MSc Project - Binding Affinity Prediction of Protein-Ligand Complex

Abdus Salam Khazi
abdus.khazi@students.uni-freiburg.de

Github Repository [6]

Supervisors: Simon Bray & Alireza Khanteymoori

September 20, 2021

Contents

1 Abstract

2 Introduction

2.1 Biological Background

Proteins are the workhorses of our body. They are necessary for many important functions in the body. Ligands are molecules that bind to proteins to form protein-ligand complexes. They can be molecules that the protein transports (e.g., a Haemoglobin transporter) or act as stimulating agents. In addition to this, they can also start/stop the protein from doing its function. The correct functioning of these protein-ligand complexes is essential for any living organism.

The study of protein-ligand complexes is an intrinsic part of the drug discovery field. It is because drugs are small molecules that act as ligands. As the drug molecules (ligands) bind to the target proteins, they can artificially influence the protein behavior. This binding between protein and ligand causes a therapeutic effect.

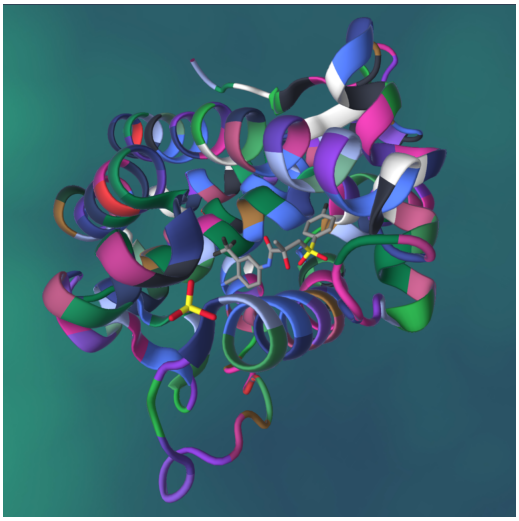


Figure 1: Haemoglobin transporter protein. [15]

When we find a target drug candidate, we have to answer questions like - How easily does the drug bind to the target protein? Does it bind to any other protein - If so, is it desirable? Does it have any unforeseen effect on the protein function? etc... To answer these questions biologists and pharmacists conduct wet-lab experiments that are expensive.

One way to reduce the cost of these experiments is to make a data-driven selection of the drugs. Using experimental data collected over many

years, one can build models to predict the behavior of the proposed drug computationally. These 'In-Silico' computational methods can aid in the elimination of undesirable drugs as well as guide the drug selection process.

Our project aims to answer one of the above questions - How well does a given drug bind to the target protein? We determine this computationally by building a machine learning model that trains on the previous data. We hope that this model will help reduce the costs of drug discovery.

2.2 Understanding Binding Affinity

The binding affinity between a protein and a ligand is quantified by the K_d, K_i and IC_{50} measures in the PDBBind Data bank. Here K_d refers to disassociation constant, K_i refers to the inhibition constant, and IC_{50} refers to inhibitory concentration 50%. The reason for having different measurements is because it is not possible to use the same measurement techniques for all biological complexes/processes.

To understand K_d, consider a protein and a ligand binding and unbinding continuously in a kinetic system. In this system, let $[P]$, $[L]$, and $[PL]$ represent the concentrations of the Protein, the Ligand, and the Protein-Ligand complex respectively. This is represented by the following equation:

$$[P] + [L] \rightleftharpoons [PL]$$

We can quantify the binding affinity K_d by using the concentrations in the above system at equilibrium.

$$K_d = \frac{[P][L]}{[PL]} = \frac{k_{-1}}{k_1}$$

where k_{-1} is the disassociation rate constant and k_1 is the association rate constant. Similarly, K_i and IC_{50} are defined using concentration albeit non-trivially. [11]

2.3 PDBBind Dataset

Over the last few decades, researchers have been successful in building a single data archive for proteins. This archive, called **Protein Data Bank** [8] , holds 3-D structural data of the proteins determined by experiments like X-ray crystallographic, Nuclear magnetic resonance (NMR), and cryoelectron microscopy (cryoEM). A subset of this data also contains information about how well a given protein and ligand bind together. It is called binding affinity between a protein and ligands. (It also contains data about protein-protein complexes that our project does not deal with) [7]

As we study the protein-ligand binding affinity here, we would like to filter out this data from the **Protein Data Bank** It is what is done by the maintainers of the **PDBBind Data Bank**. [13] Using the curated protein-ligand affinity data present in the PDBBind Data bank, we build a machine learning model that learns to predict the affinity.

3 Problem Formulation

3.1 Problem Overview

The problem that we are solving is - *Given $K_d/K_i/IC_{50}$ for various complexes in the PDBBind Data bank, can we predict this affinity measure for new protein-ligand complexes?* Figure 2 shows how the Protein-Ligand problem can be classified.

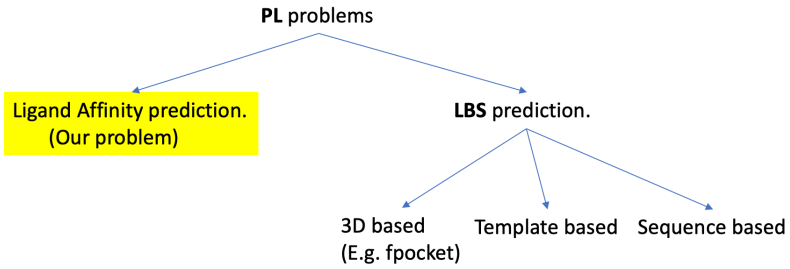


Figure 2: Protein-Ligand problem classifcation.

The binding of proteins and ligands is heavily influenced by their respective 3D structures. Figure 3 illustrates a hypothesis called *Lock and Key*. It is very crucial that the shape of the protein's binding location and the shape of the ligand be complementary for the binding.

The other parts of the protein are not involved in the binding process directly. Hence, we only get the features of the binding location. Any potential binding location in the 3D structure of a protein is called a pocket. We use a 3D based LBS prediction package called *fpocket* to find the potential pockets. A submodule in the package called *dpocket* is used to extract the binding pocket's features.

To give a plug-and-play input to our model, we keep the features of proteins and ligands distinct till the training phase. That is, we do not use the combined features in any pre-processing of the ML pipeline e.g. feature reduction. This helps our model to provide the binding affinity between any protein and any ligand.

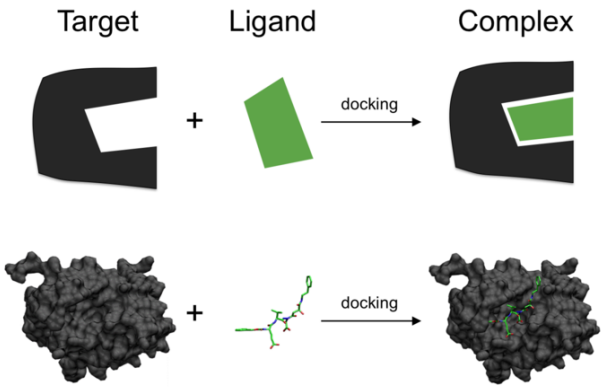


Figure 3: Lock and Key hypothesis in molecular docking. [12]

3.2 Overview of file formats

The **PDBBind Data bank** extracts information about the PL complexes from the **Protein Data bank** and creates the following files for every complex

- **PDB Format** - For the Protein.
- **Mol2** - For the ligand.
- **SDF** - For the ligand.

All of the above formats contain the 3D information that is essential in the prediction of the binding affinity. All the above mentioned formats use the **XYZ format** internally to represent the 3D structure of their molecules.

3.2.1 XYZ format

XYZ format is a chemical file format that represents the geometry of a molecule. It specifies the number of atoms and their Cartesian X, Y, Z coordinates hence the name XYZ format. The following text illustrates the XYZ format. Section 9.1 gives an example. [18]

```
<number of atoms>
comment line
<element> <X> <Y> <Z>
...
```

The unit of distance used is Angstrom (Å). $1\,\text{Å} = 10^{-10}$ m. [18]

3.2.2 PDB format

PDB format is a human-readable file format used to represent the protein molecules (macromolecules). Within the PDB format, the coordinates of atoms are represented like the XYZ format [see 3.2.1]. Because of the 3D information in this format, molecular visualization of proteins is possible with specialized software. It also contains information about atomic connectivity and the protein's primary, secondary, tertiary, and quaternary structures. [14] [4] Please see [1] for an example pdb file.

3.2.3 Structure Data File (SDF) Format

SDF format file is a Chemical Table file (CT File) that contains structure of the molecule in the X,Y,Z format. It contains information like atomic bonds, connectivity information, molecular weight, and molecular formula. [17] Section 9.2 illustrates the SDF file format.

3.2.4 Mol2 format

Similar to SDF format, Mol2 also represents the 3D structure of a molecule in the X,Y,Z format. It contains the atomic bond and connectivity information but does not contain the other data like molecular weight and formula. We use the ligands given in this format because more ligands in mol2 format could be processed with the RDKit feature extractor. Section 9.3 illustrates the Mol2 file format.

3.2.5 SMILES format

SMILES is an acronym for Simplified Molecular-Input Line-Entry System. It represents a molecule using an ASCII string. Using the 3D data in SDF and Mol2 formats, we can create an atomic graph representation. Using this graph the SMILES string for the molecule can be generated. The SMILES format itself is not very helpful for us as we lose the 3D structural information after converting to it. [16]

3.3 Extraction of features

As the file formats are different for proteins and ligands, we use different tools to extract their features.

3.3.1 Ligand Features using RDKit

RDKit is an open-source cheminformatics software [9]. The core data structures and algorithms of RDKit are written in C++ and the python wrappers are generated using Boost.Python. Using the module, *RDKit.Chem.Descriptors* we extract 402 features for each ligand [2]. Each descriptor value is taken as a real number, hence the input space of ligand features is $\mathbf{R}^{402}$ before any feature elimination.

3.3.2 Protein Features using fpocket/dpocket descriptors

Fpocket stands for "Find pocket" whereas *Dpocket* stands for "Describe pocket". *Fpocket* uses 3D Voronoi tessalation and the concepts of "Alpha Spheres" to find out pockets in the protein structure [5] [19]. Given a protein PDB file, we can extract the descriptors of all pockets in the protein. For every pocket, we get 55 descriptors in total which are taken as real values, $\mathbf{R}$. Hence the input space for protein features is $\mathbf{R}^{55}$.

To get the descriptors of the ligand-binding pockets, we use *Dpocket*. *Dpocket* is provided with the protein PDB file and the ligand ID of the PL complex as input. It generates 3 files as output files, namely, dpout_fpocketp.txt, dpout_fpocketnp.txt, and dpout_explicitp.txt.

- **dpout_fpocketp.txt**. This file contains the descriptors (a.k.a features) of all pockets that are considered to be binding pockets based on a binding criterion. The ligand can bind at different locations (pockets) in the protein 3D structure. Hence, there may be features of more than 1 pocket in this file.

- **dpout_fpocketnp.txt**. This file contains the descriptors of all pockets that are non-binding according to the criteria.

- **dpout_explicitp.txt**. An explicit pocket is defined as a pocket consisting of all vertices/atoms situated at a specific distance from the ligand in the PL complex. This distance is 4 Å by default. This file contains the descriptors of all explicit pockets in the PL complex.

In our project, we do not use dpout_fpocketnp.txt as they contain non-binding pockets. In the other 2 files, we prefer using pocket descriptors given by dpout_fpocketp.txt as explicitly defined pockets are heavily biased towards the ligand.

4 Feature selection

4.1 Requirement of feature selection

Both the protein and the ligand are equally responsible for the affinity of the PL complex. Hence we concatenate their descriptors to get a high dimensional $\mathbf{R}^{457}$ input for our model. There are a couple of issues with using all of the descriptors.

- The amount of data is not very large. For example, we could only get ≈ 35000 data points after concatenating the ligand features with the *fpocketp* pocket descriptors.
- The number of ligand descriptors >> protein descriptors. This creates a data imbalance and may lead the model to select only ligand features for their prediction.

Hence we need some methods to reduce the input dimensions.

4.2 Feature Family analysis

All the features extracted from the pdb and the ligand files can be classified into various families. Some of the important ones are - AUTOCORR2d_, Chi, EState_VSA, PEOE_VSA, SMR_VSA, SlogP_VSA, VSA_EState and fr_. Figure 4 shows correlation matrix of 2 of the most interesting families AUTOCORR2d_ and Chi. As we can see from the heatmap there is heavy correlation within the families. Hence we also require a strategy to deal with this. The correlation heat maps for the other families are illustrated in Section 9.4.

4.3 Feature Selection Strategies

We try to select features of the protein and the ligand separately. This helps us make our model plug-and-play as discussed in section 3.1. The best combination, i.e Global Optima, cannot be obtained practically by brute force algorithms. This is because we would have to try $\binom{402}{k_1} * \binom{55}{k_1}$ $(k_1, k_2 \in \mathbf{I}^+)$ possibilities which is impractical. In the next subsections we discuss a few feature selection strategies that were considered in our project.

4.3.1 Selection by output correlation

The *pearson* and *spearman* correlations of each feature were calculated against the output variable. We assumed that the features were either lin-

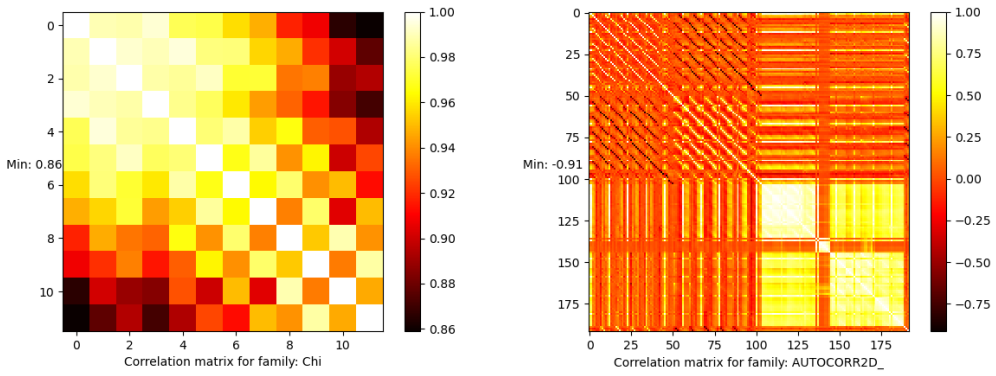


Figure 4: Correlation Heat Map

early related to the output (in the case of Pearson correlation) or had a monotonic relationship (in the case of Spearman correlation). 20 from the ligand descriptors that were most correlated to the output were selected as inputs to the model. Similar 20 best features from the protein descriptors were selected.

4.3.2 Using genetic algorithms [3]

Since the feature space is a non-continuous problem of combinatorial complexity, we also studied genetic feature selection algorithms. We represent each feature by a binary number, 1 for the inclusion of the feature and 0 for the exclusion. Let $\mathbb{B}$ be a binary number $\{0, 1\}$. Each feature selection $p \in \mathbb{B}^{456}$ (401 for ligands + 55 for proteins) is called a chromosome. A "population" of n chromosomes is maintained. For each generation, the best pairs of chromosomes are selected as parents. The next generation is created by crossover and mutation of the chromosomes. Algorithm 1 below gives complete pseudocode for this.

The scoring function used to select the best chromosome can vary according to the type of the model being fit. [See Section 6]

4.3.3 Manual Feature selection

A selected list of features were given to us by Simon Bray. We investigated manually selected 121 ligand descriptors. We continued to use all of the

Algorithm 1 Selection of features in our model using genetic algorithm [3]

procedure GENETIC_ALGORITHM_BASED_SELECTOR
    scoringfunction ← Get model specific scoring function
    $population = \{C_1, C_2, C_3 ... C_n\} \in \mathbb{B}^{456}$ (initial chromosomes).
    best ← C_1 *// Arbitrarily initialized*
    $i \leftarrow 0$
    gen ← number of generations to run.
    for $i < gen$ `with step 1` **do**
        $\{S_1, S_2, S_3 ... S_n\}$ ← scoringfunction($population$) $\forall S_i \in \mathbf{R}$.
        // Do a tournament selection for the best chromosomes
        $genetically\_better\_population$ ← empty list
        for $j < len(population)$ **do**
            Set ← random_k_selections($population$)
            $c \leftarrow best$(Set) *// Based on scoringfunction.*
            $genetically\_better\_population.add(c)$
        end for
        $children$ ← empty list
        for $j < len(population)$ `with step 2` **do**
            $P_1, P_2 \leftarrow population[j], population[j+1]$
            $c_1, c_2 \leftarrow crossover(P_1, P_2)$
            $c_1 \leftarrow mutation(c_1)$
            $c_2 \leftarrow mutation(c_2)$
            $children.add(c_1, c_2)$
        end for
        $population \leftarrow children$
    end for
    return best($population$)
end procedure

protein descriptors as they were only a few in number as compared to the ligand descriptors.

5 Testing

5.1 Reproducibility

Reproducible ML models are very crucial for verifying any project or research results. Due to the stochastic nature of many ML training processes, reproducing the exact model (and consequently the exact output) is a challenge. Two methods can be employed to produce verifiable results:

- Training many models and reporting the average results.
- Controlling the randomness of the trained models. It is done by setting the seed of the pseudo-random algorithms.

We use the second approach in our project. Every script, when executed, reports an execution ID. This is the random seed used during the execution. If we want to reproduce the exact results, we give this execution ID to the script as the first argument.

5.2 Model Quality Analysis

We are trying to predict the following function

$$\text{Binding affinity prediction} : \mathbf{R}^n \mapsto \mathbf{R} \text{ where } n \in \mathbf{I}^+$$

Since the input space is multi-dimensional, we cannot fully visualize our model as a function of the input space. To get around this using the following methods

- We report *Coefficient of determination*, $R^2 \in (-\infty, 1.0]$ where 1.0 is the best score. [10]

$$R^2(y, \hat{y}) = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2} \text{ where } \bar{y} = \frac{1}{n}\sum_{i=1}^{n} y_i \text{ and } \sum_{i=1}^{n}(y_i - \hat{y}_i)^2 = \sum_{i=1}^{n} \epsilon_i^2$$

- For visualizing the results, we plot a 2D plot of expected values against the model's output.

A perfect model would have all the points on the $y = x$ line. This corresponds to R^2 score of 1.0. Figure 5, shows the validation accuracy of a sample *Random Forest Regressor* model. *y_validate* (x axis) represents the actual validation data. *y_validate_pred* (y axis) represents the predicted output from the model.

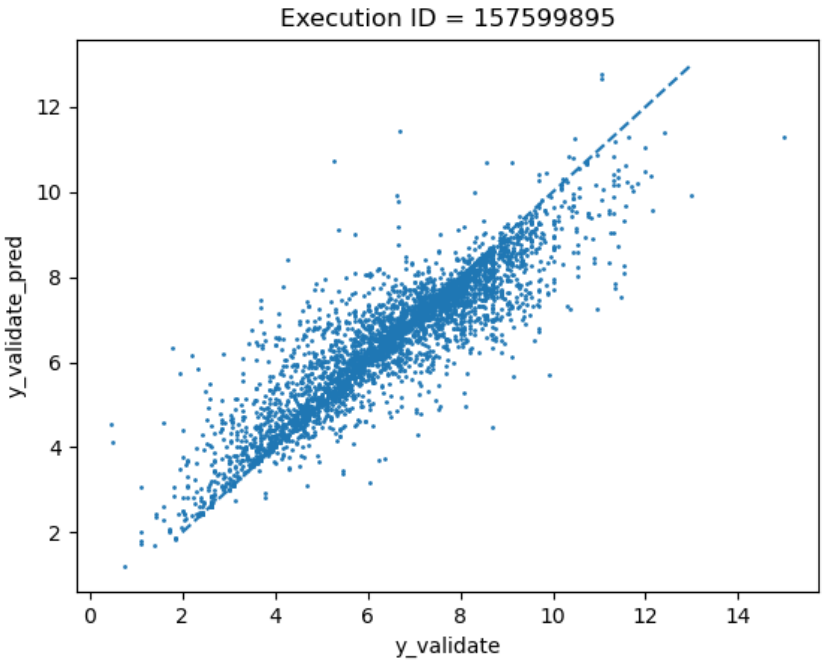


Figure 5: (Sample) Visualizing accuracy. $R^2 \approx 0.805$.

6 Machine Learning Models

We decided to use the following machine learning models for our problem:

- Linear Regression
- Support Vector Regression
- Rotation Forest Regression
- Random Forest Regression

By far, the most impressive performance was given by Random Forest Regression. Hence we spent more time on Random Forest regression to understand the reasons for its performance.

Because of the lack of data, Deep Neural Networks were not suitable for our problem. For example, if we used all the features to train a simple DNN model of size $[457, 20, 10, 1]$, there would be 9350 parameters to train. As we only have 16,000 data points to train our model, the deep learning model would over-fit drastically.

6.1 Data Preprocessing

Before fitting the model, we did a preliminary analysis of our data. This was necessary to train our models more reliably.

6.1.1 Data cleaning

Using Principle component analysis (PCA) each feature's contribution in the variation of the data was analyzed. During this analysis, 2 issues were found:

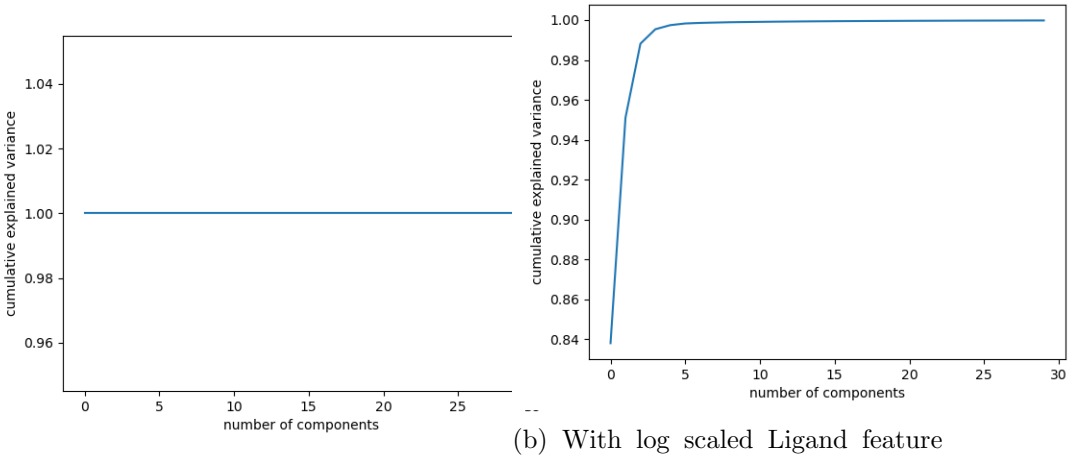
- There was a ligand feature named IPC that was having extremely small and extremely huge values e.g $k * 10^{39}$ where $k \in (0, 1]$. Hence we scaled this feature using a logarithm function. This was done for numerical safety during the training of our models. The effect of this scaling on the cumulative PCA is shown in Figure 6.

- There were a lot of NaN (Not a number) and zeros in our data. Perhaps this was because the measurements done in the experiments were not recorded completely and the RDKit feature extractor could not extract these values reliably. While the presence of 0s was harmless, we made sure that we removed all NaN values before we input the data into our model. Figure 7 shows cumulative PCA of the protein features.

It must be noted that the scaling of ligand feature IPC must be done before we use the model for the prediction of new data. This is a limitation of our model.

6.1.2 Dealing with measurement resolution

During the measurement of the Protein-Ligand complexes, the measurements can be taken at various resolutions (in Å units). The accuracy of the measurement is inversely proportional to the resolution. As shown in



(a) With original Ligand feature IPC

(b) With log scaled Ligand feature IPC

Figure 6: Cumulative PCA of ligand features

a small excerpt of the PDB Databank INDEX file, we have a resolution measurement for each of the complexes (or data points).

```
# ================================================================================
# List of protein-ligand complexes with known binding data in PDBbind v.2019
# 17679 protein-ligand complexes in total, sorted by binding data
# Latest update: Dec 2019
# PDB code, resolution, release year, -logKd/Ki, Kd/Ki, reference, ligand name
# ================================================================================
3zzf  2.20  2012   0.40  Ki=400mM      // 3zzf.pdf (NLG)
3gww  2.46  2009   0.45  IC50=355mM    // 3gwu.pdf (SFX)
1w8l  1.80  2004   0.49  Ki=320mM      // 1w8l.pdf (1P3)
```

Using the resolution, we devised the following 2 ways to calculate the weight of each data point.

- Hyperbolic formula: $W_i = \frac{\max \mathrm{R}_{1\ldots\mathrm{n}}}{R_i}$
- Linear formula: $W_i = (\max \mathrm{R}_{1\ldots\mathrm{n}} + 1) - R_i$

Where R_i is a resolution of a given data point. We found that $\max \mathrm{R}_{1\ldots\mathrm{n}} \approx 5$ Å in our data. It is necessary to add 1 to the max resolution in the linear formula to avoid data points with 0 weight. The formulae are depicted

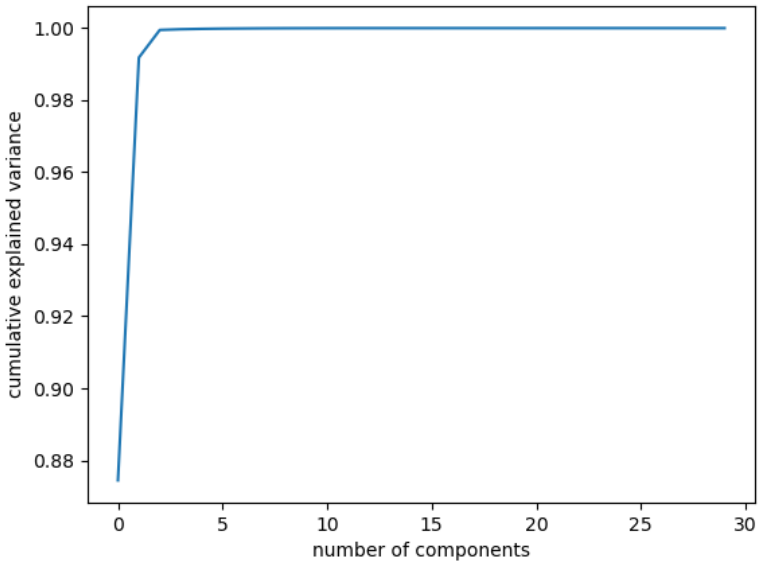


Figure 7: cumulative PCA of protein features

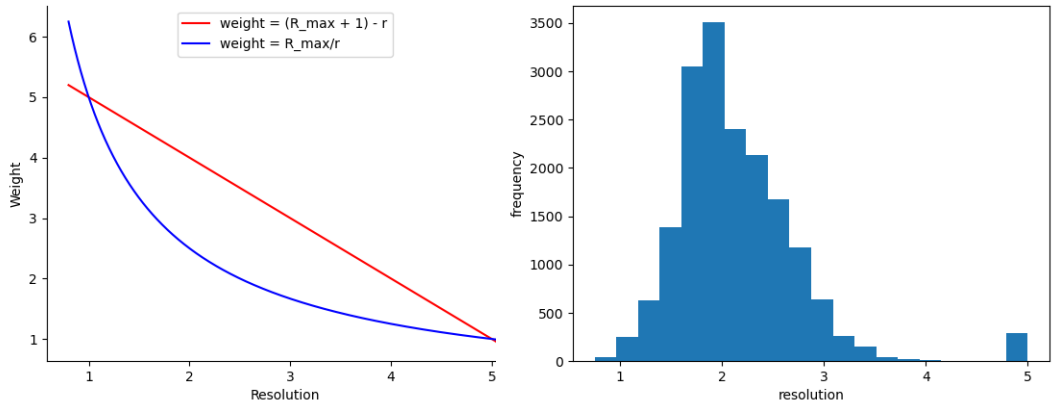
in Figure 8a. Figure 8b shows the distribution of data points as a function of resolution. The resolution of the measurements ranges from 1 to 5 approximately.

Figure 9 shows the distribution of the weights obtained from our data for the linear and hyperbolic case. In Figure 9b, we see that linear weighting gives us data points with a higher weight as compared to hyperbolic weighting. This makes intuitive sense because as shown Figure 8b, most of the data points have a resolution around 2 Å. Since linear weighting around 2 Å is higher, the linearly weighted data points will have a higher weight on average.

We employed 2 strategies to use the calculated weights in our models.

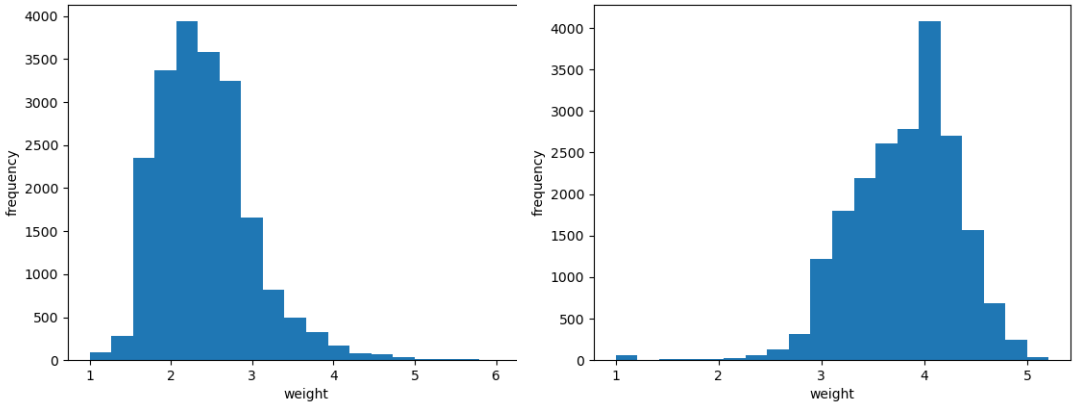
- Duplicating the data points.
- Using the weights directly in the fit function.

After the duplication, we get ≈39,000 data points for the linear weighting formula and training and ≈62,000 for the hyperbolic weighting.



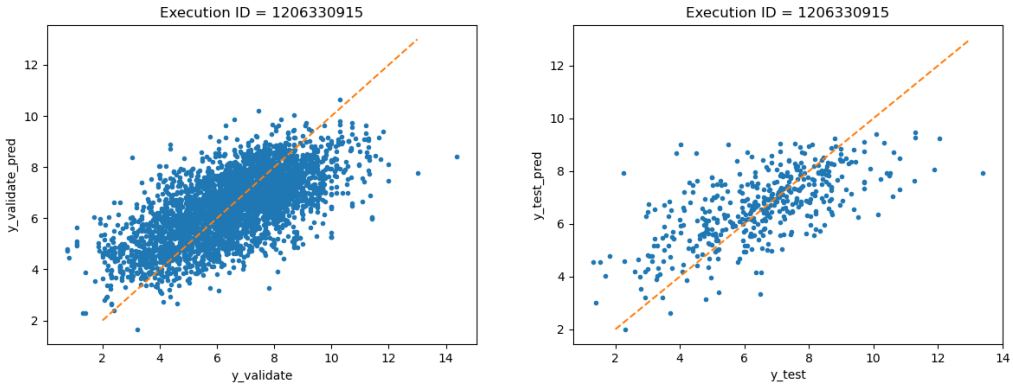
(a) Weight calculation formulae (b) Resolution distribution

Figure 8: Weight calculation and resolution distribution



(a) Hyperbolic weighting (b) Linear weighting

Figure 9: Weighting of our data points (Excluding the test set)



(a) Validation Accuracy (b) Test Accuracy

Figure 10: Best Linear Model. Uses all 457 features with Hyperbolic weighting.

6.2 Linear regression

Linear regression fits a linear model to a given data. It tries to minimize the square of errors between the predicted output value and the actual output value. This is the cheapest model (computationally) that we used to fit our data. Table 1 shows the overview of our analysis with this model. Figure 10 shows the best results obtained with linear regression.

Features Selected	**Training**	**Validation**	**Testing**
457 (all)	0.456	0.433	0.412
457 (Hyperbolic)	**0.452**	**0.431**	**0.420**
457 (Hyperbolic duplication)	0.465	0.424	0.419
457 (Linear)	0.455	0.430	0.416
457 (Linear Duplication)	0.460	0.428	0.407
49 (Genetic + Hyperbolic)[1]	≈0.373	≈0.393	≈0.402
40 (Pearson + Hyperbolic)	0.288	0.276	0.286
40 (Spearman + Hyperbolic)	0.291	0.280	0.289
176 (manual + Hyperbolic)	0.364	0.342	0.389

Table 1: R^2 scores of the Linear Regression Model

[1] Approximately reproducible as the reproducibility module was introduced later.

The reason for the low R^2 score is that linear models assume strong linearity between the input variables and the output variable. This is not always true. One anomaly in the results is that there are some cases where the validation and testing results seem to be better than the training result. However, this is not always the case. The results highly depend on the seed. If the seed changes, the train/test data split gives us different data for training and different data for validation. What is important for the validity of our model is that the results remain in a reasonable range.

In the linear regression model, we used genetic algorithms to reduce both the large feature space and to eliminate correlated features within the families.

6.2.1 Score function of genetic feature selection

Since the fitting of the linear regression model was very cheap, we could afford to refit the model every time in our genetic algorithm. We had 2 objectives at hand

- Get the best performing model.
- Reduce the number of input features to make our model simple and explainable.

The above objectives are not always against each other. This is because removing a feature that has no correlation (or random correlation) with the output may improve the model whereas removing a feature that the output is highly correlated on degrades the model.

Hence, taking inspiration from the hypervolume-based multi-objective optimization, we designed the following score function.

$$\text{score} = \mathbf{R}^2\text{score} * \text{Features Eliminated}$$

Figure 11 gives us an intuition about this function. Here the score function represents the area of the square formed between a given point and the origin. Trying to improve the score means, it will try to eliminate more features as well as try to get a higher R^2 score. In the example below, the point a is preferred over b as it gives almost the same R^2 score and eliminates a larger number of features. Another advantage of this score function is that the scale of both axes is irrelevant.

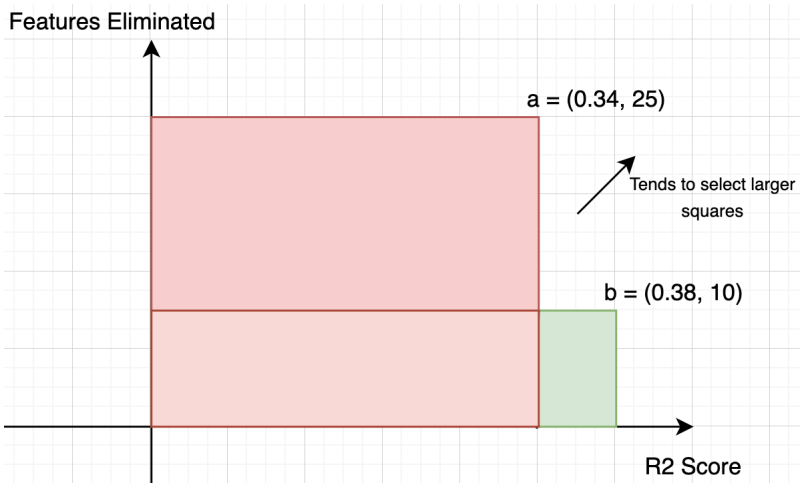


Figure 11: Genetic Algorithm score function representation.

6.2.2 Initialization strategies of population

Two main strategies were used to initialize the population for the genetic algorithm

- **Random Initialization**. A population of 2400 chromosomes was randomly initialized and run for 500 generations.
- **Specific Initialization**. Here, the initial population was kept at 457. The algorithm was run for 2000 generations. Each chromosome in the population had 1 distinct feature excluded. The following represents the initialization of the chromosomes.

  $c_1 = [01111.....11]$
  $c_2 = [10111.....11]$
  $c_3 = [11011.....11]$
  ...
  $c_{457} = [11111.....10]$

  The rationale was that using the above score function we could go to a local minimum by eliminating the worst performing features rather than trying to select the best performing features as in the case of random initialization.

Both the initialization strategies gave similar results as shown in Table 1

6.3 Random Forest Regression

Random forest regression is an ensemble ML model of regression trees. When training each tree, the algorithm finds out which feature value can divide the data into 2 groups to yield the lowest sum of squared values on both sides (The criterion can be different as well e.g. lowest absolute error). Then each of the sub-data is split recursively till a stopping criterion (e.g a set number of data points in the leaf).

For each tree, a subset of the data is used for training. The subset of data is sampled with replacement a.k.a bagging. After training, the bunch of "Experts" that are good at different data subsets predict the output of new given inputs. The average of the predicted outputs is the result of the whole random forest regressor.

This model is non-linear. One advantage is that there is no need for any assumption about the data. Moreover, random forests can handle categorical features together with the real-valued features very easily. On the negative side, the function represented by the ensemble cannot be easily represented and has to be interpreted as a black-box function.

Figure 12 shows the validation and testing accuracy obtained by a random forest with manually selected features. This model outperforms the other features selection strategies.

Table 2 gives an overview of our model. The table shows that features selected by output correlation (Spearman and Pearson correlations) seem to generalize well to the test data. However, we cannot consider this as a good model because the validation error and the test error are not in agreement with each other. Moreover, the correlated feature selection assumes that the data features have a strong linear or monotonic relation with the output which may not be correct in our case. Hence, we do not consider these models as the best models.

6.3.1 Dealing with correlated features

In Section 4.2 we discussed that within some of the families of features there are heavily correlated features. One way to deal with this would be to just take one feature from a family and exclude the rest. However, this would not be a good strategy for generalization. This is because in the future if this one feature is not measured, our model would be rendered useless.

For this reason, we made use of the stochasticity of the random forest to our advantage. During the creation of a regression tree, the random forest model tries to get the best (feature, value) tuple to divide the data. This

Features Selected	Training	Validation	OOB score	Testing
457 (all)	0.961	0.790	0.793	0.540
457 (Hyperbolic)	0.961	0.778	0.796	0.555
457 (Linear)	0.960	0.800	0.790	0.542
40 (Spearman)	0.930	0.677	0.672	0.880
40 (Pearson)	0.925	0.651	0.646	0.870
229 (Genetic Elitism)	0.958	0.791	0.792	0.537
394 (Genetic Normal)	0.960	0.808	0.791	0.543
176 (manual)	**0.947**	**0.737**	**0.734**	**0.579**

Table 2: Random Forest Regression R^2 Score table

is known as the split criteria. We force it to randomly select only 20% of the features for deciding on the split criteria. More specifically, we use the following values

- 400 Trees (n_estimators)
- 20% of feature selection. (max_features)
- A minimum of 2 data points for each leaf. (min_samples_leaf)

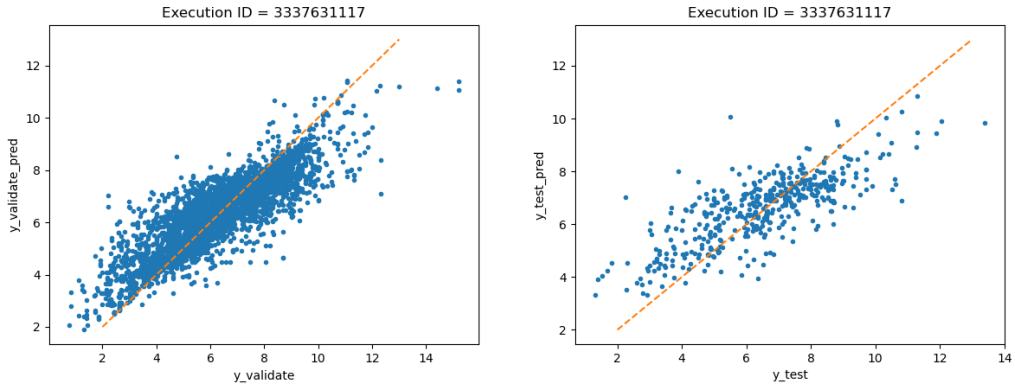
These hyperparameters were determined after some empirical testing. We increased the number of estimators as we increased the stochasticity of our ensemble. Our model would hence be dependent on the entire family of features and not rely on a single feature heavily.

6.3.2 Dealing with measurement resolution

We trained our model by duplicating the data according to their weights. After training we got the following results - Training R^2 score = 0.995, Validation R^2 score = 0.8298 and Out Of Bag (OOB) score = 0.977.

We found the following issues with this strategy:

- The out-of-bag error (OOB score) is erroneously high here. This is because of repeated data points. Many points are both inside the selected bag and outside it.
- Due to duplication of data points, the trees may end up being more correlated to each other. This will affect the generalization of our model.



(a) Validation Accuracy (b) Testing Accuracy

Figure 12: Random Forest Regressor with 176 manually selected features and hyperbolic weighting.

- As training this model is costly, duplicating the data points further increases the cost of training.

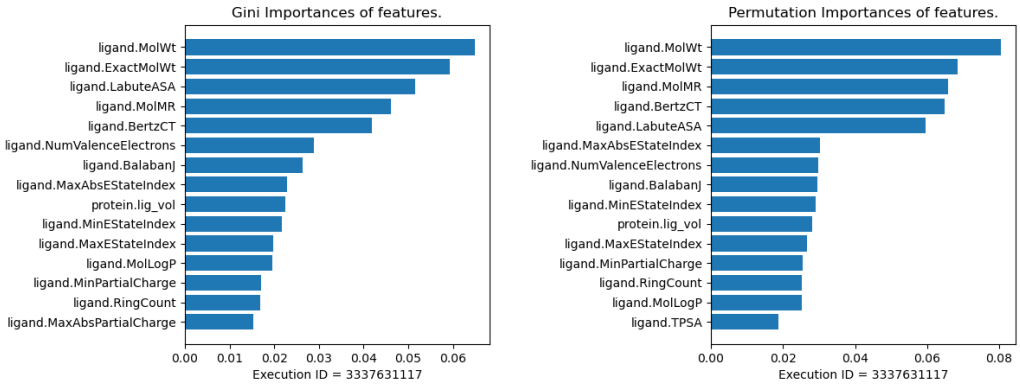
Hence, we conclude that duplication of the data based on their weights is a wrong strategy for this model.

When we used our weight directly in the fit function, we could not get any noticeable improvement in our model. We think this is because the data is already skewed towards the ≈2 Å data points. This can be seen in Figure 8b

6.3.3 Feature Importance calculation

The Random Forest regression model fitting is very expensive as compared to the simple linear regression. Hence the same strategy cannot be used for feature selection as used in the linear models. We used the following strategies to determine the importance of features in the model.

- **Gini Importance (or) Decrease in impurity**. This is calculated by the model itself. For every feature, it calculates how much decrease in the split criterion the feature contributes in the entire ensemble (i.e in every node it's used in all the trees). The disadvantage is that they cannot be reliable when the features have a high cardinality. Figure 13a illustrates this importance.



(a) Gini importance

(b) Permutation importance

Figure 13: Feature Importance calculation of Random Forest Regressor. (With manual feature selection)

- **Permutation Importance** - This is a model agnostic method. It tries to calculate the reduction in the accuracy we shuffle each feature. The importance of the features is directly proportional to the amount of reduction in the accuracy. The disadvantage is that since each feature is checked independently, correlated input features make the results unreliable. If feature A and feature B are correlated, the shuffling of A may not impact the model as the model can rely on B for its prediction. Figure 13b illustrates this importance.

6.3.4 Permutation Importance and genetic algorithm

To overcome the issue with the above feature importance selection, we used a combination of permutation importance and genetic algorithm to find out the importance of the selected features. This is because it is very expensive to refit the model and cheap enough to check the results of shuffling the model.

We used 2 different strategies to get them for our genetic algorithms -

- **Using specific scoring function**. In Section 6.2.1, we discussed intuition behind the scoring function used in our genetic model. In the random forest model, however, this scoring function selected a lot of features (> 400). Hence we researched the following scoring function

in an effort to reduce the number of features.

$$\text{score} = \mathbf{R}^2 * \text{Features Eliminated}^2$$

We used a stronger signal for the number of eliminated features. This scoring function, however, turned out to be too strong and non of the features were selected.

- **Elitism**. Elitism is a strategy used in the genetic algorithms to select the top few elements from each generation for the next generation in addition to the newly created children. This keeps the best performing subset of the population always in the current generation in an effort to improve them further by cross-over or mutation.

6.4 Support Vector Regression

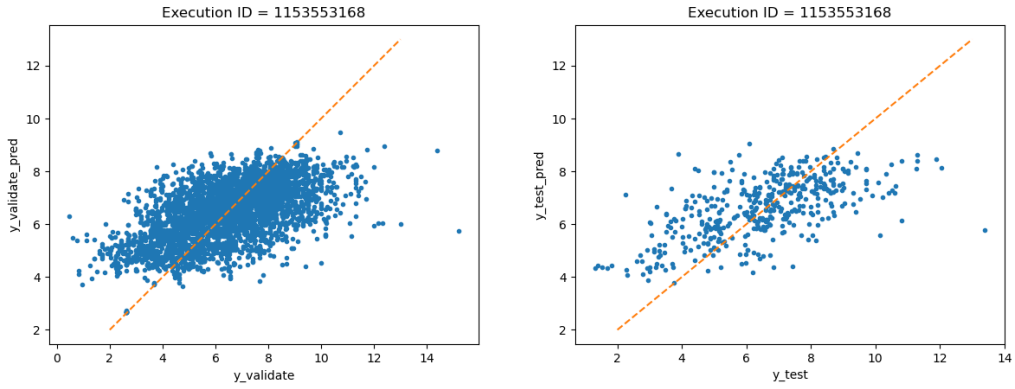
Support vector regression is a non-linear regression method that fits a curve such that the margin of error between the model's prediction and the original value is kept within a minimum range. Any error higher than this minimum range is penalized during the training of the model. We trained support vector regression with the RBF kernel.

Features Selected	Training	Validation	Testing
457 (all)	0.310	0.308	0.356
457 (Hyperbolic)	0.325	0.326	0.371
457 (Linear)	**0.336**	**0.334**	**0.374**
40 (Spearman + Linear)	0.256	0.286	0.262
40 (Pearson + Linear)	0.199	0.200	0.199
40 (Spearman + Hyperbolic)	0.252	0.266	0.254
40 (Pearson + Hyperbolic)	0.169	0.169	0.169
176 (manual)	0.306	0.298	0.358

Table 3: Table showing R^2 Scores for SVR

We did not use the duplication of data strategy for this model because the training time complexity is $O(n^2)$ where n is the number of data points. However, we did use data point weights during the fit function of the training.

The results are reported in Table 3. The linear weighting of the data gave the best results in SVR. The reason testing R^2 score is around the same range as the training score maybe because the testing data is of better



(a) Validation accuarcy

(b) Testing accuracy

Figure 14: SVR accuracy visualization for all features 457 and Linear weighting

quality data as compared to the validation data. However, this varies very highly depending on the random seed used. What we must observe is that the approximate range in which the values are obtained is small.

As the performance of the model was not on par with other models, we did not run the costly genetic algorithm feature selection on SVR. Moreover, the time it took to get the R^2 score (validation and testing) was very high and was not suitable to be used in a permutation-based genetic algorithm as discussed in the random forest section. The issues of within family feature correlation as well as feature selection were hence dealt with by either using output correlation or by manual feature selection.

Figures 14a and 14b visualize the validation and testing accuracy of the best SVR model.

6.5 Rotation Forests

In Section 6.3 we analyzed different strategies to improve the performance of our model. While fitting a tree, the random forest model tries to get a (feature, value) tuple that best divides our data. The criterion is to reduce the entropy of the divided data. This is done recursively till some set criteria (E.g., the leaf nodes should contain 7 data points). However, each (feature, value) only represents an axis-aligned hyperplane.

If there are some real-valued features in our data set, we could reduce the complexity of our tree by using a hyperplane that is not axis aligned.

For example consider approximation of the line $y = x$. To approximate this using an axis-aligned hyperplane, we would require a deep tree of high complexity. If instead, we transform our data such that the line $y = x$ transforms to $y = k$ where $k \in \mathbf{R}$, we only require 1 (feature, value) tuple.

This can be accomplished by calculating the eigenvector of our data and transforming our feature space such that the basis vectors are the eigenvectors of the data. Rotation forests use this concept to reduce the complexity of their trees and to get a better quality model. They were first proposed for classification. We adapted the implementation for the regression case.

The training R^2 score was comparable to that of the random forest regressor. However, we did not see much improvement in the accuracy. One of the reasons, we hypothesize, is that rotation forests are shown to be better than random forests only for the continuous feature values. However, our data set has both real-valued and discrete values, although the discrete values are encoded as the real-values for our model. Hence we conclude that this model is not the best model for our problem.

Table 4 gives an overview of the performance of Rotation forests. The implemented model does not have an option to train weighted data. We do not consider the models trained by using features selected by output correlation good models for similar reasons as discussed in Section 6.3.

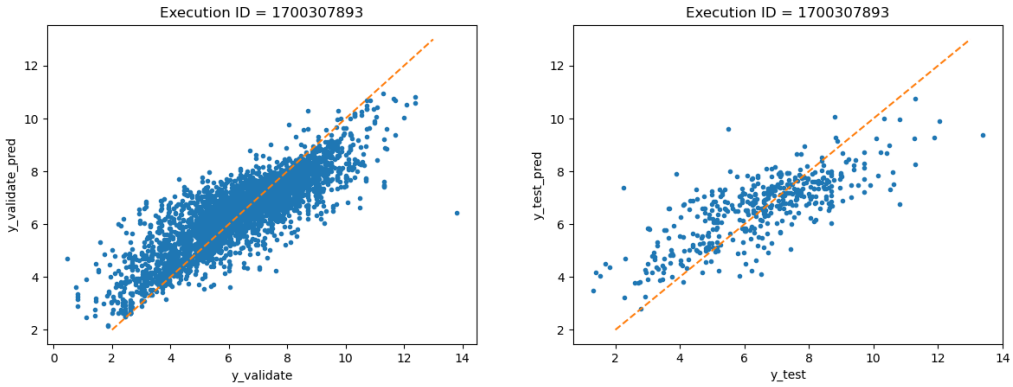
Features Selected	**Training**	**Validation**	**Testing**
457 (all)	0.967	0.756	0.543
457 (Hyperbolic)	NA	NA	NA
457 (Linear weighting)	NA	NA	NA
40 (Person)	0.948	0.596	0.878
40 (Spearman)	0.951	0.667	0.896
176 (manual)	**0.960**	**0.724**	**0.572**

Table 4: Rotation Forest R^2 Score overview

Figures 15a and 15b visualize the validation and testing accuracy of the best Rotation Forest Model.

7 Discussion

In this project, we tried to determine the protein-ligand binding affinity by using spacial features extracted by RDKit and D-Pocket. We believe that 2D/3D features made our data more accessible to simple machine learning



(a) Validation accuarcy

(b) Testing accuracy

Figure 15: Rotation Forest Accuracy visualization for manually selected features (176). The rotation forest implementation does not support data weighting.

models. It was crucial because the complex models that directly get the 3D features from the data (E.g., 3D convolutions) require a lot of data which is expensive to accumulate.

We spent most of the time analyzing and studying the most suitable linear model (Linear Regression) and non-linear model (Random Forest Regression) for our extracted data. Our study also included feature selection methods like genetic algorithms. We found that our extracted protein-ligand data had the following interesting properties

- The data does not have a symmetric Gaussian distribution. In other words, the ranges of values in each dimension were very different.
- The features are both real-valued numbers as well as discrete. The discrete values are also represented as real numbers.
- There are feature families in the ligand descriptors that have highly correlated features.

While Linear Regression did give reasonable results, it was not a very reliable model because it assumed that the data was linearly related to the binding affinity. Nevertheless, we were able to run genetic algorithms on this model to get the best ≈ 50 features. We failed to do this in the case of the Random Forest Regression model.

From our study of genetic algorithms, we conclude that it was not very helpful in improving the performance of our model. However, we were able to reduce (eliminate) around 200 features for the Random Regression Model and around 400 features in the Linear Regression without hampering the performance to a large degree. One issue we found with genetic algorithms is that they make the model very reliant on the selected features. This hard constraint may not be good when the data collection process is expensive and error-prone.

We found that the Random Forest Regression could deal with all the above specific properties of our data. Random Forest Regression deals with both Real valued and discrete input features because the main aim of a fitting tree (in the random forest) is to reduce the entropy of the divided data. Since all the discrete features in our data were extracted as real-valued numbers, the trees can easily find a midpoint between the discrete values to divide the data. Interestingly, we could also use the feature correlation within families to our advantage to improve the robustness of our model. By increasing the randomness of our feature selection during the data split we force our model to rely less on a specific feature. Hence a wrong measurement or a corrupted feature would not hamper our model significantly.

Since we are getting an R^2 score of > 0.5 consistently, the binding affinity predicted by our model can be used for any new PL complex can be taken as a reference for drug-testing decisions. For example, one can use the results to prioritize testing of new ligands in a wet lab experiment to save resources for the important ligands.

As with any model, our model and approach do have limitations. Firstly, Random Forest Regression models cannot be explained easily. With hundreds of trees in our random forest, the model at best can only be treated as a Black Box. Because of its complexity, we were not able to show why some features were very important as compared to the others (E.g., in the feature importance calculation). Secondly, our model relies heavily on the ligand features. The model, hence, assumes that the binding only slightly depends on the protein features. On the contrary, the protein and ligand features are both equally responsible for the binding affinity. Also, we could not perform hyper-parameter optimization, in conjunction with the genetic feature selection algorithms as it is prohibitively expensive.

8 Conclusion

In this project, we studied methods and preprocessing techniques to predict the protein-ligand binding affinity. In conclusion, we suggest using Random Forest Regression for this problem because of the ease of training and the generalization capability. We believe that a random forest regression could be used as a baseline to compare any new models.

In further work, we believe that there is scope for improving the feature selection process of our model. Our model depends heavily on the ligand features as compared to the protein features. We believe it is because of the data extracted by RDKit and D-Pocket. Further investigation on new models that do not have this limitation would be helpful. One way could be to build models with all protein features and one family of ligand features. The most important features in each family may can as surrogates for the whole family. We also think more study needs to be conducted to create explainable models. It would be both accepted by the Cheminformatics community and would be helpful to further the understanding of binding affinity.

References

[1] Protein Data Bank. Example Protein - 2Y07. Link. [Online; accessed 1-Aug-2021].

[2] Uni-Freiburg) Björn Grüning (Department of Bioinformatics. Galaxy Tool wrappers. Link. [Online; accessed 1-Aug-2021].

[3] Jason Brownlee. Simple Genetic Algorithm From scratch. Link. [Online; accessed 28-June-2021].

[4] TMP Chem. Computational Chemistry 1.2 - PDB File Format. Link. [Online; accessed 22-July-2021].

[5] Vincent Le Guilloux and Peter Schmidtke. fpocket User Manual. Link. [Online; accessed 2-Aug-2021].

[6] Abdus Salam Khazi. Code for the whole project. Link. [Online; accessed 22-July-2021].

[7] PDBank. PDBank History. Link. [Online; accessed 22-July-2021].

[8] PDBank. PDBank Homepage. Link. [Online; accessed 22-July-2021].

[9] RDKit. RDKit: Open-Source Cheminformatics Software. Link. [Online; accessed 1-Aug-2021].

[10] scikit learn. R2 score, the coefficient of determination. Link. [Online; accessed 22-July-2021].

[11] The Science Snail. Difference between Ki, Kd, IC50 and EC50 values. Link. [Online; accessed 22-July-2021].

[12] Wikipedia. Docking (molecular). Link. [Online; accessed 1-Aug-2021].

[13] Wikipedia. PDBbind database. Link. [Online; accessed 22-July-2021].

[14] Wikipedia. Protein Data Bank (file format). Link. [Online; accessed 22-July-2021].

[15] Wikipedia. Protein–ligand complex. Link. [Online; accessed 24-June-2021].

[16] Wikipedia. Simplified molecular-input line-entry system. Link. [Online; accessed 1-Aug-2021].

[17] Wikipedia. Structure Data File format. Link. [Online; accessed 1-Aug-2021].

[18] Wikipedia. XYZ Format. Link. [Online; accessed 22-July-2021].

[19] Gaming World. Procedural Terrain Generation with Unity : What is Voronoi Tessellation. Link. [Online; accessed 2-Aug-2021].

9 Appendix

9.1 XYZ File format

The following represents the pyridine molecule in the XYZ format.

```
11

C       -0.180226841      0.360945118     -1.120304970
C       -0.180226841      1.559292118     -0.407860970
C       -0.180226841      1.503191118      0.986935030
N       -0.180226841      0.360945118      1.29018350
C       -0.180226841     -0.781300882      0.986935030
C       -0.180226841     -0.837401882     -0.407860970
```

```
H       -0.180226841      0.360945118     -2.206546970
H       -0.180226841      2.517950118     -0.917077970
H       -0.180226841      2.421289118      1.572099030
H       -0.180226841     -1.699398882      1.572099030
H       -0.180226841     -1.796059882     -0.917077970
```

9.2 SDF File format

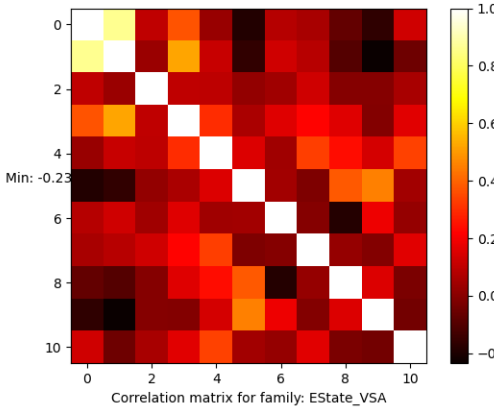
```
2uzn_ligand

Created by X-TOOL on Fri Nov 18 14:55:27 2016
 37 39  0  0  0  0  0  0  0  0999 V2000
    7.1480   60.4530    6.6830  O 0  0  0  1  0  1
    6.0470   60.1670    7.5640  S 0  0  0  1  0  4
    .......
    .......
   -2.6338   67.4589    8.1225  H 0  0  0  1  0  1
  1  2  2  0  0  2
  2  3  2  0  0  2
  .......
  .......
 23 37  1  0  0  2
M  END
> <MOLECULAR_FORMULA>
C15H13N3O4S2

> <MOLECULAR_WEIGHT>
363.3

> <NUM_HB_ATOMS>
7

> <NUM_ROTOR>
1

> <XLOGP2>
1.31
```

9.3 MOL2 File Format

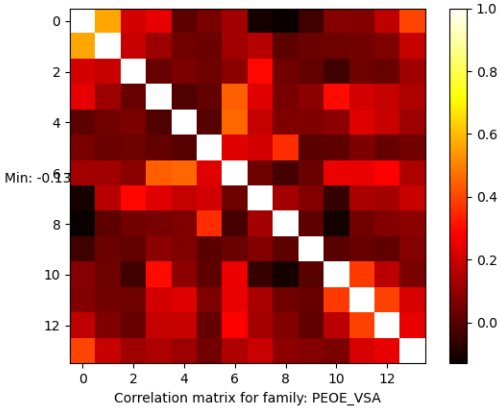
```
###
### Created by X-TOOL on Fri Sep 26 17:34:18 2014
###

@<TRIPOS>MOLECULE
1fo2_ligand
   25    25     1     0     0
SMALL
GAST_HUCK


@<TRIPOS>ATOM
      1 C4          39.0090   40.2680   25.5130 C.3       1 DMJ          0.1280
      2 O4          39.2170   40.5810   26.8980 O.3       1 DMJ         -0.3835
     .......
     25 H14         38.0787   41.8134   21.3802 H         1 DMJ          0.2097
@<TRIPOS>BOND
     1    1    9 1
     2    1    3 1
    .......
    25   11   25 1
@<TRIPOS>SUBSTRUCTURE
     1 DMJ          1
```

9.4 Correlation Heat Maps

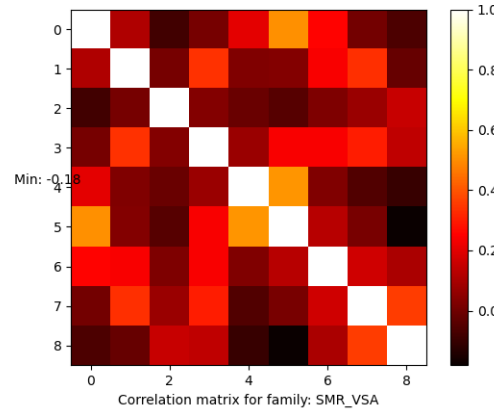
This section shows the correlation heat maps of the most interesting feature families in the our dataset.



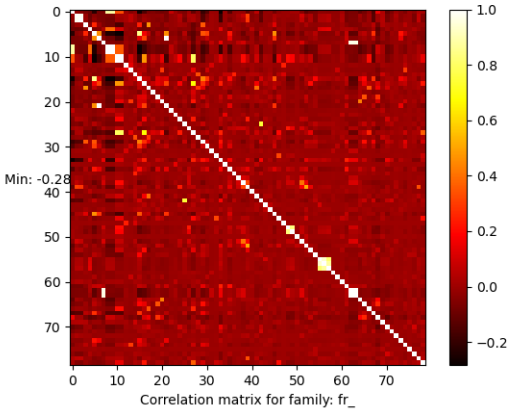
(a) Family EState_VSA



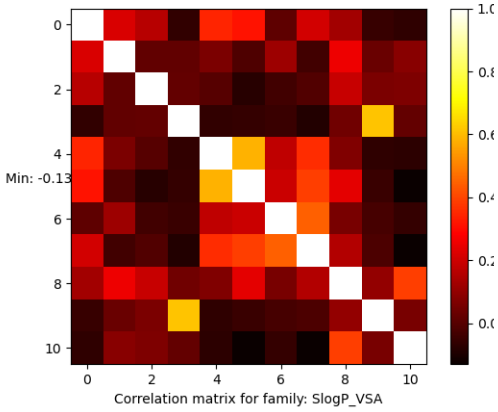
(b) Family PEOE_VSA



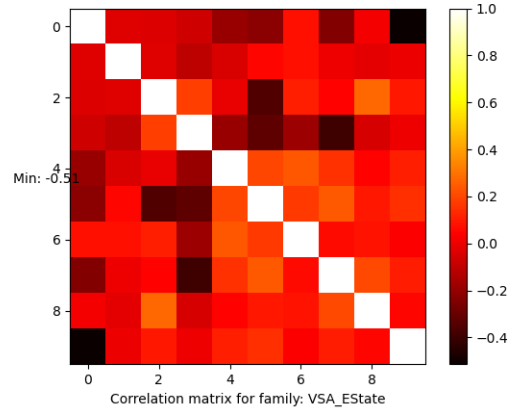
(c) Family SMR_VSA



(d) Family fr_



(e) Family SlogP_VSA



(f) Family VSA_EState

Figure 16: Correlation Heat Maps