Probing the Intrinsic Resistome of *Mycobacterium tuberculosis*

Mapping of Chemical Space: Modeling and Simulation of Metabolism and Predictive Models of Influx as Filters for Drug-like Molecules

A Major Project dissertation submitted
in partial fulfilment of the requirement for the degree of

Master of Technology
In
Bioinformatics

Submitted by

JAYA UNIYAL

(2K12/BIO/09)

Delhi Technological University, Delhi, India

Under the supervision of

Dr. V. K. Singh

Department of Biotechnology
Delhi Technological University
(Formerly Delhi College of Engineering)
Shahbad Daulatpur, Main Bawana Road,

Delhi-110042, INDIA

CERTIFICATE

This is to certify that the M. Tech. dissertation entitled **"Probing the Intrinsic Resistome of *Mycobacterium tuberculosis*. Mapping of Chemical Space: Modeling and Simulation of Metabolism and Predictive Models of Influx as Filters for Drug-like Molecules"**, submitted by **JAYA UNIYAL (2K12/BIO/09)** in partial fulfilment of the requirement for the award of the degree of Master of Technology, Delhi Technological University (Formerly Delhi College of Engineering, University of Delhi), is an authentic record of the candidate's own work carried out by her under my guidance.

The information and data enclosed in this dissertation is original and has not been submitted elsewhere for honouring of any other degree.

Date:

Dr. V.K.Singh
Department of Bio-Technology
Delhi Technological University
(Formerly Delhi College of Engineering, University of Delhi)

DECLARATION

I, Jaya Uniyal hereby declare that the report entitled "**Probing the Intrinsic Resistome of *Mycobacterium tuberculosis*. Mapping of chemical space: Mapping of Chemical Space: Modeling and Simulation of Metabolism and Predictive Models of Influx as Filters for Drug-like Molecules**" submitted in partial fulfilment of the requirement for the award of the degree of Master of Technology, Delhi Technological University (Formerly Delhi College of Engineering, University of Delhi), is a record of original and independent research work done by me under the supervision and guidance of Dr. Andrew M Lynn, Associate Professor, School of Computational & Integrative Sciences(SCIS),Jawaharlal Nehru University(JNU),New Delhi and the thesis has not formed the basis of the award of any Degree/Diploma/Associateship/Fellowship or other similar title to any candidate of any university/institution.

Date: Signature of candidate

ACKNOWLEDGEMENT

He who thanks but with the lips
Thanks but in part;
The full, the true Thanksgiving
Comes from the heart.

I am grateful to the Almighty for having given me the strength and perseverance to accomplish my objectives. With him by my side the whole process was easier. My belief in him kept me moving towards the completion of this project.

Next I would like to acknowledge my beloved family who stood by me and gave me all the more patience and strength. A big thank you to my dad, who despite his busy schedules, took out time to hear my issues and complaints and constantly encouraged and supported me. The blessings of my grandparents were all the way with me. I am indebted to the love of my family and friends throughout my journey.

I express my sincere gratitude and admiration to my advisor, **Dr. V.K.Singh**, for his guidance, support, and mentorship that has helped me to successfully complete the project.

I profusely thank **Dr. Andrew M. Lynn**, Associate Professor, Jawaharlal Nehru University, New Delhi, India for his unparalleled support and inspiration. I consider myself fortunate and highly indebted to him for his precious guidance, various innovative ideas and invaluable suggestions. I am thankful to him for his exceedingly helpful nature. It would not have been possible for me to finish this work without his constant support and motivations. He showed immense patience in dealing with me and rectified all the mistakes, committed by me.

I thank my fellow labmates in **Lynn Group**: School of computing and integrative Sciences for the stimulating discussions, for the sleepless nights we were working together before deadlines, and for all the fun we have had in the last one year. The role of **Anmol Sir, Deepak Sir, Pankaj Sir, Shawez Sir,** and **Swati Mam** was critical and of utmost importance throughout. They helped me understand the basics as well as the advanced fundas and were available day in and day out.

A special mention to the people without whose constant support this journey and this work would have been incomplete. Firstly I would like to thank my constant support and strength,

Manu Kandpal, who was there at every step and progress of this work. My buddies, **Rishi Srivastava, Basharat Bhat** and **Kashif Nawaz** who were my point of advice and I could always depend on them. Most of our coffee breaks turned out to be scientific discussions that provided insights to my work.

To my friends **Yashna Paul** , **Neha Nagpal, Dhwani Dholakia and Mayank Guptai,** thank you for your thoughtful consideration, unconditional love and support that made my work easy and pleasant.

Last but not the least, I owe a sincere thanks to Bioinformatics staff of Delhi Technological University for providing a secure and healthy environment for growth in education and life.

Once again I thank all the people who helped me with their advice, support and encouragement.

Jaya Uniyal
2K12/BIO/09

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

MTB	*Mycobacterium tuberculosis*
MDR	Multi Drug Resistant
XDR	Extensively Drug Resistant
INH	Isoniazid
RMP	Rifampicin
DEM	Dead End Metabolite
RNP	Root Non Produced
RNC	Root non Consumed
FBA	Flux Balance Analysis
HTS	High Throughput Screening
QSAR	Quantitative Structure Activity Relationship
SVM	Support Vector Machine
KNN	k-Nearest Neighbour
PLS	Partial Least Square
GBM	Generalized Boosted Method
GLM	Generalized Linear Model
RF	Random Forest
CARET	Classification and Regression tool
COBRA	Constraint Based Reconstruction and Analysis
SE	Sensitivity
SP	Specificity

Probing the Intrinsic Resistome of *Mycobacterium tuberculosis*

Mapping of chemical space: Modeling and simulation of metabolism and predictive models of influx as filters for drug-like molecules.

Jaya Uniyal

Delhi Technological University, Delhi, India

1. ABSTRACT

Tuberculosis (TB) is one of the most serious infectious diseases worldwide. Approximately one-third of the people are infected with *Mycobacterium tuberculosis* (Mtb), making tuberculosis (TB) one of the most prevalent infectious diseases in the world. Anti-tuberculosis (TB) drug resistance is a major public health problem that threatens progress made in TB care and control worldwide. Intrinsic resistance to antibiotics is posing severe threats to the treatments currently available for TB rendering them ineffective and incomplete.

To deal with this scenario, efforts have to be directed towards mechanisms that are underexplored in the organism for the purpose of developing an effective and adequate treatment regime. Herein, we propose metabolic pathways of the mycobacterium as the choice of target for identifying and developing novel drug molecules that may act as potential inhibitors of the organism. In this study, we propose to explore the chemical space within Mtb to identify essential metabolites, and use this information to develop influx models for the organism.

Mtb presents a plethora of molecular targets that hold a great potential for therapeutic intervention in the post genomic era. These opportunities present the Mtb genome as a highly druggable genome. Studies have identified proteins critical for the survival for *M.tuberculosis* that are likely to have high rates of success as drug candidates. Many of the druggable proteins are enzymes that control several metabolic processes within the cells by catalyzing the reactions converting nutrients into energy and new molecules which are in a way crucial for the survival of the microbe.

Because metabolism is fundamental in sustaining microbial life, drugs that target pathogen-specific metabolites, enzymes and pathways can be very effective. In particular, the metabolic challenges faced by intracellular pathogens, such as *Mycobacterium tuberculosis*, residing in the infected host provide novel opportunities for therapeutic intervention (Fang et al, 2009).

Keywords: TB,druggable genome, metabolic pathways.

2. INTRODUCTION

Tuberculosis (TB) is one of the most serious infectious diseases worldwide. The World Health Organization predicts that between 2002 and 2020, 36 million people will have died from TB. Approximately one-third of the people are infected with *Mycobacterium tuberculosis* (Mtb), making tuberculosis (TB) one of the most prevalent infectious diseases in the world. Anti-tuberculosis (TB) drug resistance is a major public health problem that threatens progress made in TB care and control worldwide.

Mycobacterium tuberculosis, the causative agent of tuberculosis (TB), is among the world's most devastating pathogens, responsible for 2–3 million deaths annually. MTB has two peculiar features that make it particularly difficult to combat. First, its thick, waxy cell wall, rich in unusual lipids, makes it impermeable to many drugs. Second, its latency, which prolongs to as much as many years until it is reactivated. When engulfed by macrophages, it can switch its metabolism and remain in a latent or persistent state inside granulomas in the lung. Current estimates are that one-third of the world's population is infected, and that the incidence of active TB is rising, in particular as a result of synergy with the HIV/AIDS pandemic. Although effective anti-TB drugs exist, treatment regime require a mixture of two to three drugs administered for at least 6 months (Card et al., 2005).

Drug-resistant TB is the man-made result of interrupted, inadequate and inappropriate TB treatment, and its spread is undermining the efforts being made to control the epidemic. Multi-drug-resistant tuberculosis (MDR-TB) is defined as tuberculosis that is resistant to at least isoniazid (INH) and rifampicin (RMP), the two most powerful first-line treatment anti-TB drugs. *M. tuberculosis* develops drug resistance exclusively through chromosomal mutations, in particular single-nucleotide polymorphisms (McGrath et al., 2014).

Genes that encode disease related proteins that can be modulated by drug-like molecules represent the druggable genome (Keller et al, 2006). M.Tb presents a plethora of molecular targets that hold a great potential for therapeutic intervention in the post genomic era. These opportunities present the Mtb genome as a highly druggable genome. Studies have identified proteins critical for the survival for M.tuberculosis that are likely to have high rates of success as drug candidates (Raman et al., 2008).

Many of the druggable proteins are enzymes that control several metabolic processes within the cells by catalyzing the reactions converting nutrients into energy and new molecules which are in a way crucial for the survival of the microbe. The enzyme–reaction relationships can be used for the reconstruction of a network of reactions, which leads to a metabolic model of metabolism (Baart et al., 2012). Modulating these enzyme reaction relationships can provide us with druggable protein targets.

Despite its high druggability, the Mtb genome also shows intrinsic resistance to antibiotics. Clinical definition of antibiotic resistance is mainly based on the bacterial response to treatment (Olivares et al, 2013). The term "Intrinsic resistome" has been used in context to resistance that the bacterium provides against the antibiotics. The causes of such resistance may be due to loss of a target, and

reduced uptake or increased efflux of the drug. Effectors for these causes maybe the acquisition of resistance genes(Horizontal gene transfer) or mutations in genes that make bacteria more resistant to antibiotics and mechanisms that selectively restrict uptake such as impermeability of the cell wall, and efflux proteins.

Importantly, using the sequenced genome of *M. tuberculosis* (Cole et al, 1998) together with literature data on known metabolic reactions, extensive metabolic network reconstructions have been carried out for this organism (Jamshidi et al, 2007; Beste et al, 2007). Analyses of these networks based on FBA reveal that they contain sufficient information to predict growth rates and identify genes that are essential for the growth of *M. tuberculosis* in select media (Jamshidi et al, 2007; Beste et al, 2007; McGrath et al, 2014).

Herein, we present a list of drug-like molecules based on the predictive model for molecules that are being influxed by the bacterium. The molecules being predicted as potent drug molecules are based on the intrinsic requirement of the bacterium and thus expected to be more active within the cell. Thus we predict that these molecules have higher uptake/influx potential within the bacterium and thus a good choice for being tested for inhibitory effect on mycobaterial growth.

3. REVIEW OF LITERATURE

3.1 Metabolism

Metabolism is regarded as one of the most complex kind of cellular processes and many attempts have been made to reconstruct the metabolic network in many organisms. The first whole-cell metabolic model was developed for the human red blood cell (RBC) (Price et al, 2003) as the culmination of two decades of work.

A metabolic pathway is defined as a set of enzyme catalyzed biochemical reactions required by a living organism that transform an initial (source/reactant) compound into a final (target/product) compound. A metabolic pathway involves the step-by-step modification of an initial molecule to form another product. Each metabolic pathway consists of a series of such biochemical reactions that are connected by their intermediates: the products of one reaction are the substrates for subsequent reactions, and so on. Metabolic pathways are often considered to flow in one direction.

These biochemical reactions are connected to each other via their reactant and product metabolites to create complex metabolic interconnections. These networks may then be analyzed using the complex calculations of network/graph theory, stoichiometric analysis and other relevant information on protein function and their enzymatic conversions (Hatzimanikatis et al, 2004). These interconnected networks ultimately allow the cell to function and grow.

Recently, metabolic pathway analysis has gained much prominence from a system's biology perspective (Klamt et al, 2003). Each cell is a complete system in itself well defined by a system boundary. This boundary identifies a system from the environment. Each system or a cell in biological terms comprises of a complex world of cellular metabolism commonly organized into metabolic pathways (Planes et al, 2009).

The metabolic network is used to self-consistently calculate the overall biomass growth rate μ, substrate uptake rates vC, and the fluxes of all metabolic reactions (Fang et al.2009). These metabolic frameworks take into consideration different aspects of enzyme chemistry, enzyme structure and metabolite structure, and demonstrate the impact of metabolic biochemistry on the systemic properties of metabolism (Hatzimanikatis et al, 2004).

3.1.1. Metabolic networks

Owing to the complexity of cellular processes, metabolic networks have gained prominent importance recently. Metabolic networks comprise of a repertoire of chemical and enzymatic reactions that aid in supporting life. These innumerate reactions help transport and convert thousands of organic compounds into molecules that carry forward the life sustaining reactions (Schilling et al., 2000). The collection of reactions and hence pathways that a metabolic network possesses determines the architecture and topology of the network.

Metabolic networks incorporate diverse data sets including genome annotations, biochemical characterizations and cell physiology experiments (Papin et al., 2003).

Most of the contribution in today's well elucidated metabolic networks has been due to the large-scale sequencing projects and genome annotation efforts. Once functional annotation of enzyme coding genes is done based on sequence similarity and positional correlation of genes, organism-specific pathways can be constructed computationally by correlating genes in the genome with gene products (enzymes) in the reference pathways (Kanehisa and Goto, 2000). With this level of description, mathematically defined pathways enable the analysis of a complete metabolic system.

The detailed analysis of these reconstructed networks involves the mathematical description of cellular properties that are measured in terms of reaction fluxes and biomass conversion or total cellular growth. These methods take into account properties of the enzyme chemistry by accounting for mass balance and thermodynamic constraints.

3.1.2. Metabolic Reconstruction

Computationally intensive reconstruction of metabolic pathway aims at representing the complete network of metabolites by interconnecting and correlating reactions and their catalysts by activities assigned to one or more genes. Reconstructed models do sometime contain inconsistencies that appear as blocked reactions that contain dead end metabolites or gap metabolites (Ponce-de-León et al, 2013). These inconsistencies usually appear in the network due to incomplete information about the reactions/pathway or due to no known functional annotations for certain reactions.

Due to these inconsistencies, there are sometimes losses of connections in the network that account for truncated networks. These truncated or broken networks accumulate gaps that create dead end metabolites in the network.

A dead-end metabolite (DEM) is defined as a metabolite that is produced by the known metabolic reactions of an organism and has no reactions consuming it, or that is consumed by the metabolic reactions of an organism and has no known reactions producing it, and in both cases has no identified transporter (Mackie et al,2013). DEMs are thus isolated compounds within a metabolic network.

These DEMs may be classified as Root not produced (RNP) and root not consumed (RNC).

A 'dead end' exists in a metabolic network if a metabolite is either only produced or only consumed in the network. If a metabolic network contains a gap, it is missing the biochemical reactions that can produce or consume the dead end metabolites (Reed et al, 2003). These DEMs as stated above are the outcome of loss of connection within the network. Any reaction that contains a DEM is found to contain no flux in any simulation condition. Network reactions that cannot carry any flux in any simulation condition are called blocked reactions. Generally, these blocked reactions are caused by missing links in the network.

A blocked reaction may also be defined as a reaction whose Flux is always zero i.e. its min and max value is zero. These reactions contain gap metabolites or dead end metabolite.

3.2 Flux Balance Analysis

The chemical reactions that occur within a cell that are a necessary as life sustaining reactions are known as metabolic reactions. Metabolites are the intermediates and the products of metabolic pathways. Flux balance analysis is a mathematical approach for analyzing the flow of these metabolites through a metabolic network (Orth et al, 2010).

FBA, as the term implies is the mathematical analysis of the flow pattern i.e. influx and efflux patterns of the metabolites within a system i.e. a living cell simulation. FBA requires knowledge about the system and its metabolic network. Thus FBA has a prerequisite for a complete system definition including the definition of reactions, and metabolites. Another prerequisite of FBA is the reconstructed metabolic pathway. FBA calculates the flow of metabolites through this metabolic network, thereby making it possible to predict the growth rate of an organism or the rate of production of a biotechnologically important metabolite (Schilling et al, 1999).

The methodology for FBA consists of complex mathematical calculations that simulate *in silico*, the actual cellular environment. Detailed methodology for FBA has been described in many publications.

Flux Balance Analysis can be best studied by classifying it in the following four steps (Kauffman et al, 2003).

1. **Defining the system boundaries:** A well-defined system containing the complete information about the reactions taking place in the system and the metabolites is required for FBA. This system should encompass all the reaction known so far.

   For any FBA experiments, a prior knowledge of its metabolic pathways is required. This information may either be retrieved through literature survey for the organisms where it has been done or should be collected by reconstructing the metabolic pathway from the annotated genome.

   Reconstruction process involves the protein to gene approach. All known proteins in the organism are searched for their corresponding genes. These then are subsequently identified for metabolic enzymes and the reactions that they might be catalyzing.

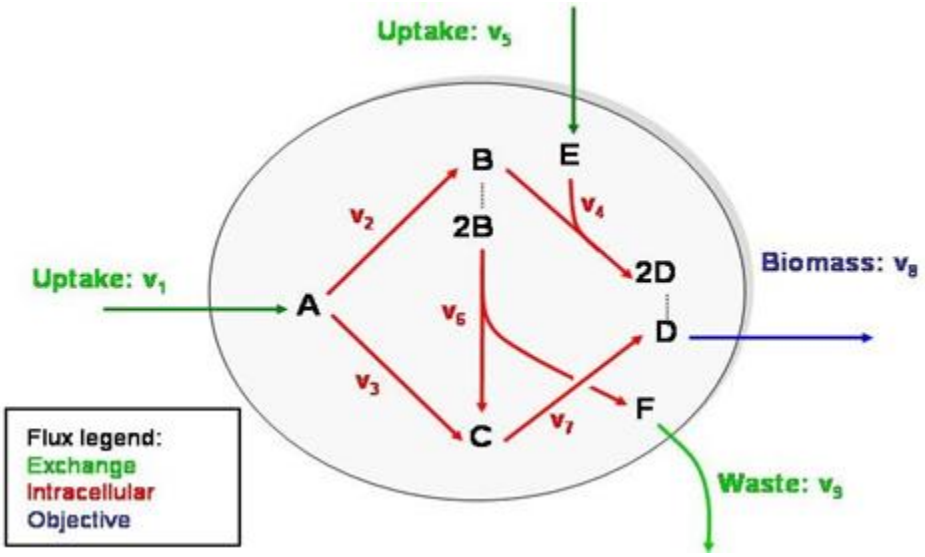


Figure 1. System based approach: system enclosed within system's boundary. FBA does not include the regulation of the pathway.

2. **To start any FBA**, the biological data should be put in a format that could be calculated upon easily. This is done with the help of matrices and vectors. The interconnection between metabolites is represented in the form of reactions that are further written as equations with reactants on the left hand side and products on the right. Eventually, reactions and the metabolites are translated to a stoichiometric matrix where a list of reactants is scored against the list of reactions. This scoring assigns a negative(-) sign for metabolites that are consumed in the reaction and a positive(+) sign for the metabolites being produced in the reaction. The numeric value gives the number of molecules consumed or produced in the process. The important feature of this representation is tabulation, a matrix of the stoichiometric coefficients of each reaction.

The main aim of FBA is to identify the metabolic fluxes in a steady state metabolic network. Since the number of reactions is far more than number of metabolites, the fluxes are said to be underdetermined. Thus, additional constraints are applied to determine the steady state fluxes within a system.

3. Defining measurable fluxes:

With no constraints, the flux distribution of a biological network may lie at any point in a solution space. When mass balance constraints imposed by the stoichiometric matrix are applied to a network, it ensures mass balance between the total amount of any compound being produced and the total amount being consumed at steady state. Every reaction can also be given upper and lower bounds, which define the maximum and minimum allowable fluxes of the reactions.

Both these constraints define an allowable solution space. The network may acquire any flux distribution within this space, but points outside this space are denied by the constraints. Throughobject$obsLevels optimization of an objective function, FBA can identify a single optimal flux distribution that lies on the edge of the allowable solution space (Orth et al, 2010).

4. Optimization of flux:

Optimization is the next step that is employed for calculating the fluxes. Metabolic networks tend to have more fluxes than there are metabolites. Thus it needs to be optimized to a level with respect to a certain objective function which here corresponds to biomass production.

3.3 Data Mining

The next aim of this work is to determine meaningful information based upon the metabolic network of *Mycobacterium tuberculosis*.

The current treatment to MTB infection involves the combination of available drugs. Also, no new drug for the disease has been found in the recent past and the currently used medications were last found in the 1960s putting tremendous pressure on the clinical and pharmacological industry. The recommended combination therapy for TB is lengthy and cumbersome since it can involve up to four drugs and requires daily medication for 6 to 12 months. Limited health care system resources often lead to treatment interruption or failure, exacerbating drug resistance problems.

Drug-resistant TB is the man-made result of interrupted, inadequate and inappropriate TB treatment, and its spread is undermining the efforts being made to control the epidemic. Multi-drug-resistant tuberculosis (MDR-TB) is defined as tuberculosis that is resistant to at least isoniazid (INH) and rifampicin (RMP), the two most powerful first-line treatment anti-TB drugs. *M. tuberculosis* develops drug resistance exclusively through chromosomal mutations, in particular single-nucleotide polymorphisms (McGrath et al., 2014).

The need for novel, more effective drugs to improve TB control is evident. Treatment of active disease needs to be shortened, simplified, and should not interfere with the administration of antiretroviral agents. It is especially desirable to identify new types of TB drugs acting on novel drug targets with no cross-resistance to existing drugs. Modern high throughput screening (HTS) systems provide an immensely powerful strategy to identify new lead compounds in a relatively short amount of time.

In recent years, chemical structures of small bio-molecules and structure derived properties have been made available and can be accessed easily. With the explosion of this data, there is a need to understand the underlying structure-activity relationships. Cheminformatics which combines use of chemical structure and computational tools helps to identify and develop novel drugs by lead identification and optimization.

QSAR focuses on finding a model that correlates the structure of a molecule with the activity. The

cheminformatics methods used in building QSAR models is divided into three groups (a) extracting descriptors from molecular structure, (b) choosing the descriptor for analyzed activity, and (c) using the values of the descriptors as independent variable to define a mapping that correlates them with the activity.

Molecular descriptors are calculated using software DRAGON6. Dragon provides almost 5,000 molecular descriptors that are divided into 29 logical blocks, each in turn divided into a number of sub-blocks to allow easy retrieval of the molecular descriptors of interest. Dagon permits to merge calculated molecular descriptors and user-defined properties for a set of molecules, providing a complete output file that can be easily loaded by any correlation analysis application. These descriptors can be used to evaluate molecular structure-activity relationships (QSAR/QSPR), similarity analysis and screening of molecular database.

The caret package in R contains functions to streamline the model training process for complex regression and classification problems. The package is used for data splitting and pre-processing, as well as unsupervised feature selection routines and methods to tune models using resampling that helps diagnose over-fitting. The methods available in train are used to build the models for classification and the arguments can be used for tuning the parameters for specific models.

Data Mining and Machine Learning:

Machine Learning is the branch of Artificial Intelligence, which is devoted to enabling the computers to learn. It is an amalgamation of the branches of computer science, engineering, and statistics and is often used in other disciplines. It has its application in many fields ranging from politics to geo-sciences and can be used to solve many problems. Any field requiring interpretation of data can benefit from machine learning techniques.

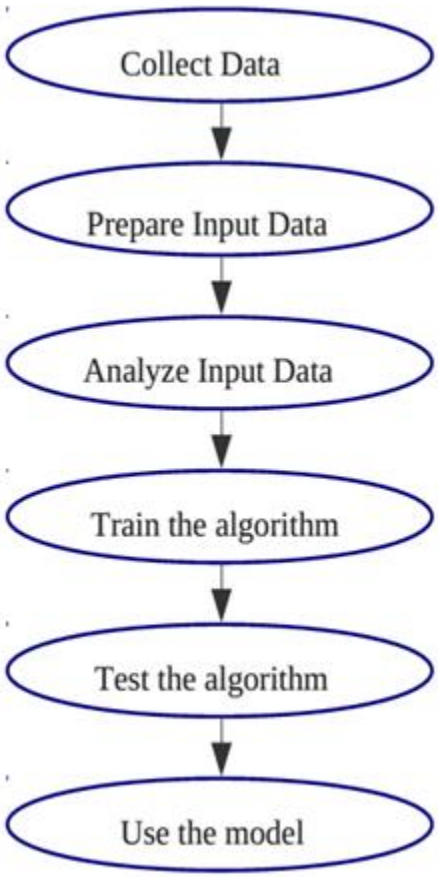
Data mining has been defined as “the non-trivial extraction of implicit, previously unknown, and potentially useful information from data”. It is the process of discovering patterns in the data, the process being either automatic or semiautomatic. Data mining is the center- piece of advanced methodologies to help the industry deal with this information overload, since the methods are both time and cost efficient.

Data Mining Algorithms

The general first step in any machine learning algorithm is to have two separate datasets: “training set” and “test set”. The algorithm to be used is first decided. The algorithm is then trained by allowing it to learn. The machine learning occurs when the program is fed with quality data called “training set” that consists of various examples. Each example in a training set has a number of features and target variables. The target variable is the one that we are aiming to predict with our machine learning

algorithms. These are known in case of training set and not known in case of test set. These have a nominal value in case of classification while in regression, the values can also be continuous.

The machine learns by establishing some relationship between the features and the target variable. Features or attributes are the individual measurements that make up a training/test example and are represented as columns. Next, the test set is fed to the program without the target variable to let the program decide the class. The predicted values are then compared with the observed values to determine the accuracy of the algorithm. The two types of learning are Supervised and Unsupervised learning.



Figue2. Data Mining Algorithm.

Application to Biological Data

In biology, the data generated is often either too complex and/or too voluminous to be processed and analyzed by traditional methods. Data Mining can improve decision making by discovering trends in large amount of complex data. Data-mining techniques can be expanded in building machine learning

classifiers to predict the function of well characterized proteins based on the features of their amino acid sequence without using homology information. Data-mining techniques and classifiers can provide intelligent predictive models based on classifiers trained and tested on known and complete data sets.

Apart from the derivation of structure–function relationships, the contributions of machine learning has its applicability in the protein folding process, resulting in soft wares that are better adapted carry on the task Text mining is another application where data mining methodologies are also increasingly applied for information retrieval from the scientific literature. Automation is more a necessity for developing new ways of searching and summarizing the literature.

Recently, the field of research in drug discovery is focused on applying data mining approaches to design and discover effective molecules to affect various disease targets with the aim at success in advanced stages to have better chance in clinical trial stages. In this regard, data mining novel approach in combining cheminformatics, intensive data handling were used together with correlation of biologic data to search for the desired biologic activity in the domain of natural products that were not explored before . It would be obvious that the insilico drug discovery based on data mining approaches play great roles in treatment of diseases and are gaining increased applications in drug discovery and development by saving several folds of cost and time parallel to increasing success rate.

Cheminformatics and QSAR

QSAR (Quantitative Structure Activity Relationship) is based on hypothesis that experimental properties are a consequence of the 3D structure of a molecule, so it focuses on finding a model that correlates the structure of a molecule with the activity. The chemical structures of small bio-molecules and their derived properties which is publicly available in the form of open database are easily accessible. With inundation of the data, screening the useful structure-activity relationships, is the need of the hour and cheminformatics provides us a suitable platform for screening.

Cheminformatics methodology

Cheminformatics combines the use of chemical structure and computational tools to help in identifying and developing novel drugs by lead identification and optimization. The cheminformatics methods used in building QSAR models is divided into three groups :

1. Extracting descriptors from molecular structure

The structure of small molecules are defined in terms of number of atoms and covalent bonds between the atoms. We cannot use the structure directly for mapping it to the activity as the structure does not explicitly contain the information and it needs to be extracted. The properties ranging from the physiochemical, quantum-chemical to geometrical and topological, are the ones that directly correlate with

the activity and are needed to be described in form of the molecular descriptors. To predict the activity an input numerical vector of features of uniform length is required. We need to convert the structure in form of well-defined set of numerical values, for effective input for various algorithms.

2. Choosing the descriptors for analyzed activity

In order to avoid inaccuracies,. The most informative descriptors that show high correlation, are used to generate the model. We tend to avoid the ones that are inter correlated, which impacts the QSAR analysis negatively.

3. Define a mapping that correlates them with the activity

A particular function needs to be defined that gives the value of the activity that quantifies successfully with the values of the descriptors. The most accurate mapping function from some wide family of functions, is usually fitted based on the information available in the training set. Mapping function used can range from linear to non-linear ones, and many other methods for carrying out the training to obtain the optimal function can be employed.

Machine learning (ML) techniques and specifically supervised learning methods have been recently adopted for virtual screening to assign nominal/numerical classifications in terms of activities. A major focus of ML methods is to automatically learn to recognize complex patterns which classify sets from input data and to make intelligent decisions based on independent datasets. Bioactivity data available from the many high throughput screens provide useful means to train machine learning classifiers as it contains binary i.e. active/inactive as well as numerical (for example IC50) values for classification of compounds. Previous studies have pointed to the usability of bioassay data available in public domain to build efficient classifiers. The recent availability of a large amount of data on biological activities of molecules, especially derived from high throughput screens now enables us to create predictive computational models. Though ML methods have proved to be a valuable tool in rapid virtual screening of large molecular libraries, they have been seldomly applied in TB drug discovery programs. Our present study aims at developing a comprehensive and systematic approach with the aid of ML techniques to build binary classification models from high throughput whole-cell screens of anti-tubercular agents. These predictive models when applied to virtual screening of large compound libraries can identify new active scaffolds that can accelerate the Mtb drug discovery process.

3.3.1 Supervised Learning

In Supervised learning, the machine learns from the supplied data having a specific target variable. The machine's task is reduced as now the machine has to find a pattern in the input data to get the values of the target variable. Here, we are telling the algorithm what to predict while giving it access to a training set containing the pre-classified data. Classification and regression are examples of supervised learning.

Classification: Classification is the process of categorizing the data Classification process requires building a classifier which is a mathematical function that is able to assign class (active/inactive) labels to instances which can be defined by a set of attributes (descriptors). In classification, we predict the class of an instance of the data. Most classification models can be built using either the numeric or nominal values. Classification algorithms can be of various types for example Bayesian bases, Genetic algorithm based or association based. Regression: Regression is the prediction of a numeric value. Regression analysis helps in understanding the relationship between variables. It analyzes the effect of change in the value of the dependent variable when one of the independent variable is changed, while the other independent variables are held fixed. Regression methods can be of various types including linear regression, nonlinear regression, multiple linear regression and step wise linear regression.

3.3.1.1 Linear regression

It assumes that the regression function is linear in the inputs X1 , . . . ,Xx. Linear models are simple and often provide an adequate and interpretative description of how the inputs affect the output. These algorithms are useful in prediction when there are small numbers of training cases, low signal-to-noise ratio or sparse data.

Our goal while using regression is to predict a numeric target value. Regression is represented by the following simple equation:

$Y= \alpha X1 +\beta X2 +.....\delta X_x$

Here the values X1 , βX2 ,.....Xx are called the independent variables and Y is the dependent variable. α, β are known as regression weights and the process of finding these regression weights is called regression. Multiple Linear Regression (MLR) models the activity to be predicted as a linear function of all descriptors. Based on the examples from the training set, the coefficients of the independent variables are estimated. These regression weights are chosen to minimize the squares of the errors between the predicted and the actual activity. The main restriction of MLR analysis is the case of large descriptors-to-compounds ratio or multi-collinear descriptors in general. This makes the problem ill-conditioned and makes the results unstable.

Generalized linear models (GLM's) are a framework for modeling a response/dependent variable Y that can be either bounded or discrete. GLM are generally useful while modeling positive quantities that has variation over a large scale and can be described using a skewed distribution. Other uses includes modeling a categorical data such as a binomial or a multinomial distribution where there are fixed number of choices that cannot be arranged in a meaningful way. Also used for modeling ordinal data such as ratings where its possible to have different outcomes and the quantity itself doesn't have any absolute meaning.

Generalized linear models allow for an arbitrary link function, g , that relates the mean of the response variable to the predictors, i.e. $E(y) =g(\beta'x)$. The link function is often related to the distribution of the

response, and in particular it typically has the effect of transforming between the range of the linear predictor and the range of the response variable.

Algorithm for Linear Regression

1. Collect the data. The weights are calculated from the training data.
2. Using the following equation, each class is represented as a linear combination of the attributes, with predetermined weights.

Y = a0 + a1X1 + a2X2 +..... + an Xn .

where X1, X2, ... , Xn are the attributes values and a0, a1, a2 , ... , an are the weights.

3. The predicted value(not the actual value) for the first instance can be written as

$$a_0 X_0 + a_1 X_1 + a_2 X_2 + \ldots\ldots + a_n X_n = \sum_0^n a_i X_i$$

4. The method of linear regression is to choose the coefficients a – there are x+1 of them- to minimize the sum of squares of these differences over all the training instances. Suppose there are x training instances, denote the ith one with a superscript (i). Then the sum of differences is

$$\sum (X - \sum_0^n a_i X_i)^2$$

where the expression inside the parentheses is the difference between the ith instances; actual class and its predicted class. This is the sum of square we are aiming to minimize by choosing the coefficients properly.

3.3.1.2 Support Vector Machines

It creates a decision hyperplane, a margin, that maximizes the distance from the hyperplane to the nearest examples from each of the classes. The line used to separate the data set is called a separating hyperplane. In 2D plots, it's just a line but when we have a data set with three dimensions, we need a plane to separate the data, known as a hyperplane which is our decision boundary. Everything on one side belongs to one class, and everything on the other side belongs to a different class. The classifier is made in such a way that the farther a data point is from the decision boundary, the more accurate is the prediction. The points which are closest to the separating hyperplane are known as support vectors. We have to maximize the distance from the separating line to the support vectors. This allows for formulating the classifier training as a constrained optimization problem. The objective function is unimodal, and can be optimized effectively to global optimum.

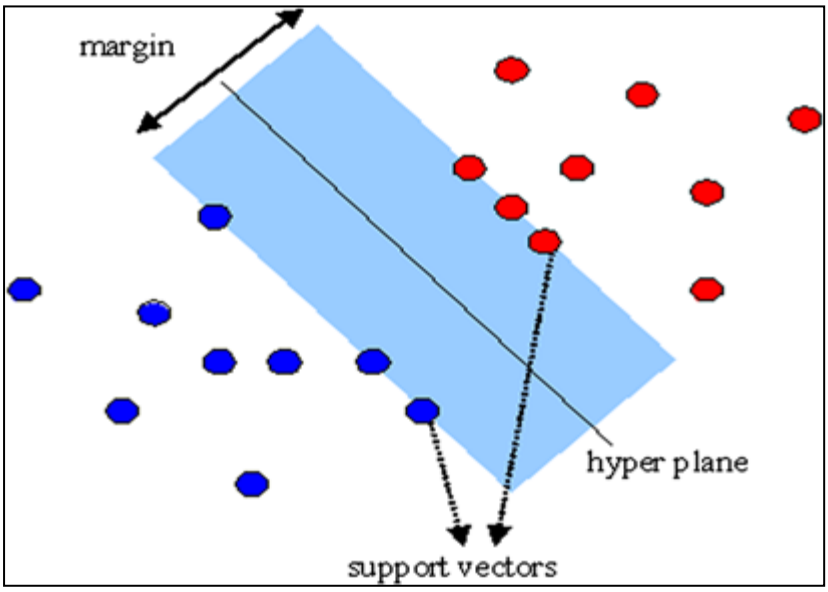


Figure3. Support vector principle.

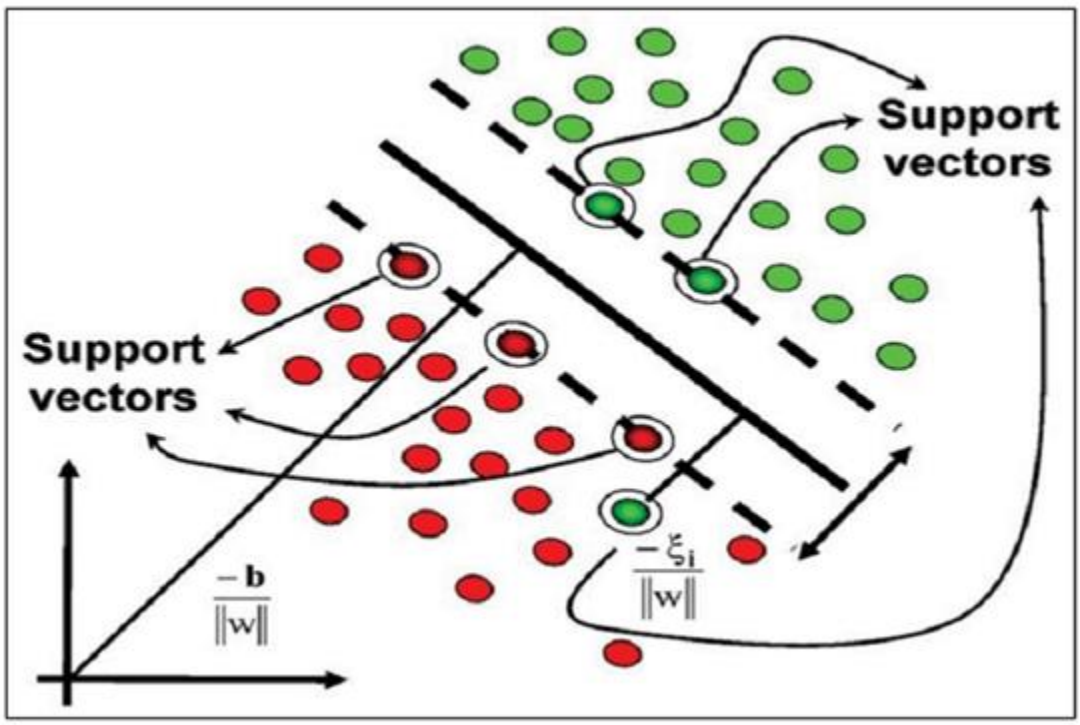


Figure 4. *SVM uses Structural Risk Minimization to compare various separation models and to eventually choose the model with the largest margin of separation.*

Compounds from different classes can be separated by linear hyperplane; such hyperplane is defined solely by its nearest compounds from the training set, the support vectors. When no linear variables are present, slack variables are introduced which are associated with the misclassified compounds and, in conjunction with the margin, are subject to optimization. The erroneous classification cannot be avoided, but it is penalized.

The SVM can be easily transformed into a non-linear classifier by employing the kernel function. The kernel function introduces an implicit mapping from the original descriptor space to a high or infinite-dimensional space. The linear hyperplane in the kernel space may be highly non-linear in the original space. The SVM method has been shown to exhibit low over training and thus allows for good generalization to the previously unseen compounds. It is also relatively robust when only a small number of examples of each class is available.

The SVM methods have been extended into Support Vector Regression (SVR) to handle the regression problems which can be used for accurate nonlinear mapping between the activity and the descriptors can be found. However, contrary to typical regression methods, the predicted values are penalized only if their absolute error exceeds certain user-specified threshold, and thus the regression model is not optimal in terms of the least-square error.

In SVM classification, weights are a biasing mechanism for specifying the relative importance of target values (classes). SVM models are automatically initialized to achieve the best average prediction across all classes. However, if the training data does not represent a realistic distribution, you can bias the model to compensate for class values that are under-represented. If you increase the weight for a class, the percent of correct predictions for that class should increase. Priors are associated with probabilistic models to correct for biased sampling procedures. SVM uses priors as a weight vector that biases optimization and favors one class over another.

Algorithm for Support vector machines

1. Classification is achieved by realizing a linear or non-linear separation surface in the input space. In Support Vector classification, the separating function can be expressed as a linear combination of kernels associated with the Support Vectors as

$$f(x)=\sum_{(x_j \in S)} a_j y_j K(x_j, x)+b$$

Where xi denotes the training patterns, yi □ {+1, −1} denotes the corresponding class labels and S denotes the set of Support Vectors.

2. The dual formulation yields

$$min_{(0 \leqslant a_i \leqslant C)} W = \frac{1}{2} \sum_{i,j} a_i Q_{ij} - \sum_i a_i + b \sum_i y_i a_i$$

where ai are the corresponding coefficients, b is the offset, Qij = yi yj K (xi, xj) is a symmetric positive definite kernel matrix and C is the parameter used to penalize error points in the inseparable case.

3. The Karush-Kuhn-Tucker (KKT) conditions for the dual can be expressed as

$$g_i = \frac{(\partial W)}{(\partial a_i)} = \sum_i Q_{ij} a_j + y_i b - 1 = y_i f(x_i) - 1$$

And

$$\frac{(\partial W)}{(\partial b)} = \sum_j y_j a_j = 0$$

This partitions the training set into S the support vector set (0 < ai < C , gi <0 , E the error se ai = C , gi < 0 t and R the well-classified set ai =0 , gi> 0.

4. If the points in error are penalized quadratically with the penalty factor C' then it has been shown that the problem reduces to that of a separable case with C =∞ . The kernel function is modified as

$$K'(x_i, x_j) = K(x_i, x_j) + \frac{1}{(C')} delta_{ij}$$

Where

$delta_{ij}$ =1 if i= j deltaij =0 otherwise

Here the SVM problem reduces to a linear separation case. The training of the SV algorithm involves solving a quadratic optimization problem which requires the use of optimization routines from numerical libraries. This step is computationally intensive, can be subject to stability problems and is non-trivial to implement.

3.3.1.3 K Nearest Neighbour

It is a supervised learning algorithm where the result of new instance query is classified based on majority of k – nearest neighbor category. The purpose of this algorithm is to classify a new object based attributes and training samples. The classifiers do not use any model to fit and only based on memory. Given a query point we find k no. of objects or closest to the query point.

The classification uses majority vote among the classification of k objects. Any ties can be broken at random. It works based on minimum distance from the query instance to the training samples to determine the k nearest neighbours. After we gather k nearest neighbours, we take simple majority of these k nearest neighbours to be the prediction of the query instance.
It requires no training and with increase in training data it converges to the optimal prediction error. For a given compound in the descriptor space, the method analyzes its k nearest neighboring compounds from the training set and predicts the activity class that is most highly represented among these neighbors. It is sensitive to the choice of metric, number of neighbors and to the number of training compounds available.

Algorithm for K-nearest neighbour
Let the input vector to be classified be called in X. Other inputs used are our full matrix of training examples called dataSet, a vector of labels called labels, and, finally, k, the number of nearest neighbors to use in the voting. The labels vector should have as many elements in it as there are rows in the dataSet matrix.

1. For every point in our dataset, calculate the distance between inX and the current point.
2. The distance calculation is given by simple Euclidean distance, where the distance between two vectors, xA and xB, with two elements, is given by

$$d=\sqrt{(xA_0-xB_0)^2-(xA_1-xB_1)^2}$$

3. Sort the distances in increasing order.

4. Take k items with lowest distances to in X, these are used to vote on the class of in X. The input k should be a positive integer.
5. Find the majority class among these items
6. Return the label of the majority class (the most frequently occurring label) as our prediction for the class of in X.

3.3.1.4 Decision Trees
It is logic-based, expert systems and each classification tree can be translated into a set of predictive rules in Boolean logic. The model consists of a tree-like structure consisting of nodes and links. Nodes are linked hierarchically, with several child nodes branching from a common parent node and a node with no children nodes is called a leaf. In each node, a test using a single descriptor is made and based on the result of the test, the algorithm is directed to one of the child nodes branching from the parent.

The decision tree is one of the most commonly used classification techniques. It has decision blocks (rectangles) and terminating blocks (ovals) where some conclusion has been reached. The right and left arrows coming out of the decision blocks are known as branches, and they can lead to other decision blocks or to a terminating block.

For building a decision tree, first a decision on the data set is made to decide the feature to be used for splitting the data. Every feature and measure is tried and the split that gives the best results is then used to split the data set into subsets. The subsets will then traverse down the branches of the first decision node. If the data on the branches belongs to the same class, then it doesn't need further splitting othewise the splitting process is repeated until all the data is classified.

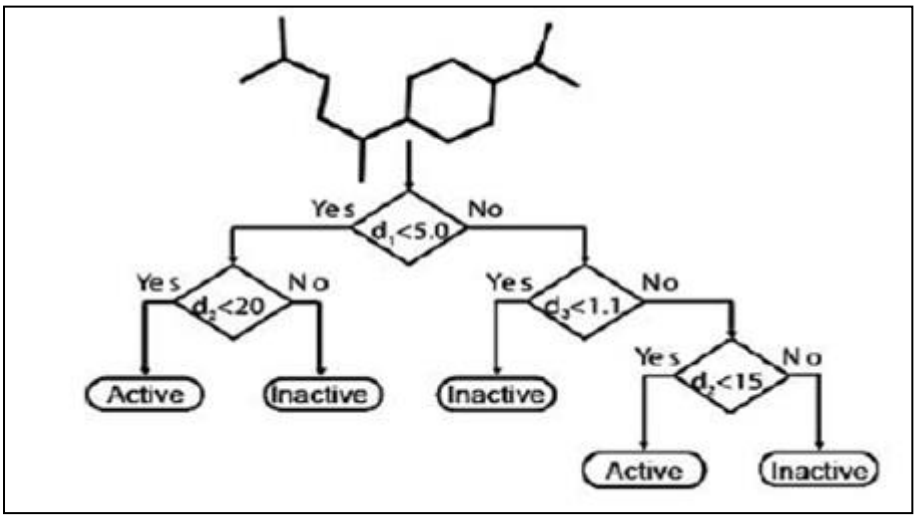


Figure 5. *General Architecture of a Decision Tree showing decision blocks in rectangle and terminating blocks in oval.*

Further in the child node, another test is performed and the traversal of the tree towards the leafs is carried out. The final decision is based on the activity class associated with the leaf. Thus, the whole decision process is based on the series of simple tests, with results guiding the path from the root of the tree to a leaf.

The training of the model consists of incremental addition of nodes. The process starts with choosing the test for the root node. A test which optimally separates the compounds into the appropriate activity classes is chosen. The main advantage of this method is that it is fast and can handle large input variables without over-fitting.

On building a decision tree many of the branches reflect anomalies in training data because of the outliers. Tree pruning methods can be used to correct the problem of over fitting of the data. The least reliable branches are pruned and it results in faster classification and improved accuracy.

Algorithm for Decision Trees:

1. If all the values of target-attributes = (True)
2. Return a leaf node and label it TRUE
3. If all the values of a target-attribute = FALSE
4. Return a leaf node and label it False
5. If the attribute list id empty then
6. Return leaf node with most common class (True/False)
7. Select the best attribute with highest information gain which acts as the best feature to split the data.
8. Split the dataset.
9. Create a branch node for each split.
10. For each value in the attribute list, let S be the number if samples, if S is not empty, attach the node returned by decision tree.

3.3.1.5 Random Forest

Random forest is a collection of un-pruned decision trees. These are often used when we have large number of datasets and a very large number of input variables. A random forest is typically made up of tens and hundreds of decision trees. The general observation is that the random forest model builder is very competitive with nonlinear classifiers such as artificial neural networks and support vector machines. However performance is often dataset dependent and so it remains useful to try a suite of approaches.

Each decision tree is built from a random subset of the training dataset, using what is called replacement in performing this sampling. That is some entities will be included more than once in the sample, and others won't appear at all. Generally about two-third of the entities will be included in the subset of the training dataset and one-third will be left out. In building each decision tree model based on a different random subset of the training dataset random subset of the available variables is used to choose how best to partition the dataset at each node. Each decision tree is built to its maximum size with no pruning performed. The resulting decision tree models of the forest represent the final ensemble model where each decision tree votes for the result and the majority wins.

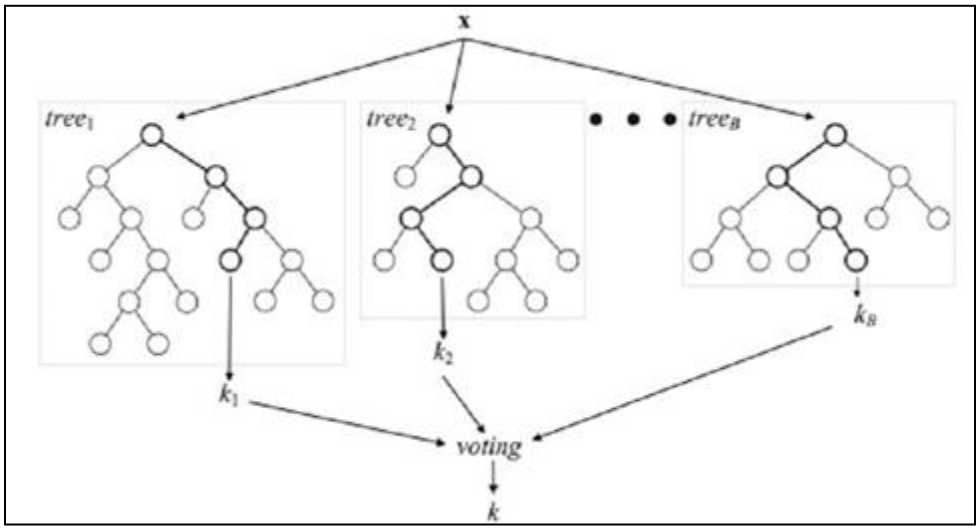


Figure 6. *A general architecture of Random Forest.*

Algorithm for Random Forest:

1. Random Forest is an ensemble of B trees [T1 (X) , ... , TB(X)] , where X = X1,..... , Xp is a p-dimensional vector of molecular descriptors or properties associated with a molecule. The ensemble produces B outputs Y1=[T1(X) , ... ,ŶB, TB(X)] , wherê Yb =1 B is the prediction for a molecule by the bth time.

2. Outputs of all trees are aggregated to produce one final prediction,Ŷ . For classification problems,Ŷ is the class predicted by the majority of trees. In regression it is the average of the individual tree predictions.

3. Given data on a set of n molecules for training,
D = [(X1, Y1,…..,(Xn,Yn))] , where Xi, i = 1,……,n. is a vector of descriptors and Yi is either the corresponding class label (e.g., active/inactive) or activity of interest. From the training data of n molecules, draw a bootstrap sample (i.e., randomly sample, with replacement, n molecules).

4. For each bootstrap sample, grow a tree with the following modification: at each node, choose the best split among a randomly selected subset of m try (rather than all) descriptors. Here m (try) is essentially the only tuning parameter in the algorithm. The tree is grown to the maximum size (i.e., until no further splits are possible) and not pruned back.

5. Repeat the above steps until (a sufficiently large number) B such trees are grown.

4. MATERIALS AND METHODS

4.1 DATA

The data required for FBA analysis should be a reconstructed metabolic network. The same had previously been described by Neema Jamshidi and Bernhard Ø Palsson in their work. Therefore, the same metabolic network was retrieved in a systems biology markup language format and used for FBA analysis.

4.2 FBA

FBA is a mathematical approach for studying the growth patterns and metabolism within an organism. To perform FBA, a MATLAB package for implementing COBRA methods. COBRA stands for COnstraint Based Reconstruction and Analysis. The manner in which microorganisms utilize their metabolic processes can be predicted using constraint-based analysis of genome-scale metabolic networks (Becker, S. et al,2007).

4.2.1 FBA Methodology

1. **Setting up MATLAB environment:**

MATLAB is a high-level language and interactive environment for numerical computation, visualization, and programming. Using MATLAB, you can analyze data, develop algorithms, and create models and applications.

The matlab installation, activation and licensing can be obtained from (http://www.mathworks.in/products/matlab/). The root privileges of the system should be priorly obtained before starting the installation process. Once the installer is started, it is a self-guided process made for easy installation.

2. Tool Box Installation

There are two options for installing the COBRA Toolbox à la carte' or bundled. The à la carte version only contains the COBRA Toolbox. The bundled version includes the COBRA Toolbox, libSBML, the SBMLToolbox, GLPK, and glpkmex. The bundled version has been tested on Mac OS X 10.6 Snow Leopard (64-bit) Ubuntu GNU/Linux Lucid (64-bit), Windows XP (32-bit), and Windows 7 (64-bit). Separate installation instructions are provided in Equipment Setup. The bundled version was used for the installation.

After successful installation, initialization of the toolbox is done by running the function initCobraToolbox(). After this, the desired paths are saved for further work. The installation is considered successful when the test suite is run. The sample data is run to check the correct functioning of the tool.

3.**Running the FBA**

Once the toolbox is ready, the COBRA-compliant SBML models are read into MATLAB. The models are loaded and saved.

>> model = readCbModel(*['iNJ.xml']***);**

>> writeCbModel(*model, 'sbml'***);**

Further modifications in the model can be done using various functions, if desired. We used the reconstructed model from Palsson lab.

Simulating the optimal growth using flux-balance analysis (FBA)

FBA is a method that calculates the flow of metabolites through a metabolic network. Growth is simulated by optimizing the model for flux through the model's biomass function.

In addition to specifying an objective, it is also necessary to define the *in silico* growth medium; this is accomplished by modifying the bounds of exchange reactions. Exchange reactions for metabolites comprising the *in silico* growth medium should have a lower bound less than 0; all other exchange reactions should have a lower bound of 0. All exchange reactions should have an upper bound greater than 0 to prevent metabolite build up. The solution returned will have units based on the units used in the model (typically mmol·gDW-1· h-1). FBA can be performed either in (A) standard (B) geometric mode:

Standard FBA is performed with:

>> a=optimizeCbModel(model)

optimizeCbModel will return a solution structure containing: the objective value 'f', the primal solution 'x', the dual solution 'y', the reduced cost 'w', a universal status flag 'stat', a solver specific status flag 'origStat', and the time to compute the solution 'time'. The primal solution, 'x' represents the flux carried by each reaction within the model. The dual solution, 'y' represents the shadow prices for each metabolite and indicates how much the addition of the corresponding metabolite will increase or decrease the objective value28, 60. The reduced cost, 'w', indicates how much each reaction affects the objective. A solver status of 1 indicates that an optimal solution was found.

Reaction deletion studies

To detect which reactions were of critical importance, deletion studies were stimulated and observed for variances from the normal standard FBA results.

Deletion studies can be easily simulated with *in silico* models. Reaction deletion methods within the Toolbox are dependent on the proper setup of the gene-reaction matrix as well as the rules defining the Boolean relationship between genes and reactions. Reactions that when deleted, affect the overall flux balance, have their upper and lower flux bounds set to zero and are therefore not functional.

The possible results from deletion studies are: 1) unchanged maximal growth, 2) reduced maximal

growth, or 3) no growth (lethal). Deletion studies can be used to predict reaction essentiality.
>> singleRxnDeletion(model)

4.3. DATA MINING

CARET Package

The CARET package (Classification and Regression Training) is a set of functions that attempt to streamline the process for creating predictive models (Kuhn M, 2008). The package contains tools for data splitting, preprocessing, model tuning using resampling and variable importance estimation using the rich set of models available in R. The package is available at the comprehensive R Archive Network at http://CRAN.Rproject.org/package=caret.

Predictive modeling methods in R :

We used the CARET function in R as data mining toolkit for analysis of the data and classification experiments. CARET incorporates comprehensive collection of machine learning algorithms for data mining tasks. It also incorporates tools for data pre-processing, classification, regression, clustering, association and visualization. The toolkit is also well-suited for developing new machine learning schemes.

Classification Algorithms

Classification refers to an algorithmic procedure that attempts to assign each input value, a given set of classes. The classification process requires building a classifier (model) which is a mathematical function that assigns class (ex. active/inactive) labels to instances defined by a set of attributes (ex. descriptors). A predictive model is one whose primary function is prediction.

Parametric regression models: Ordinary/generalized/robust regression models; neural networks; partial least squares; projection pursuit regression; multivariate adaptive regression splines; principal component regression.
Sparse/penalized models: Ridge regression; the lasso; the elastic net; generalized linear models; partial least squares; nearest shrunken centroids; logistic regression.
Kernel methods: Support vector machines, relevance vector machines; least squares support vector machine; Gaussian processes.
Ensembles: Random Forest; boosting (trees, linear models, generalized additive models, generalized linear models); bagging(trees, multivariate adaptive regression splines).
Prototype methods: k nearest neighbors; learned vector quantization.
Others: Naive Bayes; Bayesian multinomial probit models.

Functions in CARET

R provides us with a variety of functions that can be used for building predictive models. The caret package has in built functions that are used for data splitting, preprocessing, feature selection, model tuning using resampling and variable importance estimation. The package provides wrapper functions for a large number of model building methods, with the facility for automating tuning.

4.3.1 Preprocessing the Data

The strength of correlation between descriptors variables can influence the relative performance of prediction methods. Preprocessing the data sets includes discarding the near-zero values and identifying the outliers. It also requires identifying the correlated predictors so as to reduce the size of the data set by excluding the predictor variables that are linearly dependent.

```
## Code for removing nzv (near-zero values)##

 nzv <- nearZeroVar(trainX)
 if(length(nzv) > 0)
{
nzvVars <- names(trainX)[nzv]
trainX <- trainX[, -nzv]
nzvText <- paste("There were ", length(nzv),  predictors that were removed due to
severely unbalanced distributions that could negatively affect the model fit",
ifelse(length(nzv) > 10, ".", paste(": ", listString(nzvVars), ".", sep = "")), sep = "")
if(pctTrain < 1) testX <- testX[, -nzv]}
else
nzvText <- ""}  else nzvText <- ""
```

```
## Code for Correlation ##

corrThresh <- .75
highCorr <- findCorrelation(cor(trainX, use = "pairwise.complete.obs"),  corrThresh)
if(length(highCorr) > 0)
{corrVars <- names(trainX)[highCorr]
trainX <- trainX[, -highCorr]}

ifelse (length(highCorr) > 10, ".",  paste(": ",  listString(highCorr), ".", sep = "")),
Removing these predictors forced all pair-wise correlations to be less than" ,
corrThresh, ".", sep = "")
```

Principal component analysis (PCA) is a multivariate technique, analyzes a data set with several inter-correlated quantitative dependent variables. This data set is used to extract information, represent the variables as a set of new orthogonal variables known as the principal components. It displays similarity patterns as points in maps. The quality of the PCA model can be evaluated using cross-validation techniques such as the bootstrap and the jackknife.

```
##  Code for PCA plot ##

#The name of the predictors from the data are input into a variable that are used in
the PCA plot
predictors <- rawData[, predictorNames, drop = FALSE]


# PCA will fail with predictors having less than 2 unique values
isZeroVar <- apply(predictors, 2, function(x) length(unique(x)) < 2)
if(any(isZeroVar))
```

```
predictors <- predictors[, !isZeroVar, drop = FALSE]


#The function prcomp in the PCA plot that is able to handle the missing values
pcaForm <- as.formula(paste("~", paste(names(predictors), collapse = "+")))
pca <- prcomp(pcaForm,  data = predictors, center = TRUE, scale. = TRUE,
na.action = na.omit)
```

4.3.2 Model Building and Resampling

The caret package has a variety of functions that are used to streamline the model building and evaluation process. The train function can be used to evaluate, using resampling, the effect of model tuning parameters on performance, choose the optimal model across these parameters and use a training set to estimate model performance. These methods are summarized in Table1.

S.No.	Classification Model	Examples
1.	Parametric regression models:	Ordinary/generalized/robust regression models; neural networks; partial least squares; projection pursuit regression; multivariate adaptive regression splines; principal component regression.
2.	Sparse/penalized models:	Ridge regression; the lasso; the elastic net; generalized linear models; partial least squares; nearest shrunken centroids; logistic regression.
3.	Kernel methods	Support vector machines; relevance vector machines; least squares support vector machine; Gaussian processes.
4.	Ensembles	Random forest; boosting (trees, linear models, generalized additive models, generalized linear models; bagging (trees, multivariate adaptive regression splines).
5.	Prototype methods	k nearest neighbours; learned vector quantization
6.	Discriminant analysis	Linear; quadratic; penalized; stabilized; sparse; mixture; regularized.
7.	Probability Based	Naive Bayes; Bayesian multinomial probit models
8.	Tree Based	CART; C4.5; conditional inference trees; node harvest.

Table1. The table list a summary of classification models with examples available in R for Predictive modeling.

The Resampling is done by incorporating boot632 cross validation rule. Bootstrapping is used to get bias-corrected (over fitting corrected) The bootstrap has other important advantages besides providing more accurate point estimates for prediction error. The bootstrap replications also provide a direct assessment of variability for estimated parameters in the prediction rule. The idea behind bootstrapping is to sample the data with replacement. A dataset of n instances is sampled n times, with replacements which gives another dataset of n instances. The chance of an instance being picked up each time is 1/n, so the probability of it not being picked up is 1- 1/n. Multiplying these probabilities to obtain a sufficient number of picking opportunities, n, and the result as follows

$$\left(1-\frac{1}{n}\right)^{n}=e^{-1}=0.368$$

where e is the base of natural logarithms, 2.7183. This gives the chance of a particular instance not being picked at all. Thus if we have a large dataset, the test set will contain about 36.8% of such instances and training set will contain about 63.2% instances. That is why it is known as the 632 bootstrap rule.

4.2.3 Model Evaluation

Parameters used for estimating performance of a model: Sensitivity & specificity are statistical measures of the performance of a binary classification test. The extractPredict() and confusionMatrix() are the functions used to calculate specificity and sensitivity. The overall effectiveness of a general classifier can be concluded by accuracy of the model.

I. Confusion matrix:

It is a matrix that gives a clear idea of the true positives, true negatives, false positives, and false negatives as the observed and predicted values in a given sample.

		Predicted Negative	Positive
Actual	Negative	TN	FP
	Positive	FN	TP

Where

TN-> number of correct predictions that an instance is negative.

FP-> number of incorrect predictions that an instance is positive.

FN-> number of incorrect predictions that an instance is negative.

TP-> number of correct prediction that an instance is positive.

```
# Function for Confusion Matrix

cm <- confusionMatrix(classPred, testY)
```

II. Sensitivity, SE or Recall:

It is the probability of classifying/predicting an active drug target as against actual active drugs. Classifiers with a large recall don't have many positive examples classified incorrectly.

$$Senstivity = SE = \frac{TP}{TP+FN} = \frac{TP}{P}$$

```
# Function for Sensitivity

sens <- sensitivity(data[, "pred"])
```

III. Specificity, SP:

It is the probability of classifying/predicting an inactive drug target against those as actual inactive drugs.

```
# Function for specificity

spec <-  specificity(data[, "pred"])
```

IV. ROC (Receiver operating curve):

It is created by plotting the fraction of true positives out of the positives (sensitivity) vs. the fraction of false positives out of the negatives(1-specificity)indices along the Y and X axes respectively. It is also known as Relative Operating Characteristic curve because it is a comparison of two operating characteristics i.e. true positive rate and false positive rate as the criterion changes. ROC analysis provides tools to select possibly optimal models and to discard suboptimal ones independently from the cost context or the class distribution.

```
#  Usage of function for plotting  ROC curve in R:

plot.roc(resPonse, preDictor,  percent=TRUE)
```

V. Area Under the Curve, AUC:

The area under the curve value reported by a ROC is equal to the probability that a classifier will rank randomly chosen positive instance higher than a randomly chosen negative instance. The ROC curve identifies the discrimination ability of the classification system. The performance of a diagnostic variable can be quantified by calculating the area under the ROC curve (AUROC). In an ideal case scenarios, the test would have an AUC of 1, whereas any random guess would have an AUROC of 0.5.

VI. Kappa Statistic:

The Kappa statistic is a measure of concordance for categorical data that measures agreement relative

to what would be expected by chance. Values of 1 indicate perfect agreement, while a value of zero would indicate a lack of agreement. Negative Kappa values can also occur, but are less common since it would indicate a negative association between the observed and predicted data. Kappa is an excellent performance measure when the classes are highly unbalanced.

4.3.4 Data Mining Methodology

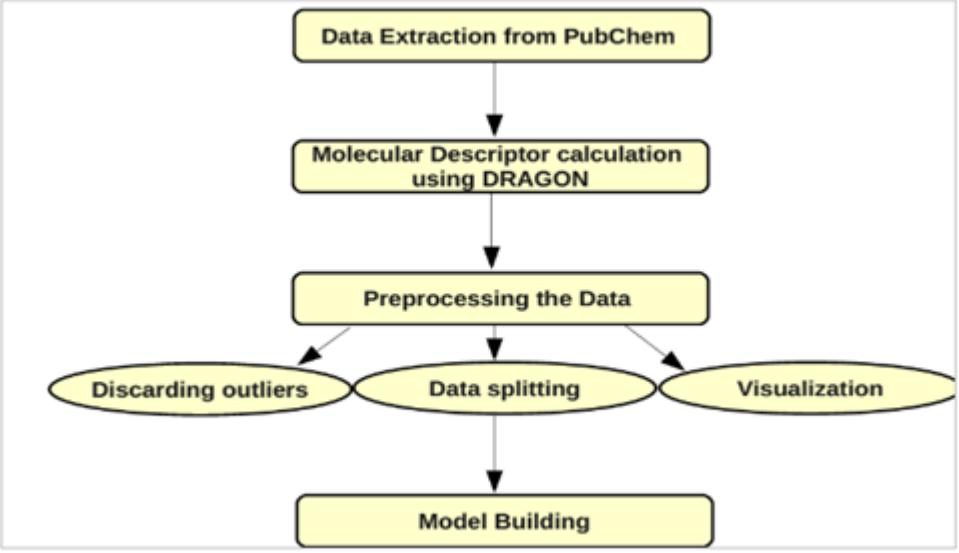


Figure 8. *The figure depicts the methodology used in building Quantitative Structure Activity Relationships models*.

1. **PubChem Bioassay**

PubChem is a public repository of chemical information including structures of small molecules and various molecules properties. It is administered as a part of the NIH Molecular Libraries Initiative (MLI). Its database , the BioAssay Database contains experimental results for some of the compounds in PubChem that have been tested in MLI screening centers of elsewhere for activity against particular biological targets. All the molecular structures for generating Classification models were retrieved from PubChem BioAssay (Wang et al., 2009) data corresponding to the BioAssay ID 'AID-1332'. The BioAssay reports 166 active and 927 inactive compounds based on QFRET –for differential inhibitors of the *Mycobacterium tuberculosis* H37Rv.

2. Descriptor Calculation

Since molecular descriptors can be used to evaluate molecular structure activity relationships as well as for similarity analysis and High Throughput Screening of molecule databases, these were calculated for the set of both active and inactive SDF files using DRAGON-6. The script

condor_dragon_descriptor.pl was used for descriptor calculation.

Dragon is a software for the calculation of molecular descriptors developed by the Milano Chemo-metrics and QSAR Research group. Molecular descriptors are the numeric representation of the Physio-chemical features extracted from various structural representation of a molecular structure. Such a quantitative representation is obtained as the result of a logical and mathematical procedure that transforms chemical information encoded with a symbolic representation of a molecule into a useful number (Gramatica et al., 2007). DRAGON calculates 4885 molecular descriptors that are divided into logical blocks.

The user can calculate not only the simplest atom type, functional group and fragment counts, but also several topological and geometrical descriptors. DRAGON was designed in order to deliver a user-friendly software where the descriptor calculation are performed with a simple logical sequence of loading of molecular files, selection of descriptors and saving of calculated descriptors.

3. Data preprocessing

In this chunk, the database is preprocessed in order to discard off the near zero values, highly correlated values, outliers, unbalanced distribution and Zero Variance Predictors. Near Zero Variance predictors are the predictors with single unique value which generally cause model failure. So they were removed using nzv function. Models can have poor performance in multicollinearity situations (ie high correlations between predictors). For removing highly correlated values, the cutoff threshold limit was given for finding out correlated values and the values sharing the highest correlation were removed using the findcorrelation function. The predictors which had a correlation above 0.75 were removed.

4. Data transformation

Once the final set of predictors was determined, the values required transformations before being used in the model. Some models, such as Partial least Squares, Neural Networks, Support Vector Machines need the predictor variables to be centered or scaled. The preprocessing function was used to determine values for predictor transformations using the training set. One of its arguments PCA (Principal Component Analysis) reduces the dimension of the dataset and presents it in the plot such a way which makes the task of interpreting the class of compounds easier.

5. Data splitting

The data was split into training set which is used for selecting model parameters, its values and model building and test set which is used to get an independent assessment of model efficacy by using the command “createDataPartition” and according to the already defined variable “pctTrain” where the percentage of the split was mentioned.

6. Model building and fitting

The train control function was used to select values of model tuning parameters and for the estimating model performance using resampling. For example, a radial basis function Support Vector machine had two tuning parameters, scale function and cost value. A set of modified datasets were created from the training samples using Boot(632 rule) (Efron et al,. 1983). Each dataset had a corresponding set of holdout samples. For each candidate tuning parameter combination , a model was fit to each resampled dataset and was used to predict the corresponding hold out samples. The performance estimates were used to evaluate which combinations of tuning parameters are appropriate. Once the final tuning values were assigned, the final model was refit using the entire training set.

This phase was executed by running the "Classification.Rnw" script using the Sweave command. The different models were built by altering the model name in the script itself. The model were saved as "modelFit.RData" for further use.

7. Prediction on the test set

The model was used to generate predictions for the test data which was initially split using the extract prediction function. The sensitivity and specificity used for characterizing the performance of the model were calculated using confusion matrix function and ROC curves was built to depict the model performance. These results were written into tex format, which were then converted into pdf files using 'texi2pdf'.

5. RESULTS AND DISCUSSION

5.1 FBA

1.Results for running standard FBA:

>> a=optimizeCbModel(model)

It gives the overall flux of the system in terms of the objective function. optimizeCbModel returns a solution structure containing: the objective value 'f', the primal solution 'x', the dual solution 'y', the reduced cost 'w', a universal status flag 'stat', a solver specific status flag 'origStat', and the time to compute the solution 'time'. The primal solution, 'x' represents the flux carried by each reaction within the model. The dual solution, 'y' represents the shadow prices for each metabolite and indicates how much the addition of the corresponding metabolite will increase or decrease the objective value[28, 60]. The reduced cost, 'w', indicates how much each reaction affects the objective. A solver status of 1 indicates that an optimal solution was found.

a =

x: [1028x1 double]

f: 0.0522

y: [826x1 double]

w: [1028x1 double]

stat: 1

origStat: 5

solver: 'glpk'

time: 0.0510

>> singleRxnDeletion(model)

This function gives out each reaction knockout flux of the system. It means that the effect of deleting a single reaction is observed as the variation in the objective when compared to the wild type case. Based on the results obtained, reactions that had a lethal effect upon their deletion were screened. The reactions that had zero reaction flux upon deletion were selected as essential reactions and were populated as listed below:

ReactionName	Reaction Description	Flux Value
AACPS10	ACP[c] + atp[c] + mocdca[c] -> amp[c] + mstrACP[c] + ppi[c]	0
AACPS3	ACP[c] + atp[c] + hdca[c] -> amp[c] + palmACP[c] + ppi[c]	0
ACACT1r	2 accoa[c] <=> aacoa[c] + coa[c]	0
ACCC	co2[c] + hexccoa[c] -> chexccoa[c] + h2o[c]	0
ACCOACr	accoa[c] + atp[c] + hco3[c] <=> adp[c] + h[c] + malcoa[c] + pi[c]	0
ACChex	coa[c] + h[c] + hexc[c] -> h2o[c] + hexccoa[c]	0
ACGAMT	uacgam[c] + udcpp[c] -> ump[c] + unaga[c]	0
ACHBS	2obut[c] + h[c] + pyr[c] -> 2ahbut[c] + co2[c]	0
AFE	agalfragund[c] + arabinan[c] -> arabinanagalfragund[c] + h2o[c]	0
AFTA	decda-tb[c] + galfragund[c] -> agalfragund[c] + decd-tb[c]	0
AGPAT160	1hdecg3p[c] + palmACP[c] -> ACP[c] + pa160[c]	0
AGPAT160190	1hdecg3p[c] + mstrACP[c] -> ACP[c] + pa160190[c]	0
AGPAT190	1msg3p[c] + mstrACP[c] -> ACP[c] + pa190190[c]	0
AHC	ahcys[c] + h2o[c] <=> adn[c] + hcys-L[c]	0
ALAALAr	2 ala-D[c] + atp[c] <=> adp[c] + alaala[c] + h[c] + pi[c]	0
ARABF	12 arab-D[c] + 8 mharab-D[c] + 12 tarab-D[c] -> arabinan[c] + 32 h2o[c]	0
ARABI	arab-D[c] <=> rbl-D[c]	0
ASNS1	asp-L[c] + atp[c] + gln-L[c] + h2o[c] -> amp[c] + asn-L[c] + glu-L[c] + h[c] + ppi[c]	0
ATPPRT	atp[c] + prpp[c] -> ppi[c] + prbatp[c]	0
BDH	bhb[c] + nad[c] <=> acac[c] + h[c] + nadh[c]	0
CDPPT160190	cdpc16c19g[c] + inost[c] -> cmp[c] + h[c] + ptd1ino160190[c]	0
CHORM	chor[c] -> pphn[c]	0
CHORS	3psme[c] -> chor[c] + pi[c]	0
CLPNS160190	2 pg160190[c] <=> clpn160190[c] + glyc[c]	0
CMCBTFL	cmcbtt[c] + fe3[e] -> fcmcbtt[c]	0
DAPDC	26dap-M[c] + h[c] -> co2[c] + lys-L[c]	0
DAPE	26dap-LL[c] <=> 26dap-M[c]	0
DASYN160	ctp[c] + h[c] + pa160[c] -> cdpdhdecg[c] + ppi[c]	0
DASYN160190	ctp[c] + h[c] + pa160190[c] -> cdpc16c19g[c] + ppi[c]	0
DASYN190190	ctp[c] + h[c] + pa190190[c] -> cdpc19c19g[c] + ppi[c]	0
DDPA	e4p[c] + h2o[c] + pep[c] -> 2dda7p[c] + pi[c]	0
DGK1	atp[c] + dgmp[c] <=> adp[c] + dgdp[c]	0
DHAD2	23dhmp[c] -> 3mop[c] + h2o[c]	0
DHDPRy	23dhdp[c] + h[c] + nadph[c] -> nadp[c] + thdp[c]	0
DHDPS	aspsa[c] + pyr[c] -> 23dhdp[c] + 2 h2o[c] + h[c]	0
DHFR	dhf[c] + h[c] + nadph[c] <=> nadp[c] + thf[c]	0
DHQD	3dhq[c] <=> 3dhsk[c] + h2o[c]	0

DHQS	2dda7p[c] -> 3dhq[c] + pi[c]	0
DMATT	dmpp[c] + ipdp[c] -> grdp[c] + ppi[c]	0
EX_fe3(e)	fe3[e] <=>	0
FACOAL160	atp[c] + coa[c] + hdca[c] <=> amp[c] + pmtcoa[c] + ppi[c]	0
FACOAL80	atp[c] + coa[c] + octa[c] <=> amp[c] + occoa[c] + ppi[c]	0
FACOALPHDCA	atp[c] + coa[c] + phdca[c] <=> amp[c] + phdcacoa[c] + ppi[c]	0
FAMPL1	amp[c] + h[c] + meroacidcyc2ACP[c] -> ACP[c] + meroacidcyc2AMP[c]	0
FAMPL2	amp[c] + h[c] + mmeroacidcyc2ACP[c] -> ACP[c] + mmeroacidcyc2AMP[c]	0
FAMPL3	amp[c] + h[c] + kmeroacidcyc2ACP[c] -> ACP[c] + kmeroacidcyc2AMP[c]	0
FAMPL4	amp[c] + h[c] + mmmeroacidcyc1ACP[c] -> ACP[c] + mmmeroacidcyc1AMP[c]	0
FAMPL5	amp[c] + h[c] + mkmeroacidcyc1ACP[c] -> ACP[c] + mkmeroacidcyc1AMP[c]	0
FAS100	3 h[c] + malcoa[c] + 2 nadph[c] + octa[c] -> co2[c] + coa[c] + dca[c] + h2o[c] + 2 nadp[c]	0
FAS120	dca[c] + 3 h[c] + malcoa[c] + 2 nadph[c] -> co2[c] + coa[c] + ddca[c] + h2o[c] + 2 nadp[c]	0
FAS140	ddca[c] + 3 h[c] + malcoa[c] + 2 nadph[c] -> co2[c] + coa[c] + h2o[c] + 2 nadp[c] + ttdca[c]	0
FAS160	3 h[c] + malcoa[c] + 2 nadph[c] + ttdca[c] -> co2[c] + coa[c] + h2o[c] + hdca[c] + 2 nadp[c]	0
FAS240_L	9 h[c] + 3 malcoa[c] + 6 nadph[c] + ocdca[c] -> 3 co2[c] + 3 coa[c] + 3 h2o[c] + 6 nadp[c] + ttc[c]	0
FAS260	3 h[c] + malcoa[c] + 2 nadph[c] + ttc[c] -> co2[c] + coa[c] + h2o[c] + hexc[c] + 2 nadp[c]	0
FAS80_L	accoa[c] + 8 h[c] + 3 malcoa[c] + 6 nadph[c] -> 3 co2[c] + 4 coa[c] + 2 h2o[c] + 6 nadp[c] + octa[c]	0
FASm1601	3 h[c] + hdca[c] + mmcoa-S[c] + 2 nadph[c] -> co2[c] + coa[c] + h2o[c] + m1ocdca[c] + 2 nadp[c]	0
FASm180	15 h[c] + 4 malcoa[c] + mmcoa-S[c] + 10 nadph[c] + octa[c] -> 5 co2[c] + 5 coa[c] + 5 h2o[c] + mocdca[c] + 10 nadp[c]	0
FASm1801	3 h[c] + m1ocdca[c] + mmcoa-S[c] + 2 nadph[c] -> co2[c] + coa[c] + dmarach[c] + h2o[c] + 2 nadp[c]	0
FASm2001	dmarach[c] + 3 h[c] + mmcoa-S[c] + 2 nadph[c] -> co2[c] + coa[c] + h2o[c] + 2 nadp[c] + tmbhn[c]	0
FASm2002	3 h[c] + mmcoa-S[c] + 2 nadph[c] + ocdca[c] -> co2[c] + coa[c] + h2o[c] + marach[c] + 2 nadp[c]	0
FASm220	arach[c] + 3 h[c] + mmcoa-S[c] + 2 nadph[c] -> co2[c] + coa[c] + h2o[c] + mbhn[c] + 2 nadp[c]	0
FASm2201	3 h[c] + mmcoa-S[c] + 2 nadph[c] + tmbhn[c] -> co2[c] + coa[c] + h2o[c] + 2 nadp[c] + tamlgnc[c]	0
FASm2202	3 h[c] + marach[c] + mmcoa-S[c] + 2 nadph[c] -> co2[c] + coa[c] + dmbhn[c] + h2o[c] + 2 nadp[c]	0
FASm240	3 h[c] + mbhn[c] + mmcoa-S[c] + 2 nadph[c] -> co2[c] + coa[c] + dmlgnc[c] + h2o[c] + 2 nadp[c]	0
FASm2401	3 h[c] + mmcoa-S[c] + 2 nadph[c] + tamlgnc[c] -> co2[c] + coa[c] + h2o[c] + 2 nadp[c] + pmhexc[c]	0
FASm2402	dmbhn[c] + 3 h[c] + mmcoa-S[c] + 2 nadph[c] -> co2[c] + coa[c] + h2o[c] + 2 nadp[c] + tmlgnc[c]	0
FASm260	dmlgnc[c] + 3 h[c] + mmcoa-S[c] + 2 nadph[c] -> co2[c] + coa[c] + h2o[c] + 2 nadp[c] + tmhexc[c]	0
FASm2601	3 h[c] + mmcoa-S[c] + 2 nadph[c] + pmhexc[c] -> co2[c] + coa[c] + h2o[c] + hmocta[c] + 2 nadp[c]	0
FASm2602	3 h[c] + mmcoa-S[c] + 2 nadph[c] + tmlgnc[c] -> co2[c] + coa[c] + h2o[c] + 2 nadp[c] + tamhexc[c]	0
FASm280	3 h[c] + mmcoa-S[c] + 2 nadph[c] + tmhexc[c] -> co2[c] + coa[c] + h2o[c] + 2 nadp[c] + tamocta[c]	0
FASm2801	3 h[c] + hmocta[c] + mmcoa-S[c] + 2 nadph[c] -> co2[c] + coa[c] + h2o[c] + hpmtria[c] + 2 nadp[c]	0
FASm2802	3 h[c] + mmcoa-S[c] + 2 nadph[c] + tamhexc[c] -> co2[c] + coa[c] + h2o[c] + 2 nadp[c] + pmocta[c]	0
FASm300	3 h[c] + mmcoa-S[c] + 2 nadph[c] + pmocta[c] -> co2[c] + coa[c] + h2o[c] + hmtria[c] + 2 nadp[c]	0
FASm320	3 h[c] + hmtria[c] + mmcoa-S[c] + 2 nadph[c] -> co2[c] + coa[c] + h2o[c] + hpmdtria[c] + 2 nadp[c]	0
FASm340	3 h[c] + hpmdtria[c] + mmcoa-S[c] + 2 nadph[c] -> co2[c] + coa[c] + h2o[c] + 2 nadp[c] + omtta[c]	0
FRRPPDIMAS	gdpfuc[c] + rrppdima[c] -> frrppdima[c] + gdp[c] + h[c]	0

G16MTM1	gdpmann[c] + ptd1ino160190[c] -> PIM1[c] + gdp[c] + h[c]	0
G16MTM10	PIM5[c] + gdpmann[c] -> PIM6[c] + gdp[c] + h[c]	0
G16MTM4	Ac1PIM2[c] + decdman1p-tb[c] -> Ac1PIM3[c] + decd-tb[c] + h[c]	0
G16MTM5	Ac1PIM3[c] + decdman1p-tb[c] -> Ac1PIM4[c] + decd-tb[c] + h[c]	0
G16MTM6	PIM1[c] + gdpmann[c] -> PIM2[c] + gdp[c] + h[c]	0
G16MTM7	PIM2[c] + gdpmann[c] -> PIM3[c] + gdp[c] + h[c]	0
G16MTM8	PIM3[c] + gdpmann[c] -> PIM4[c] + gdp[c] + h[c]	0
G16MTM9	PIM4[c] + gdpmann[c] -> PIM5[c] + gdp[c] + h[c]	0
G1PACT	accoa[c] + gam1p[c] -> acgam1p[c] + coa[c] + h[c]	0
G1PTT	dttp[c] + g1p[c] + h[c] -> dtdpglu[c] + ppi[c]	0
G3PAT160	glyc3p[c] + palmACP[c] -> 1hdecg3p[c] + ACP[c]	0
G3PAT190	glyc3p[c] + mstrACP[c] -> 1msg3p[c] + ACP[c]	0
GALFT	h2o[c] + ragund[c] + 30 udpgalfur[c] -> galfragund[c] + 30 h[c] + 30 udp[c]	0
GALKr	atp[c] + gal[c] <=> adp[c] + gal1p[c] + h[c]	0
GALT	gal1p[c] + h[c] + utp[c] <=> ppi[c] + udpgal[c]	0
GF6PTAr	f6p[c] + gln-L[c] <=> gam6p[c] + glu-L[c]	0
GFUCS	gdpddman[c] + h[c] + nadph[c] -> gdpfuc[c] + nadp[c]	0
GLNS	atp[c] + glu-L[c] + nh4[c] -> adp[c] + gln-L[c] + h[c] + pi[c]	0
GLUR	glu-D[c] <=> glu-L[c]	0
GMAND	gdpmann[c] -> gdpddman[c] + h2o[c]	0
GMT1	decd-tb[c] + gdpmann[c] -> decdman1p-tb[c] + gdp[c]	0
GMT2	3 gdpmann[c] + harab-D[c] -> 3 gdp[c] + 3 h[c] + mharab-D[c]	0
GRTT	grdp[c] + ipdp[c] -> frdp[c] + ppi[c]	0
HAS	arab-D[c] + 5 decda-tb[c] -> 5 decd-tb[c] + harab-D[c]	0
HISTD	h2o[c] + histd[c] + 2 nad[c] -> 3 h[c] + his-L[c] + 2 nadh[c]	0
HISTP	h2o[c] + hisp[c] -> histd[c] + pi[c]	0
HSTPT	glu-L[c] + imacp[c] -> akg[c] + hisp[c]	0
IG3PS	gln-L[c] + prlp[c] -> aicar[c] + eig3p[c] + glu-L[c] + h[c]	0
IGPDH	eig3p[c] -> h2o[c] + imacp[c]	0
ILETA	akg[c] + ile-L[c] <=> 3mop[c] + glu-L[c]	0
INSH	h2o[c] + ins[c] -> hxan[c] + rib-D[c]	0
IPMD	3c2hmp[c] + nad[c] -> 3c4mop[c] + h[c] + nadh[c]	0
IPPMIa	3c2hmp[c] <=> 2ippm[c] + h2o[c]	0
IPPMIb	2ippm[c] + h2o[c] <=> 3c3hmp[c]	0
IPPS	3mob[c] + accoa[c] + h2o[c] -> 3c3hmp[c] + coa[c] + h[c]	0
KARA2i	2ahbut[c] + h[c] + nadph[c] -> 23dhmp[c] + nadp[c]	0
LEUTA	akg[c] + leu-L[c] <=> 4mop[c] + glu-L[c]	0
MAN6PI	man6p[c] <=> f6p[c]	0
MANAT1	PIM1[c] + pmtcoa[c] -> Ac1PIM1[c] + coa[c]	0

MANAT3	Ac1PIM2[c] + pmtcoa[c] -> Ac2PIM2[c] + coa[c]	0
MCBTS	atp[c] + bhb[c] + h[c] + 2 lys-L[c] + 2 o2[c] + odecoa[c] + salc[c] + thr-L[c] -> adp[c] + co2[c] + coa[c] + 5 h2o[c] + mcbts[c] + pi[c]	0
MCBTS3	atp[c] + bhb[c] + 2 lys-L[c] + 2 nadp[c] + 3.5 o2[c] + occoa[c] + salc[c] + ser-L[c] -> adp[c] + cmcbtt[c] + co2[c] + coa[c] + 5 h2o[c] + 2 h[c] + 2 nadph[c] + pi[c]	0
MCOATA	ACP[c] + malcoa[c] <=> coa[c] + malACP[c]	0
METAT	atp[c] + h2o[c] + met-L[c] -> amet[c] + pi[c] + ppi[c]	0
MI1PP	h2o[c] + mi1p-D[c] -> inost[c] + pi[c]	0
MI1PS	g6p[c] -> mi1p-D[c]	0
MN6PP	h2o[c] + man6p[c] -> man[c] + pi[c]	0
MYC1CYC1	amet[c] + meroacidACP[c] -> ahcys[c] + h[c] + meroacidcyc1ACP[c]	0
MYC1CYC2	amet[c] + mmeroacidACP[c] -> ahcys[c] + h[c] + mmeroacidcyc1ACP[c]	0
MYC1CYC3	amet[c] + kmeroacidACP[c] -> ahcys[c] + h[c] + kmeroacidcyc1ACP[c]	0
MYC1CYC4	amet[c] + mmmeroacidACP[c] -> ahcys[c] + h[c] + mmmeroacidcyc1ACP[c]	0
MYC1CYC5	amet[c] + mkmeroacidACP[c] -> ahcys[c] + h[c] + mkmeroacidcyc1ACP[c]	0
MYC1M1	amet[c] + h2o[c] + mmeroacidACP[c] -> ahcys[c] + h[c] + mmmeroacidACP[c]	0
MYC1M2	amet[c] + h2o[c] + kmeroacidACP[c] -> ahcys[c] + h[c] + mkmeroacidACP[c]	0
MYC2CYC1	amet[c] + meroacidcyc1ACP[c] -> ahcys[c] + h[c] + meroacidcyc2ACP[c]	0
MYC2CYC2	amet[c] + mmeroacidcyc1ACP[c] -> ahcys[c] + h[c] + mmeroacidcyc2ACP[c]	0
MYC2CYC3	amet[c] + kmeroacidcyc1ACP[c] -> ahcys[c] + h[c] + kmeroacidcyc2ACP[c]	0
MYCON1	chexccoa[c] + 2 h2o[c] + meroacidcyc2AMP[c] -> amp[c] + co2[c] + coa[c] + 2 h[c] + mycolate[c]	0
MYCON2	chexccoa[c] + 2 h2o[c] + mmeroacidcyc2AMP[c] -> amp[c] + co2[c] + coa[c] + 2 h[c] + mmycolate[c]	0
MYCON3	chexccoa[c] + 2 h2o[c] + kmeroacidcyc2AMP[c] -> amp[c] + co2[c] + coa[c] + 2 h[c] + kmycolate[c]	0
MYCON4	chexccoa[c] + 2 h2o[c] + mmmeroacidcyc1AMP[c] -> amp[c] + co2[c] + coa[c] + 2 h[c] + mmmycolate[c]	0
MYCON5	chexccoa[c] + 2 h2o[c] + mkmeroacidcyc1AMP[c] -> amp[c] + co2[c] + coa[c] + 2 h[c] + mkmycolate[c]	0
MYCSacp50	33 h[c] + hexc[c] + 12 malACP[c] + 24 nadph[c] -> 11 ACP[c] + 12 co2[c] + 13 h2o[c] + meroacidACP[c] + 24 nadp[c]	0
MYCSacp56	42 h[c] + hexc[c] + 15 malACP[c] + 30 nadph[c] -> 14 ACP[c] + 15 co2[c] + 16 h2o[c] + mmeroacidACP[c] + 30 nadp[c]	0
MYCSacp58	47 h[c] + hexc[c] + 16 malACP[c] + 30 nadph[c] -> 15 ACP[c] + 16 co2[c] + 17 h2o[c] + kmeroacidACP[c] + 30 nadp[c]	0
NDPK4	atp[c] + dtdp[c] <=> adp[c] + dttp[c]	0
O16RHAT	dtdprmn[c] + unaga[c] -> dtdp[c] + h[c] + ragund[c]	0
OMCDC	3c4mop[c] + h[c] -> 4mop[c] + co2[c]	0
PAPPT1	uAgla[c] + udcpp[c] -> uaAgla[c] + ump[c]	0
PAPPT2	uGgla[c] + udcpp[c] -> uaGgla[c] + ump[c]	0
PAPPT3	udcpp[c] + ugmda[c] -> uagmda[c] + ump[c]	0
PATS	5 h[c] + ocdca[c] + 4 tmlgnc[c] + tre[c] -> 5 h2o[c] + pat[c]	0
PDIMAS	2 h[c] + phthiocerol[c] + 2 tamocta[c] -> 2 h2o[c] + pdima[c]	0
PGAMT	gam1p[c] <=> gam6p[c]	0
PGLS	4 amet[c] + frrppdima[c] -> 4 ahcys[c] + 4 h[c] + mfrrppdima[c]	0
PGMT	g1p[c] <=> g6p[c]	0
PGPP160	h2o[c] + pgp160[c] -> pg160[c] + pi[c]	0

PGPP160190	h2o[c] + pgp160190[c] -> pg160190[c] + pi[c]	0
PGPP190	h2o[c] + pgp190[c] -> pg190[c] + pi[c]	0
PGPPT3	udcpp[c] + uggmda[c] -> ugggmda[c] + ump[c]	0
PGSA160	cdpdhdecg[c] + glyc3p[c] -> cmp[c] + h[c] + pgp160[c]	0
PGSA160190	cdpc16c19g[c] + glyc3p[c] -> cmp[c] + h[c] + pgp160190[c]	0
PGSA190	cdpc19c19g[c] + glyc3p[c] -> cmp[c] + h[c] + pgp190[c]	0
PHDCATA	ACP[c] + phdcacoa[c] <=> coa[c] + phdcaACP[c]	0
PHETA1	akg[c] + phe-L[c] <=> glu-L[c] + phpyr[c]	0
PHTHDLS	h[c] + prephth[c] -> co2[c] + phthdl[c]	0
PHTHDLS2	h[c] + prepphth[c] -> co2[c] + pphthdl[c]	0
PHTHS	amet[c] + nadph[c] + phthdl[c] -> ahcys[c] + nadp[c] + phthiocerol[c]	0
PHTHS2	amet[c] + nadph[c] + pphthdl[c] -> ahcys[c] + nadp[c] + pphthiocerol[c]	0
PMANM	man1p[c] <=> man6p[c]	0
PPDIMAS	2 h[c] + pphthiocerol[c] + 2 tamocta[c] -> 2 h2o[c] + ppdima[c]	0
PPND	nad[c] + pphn[c] -> 34hpp[c] + co2[c] + nadh[c]	0
PPNDH	h[c] + pphn[c] -> co2[c] + h2o[c] + phpyr[c]	0
PPTGS_TB1	uaaAgla[c] -> h[c] + peptido-TB1[c] + udcpdp[c]	0
PPTGS_TB2	uaaGgla[c] -> h[c] + peptido-TB2[c] + udcpdp[c]	0
PRAMPC	h2o[c] + prbamp[c] -> prfp[c]	0
PRATPP	h2o[c] + prbatp[c] -> h[c] + ppi[c] + prbamp[c]	0
PREPPACPH	h2o[c] + prepphthACP[c] -> ACP[c] + h[c] + prepphth[c]	0
PREPTHS2	14 h[c] + 2 malcoa[c] + 2 mmcoa-S[c] + 10 nadph[c] + phdcaACP[c] -> 2 co2[c] + 4 coa[c] + 5 h2o[c] + 10 nadp[c] + prepphthACP[c]	0
PRMICIi	prfp[c] -> prlp[c]	0
PSCVT	pep[c] + skm5p[c] <=> 3psme[c] + pi[c]	0
RBK_Dr	atp[c] + rbl-D[c] <=> adp[c] + h[c] + ru5p-D[c]	0
RPPDIMAS	dtdprmn[c] + ppdima[c] -> dtdp[c] + h[c] + rppdima[c]	0
RRPPDIMAS	dtdprmn[c] + rppdima[c] -> dtdp[c] + h[c] + rrppdima[c]	0
SALCS	chor[c] -> pyr[c] + salc[c]	0
SDPDS	h2o[c] + sl26da[c] -> 26dap-LL[c] + succ[c]	0
SDPTA	akg[c] + sl26da[c] <=> glu-L[c] + sl2a6o[c]	0
SHK3Dr	3dhsk[c] + h[c] + nadph[c] <=> nadp[c] + skm[c]	0
SHKK	atp[c] + skm[c] -> adp[c] + h[c] + skm5p[c]	0
TAS	arab-D[c] + 3 decda-tb[c] -> 3 decd-tb[c] + tarab-D[c]	0
TATS	4 h[c] + hdca[c] + hpmtria[c] + 2 omtta[c] + tre[c] -> 4 h2o[c] + tat[c]	0
TATSO	4 h[c] + hdca[c] + hpmtria[c] + 2 omtta[c] + tres[c] -> 4 h2o[c] + sl1[c]	0
TDMS1	2 h[c] + mmycolate[c] + mycolate[c] + tre[c] -> 2 h2o[c] + tdm1[c]	0
TDMS2	2 h[c] + mmmycolate[c] + mycolate[c] + tre[c] -> 2 h2o[c] + tdm2[c]	0
TDMS3	2 h[c] + mkmycolate[c] + mycolate[c] + tre[c] -> 2 h2o[c] + tdm3[c]	0
TDMS4	2 h[c] + kmycolate[c] + mycolate[c] + tre[c] -> 2 h2o[c] + tdm4[c]	0

TDPDRE	dtdp4d6dg[c] -> dtdp4d6dm[c]	0
TDPDRR	dtdp4d6dm[c] + h[c] + nadph[c] -> dtdprmn[c] + nadp[c]	0
TDPGDH	dtdpglu[c] -> dtdp4d6dg[c] + h2o[c]	0
THDPS	h2o[c] + succoa[c] + thdp[c] -> coa[c] + sl2a6o[c]	0
TMDS	dump[c] + mlthf[c] -> dhf[c] + dtmp[c]	0
TMHAS1	tdm1[c] + tdm2[c] + tre[c] -> 2 h2o[c] + tmha1[c]	0
TMHAS2	tdm1[c] + tdm3[c] + tre[c] -> 2 h2o[c] + tmha2[c]	0
TMHAS3	tdm1[c] + tdm4[c] + tre[c] -> 2 h2o[c] + tmha3[c]	0
TMHAS4	tdm2[c] + tdm3[c] + tre[c] -> 2 h2o[c] + tmha4[c]	0
TRE6PS	g6p[c] + udpg[c] -> h[c] + tre6p[c] + udp[c]	0
TRESULT	paps[c] + tre[c] -> h[c] + pap[c] + tres[c]	0
TYRTA	akg[c] + tyr-L[c] <=> 34hpp[c] + glu-L[c]	0
UAAGDS	26dap-M[c] + atp[c] + uamag[c] -> adp[c] + h[c] + pi[c] + ugmd[c]	0
UAAGLS1	atp[c] + lys-L[c] + uamag[c] -> adp[c] + h[c] + pi[c] + uAgl[c]	0
UAAGLS2	atp[c] + lys-L[c] + uamag[c] -> adp[c] + h[c] + pi[c] + uGgl[c]	0
UAGCVT	pep[c] + uacgam[c] -> pi[c] + uaccg[c]	0
UAGDP	acgam1p[c] + h[c] + utp[c] -> ppi[c] + uacgam[c]	0
UAGPT1	uaAgla[c] + uacgam[c] -> h[c] + uaaAgla[c] + udp[c]	0
UAGPT2	uaGgla[c] + uacgam[c] -> h[c] + uaaGgla[c] + udp[c]	0
UAGPT3	uacgam[c] + uagmda[c] -> h[c] + uaagmda[c] + udp[c]	0
UAMAGS	atp[c] + glu-D[c] + uama[c] -> adp[c] + h[c] + pi[c] + uamag[c]	0
UAMAS	ala-L[c] + atp[c] + uamr[c] -> adp[c] + h[c] + pi[c] + uama[c]	0
UAMRH	h[c] + nadph[c] + o2[c] + uamr[c] -> h2o[c] + nadp[c] + ugmr[c]	0
UAPGR	h[c] + nadph[c] + uaccg[c] -> nadp[c] + uamr[c]	0
UDCPDP	h2o[c] + udcpdp[c] -> h[c] + pi[c] + udcpp[c]	0
UDPGALM	udpgal[c] -> udpgalfur[c]	0
UGAGDS	26dap-M[c] + atp[c] + ugmag[c] -> adp[c] + h[c] + pi[c] + uggmd[c]	0
UGGPT3	uacgam[c] + ugggmda[c] -> h[c] + udp[c] + ugagmda[c]	0
UGLDDS1	alaala[c] + atp[c] + uAgl[c] -> adp[c] + pi[c] + uAgla[c]	0
UGLDDS2	alaala[c] + atp[c] + uGgl[c] -> adp[c] + pi[c] + uGgla[c]	0
UGMAGS	atp[c] + glu-D[c] + ugma[c] -> adp[c] + h[c] + pi[c] + ugmag[c]	0
UGMAS	ala-L[c] + atp[c] + ugmr[c] -> adp[c] + h[c] + pi[c] + ugma[c]	0
UGMDDS	alaala[c] + atp[c] + ugmd[c] -> adp[c] + h[c] + pi[c] + ugmda[c]	0
UGMDDS2	alaala[c] + atp[c] + uggmd[c] -> adp[c] + h[c] + pi[c] + uggmda[c]	0
VALTA	akg[c] + val-L[c] <=> 3mob[c] + glu-L[c]	0
biomass_Mtb_9_60atp		0

Table 2. List of essential reactions from singleReactions deletion function:

These reactions were sought as essential not just on the basis of the flux values but also by cross validating the biological pathways in which they occur. For E.g. the amino acid pathways: these reactions are eventually incorporated into functional proteins

Finally, from these reactions, the metabolites were screened and selected as essential metabolites. Since the metabolites were repeating in multiple reactions, only one occurrence was recorded and the final list of 44 metabolites was made.

No.	ID	NAME
1	3593277	alpha-ketobutyric acid
2	23672314	alpha-ketoglutaric acid
3	979	4-hydroxyphenylpyruvic acid
4	7339	Phosphoribosyl Pyrophosphate
5	6971017	acetoacetic acid
6	5490374	acyl carrier protein (65-74)
7	23615194	adenosine 3'-phosphate-5'-phosphate
8	60961	Adenosine
9	6022	Adenosine Diphosphate
10	6083	Adenosine Monophosphate
11	5957	Adenosine Triphosphate
12	6131	Cytidine Monophosphate
13	317	Coenzyme A
14	66308	Arabinose
15	45479346	trans,octacis-decaprenylphospho-beta-D-arabinofuranose
16	6036	Galactose
17	6994968	2'-deoxyguanosine 5'-phosphate
18	151261	ribulose
19	65063	2'-deoxyuridylic acid
20	18396	Guanosine Diphosphate Mannose
21	504166	hexadecanoate
22	6021	Inosine
23	5950	Alanine
24	33032	Glutamic Acid
25	439398	l-histidinol phosphate
26	165271	Histidinol
27	5962	Lysine
28	6140	Phenylalanine
29	6057	Tyrosine
30	15983949	NADP
31	42609791	NADP
32	5893	NAD
33	3033836	Octadecanoate
34	997	phenylpyruvic acid

35	52941750	Phosphatidylglycerol(16:0/18:1)
36	46873832	phosphatidylglycerophosphate
37	45480609	phosphatidylinositol mannoside
38	3541112	3-Hydroxybutyric Acid
39	34756	S-Adenosylmethionine
40	123909	methylmalonyl-coenzyme A
41	439161	succinyl-coenzyme A
42	25243858	UDP-N-acetylmuramoyl-L-alanyl-D-glutamate
43	46173748	UDP-N-acetylmuramoyl-L-alanyl-D-glutamyl-L-lysine
44	46931082	UDP-N-acetylmuramoyl-L-alanyl-gamma-D-glutamyl-L-lysine

Table 3. List of molecules identified as essential from single reaction deletion function:

5.2 DATA MINING

5.2.1 Dataset Extraction

In this study, the data downloaded consists of all the available 3D structures of the targets that are known to be active or inactive against *Mycobacterium tuberculosis* from the bioassay screen: AID: 1332. The data had two classes "Active" and "Inactive" containing a total of 166 and 927 molecules respectively. The bio activity values are appended in the data as class attribute labeled as Level. The dataset is then used as the input for calculating the Molecular Descriptors.

5.2.2 Calculation of Molecular Descriptors

The software used to calculate the Molecular descriptors is DRAGON6. Dragon6 provides 4,885 molecular descriptors divided into 29 logical blocks, each in turn is divided into a number of sub blocks making it easier to retrieve the molecular descriptors of interest. The Dragon software can also be used to merge the calculated molecular descriptors and the user-defined properties for a set of molecules, which enables us to extract a complete output file which can be easily loaded by any correlation analysis application. These descriptors can be used for various purposes – for example: to evaluate molecular structure-activity relationships (QSAR/QSPR), similarity analysis and screening of molecule database.

The software was licensed for academic use, and installed on three computers available in lab. The computers all had variants of the linux operating system, running CentOS or Fedora. Although the software can use files containing an unlimited number of molecules as input, in practice the presence of an error in the molecular SDF format can cause the application to exit with a segmentation fault, losing the calculated results. For this reason, the input files are split smaller files, each containing 150 molecules, and the results merged. The molecules showing errors in the sdf format were pruned out. Descriptors were calculated for all actives (166) and inactive (927) molecules separately using this method. Dragon software requires a XML template file with input

details of descriptors and generates an output file using the following command:

```
system "nohup /usr/share/dragon6/dragon6shell -s $file.drs > $file.err &"
```

5.2.3 Data Preprocessing and Splitting

Data in the form of a comma-delimited file is read into R, after loading the caret package. The preprocessing of the data is done in order to discard the near-zero values, highly correlated values, missing values, outliers, unbalanced distribution and zero variance predictors. Columns containing missing values cause errors during model-building and are removed using standard R functions while reading in the data. The removal of near zero values and highly-correlated columns from the data is useful in the case of large datasets as they provide redundant information for discrimination, and increase the time for building the models.

In case of removal of highly correlated values, the cutoff threshold limit is given for finding out correlated values and the values sharing the highest correlation are removed. In the R script used the following modules are implemented: loading of the original data sets, preprocessing the data, principal components analysis of resultant data and finally data-splitting into training (for selecting model parameters, its values and model building) and test set (used to get an independent assessment of model efficacy).

The data is randomly split in about 80:20 ratio where 80 % of the data is used as training set and 20% of the data is used as the test set. The data set used in this study for the training set consisted of 870 samples variables and the test set consisted of 217 samples variables. Both had a total of 1823 predictor variables. The two files were labeled as trainClass.RData and testClass.RData, respectively.

5.2.4 Model Building

Model building was done using an R script in which various methods available can be used to build models, and the respective grid size can be easily defined. The run time of the methods differ owing to difference in algorithms used and grid size. The caret package in R, being a wrapper for a large number of methods, is ideal to implement our objective of comprehensively covering model building methods in cheminformatics. The methods, corresponding to those described in the theory section of this dissertation, are k nearest neighbor, svmRadial, random forest, Naive Bayes, boosting and linear models such as generalized linear model, partial least-square, etc. The methods besides differing in their accuracy levels and overall prediction, also differ in their computational requirements.

Initially the models were tuned on the training set that consisted of 870 sample and 1823 predictor variables. The breakdown of the outcome data classes were: “active” (n=133) and “inactive” (n=737). The models were then build on a test set of 217 samples and 1823 predictors variable,

with unlabeled outcomes.

Components for estimating performance of a model

1. **Confusion Matrix:** It is a matrix that contains information about actual and predicted classifications done by a classification system. Performance of such systems is commonly evaluated using the data in the matrix. The following table shows the confusion matrix for a two classifiers.

		Predicted Negative	Positive
Actual	Negative	TN	FP
	Positive	FN	TP

Where
TN-> number of correct predictions that an instance is negative.
FP-> number of incorrect predictions that an instance is positive.
FN -> number of incorrect predictions that an instance is negative.
TP-> number of correct prediction that an instance is positive.

2. <u>Sensitivity:</u>

SE, which is the probability of classifying/predicting an active drug target as against actual active drugs

$SE = TP / (TP + FN) = TP / P.$

3. Specificity:

SP, is the probability of classifying/predicting an inactive drug target against those as actual inactive drugs.

$SP = TN / (FP + TN) = TN / P.$

4. Kappa statistics:

It is a measure of concordance for categorical data that measures agreement relative to what would be expected by chance. Value of 1 indicate perfect agreement, while a value of zero would indicate a lack of agreement. Negative Kappa value can be also occur, but are less common since it would indicate a negative association between the observed and predicted data. Kappa is an excellent performance measure when the classes are highly unbalanced.

5. ROC Graph:

Receiver Operating Characteristic Curve is a graphical plot of the sensitivity or true positive rate versus false positive rate or (1- specificity) for a binary classifier system as its discrimination threshold is varied. It is also known as Relative Operating Characteristic Curve because it is a comparison of two operating characteristics i.e. true positive rate and false positive rate as the criterion changes. ROC analysis provides tools to select possibly optimal models and to discard suboptimal ones independently from the cost context or the class distribution.

5.3 GRAPH RESULTS

1. **Confusion matrix for the models :**

		Observed Values	
		Active	**Inactive**
Random Forest	**Active**	56	12
	Inactive	11	172
Bagged Tree	**Active**	20	12
	Inactive	13	172
PLS	**Active**	18	11
	Inactive	15	173
svmRadial	**Active**	20	7
	Inactive	13	177
rPart	**Active**	10	9
	Inactive	23	175
GBM	**Active**	17	7
	Inactive	16	177

One of the ROC curves for the test set consisting 217 molecules is depicted below. The model being shown is based on Partial Least Square test method.

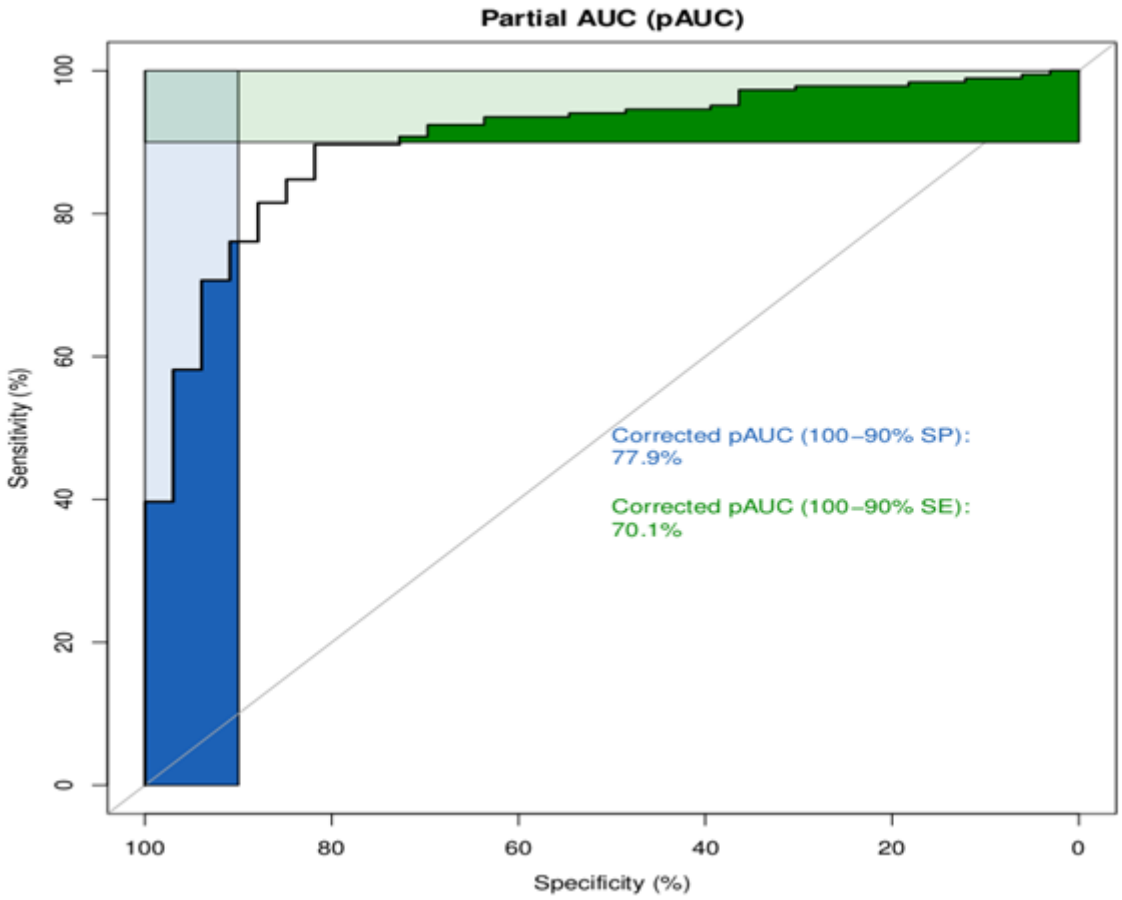


Figure 9. *ROC curve on test set for PLS.*

MODEL NAME	ACCURACY	SENSTIVITY	SPECIFICITY	ROC
Random forest	0.899	0.915	0.485	0.973
Treebag	0.885	0.892	0.606	0.935
svmRadial	0.908	0.902	0.606	0.962
rPart	0.853	0.723	0.303	0.951
PLS	0.88	0.904	0.545	0.94
GBM	0.894	0.881	0.515	0.962
GLM	0.5530	0.537	0.515	0.560

Table 4. Table shows the performance score of the Preditive Models

All the models built using CARET package gave high performance on test set. The highest performance was shown by Treebag and lowest by Glm. The models gave an average accuracy of 0.8388 and AUC of 0.897.
These results suggest that the Classification models built can be successfully applied to the unknown datasets to separate the data into set of positives and negatives.

The outcome of the prediction on the listed essential molecules is listed below.

	ID	NAME	rPart	baggedTree
1	3593277	alpha-ketobutyric acid	inactive	active
2	23672314	alpha-ketoglutaric acid	inactive	active
3	979	4-hydroxyphenylpyruvic acid	inactive	active
4	7339	Phosphoribosyl Pyrophosphate	active	active
5	6971017	acetoacetic acid	inactive	active
6	5490374	acyl carrier protein (65-74)	active	active
7	23615194	adenosine 3'-phosphate-5'-phosphate	active	active
8	60961	Adenosine	active	active
9	6022	Adenosine Diphosphate	active	active
10	6083	Adenosine Monophosphate	active	active
11	5957	Adenosine Triphosphate	active	active
12	6131	Cytidine Monophosphate	active	active

13	317	Coenzyme A	active	active
14	66308	Arabinose	active	active
15	45479346	trans,octacis-decaprenylphospho-beta-D-arabinofuranose	active	active
16	6036	Galactose	active	active
17	6994968	2'-deoxyguanosine 5'-phosphate	active	active
18	151261	ribulose	active	active
19	65063	2'-deoxyuridylic acid	active	active
20	18396	Guanosine Diphosphate Mannose	active	active
21	504166	hexadecanoate	inactive	active
22	6021	Inosine	active	active
23	5950	Alanine	inactive	active
24	33032	Glutamic Acid	inactive	active
25	439398	l-histidinol phosphate	active	active
26	165271	Histidinol	active	active
27	5962	Lysine	inactive	active
28	6140	Phenylalanine	inactive	active
29	6057	Tyrosine	inactive	active
30	15983949	NADP	active	active
31	42609791	NADP	active	active
32	5893	NAD	active	active
33	3033836	Octadecanoate	inactive	active
34	997	phenylpyruvic acid	active	active
35	52941750	Phosphatidylglycerol(16:0/18:1)	active	active
36	46873832	phosphatidylglycerophosphate	active	active
37	45480609	phosphatidylinositol mannoside	active	active
38	3541112	3-Hydroxybutyric Acid	inactive	active
39	34756	S-Adenosylmethionine	active	active
40	123909	methylmalonyl-coenzyme A	active	active
41	439161	succinyl-coenzyme A	active	active
42	25243858	UDP-N-acetylmuramoyl-L-alanyl-D-glutamate	active	active
43	46173748	UDP-N-acetylmuramoyl-L-alanyl-D-glutamyl-L-lysine	active	active
44	46931082	UDP-N-acetylmuramoyl-L-alanyl-gamma-D-glutamyl-L-lysine	active	active
NC_1		Negative Control_1	inactive	inactive
NC_2		Negative Control_2	inactive	inactive
NC_3		Negative Control_3	inactive	inactive
NC_4		Negative Control_4	inactive	inactive
NC_5		Negative Control_5	inactive	inactive
NC_6		Negative Control_6	inactive	inactive
NC_7		Negative Control_7	inactive	inactive

NC_8		Negative Control_8	inactive	inactive
NC_9		Negative Control_9	inactive	inactive
NC_10		Negative Control_10	inactive	inactive
NC_11		Negative Control_11	inactive	inactive
NC_12		Negative Control_12	inactive	inactive
NC_13		Negative Control_13	inactive	inactive
NC_14		Negative Control_14	inactive	inactive
NC_15		Negative Control_15	inactive	inactive
NC_16		Negative Control_16	inactive	inactive
NC_17		Negative Control_17	inactive	inactive
NC_18		Negative Control_18	inactive	inactive
NC_19		Negative Control_19	inactive	inactive
NC_20		Negative Control_20	inactive	inactive
NC_21		Negative Control_21	inactive	inactive
NC_22		Negative Control_22	inactive	inactive
NC_23		Negative Control_23	inactive	inactive
NC_24		Negative Control_24	inactive	inactive
NC_25		Negative Control_25	inactive	inactive
NC_26		Negative Control_26	inactive	inactive
NC_27		Negative Control_27	inactive	inactive
NC_28		Negative Control_28	inactive	inactive
NC_29		Negative Control_29	inactive	inactive
NC_30		Negative Control_30	inactive	inactive

Table 5. Prediction results for the essential metabolite dataset

6. CONCLUSION

We have explored two techniques to identify potential lead molecules for rational drug design. Flux Balance Analysis can be used to systematically knock-out proteins and identify essential reactions. We have also employed a systematic and comprehensive approach to build supervised classification based predictive models for anti-tubercular agents from publicly available bioassay datasets for in-vitro screens for Mtb inhibitors and then Analysis using them onto metabolites identified as potential targets from FBA. In contrast with the conventional target-based screening approaches, these models are target-agnostic as they are based on whole-cell screening experiments. We have used these classification models for anti-tubercular agent.

The thesis defines a standard protocol for deploying data-mining methods in conjunction with metabolic flux analysis. Bioassay datasets, containing the screening results to identify active molecules were used for model-building. Models were built using the methods including partial least square, generalized linear model, k-nearest neighbor, Boosting (glmboost), Bagging (treebag), random forest and support vector machines. The developed models performed well in identifying active molecules against *M. tuberculosis* and are easily extendable to other data sets.

When used in screening, care is taken to minimize the number of false positives. The larger the number of false positives increases the cost of experimental validation as a downstream step in drug discovery.

The prediction when extended to metabolites within the Mtb metabolic network provided insights about the druggability of those molecules along with its influx details. This determines which molecules have high influx rates and also show a promising drug potential.

These flux based predictive models can now aid in the virtual screening of large molecular libraries to mine novel chemical scaffolds and thereby trigger an accelerated drug discovery for Mtb.

7. FUTURE PERSPECTIVES

Mtb presents a plethora of molecular targets that hold a great potential for therapeutic intervention in the post genomic era. These opportunities present the Mtb genome as a highly druggable genome. Herein, we are proposing the potential models of influx for drug like molecules.

Since metabolism is fundamental in sustaining microbial life, drugs that target pathogen-specific metabolites, enzymes and pathways can be. Studies have identified proteins critical for the survival for *M.tuberculosis* that are likely to have high rates of success as drug candidates. Many of the druggable proteins are enzymes that control several metabolic processes within the cells by catalyzing the reactions converting nutrients into energy and new molecules which are in a way crucial for the survival of the microbe.

As a proof of concept, the molecules selected as essential through the FBA filter were datasets scored with models using Treebag(Bagged Tree method) and rPart(Recursive Partitioning), along with negative controls- which were plant metabolites not present in either the host or pathogen. Most molecules being predicted as actives were also influxed within the cell providing a new approach to design novel drug targets or scaffolds to treat the Mtb infection.

Future extensions of this work can explore further filters, such as the removal of metabolites that are essential to the host and addition of known drug-like molecules or known drugs which can be re-scored for their potential efficacy against *M. tuberculosis*.

8. REFERENCES

1. Fang, X.; Wallqvist, A.; Reifman, J. (2009). A systems biology framework for modeling metabolic enzyme inhibition of *Mycobacterium tuberculosis*. BMC systems biology, ***3***, 92.

2. Jamshidi, N; Palsson, BO. (2007) Investigating the metabolic capabilities of *Mycobacterium tuberculosis* H37Rv using the *in silico* strain iNJ661 and proposing alternative drug targets. BMC Syst Biol , **1**,26

3. Beste, DJ; Hooper, T; Stewart, G; Bonde, B; Avignone-Rossa, C; Bushell, ME; Wheeler, P; Klamt, S; Kierzek, AM; McFadden, J. (2007) GSMN-TB: a web-based genome-scale network model of *Mycobacterium tuberculosis* metabolism. Genome Biol, **8**:R89.

4. Cole,ST; Brosch, R; Parkhill, J; Garnier, T; Churcher, C; Harris, D; Gordon, SV; Eiglmeier, K; Gas, S; Barry, CE 3rd; *et al*. (1998). Deciphering the biology of *Mycobacterium tuberculosis* from the complete genome sequence. Nature, **393**,537-544.

5. McGrath, M; van Pittius, NG; van Helden, PD; Warren; Warner, DF. (2014). Mutation rate and the emergence of drug resistance in *Mycobacterium tuberculosis*. Journal of Antimicrobial Chemotherapy, **69(2),** 292-302.

6. Baart, GJ; Martens, DE. (2012). Genome-scale metabolic models: reconstruction and analysis. Methods Mol Biol.**799,**107-26.

7. Olivares, J; Bernardini, A; Garcia-Leon, G, Corona, F; Sanchez MB; Martinez, JL. (2013). The intrinsic resistome of bacterial pathogens. Front Microbiol. **4,**103.

8. Raman, K; Yeturu, K; Chandra, N. (2008). targetTB: a target identification pipeline for *Mycobacterium tuberculosis* through an interactome, reactome and genome-scale structural analysis. BMC Syst Biol.**2,**109.

9. Keller, TH; Pichota, A; Yin, Z. (2006). A practical view of 'druggability'. Curr OpinChem Biol.**10(4),**357-61.

10. Klamt, S; Stelling, J. (2003). Two approaches for metabolic pathway analysis?. Trends in biotechnology, **21(2)**, 64-69.

11. Planes, FJ; Beasley, JE. (2009). Path finding approaches and metabolic pathways. Discrete Applied Mathematics, **157(10),** 2244-2256.

12. Hatzimanikatis, V; Li, C; Ionita, JA; Broadbelt, LJ. (2004). Metabolic networks: enzyme function and metabolite structure. Current Opinion in Structural Biology, **14(3),** 300-306.

13. Price, ND; Papin, JA; Schilling, CH; Palsson, BO. (2003). Genome-scale microbial *in silico* models: the constraints-based approach. Trends in biotechnology, **21(4),** 162-169.

14. Fang, X; Wallqvist, A; Reifman, J. (2009). A systems biology framework for modeling metabolic enzyme inhibition of *Mycobacterium tuberculosis*. BMC systems biology, **3(1),** 92.

15. Giannoulatou, E; Hein, J. (2006). Evolution of metabolic networks.

16. Ponce-de-León, M; Montero, F; Peretó, J. (2013). Solving gap metabolites and blocked reactions in genome-scale models: application to the metabolic network of *Blattabacterium cuenoti*. BMC systems biology, **7(1),** 114.

17. Mackie, A; Keseler, IM; Nolan, L; Karp, PD; Paulsen, IT. (2013). Dead End Metabolites-Defining the Known Unknowns of the E. coli Metabolic Network. PloS one**, 8(9),** e75210.

18. Reed, JL; Vo, TD; Schilling, CH; Palsson, BO. (2003). An expanded genome-scale model of Escherichia coli K-12 (iJR904 GSM/GPR). Genome Biol, **4(9),** R54.

19. Orth, JD; Thiele, I; Palsson, BO. (2010). What is flux balance analysis?. Nature biotechnology, **28(3),** 245-248.

20. Jamshidi, N; Palsson, BO. (2007). Investigating the metabolic capabilities of *Mycobacterium tuberculosis* H37Rv using the in silico strain iNJ661 and proposing alternative drug targets. BMC systems biology, **1(1),** 26.

21. Schilling, CH; Schuster, S; Palsson, BO; Heinrich, R. (1999). Metabolic pathway analysis: basic concepts and scientific applications in the post-genomic era. Biotechnology progress, **15(3),** 296-303.

22. Kauffman, KJ. Prakash, P; Edwards, JS. (2003). Advances in flux balance analysis. Current opinion in biotechnology, **14(5),** 491-496.

23. Krieger CJ; Zhang. P; Mueller LA; Wang A; Paley S; Arnaud M; Pick J; Rhee S; Karp PD. (2006) Metacyc: A multiorganism database of metabolic pathways and enzymes, Nucleic Acids Research, **32(1),**D438 -4

24. Green ML; Karp PD. (2007). Using genome -context data to identify specific types of functional associations in pathway/genome databases. Bioinformatics Research Group, SRI International, Menlo Park, CA 94025, USA, **23(13),** 205-11.

25. Green ML; Karp PD. (2004). A bayesian method for identifying missing enzymes in predicted metabolic pathway databases, BMC Bioinformatics ***5*(1),** 76.

26. Ekins, S; Shimada, J; Chang, C. (2006). Application of data mining approaches o drug delivery. Advanced drug delivery reviews, **58(12),** 1409-1430.

27. Becker, SA; Feist, AM; Mo, ML; Hannum, G; Palsson, BO; Herrgard, MJ. (2007). Quantitative prediction of cellular metabolism with constraint-based models: the COBRA Toolbox. Nature protocols, **2(3),** 727-738.

28. Gramatica, P. (2007). Principles of QSAR models validation: internal and external. *QSAR & combinatorial science*, ***26*(5),** 694-701.

APPENDIX-A

Summary of codes for FBA in MATLAB and their usage and references for their downloads

FUNCTION	APPLICATION	REFERENCE
writeCbModel	writeCbModel Write out COBRA models in various formats	http://opencobra.sourceforge.net/openCOBRA/opencobra_documentation/cobra_toolbox_2/index.html
readCbModel	readCbModel Read in a constraint-based model	http://opencobra.sourceforge.net/openCOBRA/opencobra_documentation/cobra_toolbox_2/index.html
optimizeCbModel	optimizeCbModel Solve a flux balance analysis problem	http://opencobra.sourceforge.net/openCOBRA/opencobra_documentation/cobra_toolbox_2/index.html
singleRxnDeletion	singleRxnDeletion Performs single reaction deletion analysis using FBA	http://opencobra.sourceforge.net/openCOBRA/opencobra_documentation/cobra_toolbox_2/index.html

APPENDIX-B

Scripts used for Data Mining

I. DRAGON Script : (condor_dragon_trainee.pl)

```
#!/usr/bin/perl -w
## We assume that all files are in the format Conformers_*sdf as
# downloaded from Pubchem. You can change this so that they pick all sdf
files
@allmegafiles=`ls -1 all.sdf`;
#process these files one at a time
while (@allmegafiles){
$megafile=shift(@allmegafiles);
#split the larger files to contain only 150 molecules at a time....
# This step uses mayachemtools, and a perlscript that splits SDF files.
#SplitSDFiles.pl should be on your path!
chomp($megafile);
$megafile=~s/\.sdf//;
system "/nfs/condor/mayachemtools/bin/SplitSDFiles.pl -m Cmpds --numcmpds 150
-r $megafile -o $megafile.sdf";
# read these files for descriptor calculations
@allfiles=`ls -1 *Part*.sdf`;
print @allfiles;
## The following loop is a script that was written for
## processing in lots of 8, now replaced with the condor submit script....
while(@allfiles){
for ($i=0; $i <= 7; $i++){
  $temp=shift(@allfiles);
#check if no value
unless($temp eq ''){push(@files,$temp)}
}
foreach $file (@files){
chomp($file);
$file=~s/\.sdf//;
print $file, "\n";
# copy file template
system "cp template.drs $file.drs";
system "perl -i -p -e s/filename/$file/ $file.drs";
open(CMD,">$file.cmd");
print CMD 'universe = vanilla',"\n";
#print CMD 'requirements = (Arch=="X86_64" || Arch=="INTEL") &&
OpSys=="LINUX"',"\n";
##print CMD 'requirements = Arch=="X86_64" && OpSys=="LINUX"',"\n";
##print CMD 'requirements = Arch=="INTEL" && OpSys=="LINUX"',"\n";
#print CMD 'requirements = machine=="dolphin.osdd.jnu.ac.in"',"\n";
print CMD 'requirements = machine=="lynn.osdd.jnu.ac.in" ||
machine=="dolphin.osdd.jnu.ac.in"',"\n";
print CMD "executable    = /nfs/condor/dragon_new/dragon6shell\n";
print CMD "output          = $file.stdout\n";
print CMD "error           = $file.err\n";
print CMD "log             = $file.log\n";
print CMD "arguments       = -s $file.drs\n";
print CMD "queue\n";
close(CMD);
```

II. Model Building Script:

```
%% Classification Modeling Script
%% Max Kuhn (max.kuhn@pfizer.com, mxkuhn@gmail.com)
%% Version: 1.00
%% Created on: 2010/10/02
%%
%% This is an Sweave template for building and describing
%% classification models. It mixes R and LaTeX code. The document can
  %% be processing using R's Sweave function to produce a tex file.
%%
%% The inputs are:
%% - the initial data set in a data frame called 'rawData'
%% - a factor column in the data set called 'class'. this should be the
%%    outcome variable
%% - all other columns in rawData should be predictor variables
%% - the type of model should be in a variable called 'modName'.
%%
%% The script attempts to make some intelligent choices based on the
%% model being used. For example, if modName is "pls", the script will
%% automatically center and scale the predictor data. There are
%% situations where these choices can (and should be) changed.
%%
%% There are other options that may make sense to change. For example,
%% the user may want to adjust the type of resampling. To find these
%% parts of the script, search on the string 'OPTION'. These parts of
%% the code will document the options.
\documentclass[12pt]{report}
\usepackage{amsmath}
\usepackage[pdftex]{graphicx}
\usepackage{color}
\usepackage{ctable}
\usepackage{xspace}
\usepackage{fancyvrb}
\usepackage{fancyhdr}
\usepackage{lastpage}
\usepackage{longtable}
\usepackage{algorithm2e}
\usepackage[
         colorlinks=true,
         linkcolor=blue,
         citecolor=blue,
         urlcolor=blue]
           {hyperref}
\usepackage{lscape}
\usepackage{Sweave}
\SweaveOpts{keep.source = TRUE}
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
\definecolor{darkgreen}{rgb}{0,0.6,0}
\definecolor{darkred}{rgb}{0.6,0.0,0}
\definecolor{lightbrown}{rgb}{1,0.9,0.8}
\definecolor{brown}{rgb}{0.6,0.3,0.3}
\definecolor{darkblue}{rgb}{0,0,0.8}
\definecolor{darkmagenta}{rgb}{0.5,0,0.5}
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
\newcommand{\bld}[1]{\mbox{\boldmath $#1$}}
\newcommand{\shell}[1]{\mbox{$#1$}}
\renewcommand{\vec}[1]{\mbox{\bf {#1}}}
```

```
\newcommand{\ReallySmallSpacing}{\renewcommand{\baselinestretch}{.6}\Large\no
rmalsize}
\newcommand{\SmallSpacing}{\renewcommand{\baselinestretch}{1.1}\Large\normals
ize}
\newcommand{\halfs}{\frac{1}{2}}
\setlength{\oddsidemargin}{-.25 truein}
\setlength{\evensidemargin}{0truein}
\setlength{\topmargin}{-0.2truein}
\setlength{\textwidth}{7 truein}
\setlength{\textheight}{8.5 truein}
\setlength{\parindent}{0.20truein}
\setlength{\parskip}{0.10truein}
\setcounter{LTchunksize}{50}
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
\pagestyle{fancy}
\lhead{}
%% OPTION Report header name
\chead{Classification Model Script}
\rhead{}
\lfoot{}
\cfoot{}
\rfoot{\thepage\ of \pageref{LastPage}}
\renewcommand{\headrulewidth}{1pt}
\renewcommand{\footrulewidth}{1pt}
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% OPTION Report title and modeler name
\title{Classification Model Script - Model building only using bag}
\author{Rishi Srivastava}
\begin{document}
\maketitle
\thispagestyle{empty}
<<dummy, eval=TRUE, echo=FALSE, results=hide>>=
# sets values for variables used later in the program to prevent the \Sexpr
error on parsing with Sweave
numSamples=''
classDistString=''
missingText=''
numPredictors=''
numPCAcomp=''
pcaText=''
nzvText=''
corrText=''
ppText=''
varText=''
splitText="Dummy Text"
nirText="Dummy Text"
# pctTrain is a variable that is initialised in Data splitting, and reused
later in testPred
pctTrain=0.8
@
<<startup, eval= TRUE, results = hide, echo = FALSE>>=
library(Hmisc)
library(caret)
versionTest <- compareVersion(packageDescription("caret")$Version,
                              "4.65")
if(versionTest < 0) stop("caret version 4.65 or later is required")

library(RColorBrewer)
```

```
listString <- function (x, period = FALSE, verbose = FALSE)
{
  if (verbose)   cat("\n      entering listString\n")
  flush.console()
  if (!is.character(x))
    x <- as.character(x)
  numElements <- length(x)
  out <- if (length(x) > 0) {
    switch(min(numElements, 3), x, paste(x, collapse = " and "),
           {
             x <- paste(x, c(rep(",", numElements - 2), " and", ""), sep =
""")
             paste(x, collapse = " ")
           })
  }
  else ""
  if (period)  out <- paste(out, ".", sep = "")
  if (verbose)  cat("      leaving  listString\n\n")
  flush.console()
  out
}
resampleStats <- function(x, digits = 3)
  {
    bestPerf <- x$bestTune
    colnames(bestPerf) <- gsub("^\\.", "", colnames(bestPerf))
    out <- merge(x$results, bestPerf)
    out <- out[, colnames(out) %in% x$perfNames]
    names(out) <- gsub("ROC", "area under the ROC curve", names(out), fixed =
TRUE)
    names(out) <- gsub("Sens", "sensitivity", names(out), fixed = TRUE)
    names(out) <- gsub("Spec", "specificity", names(out), fixed = TRUE)
    names(out) <- gsub("Accuracy", "overall accuracy", names(out), fixed =
TRUE)
    names(out) <- gsub("Kappa", "Kappa statistics", names(out), fixed = TRUE)
    out <- format(out, digits = digits)
    listString(paste(names(out), "was", out))
  }
twoClassNoProbs <- function (data, lev = NULL, model = NULL)
{
  out <- c(sensitivity(data[, "pred"], data[, "obs"], lev[1]),
           specificity(data[, "pred"], data[, "obs"], lev[2]),
           confusionMatrix(data[, "pred"], data[, "obs"])$overall["Kappa"])
  names(out) <- c("Sens", "Spec", "Kappa")
  out
}
##OPTION: model name: see ?train for more values/models
modName <- "PLS"
## No longer dealing with raw data - this contains the preprocessed training
dataset
load("/nfs/condor/caret/rishi/data/trainClass.RData")
load("/nfs/condor/caret/rishi/data/testClass.RData")
rawData <- trainX
rawData$outcome <- trainY
@
%% This is a test line - to find a way to comment text outside the code
chunk, which has inserted values from R variables. We are using the method:
\Sexpr{modName}.
\section*{Data Sets}\label{S:data}
```

```
%% OPTION: provide some background on the problem, the experimental
%% data, how the compounds were selected etc
<<getDataInfo, echo = FALSE, results = hide>>=
if(!any(names(rawData) == "outcome")) stop("a variable called outcome should
be in the data set")
if(!is.factor(rawData$outcome)) stop("the outcome should be a factor vector")
## OPTION: when there are only two classes, the first level of the
##         factor is used as the "positive" or "event" for calculating
##         sensitivity and specificity. Adjust the outcome factor
accordingly.
numClasses <- length(levels(rawData$outcome))
numSamples <- nrow(rawData)
numPredictors <- ncol(rawData) - 1
predictorNames <- names(rawData)[names(rawData) != "outcome"]
isNum <- apply(rawData[,predictorNames, drop = FALSE], 2, is.numeric)
if(any(!isNum)) stop("all predictors in rawData should be numeric")
classTextCheck <- all.equal(levels(rawData$outcome),
make.names(levels(rawData$outcome)))
if(!classTextCheck) warning("the class levels are not valid R variable names;
this may cause errors")
## Get the class distribution
classDist <- table(rawData$outcome)
classDistString <- paste("``",
                         names(classDist),
                         "'' ($n$=",
                         classDist,
                         ")",
                         sep = "")
classDistString <- listString(classDistString)
@
<<missingFilter, eval=FALSE, echo = FALSE, results = hide>>=
colRate <- apply(rawData[, predictorNames, drop = FALSE],
                 2, function(x) mean(is.na(x)))
##OPTION thresholds can be changed
colExclude <- colRate > .20
missingText <- ""
if(any(colExclude))
  {
    missingText <- paste(missingText,
                         ifelse(sum(colExclude) > 1,
                                " There were ",
                                " There was "),
                         sum(colExclude),
                         ifelse(sum(colExclude) > 1,
                                " predictors ",
                                " predictor "),
                         "with an excessive number of ",
                         "missing data. ",
                         ifelse(sum(colExclude) > 1,
                                " These were excluded. ",
                                " This was excluded. "))
    predictorNames <- predictorNames[!colExclude]
    rawData <- rawData[, names(rawData) %in% c("outcome", predictorNames),
drop = FALSE]
  }
rowRate <- apply(rawData[, predictorNames, drop = FALSE],
                 1, function(x) mean(is.na(x)))
rowExclude <- rowRate > .20
```

```
if(any(rowExclude)) {
    missingText <- paste(missingText,
                         ifelse(sum(rowExclude) > 1,
                                " There were ",
                                " There was "),
                         sum(colExclude),
                         ifelse(sum(rowExclude) > 1,
                                " samples ",
                                " sample "),
                         "with an excessive number of ",
                         "missing data. ",
                         ifelse(sum(rowExclude) > 1,
                                " These were excluded. ",
                                " This was excluded. "),
                         "After filtering, ",
                         sum(!rowExclude),
                         " samples remained.")
    rawData <- rawData[!rowExclude, ]
    hasMissing <- apply(rawData[, predictorNames, drop = FALSE],
                        1, function(x) mean(is.na(x)))
  } else {
        hasMissing <- apply(rawData[, predictorNames, drop = FALSE],
                            1, function(x) any(is.na(x)))
        missingText <- paste(missingText,
                             ifelse(missingText == "",
                                "There ",
                                "Subsequently, there "),
                             ifelse(sum(hasMissing) == 1,
                                    "was ",
                                    "were "),
                             ifelse(sum(hasMissing) > 0,
                                    sum(hasMissing),
                                    "no"),
                             ifelse(sum(hasMissing) == 1,
                                    "sample ",
                                    "samples "),
                             "with missing values.")
  }
@
The initial data set consisted of \Sexpr{numSamples} samples and
\Sexpr{numPredictors} predictor variables. The breakdown of the
outcome data classes were: \Sexpr{classDistString}.
%% \Sexpr{missingText}
<<pca, eval=FALSE, echo = FALSE, results = hide>>=
predictors <- rawData[, predictorNames, drop = FALSE]
## PCA will fail with predictors having less than 2 unique values
isZeroVar <- apply(predictors, 2,
                   function(x) length(unique(x)) < 2)
if(any(isZeroVar)) predictors <- predictors[, !isZeroVar, drop = FALSE]
## For whatever, only the formula interface to prcomp
## handles missing values
pcaForm <- as.formula(
                      paste("~",
                            paste(names(predictors), collapse = "+")))
pca <- prcomp(pcaForm,
              data = predictors,
              center = TRUE,
              scale. = TRUE,
```

```
              na.action = na.omit)
## OPTION: the number of components plotted/discussed can be set
numPCAcomp <- 3
pctVar <- pca$sdev^2/sum(pca$sdev^2)*100
pcaText <- paste(round(pctVar[1:numPCAcomp], 1),
                 "\\\\%",
                 sep = "")
pcaText <- listString(pcaText)
@
%% To get an initial assessment of the separability of the classes,
%% principal component analysis (PCA) was used to distill the
%% \Sexpr{numPredictors} predictors down into \Sexpr{numPCAcomp}
%% surrogate variables (i.e. the principal components) in a manner that
%% attempts to maximize the amount of information preserved from the
%% original predictor set. Figure \ref{F:inititalPCA} contains plots of
%% the first \Sexpr{numPCAcomp} components, which accounted for
%% \Sexpr{pcaText} percent of the variability in the original predictors
%% (respectively).
%% OPTION: remark on how well (or poorly) the data separated
%% \setkeys{Gin}{width = 0.8\textwidth}
%% \begin{figure}[p]
%%  \begin{center}
<<pcaPlot, eval = FALSE, echo = FALSE, results = hide, fig = TRUE, width = 8,
height = 8>>=
trellis.par.set(caretTheme(), warn = TRUE)
if(numPCAcomp == 2)
  {
    axisRange <- extendrange(pca$x[, 1:2])
    print(
          xyplot(PC1 ~ PC2,
                 data = as.data.frame(pca$x),
                 type = c("p", "g"),
                 groups = rawData$outcome,
                 auto.key = list(columns = 2),
                 xlim = axisRange,
                 ylim = axisRange))
  } else {
    axisRange <- extendrange(pca$x[, 1:numPCAcomp])
    print(
          splom(~as.data.frame(pca$x)[, 1:numPCAcomp],
                type = c("p", "g"),
                groups = rawData$outcome,
                auto.key = list(columns = 2),
                as.table = TRUE,
                prepanel.limits = function(x) axisRange
                ))
      }
@
%%    \caption[PCA Plot]{A plot of the first \Sexpr{numPCAcomp}
%%      principal components for the original data set.}
%%    \label{F:inititalPCA}
%%  \end{center}
%% \end{figure}
<<initialDataSplit, eval = FALSE,  results = hide, echo = FALSE>>=
  ## OPTION: in small samples sizes, you may not want to set aside a
  ## training set and focus on the resampling results.
  pctTrain <- 1
if(pctTrain < 1)
```

```
  {
    ## OPTION: seed number can be changed
    set.seed(1)
    inTrain <- createDataPartition(rawData$outcome,
                                   p = pctTrain,
                                   list = FALSE)
    trainX <- rawData[ inTrain, predictorNames]
    testX  <- rawData[-inTrain, predictorNames]
    trainY <- rawData[ inTrain, "outcome"]
    testY  <- rawData[-inTrain, "outcome"]
    splitText <- paste("The original data were split into ",
                       "a training set ($n$=",
                       nrow(trainX),
                       ") and a test set ($n$=",
                       nrow(testX),
                       ") in a manner that preserved the ",
                       "distribution of the classes.",
                       sep = "")
    isZeroVar <- apply(trainX, 2,
                       function(x) length(unique(x)) < 2)
    if(any(isZeroVar))
      {
        trainX <- trainX[, !isZeroVar, drop = FALSE]
        testX <- testX[, !isZeroVar, drop = FALSE]
      }
  } else {
    trainX <- rawData[, predictorNames]
    testX  <- NULL
    trainY <- rawData[, "outcome"]
    testY  <- NULL
    splitText <- "The entire data set was used as the training set."
  }
trainDist <- table(trainY)
nir <- max(trainDist)/length(trainY)*100
niClass <- names(trainDist)[which.max(trainDist)]
nirText <- paste("The non--information rate is the accuracy that can be ",
                 "achieved by predicting all samples using the most ",
                 "dominant class. For these data, the rate is ",
                 round(nir, 2), "\\\\% using the ``",
                 niClass,
                 "'' class.",
                 sep = "")
@
%% \Sexpr{splitText}
%% \Sexpr{nirText}
<<nzv, eval=FALSE, results = hide, echo = FALSE>>=
## OPTION: other pre-processing steps can be used
ppSteps <- caret:::suggestions(modName)
set.seed(2)
if(ppSteps["nzv"])
  {
    nzv <- nearZeroVar(trainX)
    if(length(nzv) > 0)
      {
        nzvVars <- names(trainX)[nzv]
        trainX <- trainX[, -nzv]
        nzvText <- paste("There were ",
                         length(nzv),
```

```
                        " predictors that were removed due to",
                        " severely unbalanced distributions that",
                        " could negatively affect the model fit",
                        ifelse(length(nzv) > 10,
                               ".",
                               paste(": ",
                                     listString(nzvVars),
                                     ".",
                                     sep = "")),
                        sep = "")
        if(pctTrain < 1) testX <- testX[, -nzv]
      } else nzvText <- ""
  } else nzvText <- ""
@
<<corrFilter, eval = FALSE, results = hide, echo = FALSE>>=
if(ppSteps["corr"])
  {
    ## OPTION:
    corrThresh <- .75
    highCorr <- findCorrelation(cor(trainX, use = "pairwise.complete.obs"),
                                corrThresh)
    if(length(highCorr) > 0)
      {
        corrVars <- names(trainX)[highCorr]
        trainX <- trainX[, -highCorr]
        corrText <- paste("There were ",
                        length(highCorr),
                        " predictors that were removed due to",
                        " large between--predictor correlations that",
                        " could negatively affect the model fit",
                        ifelse(length(highCorr) > 10,
                               ".",
                               paste(": ",
                                     listString(highCorr),
                                     ".",
                                     sep = "")),
                         " Removing these predictors forced",
                         " all pair--wise correlations to be",
                         " less than ",
                         corrThresh,
                         ".",
                         sep = "")
        if(pctTrain < 1) testX <- testX[, -highCorr]
      } else corrText <- ""
  }else corrText <- ""
@
<<preProc, eval = FALSE, echo = FALSE, results = hide>>=
ppMethods <- NULL
if(ppSteps["center"]) ppMethods <- c(ppMethods, "center")
if(ppSteps["scale"]) ppMethods <- c(ppMethods, "scale")
if(any(hasMissing) > 0) ppMethods <- c(ppMethods, "knnImpute")
##OPTION other methods, such as spatial sign, can be added to this list
if(length(ppMethods) > 0)
  {
    ppInfo <- preProcess(trainX, method = ppMethods)
    trainX <- predict(ppInfo, trainX)
    if(pctTrain < 1) testX <- predict(ppInfo, testX)
    ppText <- paste("The following pre--processing methods were",
```

```
                    " applied to the training",
                    ifelse(pctTrain < 1, " and test", ""),
                    " data: ",
                    listString(ppMethods),
                    ".",
                    sep = "")
    ppText <- gsub("center", "mean centering", ppText)
    ppText <- gsub("scale", "scaling to unit variance", ppText)
    ppText <- gsub("knnImpute",
                   paste(ppInfo$k, "--nearest neighbor imputation", sep =
"",
                   ppText)
    ppText <- gsub("spatialSign", "the spatial sign transformation", ppText)
    ppText <- gsub("pca", "principal component feature extraction", ppText)
    ppText <- gsub("ica", "independent component feature extraction", ppText)
    } else {
      ppInfo <- NULL
      ppText <- ""
    }
predictorNames <- names(trainX)
if(nzvText != "" | corrText != "" | ppText != "")
  {
    varText <- paste("After pre--processing, ",
                     ncol(trainX),
                     "predictors remained for modeling.")
  } else varText <- ""
@
%% \Sexpr{nzvText} \Sexpr{corrText} \Sexpr{ppText} \Sexpr{varText}
\clearpage
\section*{Model Building}
<<setupWorkers, echo = FALSE, results = tex>>=
numWorkers <- 1
##OPTION: turn up numWorkers to use MPI
if(numWorkers > 1)
  {
    mpiCalcs <- function(X, FUN, ...)
      {
        theDots <- list(...)
        parLapply(theDots$cl, X, FUN)
      }
    library(snow)
    cl <- makeCluster(numWorkers, "MPI")
  }
@
<<setupResampling, echo = FALSE, results = hide>>=
##OPTION: the resampling options can be changed. See
##        ?trainControl for details
resampName <- "boot632"
resampNumber <- 10
numRepeat <- 1
resampP <- .75
modelInfo <- modelLookup(modName)
if(numClasses == 2)
  {
    foo <- if(any(modelInfo$probModel)) twoClassSummary else twoClassNoProbs
  } else foo <- defaultSummary
set.seed(3)
ctlObj <- trainControl(method = resampName,
```

```
                        number = resampNumber,
                        repeats = numRepeat,
                        p = resampP,
                        classProbs = any(modelInfo$probModel),
                        summaryFunction = foo)
##OPTION select other performance metrics as needed
optMetric <- if(numClasses == 2 & any(modelInfo$probModel)) "ROC" else
"Kappa"
if(numWorkers > 1)
  {
    ctlObj$workers <- numWorkers
    ctlObj$computeFunction <- mpiCalcs
    ctlObj$computeArgs <- list(cl = cl)
  }
@
<<setupGrid, results = hide, echo = FALSE>>=
##OPTION expand or contract these grids as needed (or
##       add more models
gridSize <- 3
if(modName %in% c("svmPoly", "svmRadial", "svmLinear", "lvq", "ctree2",
"ctree")) gridSize <- 5
if(modName %in% c("earth", "fda")) gridSize <- 7
if(modName %in% c("knn", "rocc", "glmboost", "rf", "nodeHarvest")) gridSize
<- 10
if(modName %in% c("nb")) gridSize <- 2
if(modName %in% c("pam", "rpart")) gridSize <- 15
if(modName %in% c("pls")) gridSize <- min(20, ncol(trainX))
if(modName == "gbm")
  {
    tGrid <- expand.grid(.interaction.depth = -1 + (1:5)*2 ,
                         .n.trees = (1:10)*20,
                         .shrinkage = .1)
  }
if(modName == "nnet")
  {
    tGrid <- expand.grid(.size = -1 + (1:5)*2 ,
                         .decay = c(0, .001, .01, .1))
  }
@
<<fitModel, results = hide, echo = FALSE, eval = TRUE>>=
##OPTION alter as needed
set.seed(4)
modelFit <- switch(modName,
                   gbm =
                   {
                     mix <- sample(seq(along = trainY))
                     train(
                           trainX[mix,], trainY[mix], modName,
                           verbose = FALSE,
                           bag.fraction = .9,
                           metric = optMetric,
                           trControl = ctlObj,
                           tuneGrid = tGrid)
                   },
                   multinom =
                   {
                     train(
                           trainX, trainY, modName,
```

```
                        trace = FALSE,
                        metric = optMetric,
                        maxiter = 1000,
                        MaxNWts = 5000,
                        trControl = ctlObj,
                        tuneLength = gridSize)
                  },
                  nnet =
                  {
                    train(
                        trainX, trainY, modName,
                        metric = optMetric,
                        linout = FALSE,
                        trace = FALSE,
                        maxiter = 1000,
                        MaxNWts = 5000,
                        trControl = ctlObj,
                        tuneGrid = tGrid)
                  },
                  svmRadial =, svmPoly =, svmLinear =
                  {
                    train(
                        trainX, trainY, modName,
                        metric = optMetric,
                        scaled = TRUE,
                        trControl = ctlObj,
                        tuneLength = gridSize)
                  },
                  {
                    train(trainX, trainY, modName,
                        trControl = ctlObj,
                        metric = optMetric,
                        tuneLength = gridSize)
                  })
@
<<modelDescr, echo = FALSE, results = hide>>=
summaryText <- ""
resampleName <- switch(tolower(modelFit$control$method),
                       boot = paste("the bootstrap (",
length(modelFit$control$index), " reps)", sep = ""),
                       boot632 = paste("the bootstrap 632 rule (",
length(modelFit$control$index), " reps)", sep = ""),
                       cv = paste("cross-validation (",
modelFit$control$number, " fold)", sep = ""),
                       repeatedcv = paste("cross-validation (",
modelFit$control$number, " fold, repeated ",
                         modelFit$control$repeats, " times)", sep = ""),
                       lgocv = paste("repeated train/test splits (",
length(modelFit$control$index), " reps, ",
                         round(modelFit$control$p, 2), "$\\%$)", sep = ""))
tuneVars <- latexTranslate(tolower(modelInfo$label))
tuneVars <- gsub("\\#", "the number of ", tuneVars, fixed = TRUE)
if(ncol(modelFit$bestTune) == 1 && colnames(modelFit$bestTune) ==
".parameter")
  {
    summaryText <- paste(summaryText,
                         "\n\n",
```

```
                          "There are no tuning parameters associated with this
model.",
                          "To characterize the model performance on the
training set,",
                          resampleName,
                          "was used.",
                          "Table \\\\ref{T:resamps} and Figure
\\\\ref{F:profile}",
                          "show summaries of the resampling results. ")
  } else {
    summaryText <- paste("There",
                          ifelse(nrow(modelInfo) > 1, "are", "is"),
                          nrow(modelInfo),
                          ifelse(nrow(modelInfo) > 1, "tuning parameters",
"tuning parameter"),
                          "associated with this model:",
                          listString(tuneVars, period = TRUE))
    paramNames <- gsub(".", "", names(modelFit$bestTune), fixed = TRUE)
    for(i in seq(along = paramNames))
      {
        check <- modelInfo$parameter %in% paramNames[i]
        if(any(check))
          {
            paramNames[i] <- modelInfo$label[which(check)]
          }
      }
    paramNames <- gsub("#", "the number of ", paramNames, fixed = TRUE)
    ## Check to see if there was only one combination fit
    summaryText <- paste(summaryText,
                          "To choose",
                          ifelse(nrow(modelInfo) > 1,
                                 "appropriate values of the tuning
parameters,",
                                 "an appropriate value of the tuning
parameter,"),
                          resampleName,
                          "was used to generated a profile of performance
across the",
                          nrow(modelFit$results),
                          ifelse(nrow(modelInfo) > 1,
                                 "combinations of the tuning parameters.",
                                 "candidate values."),
                          "Table \\\\ref{T:resamps} and Figure
\\\\ref{F:profile} show",
                          "summaries of the resampling profile. ",
"The final model fitted to the entire training set was:",

listString(paste(latexTranslate(tolower(paramNames)), "=",
modelFit$bestTune[1,]), period = TRUE))
  }
@
\Sexpr{summaryText}
<<resampTable, echo = FALSE, results = tex>>=
tableData <- modelFit$results
if(all(modelInfo$parameter == "parameter"))
  {
    tableData <- tableData[,-1, drop = FALSE]
```

```
    colNums <- c( length(modelFit$perfNames), length(modelFit$perfNames),
length(modelFit$perfNames))
    colLabels <- c("Mean", "Standard Deviation","Apparant")
    constString <- ""
    isConst <- NULL
  } else {
    isConst <- apply(tableData[, modelInfo$parameter, drop = FALSE],
                     2,
                     function(x) length(unique(x)) == 1)
    numParamInTable <- sum(!isConst)
    if(any(isConst))
      {
        constParam <- modelInfo$parameter[isConst]
        constValues <- format(tableData[, constParam, drop = FALSE], digits =
4)[1,,drop = FALSE]
        tableData <- tableData[, !(names(tableData) %in% constParam), drop =
FALSE]
        constString <- paste("The tuning",
                             ifelse(sum(isConst) > 1,
                                    "parmeters",
                                    "parameter"),
                             listString(paste("``", names(constValues), "'",
sep = "")),
                             ifelse(sum(isConst) > 1,
                                    "were",
                                    "was"),
                             "held constant at",
                             ifelse(sum(isConst) > 1,
                                    "a value of",
                                    "values of"),
                             listString(constValues[1,]))
      } else constString <- ""
    cn <- colnames(tableData)
    for(i in seq(along = cn))
      {
        check <- modelInfo$parameter %in% cn[i]
        if(any(check))
          {
            cn[i] <- modelInfo$label[which(check)]
          }
      }
    colnames(tableData) <- cn
    colNums <- c(numParamInTable,
                 length(modelFit$perfNames),
                 length(modelFit$perfNames),
                 length(modelFit$perfNames))
    colLabels <- c("", "Mean", "Standard Deviation", "Apparant")
  }
colnames(tableData) <- gsub("SD$", "", colnames(tableData))
colnames(tableData) <- gsub("Apparent$", "", colnames(tableData))
colnames(tableData) <- latexTranslate(colnames(tableData))
rownames(tableData) <- latexTranslate(rownames(tableData))
latex(tableData,
      rowname = NULL,
      file = "",
      cgroup = colLabels,
      n.cgroup = colNums,
      where = "h!",
```

```
      digits = 4,
      longtable = nrow(tableData) > 30,
      caption = paste(resampleName, "results from the model fit.",
constString),
      label = "T:resamps")
@
\setkeys{Gin}{ width = 0.9\textwidth}
\begin{figure}[b]
  \begin{center}
<<profilePlot, echo = FALSE, fig = TRUE, width = 8, height = 6>>=
  trellis.par.set(caretTheme(), warn = TRUE)
if(all(modelInfo$parameter == "parameter") | all(isConst) | modName == "nb")
  {
    resultsPlot <- resampleHist(modelFit)
    plotCaption <- paste("Distributions of model performance from the ",
                         "training set estimated using ",
                         resampleName)
  } else {
    if(modName %in% c("svmPoly", "svmRadial", "svmLinear"))
      {
        resultsPlot <- plot(modelFit,
                            metric = optMetric,
                            xTrans = function(x) log10(x))
        resultsPlot <- update(resultsPlot,
                              type = c("g", "p", "l"),
                              ylab = paste(optMetric, " (", resampleName,
")", sep = ""))
      } else {
        resultsPlot <- plot(modelFit,
                            metric = optMetric)
        resultsPlot <- update(resultsPlot,
                              type = c("g", "p", "l"),
                              ylab = paste(optMetric, " (", resampleName,
")", sep = ""))
      }
   plotCaption <- paste("A plot of the estimates of the",
                        optMetric,
                        "values calculated using",
                        resampleName)
  }
print(resultsPlot)
@
   \caption[Performance Plot]{\Sexpr{plotCaption}.}
    \label{F:profile}
  \end{center}
\end{figure}
<<stopWorkers, echo = FALSE, results = hide>>=
if(numWorkers > 1) stopCluster(cl)
@
<<testPred, results = tex, echo = FALSE>>=
  if(pctTrain < 1)
  {
    cat("\\clearpage\n\\section*{Test Set Results}\n\n")
    classPred <- predict(modelFit, testX)
    cm <- confusionMatrix(classPred, testY)
    values <- cm$overall[c("Accuracy", "Kappa", "AccuracyPValue",
"McnemarPValue")]
    values <- values[!is.na(values) & !is.nan(values)]
```

```
    values <- c(format(values[1:2], digits = 3),
                format.pval(values[-(1:2)], digits = 5))
    nms <- c("the overall accuracy", "the Kappa statistic",
                         "the $p$--value that accuracy is greater than the no--
information rate",
                         "the $p$--value of concordance from McNemar's test")
    nms <- nms[seq(along = values)]
    names(values) <- nms
    if(any(modelInfo$probModel))
      {
        classProbs <- extractProb(list(fit = modelFit),
                                  testX = testX,
                                  testY = testY)
        classProbs <- subset(classProbs, dataType == "Test")
        if(numClasses == 2)
          {
            tmp <- twoClassSummary(classProbs, lev = levels(classProbs$obs))
            tmp <- c(format(tmp, digits = 3))
            names(tmp) <- c("the sensitivity", "the specificity",
                            "the area under the ROC curve")
            values <- c(values, tmp)
          }
        probPlot <- plotClassProbs(classProbs)
      }
    testString <- paste("Based on the test set of",
                         nrow(testX),
                         "samples,",
                         listString(paste(names(values), "was", values),
period = TRUE),
                         "The confusion matrix for the test set is shown in
Table",
                         "\\\\ref{T:cm}.")
    testString <- paste(testString,
                         " Using ", resampleName,
                         ", the training set estimates were ",
                         resampleStats(modelFit),
                         ".",
                         sep = "")
    if(any(modelInfo$probModel)) testString <- paste(testString,
                                                     "Histograms of the class
probabilities",
                                                     "for the test set
samples are shown in",
                                                     "Figure
\\\\ref{F:probs}",
                                                     ifelse(numClasses == 2,
                                                            " and the test
set ROC curve is in Figure \\\\ref{F:roc}.",
                                                            "."))
    latex(cm$table,
          title = "",
          file = "",
          where = "h",
          cgroup = "Observed Values",
          n.cgroup = numClasses,
          caption = "The confusion matrix for the test set",
          label = "T:cm")
  } else testString <- ""
```

```
@
\Sexpr{testString}
<<classProbsTex, results = tex, echo = FALSE>>=
  if(any(modelInfo$probModel))
  {
    cat(
        paste("\\begin{figure}[p]\n",
              "\\begin{center}\n",
              "\\includegraphics{classProbs}",
              "\\caption[PCA Plot]{Class probabilities",
              "for the test set. Each panel contains ",
            "separate classes}\n",
              "\\label{F:probs}\n",
              "\\end{center}\n",
              "\\end{figure}"))
  }
  if(any(modelInfo$probModel) & numClasses == 2)
  {
    cat(
        paste("\\begin{figure}[p]\n",
              "\\begin{center}\n",
              "\\includegraphics[clip, width = .8\\textwidth]{roc}",
              "\\caption[ROC Plot]{ROC Curve",
              "for the test set.}\n",
              "\\label{F:roc}\n",
              "\\end{center}\n",
              "\\end{figure}"))
  }
@
<<classProbsTex, results = hide, echo = FALSE>>=
  if(any(modelInfo$probModel))
  {
    pdf("classProbs.pdf", height = 7, width = 7)
    trellis.par.set(caretTheme(), warn = FALSE)
    print(probPlot)
    dev.off()
  }
  if(any(modelInfo$probModel) & numClasses == 2)
  { resPonse<-testY
    preDictor<-classProbs[, levels(trainY)[1]]
    pdf("roc.pdf", height = 8, width = 8)
# from pROC example at http://web.expasy.org/pROC/screenshots.htm
    plot.roc(resPonse, preDictor, # data
         percent=TRUE, # show all values in percent
         partial.auc=c(100, 90), partial.auc.correct=TRUE, # define a partial
AUC (pAUC)
         print.auc=TRUE, #display pAUC value on the plot with following
options:
         print.auc.pattern="Corrected pAUC (100-90%% SP):\n%.1f%%",
print.auc.col="#1c61b6",
         auc.polygon=TRUE, auc.polygon.col="#1c61b6", # show pAUC as a
polygon
         max.auc.polygon=TRUE,     max.auc.polygon.col="#1c61b622", # also
show the 100% polygon
         main="Partial AUC (pAUC)")
    plot.roc(resPonse, preDictor,
         percent=TRUE, add=TRUE, type="n", # add to plot, but don't re-add
the ROC itself (useless)
```

```
         partial.auc=c(100, 90), partial.auc.correct=TRUE,
         partial.auc.focus="se", # focus pAUC on the sensitivity
         print.auc=TRUE, print.auc.pattern="Corrected pAUC (100-90%%
SE):\n%.1f%%", print.auc.col="#008600",
         print.auc.y=40, # do not print auc over the previous one
         auc.polygon=TRUE, auc.polygon.col="#008600",
         max.auc.polygon=TRUE, max.auc.polygon.col="#00860022")
    dev.off()
  }
# commenting old roc commands
#  {
#    rocPoints <- as.data.frame(
#                                roc(classProbs[, levels(trainY)[1]],
#                                    testY,
#                                    positive =  levels(trainY)[1]))
#    pdf("roc.pdf", height = 8, width = 8)
#    plot(1 - rocPoints$specificity, rocPoints$sensitivity,
#         ylab = "Sensitivity", xlab = "1 - Specificity",
#         type = "n",
#         main = paste("AUC:", round(aucRoc(rocPoints), 3)))
#    abline(0, 1, lty = 2, col = "darkgrey")
#    points(1 - rocPoints$specificity, rocPoints$sensitivity,
#         type = "S")
#
#    dev.off()
#  }
@
\section*{Versions}
<<versions, echo = FALSE, results = tex>>=
toLatex(sessionInfo())
@
<<save-data, echo = FALSE, results = hide>>=
## change this to the name of modName....
plsFit<-modelFit
save(plsFit,file="plsFit.RData")
@
The model was built using Partial Least Squares and is saved as plsFit.RData
for reuse. This contains the variable plsFit.
\end{document}
```