PREDICTING PROTEIN STABILITY CHANGE UPON SINGLE POINT MUTATION USING MULTI-INSTANCE REGRESSION: A LOCAL CONFORMATIONAL ANALYSIS APPROACH

by

AKUL DEWAN

(Under the Direction of KHALED M. RASHEED)

ABSTRACT

The prediction of stability change caused by a mutation in a protein structure is of vital importance for protein design and analysis. Several attempts have been made to predict these energy changes by analyzing the global conformational properties of a structure. To date, none of the research has focused solely on studying the effect of local conformational properties of a mutated residue to the final stability change. In my thesis I use *multi-instance regression learning with output aggregation* to learn and predict the energy change using the information from the local environment of the mutated residue. This research shows a high degree of correlation between the expected and predicted values of energy changes and a quantum leap from the current state-of-the-art.

INDEX WORDS: machine learning, protein stability, conformational analysis

PREDICTING PROTEIN STABILITY CHANGE UPON SINGLE POINT MUTATION USING MULTI-INSTANCE REGRESSION: A LOCAL CONFORMATIONAL ANALYSIS APPROACH

by

AKUL DEWAN

B.Tech, Uttar Pradesh Technical University, India, May 2011

A Thesis Submitted to the Graduate Faculty of The University of Georgia in Partial Fulfillment of the Requirements for the Degree

MASTER OF SCIENCE

ATHENS, GEORGIA

PREDICTING PROTEIN STABILITY CHANGE UPON SINGLE POINT MUTATION USING MULTI-INSTANCE REGRESSION: A LOCAL CONFORMATIONAL ANALYSIS APPROACH

by

AKUL DEWAN

Major Professor:	KHALED M. RASHEED
Committee:	ZACHARY WOOD
	WALTER D. POTTER

Electronic Version Approved:

Julie Coffield

Interim Dean of the Graduate School

The University of Georgia

DECEMBER 2014

DEDICATION

Dedicated to my late grandparents.

ACKNOWLEDGEMENTS

This work has been done with an intent to give my bit to the society. Machine Learning is a power field; nothing can be more fruitful than to use this for welfare of society. All over the world millions of dollars are spent to decipher the stability and structure of proteins; unfortunately, less has been discovered so far. Study of proteins can unlock door to a better diagnosis and treatment of many life threatening diseases including Cancer.

This work would not have made it to this stage without constant encouragement, support and guidance from Dr. Khaled M. Rasheed, Dr. Zachary Wood and Dr. W. Don Potter. I would like to thank Dr. Khaled M. Rasheed for introducing me to the amazing field of Machine Learning and allowing me be a part of several important projects throughout my course of study at UGA. I also thank Dr. Zachary Wood for his undeterred support, his faith in my work and most importantly his willingness to go out of his way and teach me fundamentals of Biochemistry and Molecular Biology.

Love, care and support of my family and friends have been my pillars of strength. I cannot thank them enough. A special thanks to Somenath Das for reviewing my thesis work.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Proteins are essential component of cellular machinery. Mutations in proteins occur due to environmental stresses, errors during cell replication, or disease states like cancer. A mutation can adversely affect protein function, leading to a disease state. It is therefore crucial to study the factors that affect the outcomes of protein mutations. In this work, we look into several conformational and chemical properties of protein structure and provide an in-depth analysis of their contribution to protein stability.

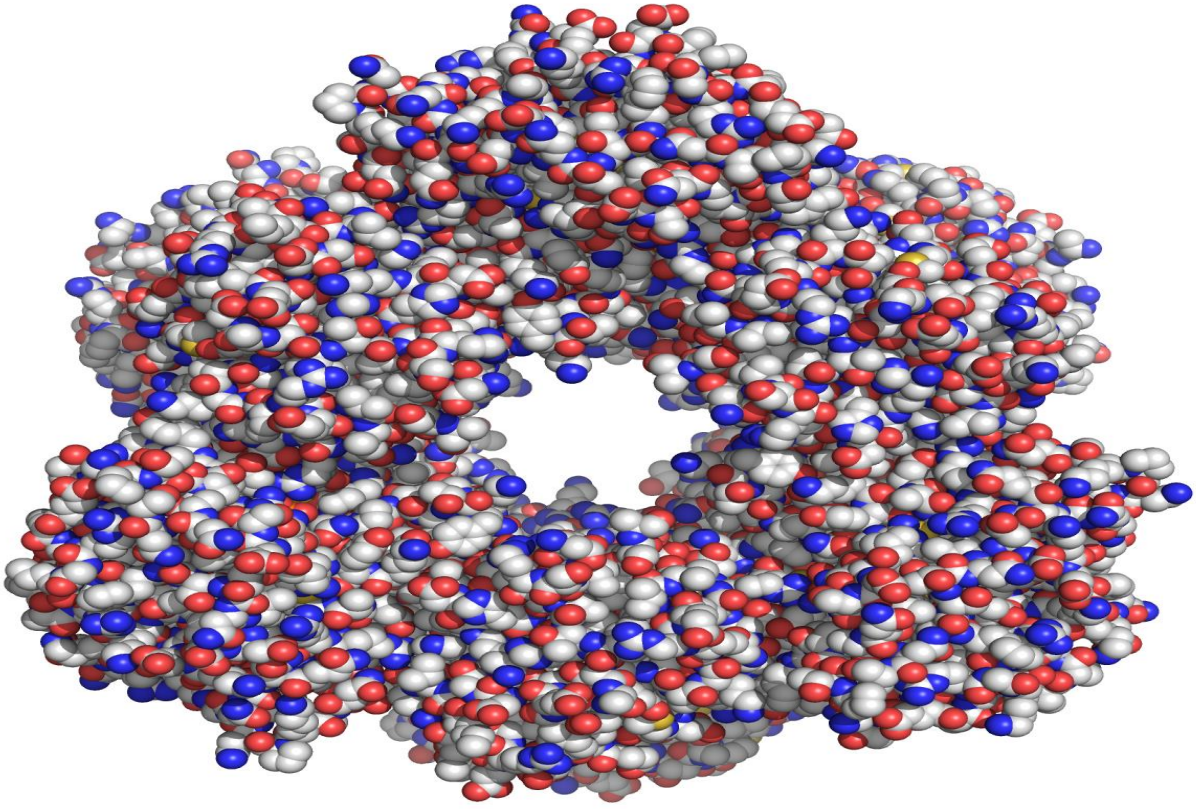

Figure 1: Three-dimensional structure of a protein.

Figure 1 shows the crystal structure of a protein. The multi-colored balls represent the atoms that make up the protein.

A protein is a hetero-polymer built from a set of 20 amino-acid residues arranged in a defined sequence that is specific for a given protein. Figure 1 shows the X-ray crystal structure of an actual protein, wherein balls represent the Van der Waals radius of individual atoms. A mutation in a protein is done at residue level, i.e. mutations are carried out by replacing a residue with another residue.

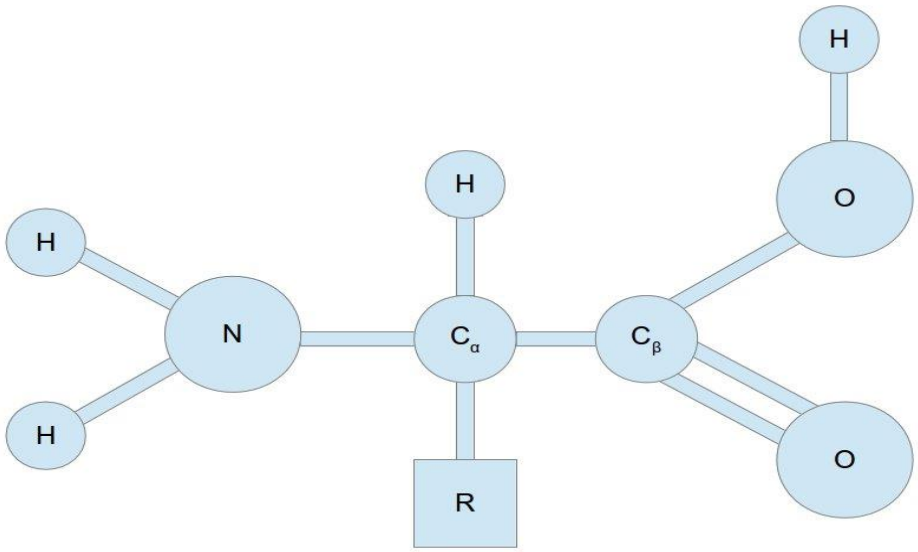

Figure 2: The basic structure of an amino acid residue

Figure 2 shows a general structure of an amino acid residue. Each residue consists of N, C, H and O group (that basically constitutes the amino-acid). R is a variable side-chain group. There are 20 distinct R-groups that are joined in a specific sequence to form a 'peptide chain' that folds into the structure of the protein.

Every amino-acid contains a common backbone made up of α-amino acid and a side-chain group named the R group. The distinctive properties of amino acids stems from the side-chain connected to C-α atom. Four major classes of amino-acids can be derived by variation of the side-chain group - namely, small hydrophobics, aromatics, polar and charged (acidic and basic residues). Consider for example, a polar side-chain (such as an amide or alcohol) is added to an amino-acid. A polarity would be induced, as an effect, in the amino-acid with a potency directly proportional to the side-chain acid group's strength. The presence of R group also plays a crucial role when a protein folds. Polar groups will require hydrogen bonding with either solvent or another polar group. Similarly, charged groups will require dipole-charge interactions with solvent or will form ionic bonds with oppositely charged amino acids.

Motivation

Noble prize winner Anfinsen's thermodynamic hypothesis states that under constant physiological conditions, the protein population converges to a structure that represents a minimum in Gibbs free energy called the native state(Rose, Fleming, Banavar, & Maritan, 2006). This means that a protein chain of amino acids spontaneously undergoes a conformational change that minimizes its chemical potential. The focus of this research is to learn what conformational and chemical features affect the stability of the native state. We have developed a

computational tool that can predict the change in stability upon mutation of a residue. This will provide biologists and biophysicists a much needed tool in which they only have to give a protein name and the mutation they are interested in analyzing (along with pH of solution in which mutation should be carried) in order to predict changes in stability. This is significant because many important proteins are difficult to study in solution, and a computational tool would greatly speed up research at a fraction of the cost and labor.

Previous Work

Several online tools are available which provide mutation stability prediction for proteins. Previous successful approaches can be divided into two categories - energy based methods and machine learning based methods. The Energy Based Methods attempt to generate an energy function that uses physical, statistical or empirical force field changes to predict change in energy upon mutation. (Bordner & Abagyan, 2004; Dehouck, Kwasigroch, Gilis, & Rooman, 2011; Gilis & Rooman, 1997; Guerois, Nielsen, & Serrano, 2002) are some of the prominent works.

Machine learning has been implemented extensively in this domain. The earliest work was done by (Capriotti, Fariselli, & Casadio, 2004); this is a neural network based approach which uses DSSP(Joosten et al., 2011) for generating the residue accessibility surface, and a set of 40 feature; wherein 20 features are used to represent (in a binary encoded form) the presence of each amino-acid near the mutated residue and 20 features represent the addition or removal of amino-acid. A similar idea of coding amino-acid presence was implemented in (Cheng, Randall, & Baldi, 2006a; and Ozen, Gönen, Alpaydan, & Haliloğlu, 2009) with the major difference in

the heuristic used to select nearest neighbors; the former is an application of SVM to this domain. It investigates 2 heuristics to find nearest neighbors - one in which 9Å radius is used around mutated residue and the other in which sequence of residues are considered; 9Å heuristic is reported as the better approach. (Ozen et al., 2009) also used 9Å as cut-off to gather nearest neighbors. It also used PAM250 (Dayhoff & Schwartz, 1978) for scoring the likelihood of stability on replacement of amino-acid.

The remainder of the thesis is organized as follows. In Chapter 2, we explain the methodology of our analysis. Chapter 2 would explicate the link between the different modules of our system. Specifically, it would explain how exactly the 3-dimensional structural information of a protein is converted into a machine learnable dataset. We developed a novel analysis algorithm named AARC (Algorithm for Analysis of Residual Conformations). In Chapter 3, a detailed explanation of AARC is given. In Chapter 4, we give details of our machine learning approach. Here again we developed MIR-OA (Multi-Instance Regression with Output Aggregation), a machine learning technique (to the best of our knowledge) never implemented in this domain of research. Lastly, Chapter 5 presents the Experimental Results and Conclusion.

CHAPTER 2

METHODOLOGY

This chapter is divided into 3 parts namely “Dataset Collection”, “Feature Generation” and “Machine Learning Module”.

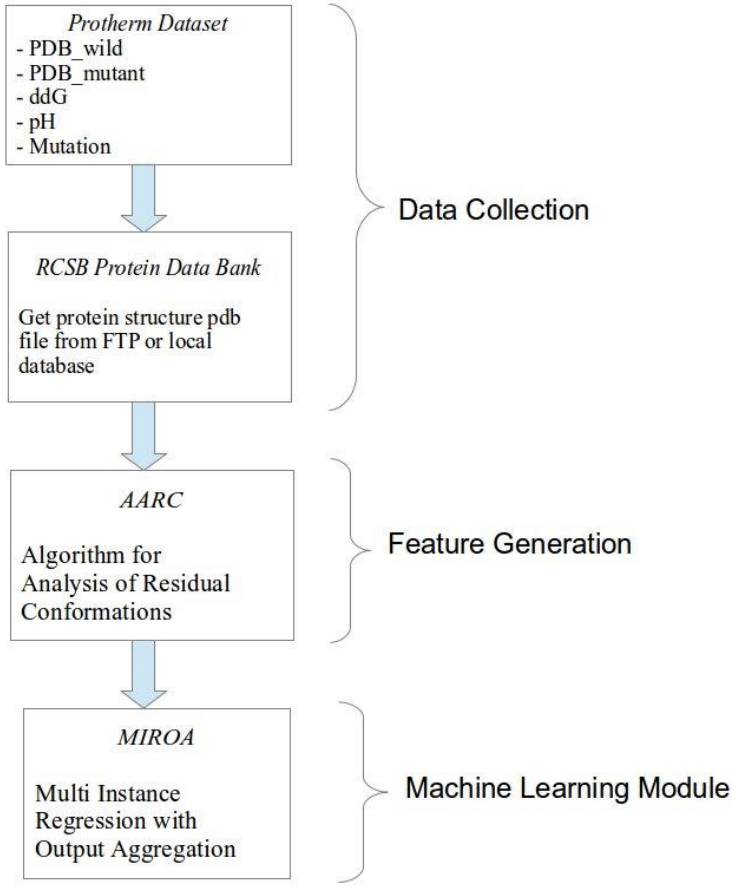

Figure 3: System Overview

The three major and discrete components of the methodology are shown above.

Figure 3 shows the overall architecture of the system. As shown, Protherm (Bava, Gromiha, Uedaira, Kitajima, & Sarai, 2004) database outset the data collection process, followed by our algorithm AARC for feature generation. AARC transforms the data in machine learnable format, which is fed into MIR-OA for generating the final machine learned model. This final model can now be used to predict results of unknown mutations.

Data Collection

Protherm is a free thermodynamic database for proteins and mutants. This database contains experimental results of mutations done on several proteins over the years by scientists all over the world. Being a combined collection of several researchers, Protherm also encompasses data generated using several reliable and unreliable experimental methodologies which inevitably induce errors and mistakes in observations. For example, a few mutations have been reported several times by different sources with each having highly varying $\Delta\Delta G$ values. Removal of such anomalies requires careful filtering of Protherm before using it. Fortunately, literature has two reliable and regularly used skimmed down versions of Protherm; these datasets were first developed by (Capriotti et al., 2004) and have been used regularly since then (Capriotti et al., 2004; Capriotti, Fariselli, & Casadio, 2005; Cheng et al., 2006; Ozen et al., 2009). The following are the two versions:

1. S1615 - The S1615 dataset contains 42 wild type proteins with 1615 experimental results. This has been used as the training set.
2. S388 – This contains 17 wild type proteins with 388 experimental results. This has been used as the test set.

It should be noted here that we are interested in observing the actual experimental results. In both these datasets there are similar protein structures used, but there is no overlap of experimental results. Our feature set does not rely upon global conformation of proteins; therefore repetition of protein structure keeps the disjunction of training, validation and testing set intact.

As shown in Figure 3, from Protherm we collect the following important information –

1. Wild type protein name
2. Mutation (which includes the residue number of wild type protein which was mutated in the experiment, the amino group name of residue to be mutated and the amino group it was mutated to)
3. Mutated protein name
4. pH value of the solution in which the experiment was conducted
5. $\Delta\Delta G$ value observed

The next step was to collect the 3-dimesional structural information for each of these protein structures. We implemented the FTP based method to download structure files in PDB format from (Berman et al., 2000). The Protein Database Files or PDB files are standard format files for representing crystallographic 3-dimensional structural information and other relevant information regarding the structure observed during crystallography. In Protein Database, every protein structure is represented by a unique 4-character identifier named PDB ID. While collecting the PDB files from FTP, we also made the local copies for reducing the fetch time during later runs. When a PDB file is searched, first the program looks for the pdb in the local

database; if not present it uses the (Berman et al., 2000) to grab the PDB file via FTP and save it into the local database.

Within each PDB file every atom is given a unique atom ID (represented by numbers), the observed B-factor(Word et al., 1999) value, the chain ID and a residue number. The chain ID and residue number are not unique for each atom. We collected atoms of only the first chain in a PDB file (since all other chains would repeat the same atoms). Also, atoms of each residue were distinguished by residue number.

A PDB file is essentially a text file, and thus for ease of reading and manipulation of these files several libraries are available. In our work we used the two most common Java libraries namely BioJava(Prlić et al., 2012) and BioShell(Gront & Kolinski, 2006, 2008). Both of these libraries provide an easy interface for reading PDB files in an object-oriented manner and also provides tools for processing several physiochemical properties.

Feature Generation

PDB files are protein structure files which only have the 3-dimensional coordinate information of each atom and each atom's temperature factor (explained in Chapter 3) information. To extract conformational features further processing is needed.

pdb_name	resi_no	voronia_property	------	mutation	ddG	pH
2LZM	33	0.065	------	?	?	?
2LZM	158	0.039	------	?	?	?
2LZM	34	0.067	------	?	?	?
2LZM	157	0.066	------	1L04	-1.1	2
2LZM	157	0.066	------	1L03	-1.3	2

Figure 4: Subsection of AARC-CSV output 2LZM

Figure 4 is a small part of AARC-CSV generated for 2LZM pdb. The ‘pdb_name’, ‘resi_no’,’mutation’,’ddG’ and ‘pH’ values combined make each row unique.

This brings us to the next step of the methodology in which we used AARC (explained in Chapter 3) on the collection of PDB files. AARC generates new Comma Separated Valued (CSV) files for each protein structure (called AARC-CSV's from hereon). The AARC-CSV of a protein contains conformational analysis values for each residue in a protein.

AARC-CSV Output Explanation

Essentially AARC-CSV is a simple CSV file with feature values of each residue in a protein. It is a protein specific CSV file. Some salient features of AARC-CSV are as follows.

- Every row of AARC-CSV represents a residue.
- Each row of AARC-CSV is unique. Although it may be observed that in some cases the ‘pdb_name’ and ‘resi_no’ are common (as shown in Figure 3), but ‘pdb_name’, ‘resi_no’,’mutation’,’ddG’ and ‘pH’ concatenation makes them unique.
- Common ‘pdb_name’ and ‘resi_no’ are due to the fact that in the Protherm dataset, a residue might have been mutated in different pH values yielding different ddG – we needed to cover all such experiments.

Once AARC completes its analysis, the AARC-CSV's are further processed to generate a single CSV file which can be understood by a machine learning algorithm. Next section describes the process of combining the AARC-CSV files to generate a machine learnable dataset and our machine learning approach.

Machine Learning Module

For machine learning we used a 'Multi-Instance Learning with Output Aggregation' (discussed in details in Chapter 4) approach. For this, we were required to develop a multi-instance dataset wherein each instance (called a bag) would have multiple attribute vectors. Each attribute vector would have a set of common values. As shown in figure 5, each bag is represented by a set of common values namely 'pdb_name', 'resi_no', 'mutation', 'ddG' and 'pH'.

pdb_name	resi_no	voronia_property	----------	overall_nearestNeighbors	mutation	ddG	pH
2LZM	38	0.159	----------	35~36~33~34~39~37~43~42~41~40~25~17~45~44	NULL	-0.1	2

Nearest neighbors of residue 38 of 2LZM protein

Conversion of AARC-CSV to machine learnable dataset

pdb_name	resi_no	voronia_property	----------	overall_nearestNeighbors	mutation	ddG	pH
2LZM	38	0.159	----------	35~36~33~34~39~37~43~42~41~40~25~17~45~44	NULL_31	-0.1	2
2LZM	38	0	----------	36~33~34~37~38~42~41~22~23~24~25~26~32~20~106~45	NULL_31	-0.1	2
2LZM	38	0	----------	35~33~34~39~37~38~42~41~40~23~24~25~32~19~45	NULL_31	-0.1	2
2LZM	38	0.065	----------	35~36~34~39~38~43~42~41~24~25~26~27~28~30~32~31~17~18~16~20~66~58~5	NULL_31	-0.1	2
2LZM	38	0.067	----------	35~36~33~39~37~38~43~42~41~40~23~24~25~26~27~32~31~17~18~20~45~44~4	NULL_31	-0.1	2
2LZM	38	0.057	----------	36~33~34~37~38~43~42~41~40~25~19~17~18~16~56~55~45~44~46	NULL_31	-0.1	2
2LZM	38	0	----------	35~36~34~39~38~42~41~40~23~24~25~19	NULL_31	-0.1	2
2LZM	38	0.018	----------	33~34~39~38~42~41~40~25~17~16~57~56~55~49~48~45~44~47~46~52~53~54~5	NULL_31	-0.1	2
2LZM	38	0.054	----------	35~36~33~34~39~37~38~43~41~40~24~25~26~27~32~17~18~56~49~48~45~44~4	NULL_31	-0.1	2
2LZM	38	0	----------	35~36~33~34~39~37~38~43~42~40~25~17~48~45~44~47~46	NULL_31	-0.1	2
2LZM	38	0	----------	36~34~39~37~38~43~42~41~25~17~45~44~47~46	NULL_31	-0.1	2
2LZM	38	0.021	----------	35~36~33~34~39~37~38~43~42~41~40~22~23~24~26~27~30~32~31~19~17~18~2	NULL_31	-0.1	2
2LZM	38	0.062	----------	33~34~39~38~43~42~41~40~24~25~26~27~28~31~19~18~15~16~14~20~58~57~5	NULL_31	-0.1	2
2LZM	38	0.038	----------	35~36~33~34~39~38~43~42~41~40~25~27~32~31~17~66~49~48~44~47~46~51~5	NULL_31	-0.1	2
2LZM	38	0.016	----------	33~34~39~38~43~42~41~40~17~56~55~49~48~45~47~46~51~52~50	NULL_31	-0.1	2

Figure 5: Conversion of AARC-CSV to machine learnable dataset

Figure 5 shows the conversion of residue number 38 of 2LZM into a bag of residues (for multi-instance learning). Note that the 'pdb_name', 'resi_no', 'mutation', 'ddG' and 'pH' of converted data corresponds to the actual row from AARC-CSV of 2LZM.

For learning, we had to develop a single file dataset which would contain a dataset on which we could implement a learning algorithm. The output of AARC could not be used directly due to two reasons – first, it produced per residue feature values whereas we were interested in performing analysis in bag format (wherein one bag had the *to be mutated* residue along with its adjacent residues), and second, the entire information was still divided into separate protein files. Therefore our objective was to combine all the information into a single CSV file in a bag format.

As discussed in chapter 1, our focus is to observe the effect of local conformations on the output of a mutation. For finding the residues which surrounded the *to be mutated* residue we developed a Nearest Neighbor feature (part of AARC, discussed in Chapter 3). Essentially from this feature we were able to gather residue numbers of adjacent residues; this was further used to grab the rows of data which were collected and put into the final dataset CSV. It is worth noting that we also used some features (like PAM250 and Blosum90) which are also common for one bag, although they do not serve any purpose in bag identification.

The following algorithm describes the methodology of generating our Multi-Instance dataset from AARC-CSV files -

MIR-OA-Dataset-Generator

1. For each wild type protein structure generate AARC-CSV files
2. Loop: for each AARC-CSV $AARC\text{-}CSV_i$
3. Loop: for each row row_i in $AARC\text{-}CSV_i$
4. If ddG_i is $\neq$ '?'
5. Collect nearest neighbor attribute value $nn\_attribute_i$
6. Loop: for each $nn\_residue_i$ in $nn\_attribute_i$
7. row_j = row data corresponding to $nn\_residue_i$ from $AARC\text{-}CSV_i$

8. Put row_j to new dataset with $ddG_i = ddG_j$, $resi\_no_i = resi\_no_j$, $ph_i = ph_j$, $pdb\_name_i = pdb\_name_j$ and $mutation_i = mutation_j$
9. Loop ends
10. Loop ends
11. Loop ends

As shown in the above algorithm, we are following output aggregation methodology (discussed in details in Chapter 4) wherein the target value of the bag is associated with each attribute vector (which are subtracted attribute vector of adjacent residues) of that bag. Several learners are implemented on this dataset.

CHAPTER 3

AARC: Algorithm for Analysis of Residual Conformations

This chapter discusses the "Algorithm for Analysis of Residual Conformations" (AARC). AARC is a residue level analysis module. This module performs comprehensive conformational analysis of each residue of each protein structure in the current dataset.

Table 1: AARC Feature Set

S.NO.	Feature Name	Final Dataset Feature Type	Source	Number of Attributes
1.	Intrusion Factor	Individual	Novel Feature	3 - Overall Residue, Main Chain, Side Chain
2.	Gap Factor	Individual	Novel Feature	3 - Overall Residue, Main Chain, Side Chain
3.	Hydrophobicity	Individual	(Widom, Bhimalapuram, & Koga, 2003)	1
4.	Nearest Neighbor	Individual	Novel Feature	3 - Overall Residue, Main Chain, Side Chain
5.	B-Factor Average	Individual	PDB File	2 - Main Chain, Side Chain
6.	Molprobity	Individual	(Chen et al., 2010)	4 – Ramachandran score, Rotamer score, Worst Angle Sigma, Worst Length Sigma
7.	Residue Distance	Individual	(Sennett, Kadirvelraj, & Wood, 2011)	3 - Overall Residue, Main Chain, Side Chain
8.	Amino-Acid Availability	Individual and Bag	(Capriotti et al., 2004)	40 – 20 Individual (for each Amino-Acid Type), 20 Bag (for each Amino-Acid Type)
9.	DSSP Feature	Individual and Bag	(Joosten et al., 2011)	4 – 2 accessibility surface individual and bag, 2 secondary structure binary individual and bag
10.	Substitution Matrices	Bag	(Dayhoff & Schwartz,	2 – pam250,

			1978; Henikoff & Henikoff, 1992)	blosum90
11.	Penalty Score	Bag	Novel Feature	1

Features processed for each residue by AARC framework

In Feature Type, an 'individual' feature implies that feature's value in final dataset is according to corresponding residue, whereas 'bag' means the feature's value is the value of *to be mutated* residue which is repeated in each attribute vector in that bag.

Along with three benchmark programs namely Voronoia (Rother, Hildebrand, Goede, Gruening, & Preissner, 2009), Molprobity (Chen et al., 2010) and DSSP (Joosten et al., 2011), we developed several novel features to analyze each residue in its wild type conformation. These features have never been used collaboratively for analysis to the best of our knowledge. Table 1 shows the features AARC processes.

Intrusion and Gap Factor

These novel features use the concepts of coordinate geometry to provide the exact volume of intrusion (steric clash) or gap (buried void or cavity) with respect to van der Waals radius around the residue's main chain and side chain (and the combined value). For this analysis, van der Waals radii of common elements found in amino acids were collected from (Bondi, 1964) . Following are explanations of these two features.

Intrusion Factor

As the name suggests, Intrusion factor provides the volume of intrusion within an atom's van der Waals radius. Given an atom a1 with van der Waals radius r1, the intrusion factor would give the approximate volume of intrusion within the sphere developed by a1 with radius r1.

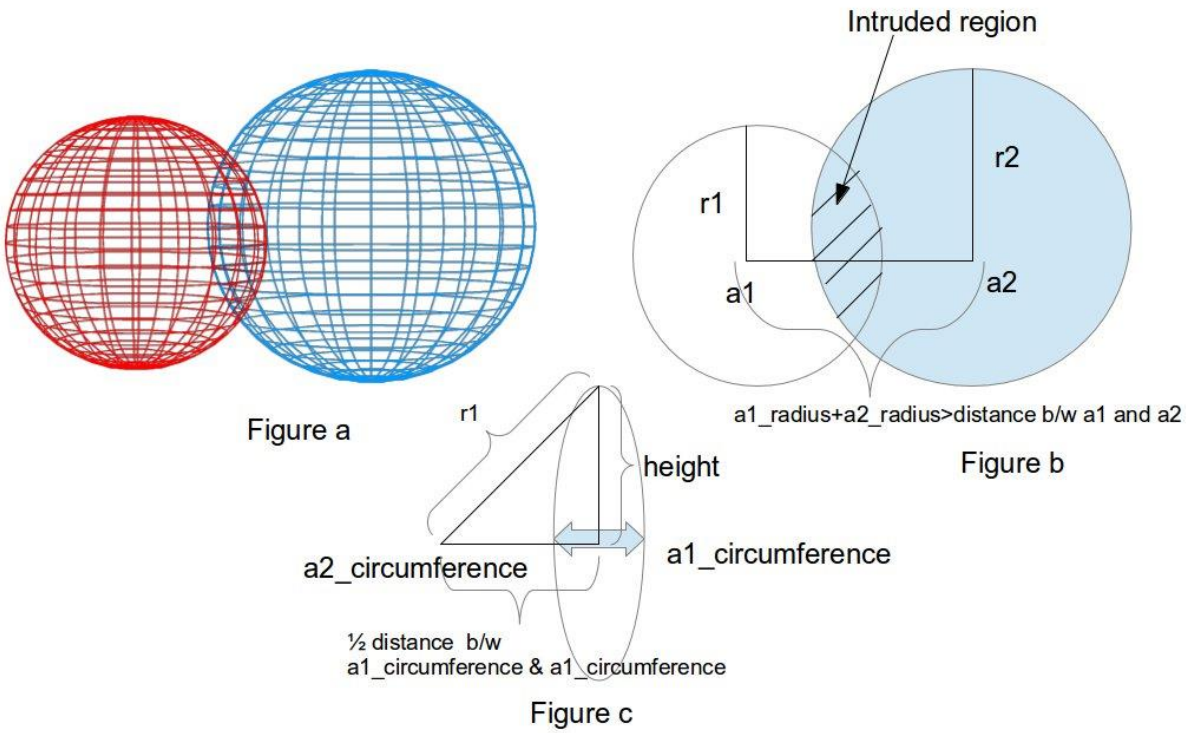

Figure 6: Intrusion Factor Diagrammatic Illustration

Figure 6 illustrates the intrusion of atom a2 with radius r2 within the van der Waals force of atom a_1 with radius r_1. The volume of intrusion is the volume of the 3-dimensional ellipsoid space which is the shaded overlap shown above. Figure 6a shows the 3-dimensional intrusion. Figure 6b explicates the precondition for intrusion factor analysis, i.e. the sum of the van der Waal radius of a1 and van der Waal radius of a_2 should be larger than the Euclidian distance

between a_1 and a_2. Lastly, Figure 6c shows the calculation of the height of the ellipsoid shaded region.

Figure 6a shows, with respect to atom a_1, how the atom a_2 intrudes into van der Waals sphere of a_1. We believe that if an atom's sphere is intruded by other atoms then it must experience some amount of strain. To quantify this strain, we considered calculating the total volume of intrusion with an assumption that intrusion volume will be directly proportional to strain.

It is possible that two atoms of different residues may not be in contact with each other or that atoms of same residue might be intruding. In such cases, we do not calculate the intrusion factor (we calculate the gap-factor for the former condition). We impose the following two preconditions for calculating the Intrusion Factor.

a. The distance between two atoms is checked. Only if the distance between two atoms is less than the sum of van der Waals radii of two atoms then the intrusion factor is calculated. Shown in figure 6b, the distance between the center of spheres of atom a_1 and atom a_2 are less than the sum of van der Waal radii of both the atoms. This is a qualifying case of intrusion factor calculation.

b. The influences of atoms of the same residue are not considered. The intrusions of atoms within the same residue are not considered for this calculation. This condition is enforced in order to prevent wrongly considering the compactness

that may be inherent within a certain kind of amino-acid. The effect of the type of amino-acid present is quantified by other factors considered in our analysis.

Often, a tightly packed residue would have many of its atoms surrounded and intruded by other atoms. For our analysis we summed the total intrusion on all the atoms within a residue. Therefore, the total volume of intrusion would be the summation of ellipsoid volumes of all such intrusions in an atom's van der Waal sphere. We calculate intrusion factors for each residue at main chain, side chain, and overall residue (main + side chain intrusion) levels.

Mathematically, intrusion within an atom can be calculated by first getting the 3-dimensional vector between the centers of the two spheres i.e.

$$a_1\text{-}a_2\text{-}center\text{-}vector[] = (a_1.\text{x} - a_2.\text{x}, a_1.\text{y} - a_2.\text{y}, a_1.\text{z} - a_2.z)$$

Where, a_i.j is j axis value of point a_i. Next using the straight line a_1-a_2-center-vector we can identify the points which are on the circumference of both the spheres. Let the farthest point of sphere a_2 in a_1 be a_2Intrusion_Point, which is calculated by the following formula.

$$a_2\text{Intrusion\_Point}[] = (a_2.\text{x} + (a_1\text{-}a_2\text{-}center\text{-}vector.x * r_2), a_2.\text{y} + (a_1\text{-}a_2\text{-}center\text{-}vector.x * r_2), a_2.\text{z} + (a_1\text{-}a_2\text{-}center\text{-}vector.y * r_2))$$

Next we find the extreme most point of sphere a_1 in a_2, let this point be a_1Circumference_Point

$$a_1\text{Circumference\_Point}[] = (a_1.\text{x} + (a_1\text{-}a_2\text{-}center\text{-}vector.x * r_1), a_2.\text{y} + (a_1\text{-}a_2\text{-}center\text{-}vector.y * r_1), a_2.\text{z} + (a_1\text{-}a_2\text{-}center\text{-}vector.z * r_1))$$

With these two points we can now work to find the volume of overlapped ellipsoid volume. The volume of an ellipsoid is –

$$Volume_{ellipsoid} = 4/3 * \pi * a * b * c$$

Where, a, b and c are 3 dimensional axes distance. In our case axis 'a' would be twice the height (shown in Figure 6c). This is calculated by using the Pythagoras theorem with two shorter sides being equal to radius r_1. We calculate height by the following formula.

The remaining two axes namely b and c would be half of the distance between a_1Circumference_Point and a_2Intrusion_Point -

$$b\_\text{or}\_c = \sqrt{(a_2\text{Intrusion\_Point}.x - a_1\text{Circumference\_Point}.x)^2 + (a_2\text{Intrusion\_Point}.y - a_1\text{Circumference\_Point}.y)^2 + (a_2\text{Intrusion\_Point}.z - a_1\text{Circumference\_Point}.z)^2}$$

We can now modify the ellipsoid formula to the following

$$a_i\text{-}a_j\text{-}volume_{\text{ellipsoid}} = 4/3 * \pi * (\text{b\_or\_c})^2 * height$$

The Intrusion-factor of one residue's main-chain would be the following -

$$Intrusion - factor_{\text{main-chain}} = \sum_{a_i \ni residue_{\text{main-chain}}} \sum_{a_j \ni residue_{\text{nearest\_neighbors}}} (a_i - a_j volume_{\text{ellipsoid}})$$

For each atom in the concerned residue's main-chain, the formula sums the a_i-a_j-$\text{volume}_{\text{ellipsoid}}$ over all the atoms of the nearest neighbors of the residue.

Gap Factor

As mentioned in Intrusion-factor, when the distance between centers of two atoms is more than the sum of the van der Waal radii of two atoms then we calculate the Gap-factor. It is observed that void around a residue affects the behavior of mutation. Consider a residue with large (or possibly unusual) gaps around it. Mutation to such a residue would display more dramatic characteristics. Gap-factor is an attempt to quantify such sites and understand the effects of large or small gaps on ΔΔG.

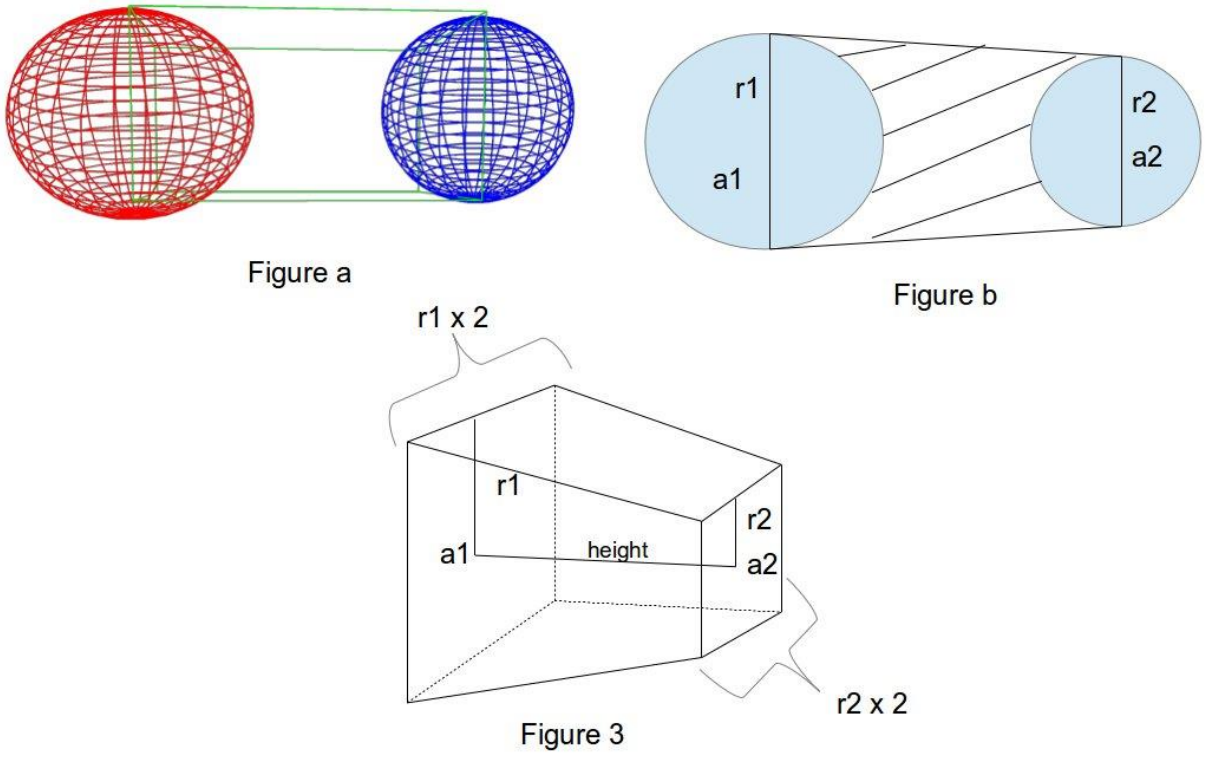

Figure 7: Gap Factor Diagrammatic Illustration

The gap factor between atom a_1 and a_2 is calculated only when they are not intruding in each other's van der Waal sphere. Figure 7a, shows the 3-dimesional area enclosed within green outlines between the spheres. This volume of trapezoidal prism is the total volume we measure to quantify the gap between spheres. Figure 7b, shows the shaded region of trapezoidal, it should be evident that since r_1 and r_2 could be different in length (because the van der Waals of sphere of different elements is different) the area we are interested in becomes the trapezoidal. Figure 7c, shows how the actual calculation of this area is done.

Figure 7 is the diagrammatic explanation of the gap-factor. There are a few points to note in this figure -

- The two atoms (a_1 and a_2) could be from two different elements. In such cases, the radii r_1 and r_2 would be different and thereby our area of interest would be a trapezoidal prism volume.
- As shown in Figure 7b, we consider the diametric plane of both spheres as the two encompassing planes of the trapezoidal volume of interest.
- The distance between a_1 and a_2 would be the height of the trapezoidal volume. We are interested in the volume of gap only. Therefore, we subtract half of the volumes of both spheres from the trapezoidal volume calculated.

Similar to the intrusion factor, the gap factor is also calculated at 3 levels namely main chain gap, side chain gap and overall gap. Also, we impose the following two constraints before calculating the gap-factor -

1. The distance between two atoms are checked. Only if the distance between the two atoms is more than the sum of van der Waals radii of the two atoms then the gap-factor is calculated. As shown in Figure 7b, the distance between the center of spheres of atom a_1 and atom a_2 are more than the sum of van der Waal radii of both the atoms. This is a qualifying case for gap factor calculation.
2. The influence of atoms of same residue are not considered. The gap of atoms within the same residue are not considered for this calculation. This condition is enforced in order to prevent wrongly considering the space that may be inherent within a certain kind of amino-acid. The effect of the type of amino-acid present is quantified by other factors considered in our analysis.

Mathematically, we can see the calculations undertaken as follows. First we calculate the distance between atom a_1 and a_2 by –

$$height=\sqrt{(a_1.\mathrm{x}-a_2.\mathrm{x})^2+(a_1.\mathrm{x}-a_2.\mathrm{y})^2+(a_1.\mathrm{z}-a_2.\mathrm{z})^2}$$

Where, $a_i.j$ is the value of a_i at axis j. Next using the radii r_1 and r_2 we can find the volumes of spheres of atom a_1 and a_2.

$$volume_{\text{sphere\_a1}}=4/3*\pi*r_1^3$$
$$volume_{\text{sphere\_a2}}=4/3*\pi*r_2^3$$

Next, we calculate the volume of the trapezoidal region –

$$volume_{trapizoid}=height*\left(((2*r_2)^2+(2*r_1)*0.334)+((4*r_1*r_2)^2*0.1667)\right)$$

The gap value between a_1 and a_2 can be found as follows -

$$a_1\text{-}a_2\text{-}gap - value = volume_{trapizoid} - volume_{\text{sphere\_a1}} * 0.5 - volume_{\text{sphere\_a2}} * 0.5$$

Lastly, the gap-factor for a residue's main-chain is found by following formula

$$Gap - factor_{\text{main-chain}} = \sum_{a_i \ni residue_{\text{main-chain}}} \sum_{a_j \ni residue_{\text{nearest\_neighbors}}} a_i\text{-}a_j\text{-}gap - value$$

In Gap-factor$_{\text{main-chain}}$, just like intrusion-factor all the atoms of main chain of residue of interest are considered. For all main-chain atoms summation of gap-factor is calculated by considering the atoms of residues in its nearest neighbors.

Hydrophobicity

Hydrophobicity is the change in free energy when a molecule is transferred from an apolar solvent to water. A positive free energy change means that the transfer is unfavorable, and is used to quantify the degree of hydrophobicity. Protein structure is very sensitive to changes in the hydrophobicity of its residues (Widom et al., 2003). A water soluble protein, in its stable state, would have hydrophobic core and polar or charged surface. This is because the hydrophobic side-chains would collapse to the core and become buried whereas the polar side-chains would react with water and be on surface or partially buried.

Table 2: Hydrophobicity Values

S.NO.	X-residue	ΔΔG
1	Ala	0.87
2	Arg	2.99
3	Asn	0.30
4	Asp	-2.46
5	Cys	1.23
6	Gln	0.30
7	Glu	-2.53
8	Gly	1.01
9	His	0.92

10	Ile	2.16
11	Leu	2.29
12	Lys	2.49
13	Met	1.71
14	Phe	2.68
15	Pro	0.90
16	Ser	0.85
17	Thr	0.95
18	Trp	2.96
19	Tyr	1.67
20	Val	1.61

In Table 2, the second column is the 3 character pdb name of a residue and the third column is the octanol-to-water free energy value. In (Wimley, Creamer, & White, 1996) the authors gave error margin values as well. For simplicity we have omitted these margins. Also, we are considering only the values for pH value of 9, whereas pH =1 values are ignored. Since our analysis involves associating the surrounding environment of a mutated residue with the effect of mutation, we considered hydrophobicity of each residue individually. This would give us a more detailed picture of the hydrophobic nature of the surrounding and the mutated residue. Considering the fact that the *to be mutated* residue is hydrophobic and is buried in core, but the surrounding residues are not hydrophobic – here the surroundings would be unstable; our idea was to capture this.

Nearest Neighbors

A novel attempt was made wherein we encoded the 3-dimensional environment of the residue *to be mutated* without any approximation methods. We wanted to protect the maximum possible granularity in the information of the environment of a *to be mutated residue*. This was done to support our hypothesis that the surrounding conformational setup of a residue is equally responsible for the mutation's behavior. Previous attempts were made in (Capriotti et al., 2004,

2005; Cheng et al., 2006; Ozen et al., 2009) to encompass the 3-dimensional environment. But we noticed that in these attempts the highly rich conformational information was reduced considerably. Given the current availability of highly efficient and accurate machine learning algorithms and huge processing powers, this step is nothing but unnecessary loss of vital information of the surroundings.

Nearest neighbor is not itself an attribute for analysis. It is an attribute that helps in developing dataset for training and testing of machine learning algorithms. That is, we use this attribute for setting up our multi instance dataset. Around each atom of a residue, a 3-dimensional sphere with 8A radius is considered. The 'residue numbers' of all the residues that fall inside this sphere are collected. Later in our processing when we perform our machine learning analysis, we use this information to collect information of the *to be mutated* residue's surroundings.

Nearest Neighbor residues are collected at 3 levels namely main-chain nearest neighbors, side-chain nearest neighbors and overall nearest neighbors. In each of these levels the atoms of the residue to be mutated are filtered for the atoms for which nearest neighbors should be considered.

B factor Average

B factor or temperature factor describes the kinetic or static disorder of an atom, and represents the flexibility of various segments of the protein. Briefly, the atoms in a crystalline structure are always vibrating depending on the flexibility of the protein segment. An atom near

the surface of a protein would not be as tightly packed as the atom buried within a core, and would be expected to have higher flexibility. In the extreme case, the atoms can be moving so much that it becomes difficult to identify the coordinates. In other words, if occupancy of any atom cannot be distinguished, i.e. its empirical electron density is low then its b-factor value will signify the low confidence in the coordinates of that particular atom. For our study this factor was used to determine flexibility as a proxy for protein dynamics. B-factor is found by the following formula.

$$B - factor_i = 8 * \pi^2 * U_i^2$$

Where,

$B\text{-}factor_i$ = B factor of the ith atom

U_i^2 = the mean square displacement of atom i.

B factors are calculated during the protein structure refinement and are listed in the PDB file with each associated atom. In our study we average the B factors for per residue analysis. We use main-chain, side-chain and overall B factor averages for our analysis.

Molprobity

Molprobity(Chen et al., 2010) is a benchmark open source program and a web-service tool. It provides structure validation service wherein they produce several results which specify which atoms and residues within a protein are more tensed or are unnaturally aligned. Molprobity performs the following analysis.

- All-atom contact analysis - This generates a ball of 0.5Å in diameter around every atom and then identifies the overlap occurring with other non-covalent bonded atoms. If there

is an overlap of more than 0.4 Å between non-donor-acceptor atoms then it signifies a clash. By this method clash/1000 atoms is reported as "clash score" for each atom. For this analysis Molprobity internally uses PROBE (Word et al., 1999).

- Torsion-angle combination using the updated Ramachandran and Rotamer analyses –

  A check is made if the residues fall in the distribution of Ramachandran backbone φ ,ψ angles (Ramachandran, Ramakrishnan, & Sasisekharan, 1963) and side chain Rotamer angle. The Ramachandran plot specifies the conformational φ ,ψ angles possible for amino-acid residue in protein; a conformation plotted outside Ramachandran plot identifies a strained conformation . The side-chain rotamer angle identifies the strain on a bond due to the current positional angle of the side-chain residues.

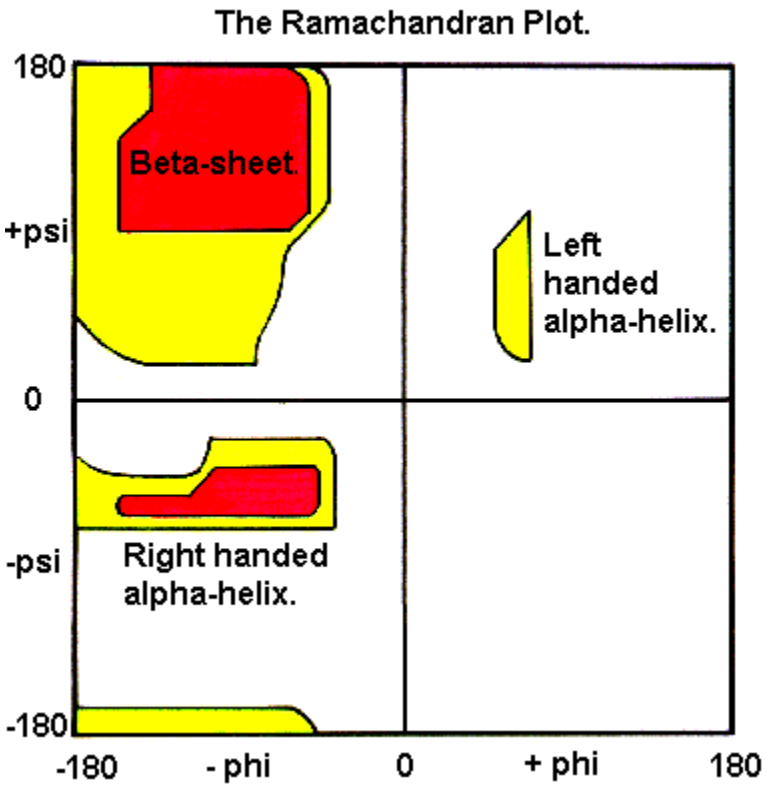

Figure 8: Ramachandran Plot

Figure 8 shows the original Ramachandran (Ramachandran et al., 1963) plot. As the number of observations has increased over the years there have been several improvements in this plot. The plot is mapped over the phi and psi angles between the atoms. If phi-psi value falls within the highlighted area then atoms are not strained.

- Covalent-geometry analyses - In this, Molprobity looks at the bond-length and bond-angle of the residues. It takes a mean of these parameters and then finds the outliers. An outlier here indicates strain in the structure, and is usually due to a modeling error, which accumulates in highly dynamic sections of the protein. Thus, these types of geometric errors are also proxies for protein dynamics.

- Molprobity score - This the final molprobity quality statistical value. It is a log-weighted combination of clash-score, percentage Ramachandran value (the not favored angles w.r.t. to Ramachandran plot) and percentage bad side-chain Rotamers. The lower the Molprobity score, the better the spot's stability.

From the abovementioned results, we take Ramachandran Score, Rotamer Score, Worst Angle (standard deviation value), Worst Length (standard deviation value). We also introduced a 'combinedScore'; it is simply an average of the abovementioned scores. For a residue i, $combinedScore_i$ is calculated as follows -

$$combinedScore_i = (rotamerScore_i + ramachandranScore_i + bondAngle_i + bondLength_i)/number\_of\_factors\_available$$

The residues at the surface of a protein do not have Rotamer scores available. Therefore, to take the average in right manner, we make sure that we adjust the denominator, i.e. 'number_of_factors_available' value according to the numerator scores available (due to constraints enforced by Molprobity).

Residue Distance

Introduced in (Sennett et al., 2011), this is a formula that uses B factor values to identify the center of motion of a protein structure. In general, the flexibility in proteins is high on the surface and is lowest near the core. The following Pearson Correlation uses this idea to identify the center of motion; main-chain B factor will display a linear increase with distance from the center of motion. The following formula is used for residue distance.

$$C_i = \frac{\sum_j \left(d_{i,j}^2 - \bar{d}^2\right)\left(B_j - \bar{B}\right)}{\left[\sum_j \left(d_{i,j}^2 - \bar{d}^2\right)^2 \sum_j \left(B_j - \bar{B}\right)^2\right]^{1/2}}$$

Here,

$d_{i,j}$ is the Euclidian distance between residue i and residue j,

d is the average distance of atoms in the given protein

B_j is the B factor value of jth atom

B is the B factor average value for the protein

In our study, we use the C-alpha atom only for calculating the distance between residues.

Amino-acid Availability Feature Set

This feature comprises a set of attributes. Here we encode the amino-acid availability. To account for the presence of a specific kind of amino-acid subset within the vicinity of the *to be mutated* residue - studies like (Capriotti et al., 2004, 2005; Ozen et al., 2009) have used a set of 20 binary features (along with few other features that we did not use) wherein they encode simple presence or absence of one particular amino-acid in the vicinity (explained in Chapter 1, Previous Work). The abovementioned literatures have claimed high accuracy using these feature sets. Therefore with the hope that we would be able to use their features along with our novel feature set we have incorporated these features in our study.

The twenty types of amino-acids are each given a fixed position in a binary array of length 20. Next we calculate if each of the 20 amino-acids is present in the 8A radius(the residues within this radius are already present from the Nearest Neighbor Feature explained above). If a residue is present then we set its corresponding array value of that amino-acid as 1

else 0. Therefore for each residue we have a binary array of length 20. Mathematically (Capriotti et al., 2004) describe this feature as -

$$V(a) = \sum_j \delta[type(j), type(a)]\rho[r(i), r(j), R]$$

Where, j goes from 0 to number of residues in a protein and [type(j),type(a)] is one when residue at position j is of type a. Also [r(i),r(j),R] is 1 when the Euclidian distance between residue i and j is less than 8A.

For completeness in our study, we repeated these 20 feature sets twice – first, we used individual values for each residue (naming it availability feature – individual), second we repeated the 20 features of the *to be mutated* residue for every residue in the bag. This provided us with a measure to learn if *to be mutated* residue is of more importance or the surrounding residues.

DSSP Feature Set

DSSP(Joosten et al., 2011) is a benchmark program used by AARC to get the water exposed surface area and secondary structure information of a residue. This program first reads the 3-dimensional structural information of a protein from a PDB file and then calculates the hydrogen bond energy of each atom. Next, to calculate the optimal hydrogen position in a structure, the actual hydrogen atoms are first removed and new hydrogen atoms are placed at 1A distance from the Nitrogen atom of backbone chain which is opposite from C=O bond. The best two hydrogen bonds processed by this are used to determine secondary structure information.

Table 3: DSSP generated secondary structure code

S.NO.	Code	Description
1	H	Alpha helix
2	B	Beta Bridge
3	E	Strands
4	G	Helix-3
5	I	Helix-5
6	T	Turn
7	S	Bend

As mentioned in Table 3, we use DSSP's output of exposed surface area and secondary structure information as individual and also as a bag feature. The exposed surface area (represented in square Angstroms) is used as-is in the individual feature and in the bag the exposed surface area of *to be mutated* residue is repeated for all the attribute vectors within a bag. As shown in Table 3, DSSP provides the secondary structure information as a single letter. We develop binary features for secondary structure by considering all letters as 0 except 'H' which is considered as 1.

Developers of DSSP provide a database version of the program and a source code version. We use the source code version where each time complete processing of PDB file is done.

Substitution Matrices

Substitution matrices provide substitution rates of amino-acids. When an amino-acid is substituted by another amino-acid the substitution matrices provide a number representing the likelihood of substitution. The likelihood is derived from homologues ancestors, i.e. substitution of a hydrophilic residue such as arginine by another hydrophilic residue such as glutamine is more likely than if replaced by leucine. In our study we used PAM250(Dayhoff & Schwartz,

1978) and Blosum90(Henikoff & Henikoff, 1992). As mentioned in Table 1, both of these features are bag features – these features are processed for *to be mutated residue* and are repeated for each attribute vector in the bag.

Penalty Score

The penalty score is another novel feature in our analysis. The volume occupied by each residue depends on the conformation and number of atoms in it. It becomes important to take into account a penalty factor with respect to volume of residue which penalizes a mutation which replaces a small volume residue by a large volume residue or vice versa.

CHAPTER 4

MIR-OA: Multi-Instance Regression with Output Aggregation

This chapter furnishes details about the machine learning module of our analysis. Here we introduce Multi-Instance Learning and explain why it was adopted in this research. Also we explain the methodology used to nullify the bias in the protein mutation datasets available.

3-dimensional protein structures are highly compact packages with large numbers of atoms arranged in a very complex yet specific manner. An analysis of these structures requires expressing the properties in as much granularity as possible. Detailed expression of features would allow the learning algorithm to better learn the decision space. Therefore, analyzing the conformational patterns of residues surrounding the mutated residue (or the residue under observation) required us to follow a "Multi-Instance Learning with Output Aggregation" approach for our machine learning analysis. By following this we were able to map the $\Delta\Delta G$ value of a mutated residue to the properties of adjacent residues; the properties which can be hypothesized as causing that particular $\Delta\Delta G$ value.

Multi-Instance Learning

Multi-Instance Learning is an extension of single instance learning. Here, each instance includes multiple attribute vectors rather than the traditional approach wherein each instance has

a single attribute vector. To such an instance with a set of multiple attribute vectors (called bag) a single target value is associated.

There are both pros and cons of this approach. On one hand, each instance can now be expressed with more details (in terms of the data patterns it holds) and also can include attribute vectors which may be weakly associated to the target value; i.e. the target value can now be associated to a group of attribute vectors (instead of just one attribute vector) each of which may or may not be strongly connected to the target value of the bag. On the other hand, this approach complicates the learning space considerably - a learner which is presented with a multi instance bag would have to learn from all the attribute vectors within that bag (which may be a complex space to learn itself). Also the learner would have to learn to distinguish one bag from the other bags (which are again sets of attribute vectors). This situation becomes worse when the target value is continuous (as in our case). Nonetheless, since our domain required high expressive power, we had to adopt this strategy (justified later).

Input Aggregation v/s Output Aggregation

For multi-instance learning, most of the effort by the machine learning community has been devoted to developing dataset transformational methods that perform transformations either during the learning phase or during the prediction phase. These methods would convert the multi-instance training/testing dataset into a flat single instance dataset. Thereby allowing one to use the already present myriad of learning algorithms for learning and prediction. Multi-instance learning has been successfully applied in several domains of which some famous ones are described in (Dietterich, 1997; Zhang & Zhou, 2007).

As mentioned earlier, there are two data transformation approaches – either to transform at the learning phase, named "Input Aggregation" or to transform at the prediction phase, named "Output Aggregation". In input aggregation a representative attribute vector for each bag is developed. This reduces one instance bag to one attribute vector and consequently the entire multi-instance dataset to a flat single instance dataset (on which now any classification algorithm can be applied). Common approaches for finding representative attribute vectors include arithmetic mean of attributes, geometric mean of attributes and K-nearest neighbor etc. Whereas in output aggregation, the dataset is flattened by adding the bag's target value to each of its attribute vector. By this, each attribute vector is now one instance in the learning dataset with a target value equal to the target value of bag (on this again any classification algorithm can be applied). Once a classification/regression model is developed on the dataset using either approach, a new bag is tested by developing the representative attribute vector of the bag in input aggregation or by taking average, median or any other aggregative method to aggregate the output predictions of each of its attribute vectors in output aggregation. We are using output aggregation (the reasons for which are discussed below).

Why Output Aggregation in this Work

Output aggregation suites our needs best. We noticed that by using input aggregation we incurred severe loss of information. The following are few points that justify our selection of output aggregation.

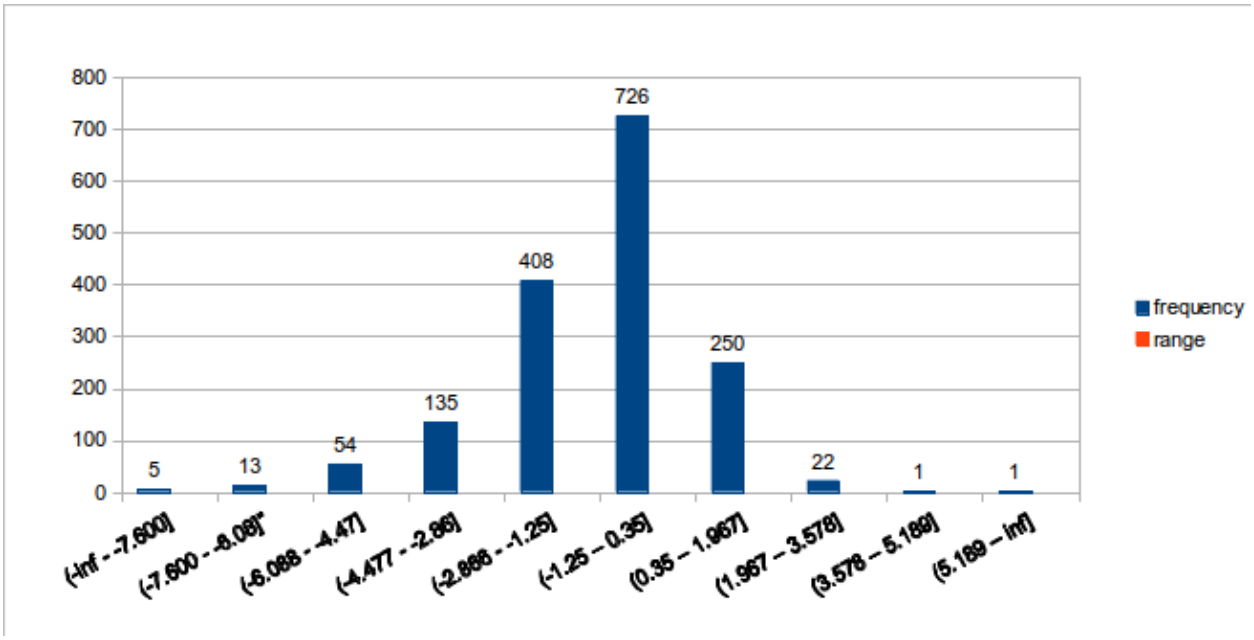

Figure 9: Distribution of Protherm ΔΔG values

Figure 9 presents a discretized distribution of ΔΔG values of S1615 dataset. Naturally occurring mutations within a protein do not have drastic ΔΔG values. This is clear in the distribution pattern of data as well where there are two large gaps of 318 and 476 between the (-1.25 – 0.35] segment and its nearest neighbors (-2.866 - -1.25] and (0.35 – 1.967] respectively.

- Consider the example of packing density we use in our analysis. We hypothesize that ΔΔG is directly related to the packing densities of adjacent residues. Now consider averaging packing densities for transformation of dataset during training. This approach might work for extreme cases wherein all (or the majority) of surrounding residue collapse or expand, but this would not work for situations where some residues expand and some collapse -which is mostly the case where ΔΔG values are near zero (i.e. not much change in protein structure).

• Most of the available experimental ΔΔG values lie between -2 and +1. Figure 9 shows the discretized distribution of the Protherm dataset. In light of the aforementioned point, it becomes necessary to keep the resolution of data unaffected during learning – which cannot be achieved in input aggregation. Whereas, in output aggregation the opposite is true; the data during training and testing is used as-is.

• Most of the data being in a close range also validates the requirement for high granularity in the training/testing instances. This would provide the learning algorithm with precise and sufficient data to learn and predict even when the data attributes are closely associated.

Output Aggregation of Prediction

As discussed above, Output aggregation allowed us to flatten the multi-instance dataset into a single instance dataset during training. Now that the learner is trained on the single instance dataset, it is capable of predicting a new dataset which is also single instance. Here we provide details on the transformation methods we use during the testing phase. We tested two approaches for output aggregation namely predicting-then-aggregating and aggregating-then-predicting.

Predicting-then-Aggregating

As mentioned earlier, our dataset was flattened and a classifier was trained on it. Each bag of attribute vectors from a test set sample in this approach was treated in a per-instance fashion, i.e. each attribute vector was treated as an instance and prediction was made on it by the classifier. Once done, the following two approaches were compared -

1. Aggregation by Average - the average value of the predictions on all instances in the bag was taken.

2. Aggregation by Median – the median value of the predictions on all instances in the bag was taken.

Aggregating-then-Predicting

Here, the attribute vector values were compressed so as to generate one representative instance for the test bag. This representative attribute vector was fed into the classifier model for prediction. The following two approaches were compared for the compression of attribute vectors of a bag -

1. Aggregation by Arithmetic Mean - the average value of each attribute in the bag was used.

2. Aggregation by Geometric Mean – the geometric mean of the values of each attribute in the bag was used.

The Predicting-then-aggregating approach performed substantially better than the aggregating-then-predicting approach. This further validates our claim regarding the issues with input aggregation (which, as mentioned earlier, transforms vectors in same way). In predicting-then-aggregating the aggregation by median worked marginally better in terms of absolute mean

error and root mean squared error but was considerably better in Pearson's correlation coefficient. These results are discussed in detail in the following chapter.

Machine Learning Algorithms

Weka (Hall et al., 2009) is an open source library for machine learning developed by 'The University of Waikato'. Weka has been developed in Java and can be easily embedded into Java projects by importing the Java packages. Weka, being a highly robust package, has several classification, clustering and attribute selection algorithms.

As a good data analysis practice, we experimented with as many algorithms as possible, including Model Trees, Random Forests, Neural-networks, Radial Basis Functions etc. We also attempted several Ensemble methods like bagging, stacking and boosting for our analysis.

Unbalanced Dataset Bias Removal

The majority of the experimental values of Protherm dataset lie in the range of -2 to +1. Learning on such an unbalanced dataset creates a bias in learning algorithms. Unbalanced datasets may result in a classical case wherein a classifier has high false positives with high true positives, i.e. most or all of the data is classified to one particular class which has the largest number of instances. Such biased classifier will show high accuracy if the test set is also biased (which is commonly the case in real world domain problems). In our case, a model developed on such a dataset would do well on an average (due to the high likelihood of a new test instance to lie between -2 to +1) but such a model would be a biased model that would fail on predicting instances outside to majority dataset range.

We introduced stratification during the training/validation phase for balancing the data. Before training/validation we divided the data into four strata $+\infty$ to 0.2, 0.2 to -1.5, -1.5 to -3.0 and -3.0 to $-\infty$ according to $\Delta\Delta G$ values and made a separate dataset for each strata. Next dataset of each strata was divided into equal sized folds. Lastly, the i^{th} fold of each strata was combined with the i^{th} folds of all other strata; this generated a new combined by strata and fold separated dataset. Training/validation was done on this dataset.

This exercise made sure that all the cross validation folds contained equal instances for training and for cross validation testing. Thereby, providing a more reliable picture of data over fitting and cross validation correlational accuracy of the learned models.

CHAPTER 5

EXPERIMENTAL RESULTS & CONCLUSION

In this final chapter we explain the experimental results and conclusions. First we illustrate the cross validation results of several regression learning algorithms on the S1615 dataset and then the test set results on the S388 dataset. At the end we would provide the conclusion and future work.

Learning and Cross Validation Results

As discussed in Chapter 4, we trained our models on the S1615 dataset. Training was done with 20 fold cross validation along with stratification. We used four methods of prediction namely prediction-by-average, prediction-by-median, prediction-by-arithmetic-mean and prediction-by-geometric-mean. Caution was kept to make sure that during training, validation and testing none of our bagged instances was split.

Table 4: Cross Validation Results of Algorithms

Algorithm Name	prediction-by-average			prediction-by-median			prediction-by-arithmetic-mean			prediction-by-geometric-mean		
	Correlation Coefficient	Mean absolute	Root mean squared	Correlation Coefficient	Mean absolute	Root mean squared	Correlation Coefficient	Mean absolute	Root mean squared	Correlation Coefficient	Mean absolute	Root mean squared
Random Forest	0.8241	0.6072	0.931	0.8244	0.6027	0.9302	0.8157	2.5011	3.467	0.7971	2.4942	3.422
Random Forest with Bagging	0.8158	0.6159	0.9586	0.8150	0.6122	0.9595	0.8022	2.504	3.477	0.7861	2.523	3.4392
REP Trees	0.6640	0.8311	1.364	0.656	0.8414	1.3891	0.6506	2.593	3.657	0.6502	2.5924	3.6594
M5P	0.6563	0.8503	1.530	0.6505	0.8592	1.5471	0.6493	2.5466	3.756	0.6484	2.5537	3.767
M5P with Bagging	0.699	0.7637	1.6992	0.6975	0.7653	1.704	0.6972	2.5498	3.936	0.6936	2.5573	3.9422
Multilayer	0.6488	1.071	1.584	0.6496	1.014	1.587	0.6481	2.808	3.907	0.644	2.835	3.955

Perceptron												
RBF	0.5797	1.001	1.354	0.5777	1.001	1.356	0.5688	4.3870	5.048	0.5589	5.5314	6.182
RBF with Bagging	0.5498	1.019	1.3966	0.5475	1.019	1.399	0.5367	4.495	5.140	0.527	5.717	6.342
Linear Regress	0.5463	1.026	1.388	0.5467	1.024	1.398	0.5480	2.518	3.197	0.5407	2.489	3.166
Linear Regress with Bagging	0.5416	1.033	1.4094	0.5418	1.032	1.409	0.532	2.529	3.211	0.5382	2.502	3.189
PLS Classifier	0.5509	1.032	1.394	0.5506	1.030	1.394	0.5528	2.532	3.222	0.5475	2.484	3.174
MLP Regress	0.5971	0.9789	1.345	0.5943	0.9817	1.349	0.5942	2.465	3.305	0.5936	2.47	3.311

Table 4 shows the 20 fold cross validation results of several regression algorithms on the S1615 dataset. Evidently, prediction-by-median method has performed significantly better than prediction-by-arithmetic-mean and prediction-by-geometric-mean, and often (but not always) better than prediction-by-average. The reason for such a discrepancy can be attributed to the way prediction-by-arithmetic-mean and prediction-by-geometric-mean is done; as shown in chapter 4 these methods require manipulation of attribute vectors before learning. As we claimed earlier, manipulation of attribute vectors reduces the granularity which is essential for learning the complex dynamics of local conformations in proteins. This result validates our decision of using Multi Instance Learning with Output Aggregation, rather than input aggregation. Also, from hereon we consider the approach of prediction-by-median as our standard prediction approach and discard others.

Random Forests algorithms had the best results in terms of correlation coefficient (0.8244) and mean absolute error (0.6027). Inherently this algorithm uses a bagging, keeping this in mind we tested the performance of all other algorithms natively and then with bagging. None of the other algorithms performed better than Random Forests. It can also be seen that

performance of decision tree based algorithms was much better than function based algorithms in terms of error. A sharp dip was observed in mean absolute error and root-mean-squared error.

Table 5: Confusion Matrix S1615 dataset

	Predicted		
Actual		Unstable	Stable
	Unstable	1111	50
	Stable	174	267

Table 5 shows the result of binarizing the collective results of the 20 fold cross validation of the S1615 dataset. The ddG values greater than or equal to zero were considered "Stable" and the ddG values less than zero were considered "Unstable". Further, "Unstable" classification was considered as positive and "Stable" classification was negative. As mentioned earlier, the total number of instances was 1602 because we removed the experimental results of 1LRP.pdb from the dataset. From table 5, following statistics can be deduced –

Sensitivity = True Positive/ (True Positive + False Positive) = 0.864

Specificity = True Negative/(True Negative + False Negative) = 0.842

Precision = True Positive/ (True Positive + False Positive) = 0.864

Accuracy = (True Positive + True Negative)/ (Total) = 0.8601

F-Measure = 2*True Positive/(2*True Positive + False Positive + False Negative) = 0.908

Outliers

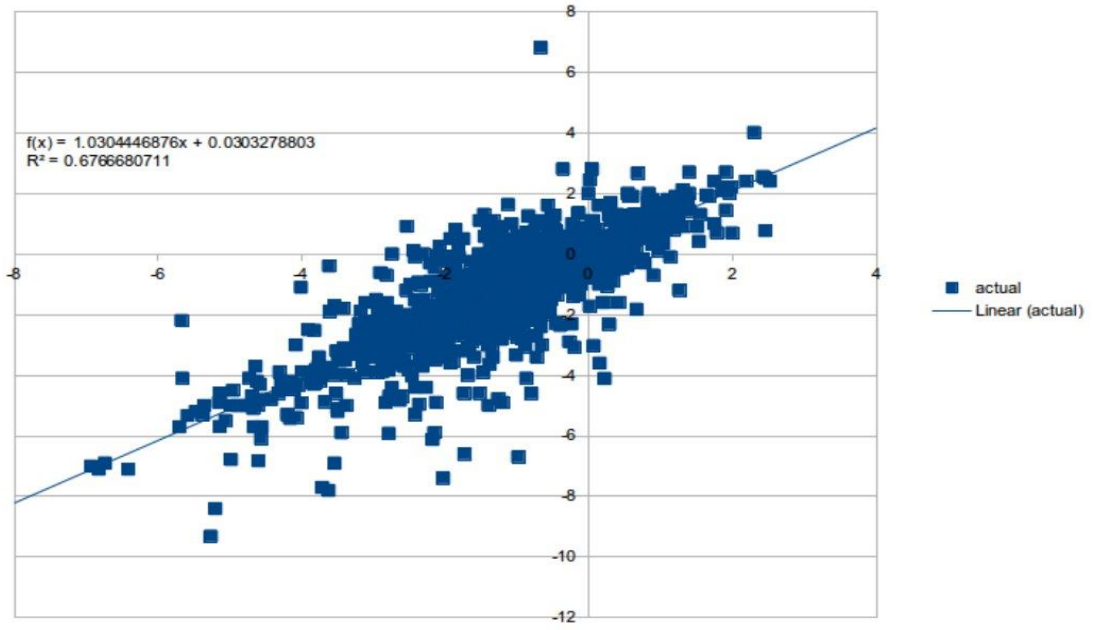

Figure 10: 20 fold Cross Validation of S1615

Figure 10 shows the correlation graph of the S1615 dataset on 20 fold cross validation using the Random Forests Algorithm. This plot was developed by plotting the predicted against the actual values of all the cross validation dataset (which ultimately is the entire S1615 dataset).

From figure 10 it can be seen that there are a few data points which the learning algorithm missed by a large margin. Analysis of points that have absolute difference of more than two between predicted and actual values by Biochemistry experts showed that the learning algorithm missed mostly in dimers. Dimers are present on the surface of a protein structure and therefore AARC is expected to develop features that depict the *to be mutated* residue as surface residue. In actuality, dimers despite being on the surface are sandwiched between two protein structures and thereby behave like buried residue on mutation. Prediction on dimers requires

further development of features which are not yet covered by AARC features. Those features are beyond the scope of this thesis work.

Testing Results

Using the information from Table 4, we developed WEKA models for the Random Forest algorithm trained on S1615. Since our model development was done in 20 folds, we essentially had 20 models; on which we ran our test dataset S388.

Table 6: Test Result on S388

fold	CC-by-median-training	CC-by-median-testing	Error-by-median-training	Error-by-median-testing	RMS-by-median-training	RMS-by-median-testing
0	0.811	0.980	0.628	0.128	0.952	0.334
1	0.893	0.985	0.621	0.134	0.891	0.285
2	0.841	0.975	0.526	0.143	0.898	0.369
3	0.777	0.983	0.668	0.131	1.013	0.300
4	0.806	0.977	0.625	0.142	0.934	0.349
5	0.782	0.973	0.673	0.137	1.169	0.384
6	0.832	0.988	0.522	0.136	0.882	0.259
7	0.805	0.985	0.548	0.129	0.869	0.287
8	0.914	0.985	0.488	0.140	0.669	0.286
9	0.744	0.979	0.625	0.133	1.079	0.340
10	0.849	0.988	0.550	0.123	0.784	0.259
11	0.847	0.987	0.598	0.114	1.005	0.263
12	0.737	0.987	0.653	0.121	1.174	0.264
13	0.813	0.978	0.776	0.134	1.092	0.342
14	0.815	0.975	0.589	0.147	0.878	0.370
15	0.834	0.980	0.608	0.135	0.933	0.325
16	0.807	0.988	0.605	0.130	0.889	0.254
17	0.846	0.988	0.629	0.124	0.875	0.254
18	0.825	0.981	0.650	0.146	0.918	0.321
19	0.899	0.993	0.463	0.119	0.693	0.205
Avg.	0.82385	0.98275	0.60225	0.1323	0.92985	0.3025

The highlighted green row in Table 6 is the average result we observed. On average, the correlation of 0.98275 was observed with only 0.13213 mean absolute error. This is by far the best result ever observed in this domain.

Table 7: S388 testing using S1615 complete dataset

CC-by-median-testing	Error-by-median-testing	RMS-by-median-testing
0.9949	0.0949	0.1595

For completeness we also developed a model on complete S1615 dataset using the Random Forest algorithm (since it performed the best) and we tested the S388 dataset on it. Table 7 shows the result. Evidently, this result would not contain any training errors as there was no cross validation.

Table 8: Confusion Matrix S388 dataset

	Predicted		
Actual		Unstable	Stable
	Unstable	339	2
	Stable	2	45

Similar to the binarization process followed in developing Table 5, the confusion matrix on the S388 dataset is shown in Table 8. Here the average of predictions made of each instance by 20 models developed by 20 fold cross validation process using random forest was done.

Table 9: Comparison of S388 dataset

Method	MCC	Accuracy	Sensitivity(+)	Specificity(+)	Sensitivity(-)	Specificity(-)
FOLDX	0.25	0.75	0.56	0.26	0.78	0.93
DFIRE	0.11	0.68	0.44	0.18	0.71	0.90
PoPMuSic	0.20	0.85	0.25	0.33	0.93	0.90
iMutant2.0	0.25	0.87	0.21	0.44	0.96	0.90
Mu-pro	0.28	0.86	0.31	0.42	0.94	0.91
MIR-OA	0.95	0.989	0.957	0.957	0.994	0.994

Finally we compare our binary prediction capability with the state-of-the-art methods. We collected the results and used the formulas mentioned in (Cheng et al., 2006) for developing the numbers listed in Table 9. As evident our results in all the comparative metrics are better than all other methods.

Conclusion

We have observed significant improvement in results over any previous work in this domain. The success thus far supports our hypothesis that local conformations have greater impact on the final outcome than global conformations.

From a machine learning perspective, it is evident that granularity of a dataset is an important factor for learning complex concepts. Our approach of multi-instance learning has never been attempted in this domain. Keeping the 3-dimensional information of the surrounding residues intact helped our learning algorithms to learn with high accuracy. The steep reduction in accuracy when using input aggregation methods (prediction-by-arithmetic-mean and prediction-by-geometric-mean) for prediction proves that compressing knowledge negatively affects learning. The combination of novel features and the novel approach of learning together helped our cause.

Future Work

This work has opened new doors for studying protein structures. The positive results encourage us to experiment more with the alternative approaches and find the optimal fit. The following are a few ideas for future work.

- We have implemented methodology on single-site mutations only. For robustness, this program should be trained on multi-site mutations as well.
- We used nearest neighbors of main-chain of *to be mutated* residue. Further study should be done at per atom level as well.
- More machine learning and output aggregation approaches can be tested on this model.
- Automation of the stratification approach can be explored. Possibly more optimal stratification can be found for the S1615 dataset.
- Learning of the larger Protherm dataset can be implemented.

REFERENCES

Bava, K. A., Gromiha, M. M., Uedaira, H., Kitajima, K., & Sarai, A. (2004). ProTherm, version 4.0: thermodynamic database for proteins and mutants. *Nucleic acids research*, *32*(Database issue), D120–1. doi:10.1093/nar/gkh082

Berman, H. M., Westbrook, J., Feng, Z., Gilliland, G., Bhat, T. N., Weissig, H., … Bourne, P. E. (2000). The Protein Data Bank. *Nucleic acids research*, *28*, 235–242. doi:10.1093/nar/28.1.235

Bondi, A. (1964). van der Waals Volumes and Radii. *The Journal of Physical Chemistry*, *68*, 441–451. doi:10.1021/j100785a001

Bordner, A. J., & Abagyan, R. A. (2004). Large-scale prediction of protein geometry and stability changes for arbitrary single point mutations. *Proteins: Structure, Function and Genetics*, *57*, 400–413. doi:10.1002/prot.20185

Capriotti, E., Fariselli, P., & Casadio, R. (2004). A neural-network-based method for predicting protein stability changes upon single point mutations. *Bioinformatics (Oxford, England)*, *20 Suppl 1*, i63–i68. doi:10.1093/bioinformatics/bth928

Capriotti, E., Fariselli, P., & Casadio, R. (2005). I-Mutant2.0: Predicting stability changes upon mutation from the protein sequence or structure. *Nucleic Acids Research*, *33*. doi:10.1093/nar/gki375

Chen, V. B., Arendall, W. B., Headd, J. J., Keedy, D. A., Immormino, R. M., Kapral, G. J., … Richardson, D. C. (2010). MolProbity: All-atom structure validation for macromolecular crystallography. *Acta Crystallographica Section D: Biological Crystallography*, *66*, 12–21. doi:10.1107/S0907444909042073

Cheng, J., Randall, A., & Baldi, P. (2006). Prediction of protein stability changes for single-site mutations using support vector machines. *Proteins*, *62*, 1125–1132. doi:10.1002/prot.20810

Dayhoff, M., & Schwartz, R. (1978). A Model of Evolutionary Change in Proteins. *In Atlas of protein sequence and structure*, 345–352. doi:10.1.1.145.4315

Dehouck, Y., Kwasigroch, J. M., Gilis, D., & Rooman, M. (2011). PoPMuSiC 2.1: a web server for the estimation of protein stability changes upon mutation and sequence optimality. *BMC bioinformatics*, *12*, 151. doi:10.1186/1471-2105-12-151

Dietterich, T. (1997). Solving the multiple instance problem with axis-parallel rectangles. *Artificial Intelligence*. doi:10.1016/S0004-3702(96)00034-3

Gilis, D., & Rooman, M. (1997). Predicting protein stability changes upon mutation using database-derived potentials: solvent accessibility determines the importance of local versus non-local interactions along the sequence. *Journal of molecular biology*, *272*, 276–290. doi:10.1006/jmbi.1997.1237

Gront, D., & Kolinski, A. (2006). BioShell - A package of tools for structural biology computations. *Bioinformatics*, *22*, 621–622. doi:10.1093/bioinformatics/btk037

Gront, D., & Kolinski, A. (2008). Utility library for structural bioinformatics. *Bioinformatics*, *24*, 584–585. doi:10.1093/bioinformatics/btm627

Guerois, R., Nielsen, J. E., & Serrano, L. (2002). Predicting changes in the stability of proteins and protein complexes: A study of more than 1000 mutations. *Journal of Molecular Biology*, *320*, 369–387. doi:10.1016/S0022-2836(02)00442-4

Hall, M., National, H., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., & Witten, I. H. (2009). The WEKA Data Mining Software : An Update. *SIGKDD Explorations*, *11*, 10–18. doi:10.1145/1656274.1656278

Henikoff, S., & Henikoff, J. G. (1992). Amino acid substitution matrices from protein blocks. *Proceedings of the National Academy of Sciences of the United States of America*, *89*, 10915–10919. doi:10.1073/pnas.89.22.10915

Joosten, R. P., Te Beek, T. A. H., Krieger, E., Hekkelman, M. L., Hooft, R. W. W., Schneider, R., … Vriend, G. (2011). A series of PDB related databases for everyday needs. *Nucleic Acids Research*, *39*. doi:10.1093/nar/gkq1105

Ozen, A., Gönen, M., Alpaydan, E., & Haliloğlu, T. (2009). Machine learning integration for predicting the effect of single amino acid substitutions on protein stability. *BMC structural biology*, *9*, 66. doi:10.1186/1472-6807-9-66

Prlić, A., Yates, A., Bliven, S. E., Rose, P. W., Jacobsen, J., Troshin, P. V., … Willis, S. (2012). BioJava: An open-source framework for bioinformatics in 2012. *Bioinformatics*, *28*, 2693–2695. doi:10.1093/bioinformatics/bts494

RAMACHANDRAN, G. N., RAMAKRISHNAN, C., & SASISEKHARAN, V. (1963). Stereochemistry of polypeptide chain configurations. *Journal of molecular biology*, *7*, 95–99. doi:10.1016/S0022-2836(63)80023-6

Rose, G. D., Fleming, P. J., Banavar, J. R., & Maritan, A. (2006). A backbone-based theory of protein folding. *Proceedings of the National Academy of Sciences of the United States of America*, *103*, 16623–16633. doi:10.1073/pnas.0606843103

Rother, K., Hildebrand, P. W., Goede, A., Gruening, B., & Preissner, R. (2009). Voronoia: Analyzing packing in protein structures. *Nucleic Acids Research*, *37*. doi:10.1093/nar/gkn769

Sennett, N. C., Kadirvelraj, R., & Wood, Z. A. (2011). Conformational flexibility in the allosteric regulation of human UDP-alpha-D-glucose 6-dehydrogenase. *Biochemistry*, *50*, 9651–9663. doi:10.1021/bi201381e

Widom, B., Bhimalapuram, P., & Koga, K. (2003). The hydrophobic effect. *Physical Chemistry Chemical Physics*. doi:10.1039/b304038k

Wimley, W. C., Creamer, T. P., & White, S. H. (1996). Solvation energies of amino acid side chains and backbone in a family of host-guest pentapeptides. *Biochemistry*, *35*, 5109–5124. doi:10.1021/bi9600153

Word, J. M., Lovell, S. C., LaBean, T. H., Taylor, H. C., Zalis, M. E., Presley, B. K., … Richardson, D. C. (1999). Visualizing and quantifying molecular goodness-of-fit: small-probe contact dots with explicit hydrogen atoms. *Journal of molecular biology*, *285*, 1711–1733. doi:10.1006/jmbi.1998.2400

Zhang, M. L., & Zhou, Z. H. (2007). ML-KNN: A lazy learning approach to multi-label learning. *Pattern Recognition*, *40*, 2038–2048. doi:10.1016/j.patcog.2006.12.019