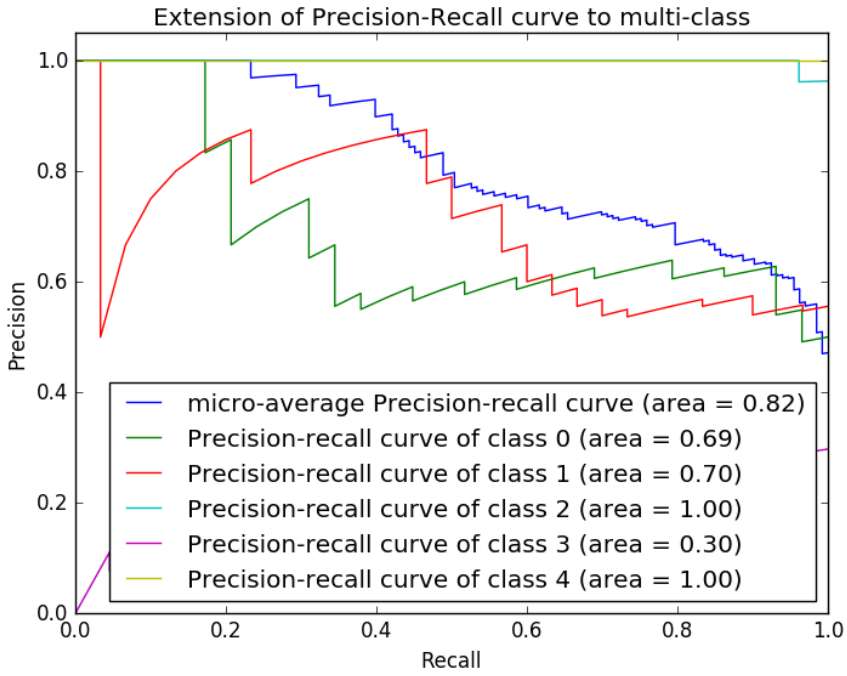

Fig.5.8 Precision Recall Curve for SVM Classifier

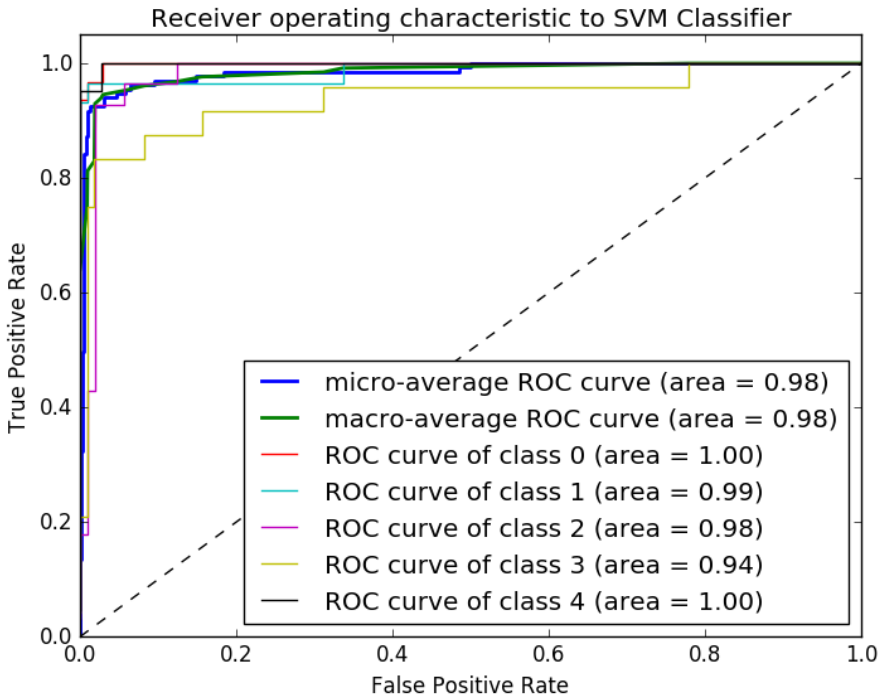

Fig.5.9 ROC Curve SVM Classifier

Findings

It is perceived from the above results the kappa statistic and prediction accuracy is high for SVM than other algorithms. The precision and recall measure for SVM is high when compared with other learning methods. Overall, the balance accuracy measure is also prominent in SVM when measured with other algorithms. The cohen's kappa is also high in SVM with 0.861 score and the time taken is minimal of 7.6 seconds.

5.5 Model IV: Predicting Muscular Dystrophy Disease using Features Related to Splicing Mutations

The exons are formed by splicing out the introns during transcription and the mutations occurred while spicing should be considered to know the alteration after the splicing process. The key idea in this experiment is to spot out discriminative descriptors from diseased gene sequences based on splicing variants and to provide an effective machine learning solution for predicting the type of muscular dystrophy disease with the splicing mutations. SNP, Gene and Exon-based discriminative features are identified and utilized to train the model.

In this experiment, the cloned gene sequences are synthesized based on the mutation position and its location on the chromosome. For instance, in the database, the mutational information for splicing mutation is specified as IVS1-5 T>G indicates (IVS- intervening sequence or introns), 1st intron and 5 nucleotides before the consensus intron site AG, where the variant occurs in nucleotide G altered to T. IVS (+ve) denotes forward strand 3' – positive numbers from G of donor site invariant and IVS (-ve) denotes backward strand 5' – Negative numbers starting from G of acceptor site.

Feature Extraction

The discriminative descriptors aids in diagnosing the identification of exonic single base substitutions that modulate splicing. Descriptors are the arrays of features that were derived based upon the genomic coordinate of the substitution in the human reference gene. Gene id, Gene symbol, chromosome number, variant exon number, exon boundary, Intron boundary, sequence length, splice site distance, PhyloP and PhastCons score, ESR Change, Donor site score, Acceptor site score, Branch site score, Splice site scores, Distance of alteration from 5' splice site, Distance of alteration from 3' splice site, scoring splice site with PWM, flanking

intron size, GC content, Exon size, Constitutive exon, Exon type and coding region score are the twenty four features identified and captured.

Variant exon number: Variant exon number gives the mutant exon's number in the target isoforms. As the exonic splicing features are captured in this work this exon number gives the position of the exon that is altered while splicing. This feature is captured through geneious pro tool.

Splice site distance: The distance of the substitution from the variant to the nearest splice site is identified and recorded as splice site distance. The splice site distance aids in identifying the severity of the disease.

PhyloP and PhastCons score: PhyloP is an evolutionary conservation element that computed the base-wise sequence conservation score of single base substitution which is calculated based on multiple sequence alignment. PhastCons is a base wise conservation element examined from probability for substitution site, based on multiple alignments. PhyloP and PhastCons scores are downloaded from the UCSC Genome Browser.

ESR Change: The regulatory sequences located within the exon and promoting exon inclusion are referred to as Exonic Splicing Regulatory (ESR) elements. ESR change identifies the change in the frequency of ESR elements with respect to single variants. To strengthen or repress the elements in the sequences Exonic Splicing Enhancers (ESE) and Exonic Splicing Silencers (ESS) is calculated using ESE finder tool. The ESR changes helps in recognizing the adjacent splice site. Counting the occurrences of nucleotides at each position within the 5' splice site is done using PWM – Position Weight Matrices that is calculated as log odds score.

Site scores (3): The 3 sites Acceptor site, branch site and donor site are altered while splicing and therefore the cut off score of these sites should be calculated to observe the change in the scores after alteration. These scores are calculated using ESE finder tool.

Scoring splice site with PWM: PWM – It is necessary to identify the score of the splice sites with the Position Weight Matrices (PWM). PWM will count the occurrences of nucleotides at each position within the 5' splice site. It is calculated as log odds score.

Flanking intron size: Flanking intron size is the length of the base pairs of the upstream and downstream introns nearby the target exon. The intron size is captured using geneious pro tool.

The features and their description are depicted in Table 5.10.

Table 5.10 Features and their Descriptions

Features	Description
Gene ID	Identifier of the gene taken from NCBI
Gene Symbol	Name of the gene involved
Chromosome Number	The chromosome involved in mutation
Variant Exon number	Mutant exon's number in the target isoforms
Length	Length of the mutated gene sequence
Exon Boundary	The position of inserted or deleted exons boundary
Intron Boundary	The position of inserted or deleted introns boundary
Splice site distance	The distance of the substitution from the variant to the nearest splice site
PhyloP and PhastCons score	Evolutionary conservation element that computed the base-wise sequence conservation score
ESR Change	change in the frequency of ESR elements
Site scores (3)	Acceptor Site, Branch site and Splice site scores
Distance of alteration from 5' splice site	Composition of A, C, G, T, AT, GC in mutated sequence.
Distance of alteration from 3' splice site	Last position of stop codon ATG
Scoring splice site with PWM	Score of the splice sites with the Position Weight Matrices
Flanking intron size	Length of the base pairs of the upstream and downstream introns
Exon size	Number of exons
Exon type	Type of exon. Initial, Internal, Terminal and Single exons
Coding region score	Score of the coding region
GC content	Calculated with the error probability set to 0 or 1

Sample feature values are shown below.

Gene ID	:	2010
Gene Symbol	:	EMD
Chromosome Number	:	23
Variant Exon number	:	1
Length	:	764
Exon Boundary	:	1
Intron Boundary	:	331
Splice site distance	:	32
PhyloP and PhastCons score	:	1.493
Acceptor Site	:	-7.3
Branch site	:	-32.7
Donor site score	:	-4.39
Distance of alteration from 5’ splice site :64		
Distance of alteration from 3’ splice site :80		
Exon size	:	330
Exon type	:	Internal
Coding region score	:	27.26
GC content	:	1

Training Dataset

Position of the spliced introns and exons are carefully examined and gene, exon and snp features are extracted as described above from 1000 gene sequences. 1000 feature vectors of dimension 24 are created and the dataset (SPM) is prepared. For each feature vector, the class label is assigned a sequence number 1 to 5 designating the category of muscular dystrophy disease. The sample training dataset is depicted below.

1756	1	23	11058	440	1	4	0	78	2	11254	32
0	0	0	0.953	2235	26.45	1					
1756	1	23	11058	440	1	4	0	78	2	11254	19
0	0	0	1.493	2235	26.45	1					
1756	1	23	11058	2381	0	1	2	275	2	11254	2
-7.3	-32.7	-4.39	3.79	2234	27.26	2					
1756	1	23	11058	1601	0	17	19	176	2	11254	2
-14.72	-39.78	-4.33	1.403	2234	27.23	2					
2010	2	23	765	263	0	1	1	330	2	738	81
-8.92	3.12	-8.733	0.838	764	6.23	3					
2010	2	23	765	81	0	1	3	330	2	738	108
-16.25	-18.72	-11.15	0.462	764	6.177	3					
3730	4	23	2742	40	0	10	13	135	2	2135	60
-12.85	-23.5	-10.1	-0.593	2741	11.74	4					
3730	4	23	2742	763	0	1	1	357	2	2135	168
-24.28	-43.2	-10.07	3.235	2741	11.74	4					
4359	40	1	747	448	0	4	5	136	2	713	6
-3.43	-9.34	-0.85	4.312	746	3.32	5					
4359	40	1	747	586	0	3	3	214	2	713	2
-16.8	-38.84	-10.63	4.754	746	3.39	5					

Building the Model

In this experiment the extracted descriptors from the diseased gene sequences are stored in *.csv files. The .csv files are converted into a numpy array as scikit – learn library accepts a numpy array in its implementation. The data frame is built with the numpy array. Normalizing the data by transforming the feature values into the range between 0 and 1 aid in scaling the input attributes for a model. Supervised learning algorithms - Naïve bayes, decision tree, ANN, SVM are employed to develop models using python library framework in scikit learn. While learning SVM model, the cost, gamma and kernel parameters are tuned to attain good results. The python script for building the model is shown below.

```
from sklearn import metrics
from sklearn.svm import SVC
# fit a SVM model to the data
model = SVC()
model.fit(X, y)
print(model)
# make predictions
expected = y
predicted = model.predict(X)
# summarize the fit of the model
print(metrics.classification_report(expected, predicted))
print(metrics.confusion_matrix(expected, predicted))
```

Performance Evaluation

The training was performed using SPM dataset and the predictive performance of the disease classification shows that SVM classifier attains an accuracy of 86.7%. The results are tabulated in Table 5.11 and depicted in Fig.5.10.

Table 5.11 Predictive Performance of the Classifiers (Splicing Mutations)

Performance criteria	Decision Tree Classifier	Artificial Neural Network	Naïve Bayes Classifier	SVM
Correctly classified instances	849	835	813	867
Incorrectly classified instances	151	165	187	133
Prediction accuracy	**84.9**	**83.5**	**81.3**	**86.7**

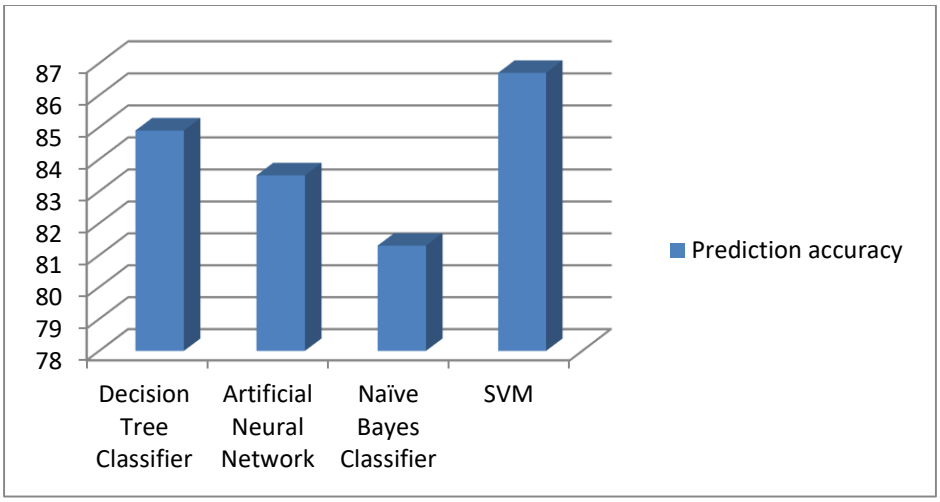

Fig.5.10 Predictive Accuracy of the Classifiers (Splicing mutations)

The performances of the classifiers are evaluated and the measures such as prediction accuracy, precision, recall, F1- score, cohen's kappa and Time taken to build the model are calculated and tabulated in Table 5.12.

Table 5.12 Performance Evaluation of the Classifiers (Splicing Mutations)

Performance criteria	Decision Tree Classifier	Artificial Neural Network	Naïve Bayes Classifier	SVM
Prediction accuracy	**84.9%**	**83.5%**	**81.3%**	**86.7%**
Precision	0.849	0.83	0.81	0.86
Recall	0.846	0.815	0.8	0.87
F1 Score	85.1	82.9	79.9	86.7
Cohen's Kappa	0.841	0.81	0.802	0.867
Time taken to build the model (in sec)	7	8	13.6	6.5

The output of the binary classifier is typically studied with the Precision-recall curves. In this multi-class classification work, the target attribute values are binarized and the class values are shaped into 1. The precision- recall curve is computed by adding some noisy features and the micro-average ROC and ROC area is calculated. The precision- recall curve is plotted for each class based on SVM linear classifier. Fig.5.11 shows the precision-recall curve for each class in SVM classifier.

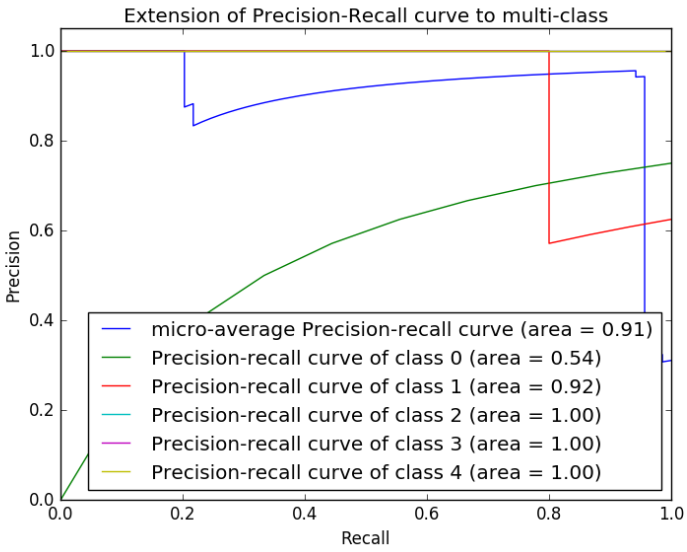

Fig.5.11 Precision Recall Curve SVM Classifier

The above figure insists that class 2, 3 and class 4 have high precision of 1.00 and class 0is curved low at the area 0.54. On the average the micro-average curve shows the value of 0.91. ROC curve is plotted for SVM linear classifier for each class. Fig.5.12 depicts the ROC curve based on SVM classifier in predicting muscular dystrophy.

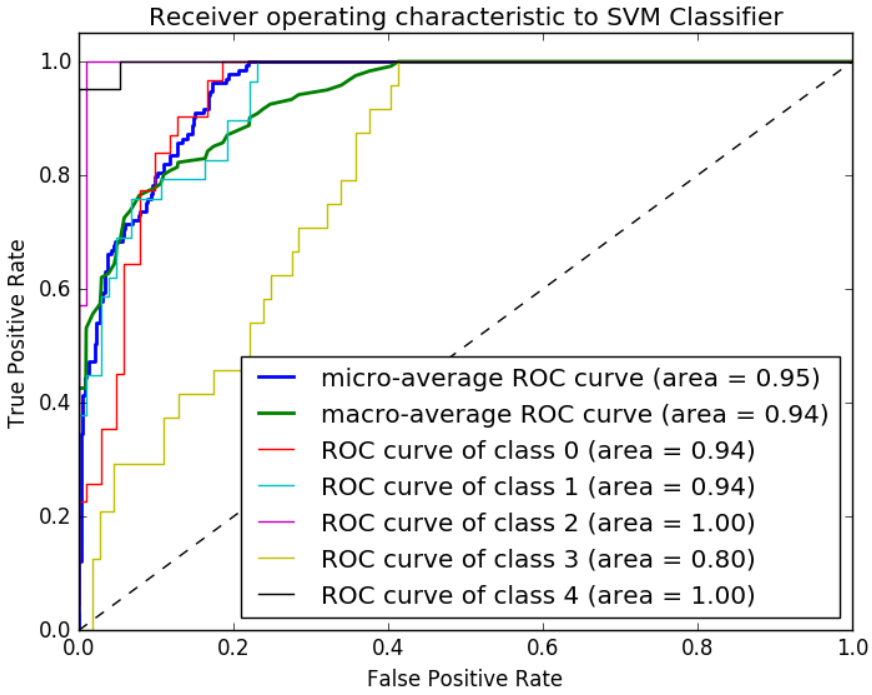

Fig.5.12 ROC Curve SVM Classifier

Findings

From the above results it is observed that the prediction accuracy of 86.7% is attained from SVM classifier. High precision and Recall measures of about 0.80 and 0.87 is attained for SVM classifier. The cohen's kappa with 0.867 score and the time taken of 6.5 seconds is promotable in SVM classifier. Overall, the balance accuracy measure is also prominent in SVM when measured with other algorithms.

5.6 Model V: Predicting Muscular Dystrophy Disease using Features Related to Aggregated Mutational Descriptors

Finally, a data driven model is developed by aggregating the features related to all kinds of mutations for predicting the disease precisely. In all the previous works, autonomous disease identification models were built based on the specific mutational features. However, the type of mutation caused in the gene sequence may not be known explicitly and hence all the mutational features are accumulated to facilitate efficient learning for predicting the disease caused by any mutation.

Training Dataset

So far in the literature no attempt was made to build disease identification model by aggregating all kind of mutational descriptors and hence it is significant to build this type of disease identification model. In this experiment, AGM dataset is formed by pooling all the mutational features described in previous sections. 106 evocative features are cumulated by eliminating the repetitive features without losing information from 132 features and feature vectors are created with labels 1 to 5 for learning the disease prediction models. Another dataset is formed by selecting the subset of attributes using the information gain selection attribute method and 73 highly ranked attributes are chosen as feature vectors.

```
1756 1 23 63 11058 4 163 487 489 732 1
1 18 8 1 11056 0 0 22106 21901.15
18806.38 1 486 33.33 20.75 23.7 22.22 44.45 55.55
1.0511945 0.8121212 0.9488055 1.1878788 1.52 0.76 0.24
1.48 1.8484848 0.9876543 1.4242424 1.3580247 0.5263158
1.1052632 1.3684211 0.8701299 0.8636364 1.1298701
1.1363636 1.2519084 0.9770992 0.1221374 1.648855
0.8787879 0.5454545 0.6666667 0.6363636 0.7384615
0.8703297 1.410989 1.0285714 1.2248062 0.8186528
0.7751938 1.1813472 1.2394366 1.1830986 0.1502347
1.42723 1.6637168 0.920354 0.7787611 0.6371681
0.8121827 0.9340102 1.4010152 0.8527919 0.6129032
1.3870968 1.2592593 0.9135802 0.1481481 1.3333333
1.1666667 0.8333333 0.7120879 0.6419753 1.2395604
1.3580247 0 0 0 0 1 10474 10474 0 1
```

1756 1 23 93 11058 4 48 142 144 449 1 1
16 3 1 2464 0 0 3980 3939.565 3381.302 1
142 21.84 29.56 33.72 14.88 63.28 36.72 0.6551724 1.6190476
1.3448276 0.3809524 1.4 1.8 0.4 0.4 1.3953488 1.3584906
0.8372093 0.9056604 0.1914894 1.8510638 0.9574468 0.3783784
1.6190476 1.6216216 0.3809524 1.025641 1.025641 0.8205128
1.1282051 0.4186047 0.9767442 1.9534884 0.4186047 0.3050847
1.9322034 2.9491525 0.6101695 0.3666667 1.0526316 1.6333333
0.9473684 1.0909091 1.8181818 0.1818182 0.9090909 1.0196078
1.1764706 1.0196078 0.7843137 0.1818182 1.0909091 1.8181818
0.9090909 1.12 0.88 0.5660377 1.4716981 0.2264151 1.4716981
1.625 0.375 0 1.5652174 0.2033898 0.4347826 1 696
696 0 1 2 275 11254 2 -7.3 -32.7 -4.39 3.79 27.26 2

2010 2 23 0 747 5 105 313 315 557 1 1
12 0 32 11056 1 2 1956 2923.671 2366.496 1
132 19.49 27.27 29.18 24.06 56.45 43.55 1.0511945 0.8121212
0.9488055 1.1878788 1.52 0.76 0.24 1.48 1.8484848 0.9876543
1.4242424 1.3580247 0.5263158 1.1052632 1.3684211 0.8664495
0.8636364 1.1335505 1.1363636 1.2519084 0.9770992 0.1221374
1.648855 0.8787879 0.5454545 0.6666667 0.6363636 0.7384615
0.8703297 1.410989 1.0285714 1.2248062 0.8186528 0.7751938
1.1813472 1.2394366 1.1830986 0.1502347 1.42723 1.6785714
0.9285714 0.75 0.6428571 0.8121827 0.9340102 1.4010152
0.8527919 0.6129032 1.3870968 1.2592593 0.9135802 0.1481481
1.3333333 1.1666667 0.8333333 0.7120879 0.6419753 1.2395604
1.3580247 2 19 20 0 1052 2094 3146 738 17
-1.162 0 3

3730 4 23 1295 2742 45 133 135 439 1 1 1
17 1 2464 0 0 3928 3907.675 3351.09 1 132
21.87 29.61 33.63 14.88 63.25 36.75 0.6551724 1.6097561 1.3448276
0.3902439 1.3658537 1.8536585 0.3902439 0.3902439 1.3953488
1.4444444 0.8372093 0.8888889 0.1914894 1.8510638 0.9574468
0.3783784 1.6190476 1.6216216 0.3809524 1.025641 1.025641
0.8205128 1.1282051 0.4186047 0.9767442 1.9534884 0.4186047
0.3050847 1.9322034 2.9491525 0.6101695 0.3666667 1.0526316
1.6333333 0.9473684 1.1162791 1.7674419 0.1860465 0.9302326
1.0196078 1.1764706 1.0196078 0.7843137 0.1818182 1.0909091
1.8181818 0.9090909 1.12 0.88 0.5555556 1.4444444 0.2222222
1.4444444 1.625 0.375 0 1.5652174 0.2033898 0.4347826 6
452 452 10 13 135 2135 60 -12.85 -23.5 -10.1 -0.593 11.74 4

```
79628 49 5 2859 3867 241 721 723 965 1 1 6
0 32 11056 1 2 -180 571.5369 380.4075 1
720 25.33 21.82 25.92 26.94 47.73 52.27 1.0511945 0.8121212
0.9488055 1.1878788 1.52 0.76 0.24 1.48 1.8484848 0.9876543
1.4242424 1.3580247 0.5263158 1.1052632 1.3684211 0.8664495
0.8636364 1.1335505 1.1363636 1.2519084 0.9770992 0.1221374
1.648855 0.8787879 0.5454545 0.6666667 0.6363636 0.7384615
0.8703297 1.410989 1.0285714 1.2248062 0.8186528 0.7751938
1.1813472 1.2394366 1.1830986 0.1502347 1.42723 1.6785714
0.9285714 0.75 0.6428571 0.8121827 0.9340102 1.4010152
0.8527919 0.6129032 1.3870968 1.2592593 0.9135802 0.1481481
1.3333333 1.1666667 0.8333333 0.7120879 0.6419753 1.2395604
1.3580247 2 5 6 0 192 541 733 0 0 11
11 1695 3778 14 -17.3 -9.1 -1.22 1.792 19.175 5
```

Building the Model

The predictive performance of the classifiers using the aggregated features is carried out in two implementations before and after feature selection. Numpy arrays are generated based on the IDM related features. The dataset is normalized using min max normalization. In all the previous experiments, autonomous disease identification models have been built based on the specific mutational features. In this experiment AGM dataset is employed for training the decision tree, naïve bayes, ANN and SVM models. The script for building the classifier is shown below.

```
import numpy as np
import pandas as pd
import io
df=pd.read_csv('C:\Users\HCL\Documents\pooled_sample_2_scikit.csv')
print df
from numpy import genfromtxt
my_data = genfromtxt('C:\Users\HCL\Documents\pooled_sample_2_scikit.csv', delimiter=',')
X = my_data[:,0:105]
y = my_data[:,106]
from sklearn import preprocessing
normalized_X = preprocessing.normalize(X)
standardized_X = preprocessing.scale(X)
```

```
from sklearn import metrics
from sklearn.svm import SVC
# fit a SVM model to the data
model = SVC()
model.fit(X, y)
print(model)
# make predictions
expected = y
predicted = model.predict(X)
# summarize the fit of the model
print(metrics.classification_report(expected, predicted))
```

Performance Evaluation

The experiments conducted on the diseased gene sequences are assessed on the model built with evaluation methods. The cross validation results of the classifiers are shown in Table 5.13 and illustrated in Fig.5.13 and Fig.5.14.

Table 5.13 Predictive Performance of the Classifiers (Pooled Features)

Performance criteria	**Decision Tree Classifier**	**Artificial Neural Network**	**Naïve Bayes Classifier**	**SVM**
Correctly classified Instance	849	835	813	867
Incorrectly classified instance	151	165	187	133
Prediction accuracy	**84.9**	**83.5**	**81.3**	**86.7**

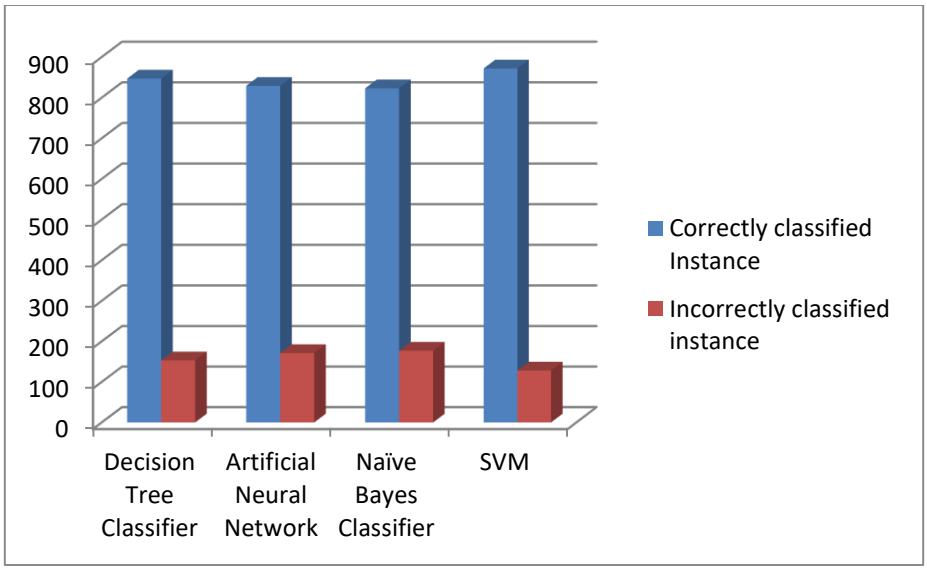

Fig.5.13 Predictive Performance of the Classifiers (Pooled Features)

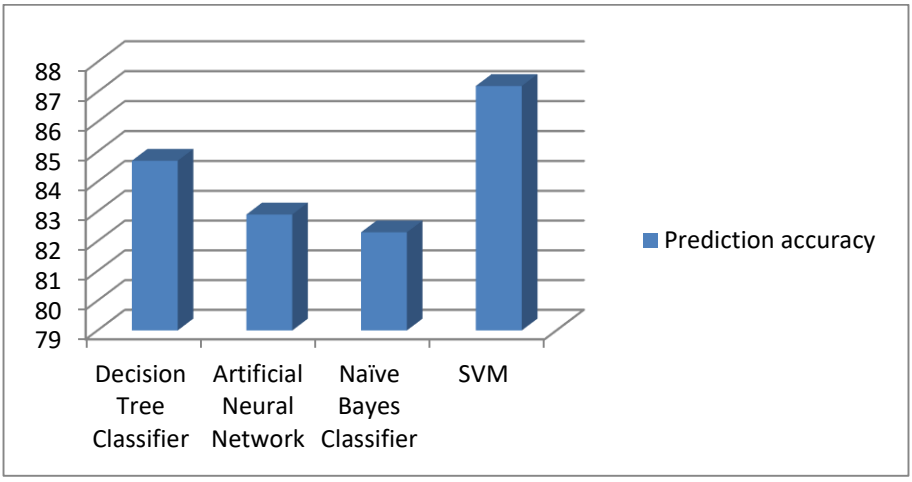

Fig.5.14 Predictive Accuracy of the Classifiers (Pooled Features)

The experiment was carried out with selected subset of attributes and a model is built using the standard pattern recognition algorithms. The performance of SVM classifier observed better accuracy of 90.3%. The performances of the classifiers are evaluated with respect to the measures such as prediction accuracy, precision, recall, F1- score, cohen's kappa and Time taken to build the model The results of the classifiers are shown in Table 5.14, Table 5.15 and illustrated in Fig.5.15.

Table 5.14 Predictive Performance of the Classifiers (After Feature Selection)

Performance criteria	Decision Tree Classifier	Artificial Neural Network	Naïve Bayes Classifier	SVM
Correctly classified Instance	847	829	823	903
Incorrectly classified instance	153	171	177	97
Prediction accuracy	**84.7**	**82.9**	**82.3**	**90.3**

Table 5.15 Performance evaluation of the Classifiers (After Feature selection)

Performance criteria	Decision Tree Classifier	Artificial Neural Network	Naïve Bayes Classifier	SVM
Prediction accuracy	**84.7%**	**82.9%**	**82.3%**	**87.2%**
Precision	0.847	0.829	0.823	0.872
Recall	0.847	0.82	0.82	0.881
F1 Score	84.1	82.1	82.1	87.2
Cohen's Kappa	0.847	0.83	0.83	0.87
Time taken to build the model (in sec)	7	9.7	9.7	5.2

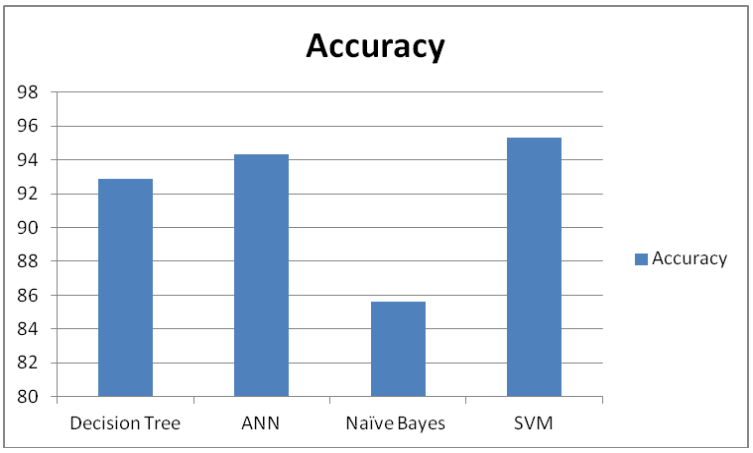

Fig.5.15 Predictive Accuracy of the Classifiers (After Feature Selection)

The comparative analysis of the models built before feature selection and after feature selection is made with respect to predictive accuracy and presented in Table 5.16 and in Fig.5.16. This comparative analysis shows that models built with high ranked features produce better results.

Table 5.16 Comparative Study of Prediction Accuracy Before and After Feature Selection

Prediction accuracy	Decision Tree Classifier	Artificial Neural Network	Naïve Bayes Classifier	SVM
Before Feature Selection	84.9	83.5	81.3	86.7
After Feature Selection	**84.7**	**82.9**	**82.3**	**90.3**

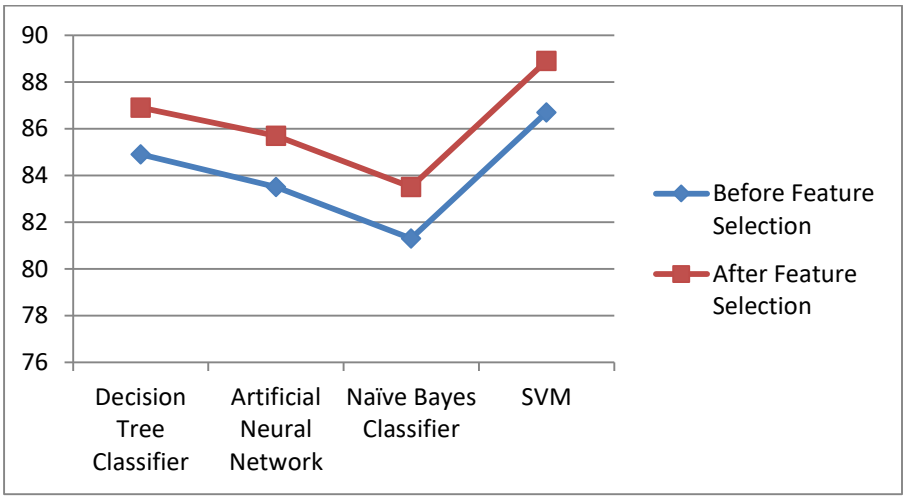

Fig.5.16 Comparative Study of Prediction Accuracy Before and After Feature Selection

From the above experiments, it was observed that the performance of the classifiers is high when training dataset contains summative features. The classification models built using non-synonymous mutational features produced an accuracy of about 84.9%. The classification models built using features related to synonymous mutations produced an accuracy of about 86%. 86.3% accuracy was attained when insertion/duplication and deletion mutational features are taken into account. Disease prediction model reached an accuracy of 86.7% when splicing mutational features are considered. When all the mutational features are pooled together, the models showed an accuracy of about 87.2%.

The output of the binary classifier is typically studied with the Precision-recall curves. In this multi-class classification work, the target attribute values are binarized and the class values are shaped into 1. The precision- recall curve is computed by adding some noisy features and the micro-average ROC and ROC area is calculated. The precision- recall curve is plotted for each class based on SVM linear classifier. Fig.5.17 shows the precision-recall curve and Fig.5.18 shows the ROC curve for each class in SVM classifier.

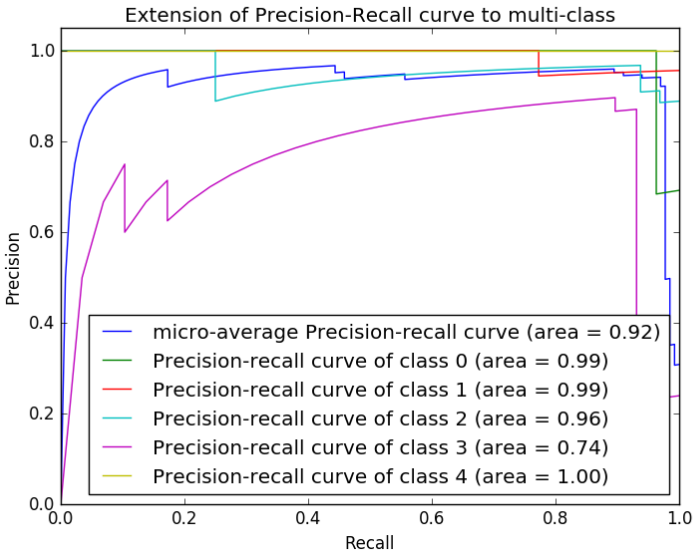

Fig.5.17 Precision Recall Curve for SVM Classifier

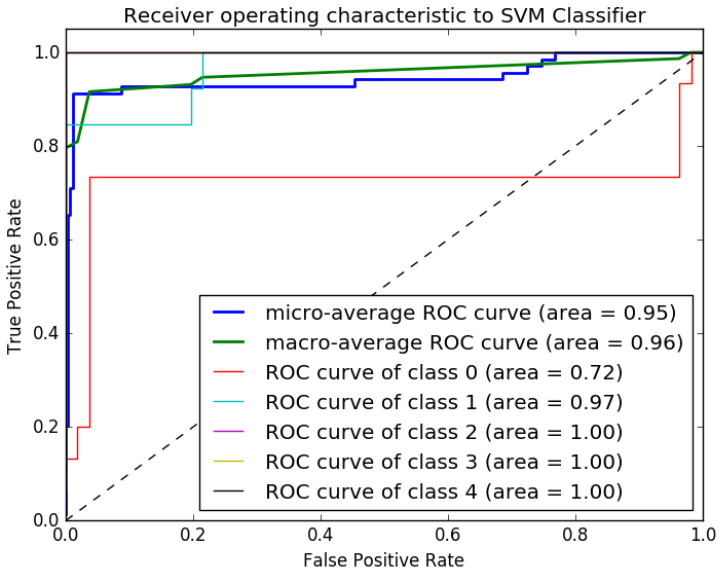

Fig.5.18 ROC Curve for SVM Classifier

Findings

Downsizing the features through feature selection expedites to improve the outcome and the prediction accuracy of the SVM classifier built using high ranked features was hoisted to 90.3%. Hence, it is observed that pooling the descriptors associated with all type of mutations produced augmented trained models for meticulous disease prediction.

5.7 Summary

This chapter demonstrates the modeling of disease identification work as the problem of learning multiclass classification system that can suits in bioinformatics environment to identify the disease effectively. It describes the implementation of shallow learning approach for identifying the genetic disease based on the mutational features. Five different models were built to identify the disease based on diverse features associated to different kind of mutations. The AGM based disease identification model is a generalized model which can identify any kind of disease effectively by aggregating all type of mutational features. The outcome of the experiments proves that, the disease identification model is effectual when the collective features are used in learning. The results shows that our method is valuable than existing disease identification procedures with respect to significant features.

Remarks

1. Paper titled “Predicting Muscular Dystrophy with Sequence based Features for Point Mutations”, is presented in the conference, IEEE Conference on research in Computational Intelligence and communication Network, IEEE CIS Kolkata chapter on Nov 2015, and published in the conference proceedings of ISBN 978-1-4673-6734-9, pp: 235 – 240.
2. Paper titled “Predicting Muscular Dystrophy through Genetic testing – A Study”, is a review paper presented in the International Conference on Innovative trends in Electronics Communication and Applications, ASDF and IIT Madras Research park, Chennai, Dec 2015and published in the conference proceedings of ISBN 978-81-929742-6-2. Vol – 01, pp: 65-71.
3. Paper titled "Muscular Dystrophy Disease Classification Using Relative Synonymous Codon Usage", has been published in the International Journal of Machine Learning and Computing vol.6, no. 2, ISSN- 2010-3700, pp. 139-144, 2016.
4. Paper titled “Identification of Rare Genetic Disorder from Single Nucleotide Variants Using Supervised Learning Technique”, has been published by International journal of control theory and applications, Vol.9, no.34, pp. 801-810, 2016. (Scopus Indexed)
5. Paper titled "Shallow Learning model for diagnosing neuromuscular disorder from splicing variants", has been published by World Journal of Engineering, Vol. 14 Issue: 4, pp.329-336, 2017. (Scopus, ISI indexed).
6. Paper titled "Data Driven Approach for Genetic Disorder Prediction by Aggregating Mutational Features", has been accepted for publication in the Asian Journal of Information Technology. (Will be published in Issue 16, Vol.3)