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 26, 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.

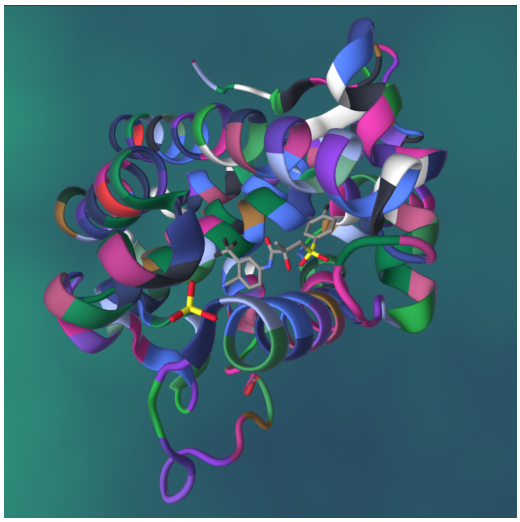


Figure 1: Haemoglobin transporter protein. [15]

When a target drug candidate is found, one has 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.

This project aims to answer one of the above questions - How well does a given drug bind to the target protein? This is determined computationally by building a machine learning model that trains on the previous data. Using this model, the goal of the project is to reduce drug discovery costs.

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:

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

The binding affinity K_d can be quantified 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 this project does not deal with) [7]

As the protein-ligand binding affinity is studied here, only the relevant data should be filtered out from **Protein Data Bank**. This is already done by the maintainers of the **PDBBind Data Bank**. [13] Using the curated protein-ligand affinity data present in the PDBBind Data bank, a machine learning model that learns to predict the affinity is built.

3 Problem Formulation

3.1 Problem Overview

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

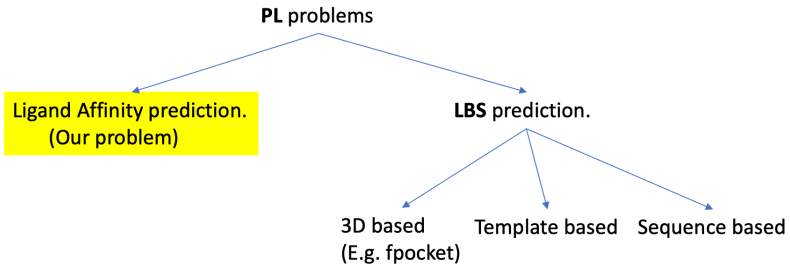


Figure 2: Protein-Ligand problem classifcation.

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

The other parts of the protein are not involved in the binding process directly. Hence, only the features of the binding location were obtained. Any potential binding location in the 3D structure of a protein is called a pocket. A 3D-based LBS prediction package called *fpocket* was used 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 the model, the features of proteins and ligands were kept distinct till the training phase. That is, the combined features were not used in any pre-processing of the ML pipeline e.g. feature reduction. This helps the model to provide the binding affinity between any protein and any ligand.

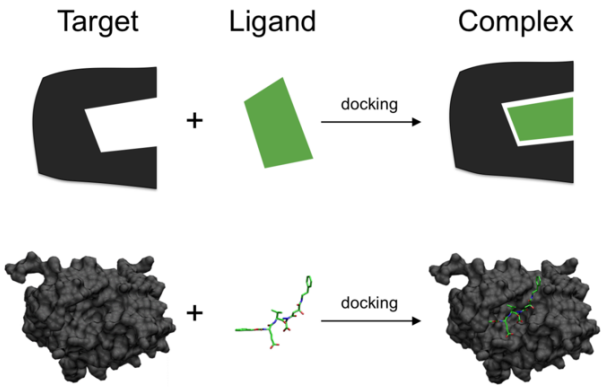


Figure 3: Lock and Key hypothesis in molecular docking. [12]

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 a 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 the 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. The ligands given in this format were used 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, an atomic graph representation could be created. Using this graph the SMILES string for the molecule can be generated. The SMILES format itself is not very helpful for us as one would 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, different tools were used 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* 402 features were extracted 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, the descriptors of all pockets in the protein could be extracted. For every pocket, 55 descriptors were obtained in total. They were 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, *Dpocket* was used. *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 the project, dpout_fpocketnp.txt files were not used as they contain the non-binding pockets. In the other 2 files, the pocket descriptors that were given by dpout_fpocketp.txt were preferred 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 their descriptors were concatenated to get a high dimensional $\mathbf{R}^{457}$ input for the model. There are a couple of issues with using all of the descriptors.

- The amount of data is not very large. For example, only $\approx$ 35000 data points could be obtained 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 some methods to reduce the input dimensions are required.

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 the correlation matrix of 2 of the most interesting families AUTOCORR2d_ and Chi. As can be seen in the heatmap there is a heavy correlation within the families. Hence a strategy to deal with this is also required. The correlation heat maps for the other families are illustrated in Section 9.4.

4.3 Feature Selection Strategies

The features of proteins and ligands were selected separately. This helps us make the 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 one 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, a few feature selection strategies that were considered in this project are discussed.

4.3.1 Selection by output correlation

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

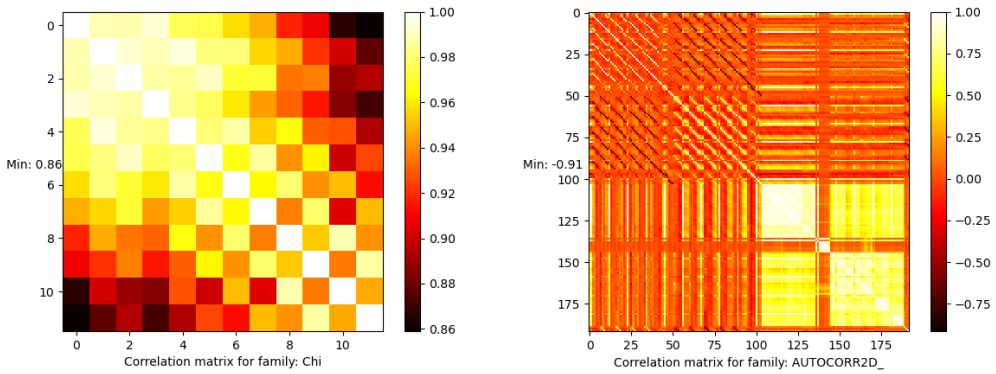


Figure 4: Correlation Heat Map

linearly 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

Since the feature space is a non-continuous problem of combinatorial complexity, genetic feature selection algorithms [3] were also studied. Each feature was represented by a binary number, 1 for the inclusion of the feature and 0 for its 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 model type. [See Section 6]

4.3.3 Manual Feature selection

A selected list of 121 ligand descriptors was given by Simon Bray. All protein descriptors were used as they were only a few in number.

Algorithm 1 Selection of features for the 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

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.

The second approach was used in this project. Every script, when executed, reports an execution ID. This is the random seed used during the execution. The exact results can be reproduced by using this execution ID as the first argument to the script.

5.2 Model Quality Analysis

The following function is predicted by the model

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

Since the input space is multi-dimensional, one cannot fully visualize the model as a function of the input space. To get around this,

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

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

- A 2D scatter plot of expected values vs the model's output is plotted.

A perfect model would have all the points on the $y = x$ line. This corresponds to a 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.

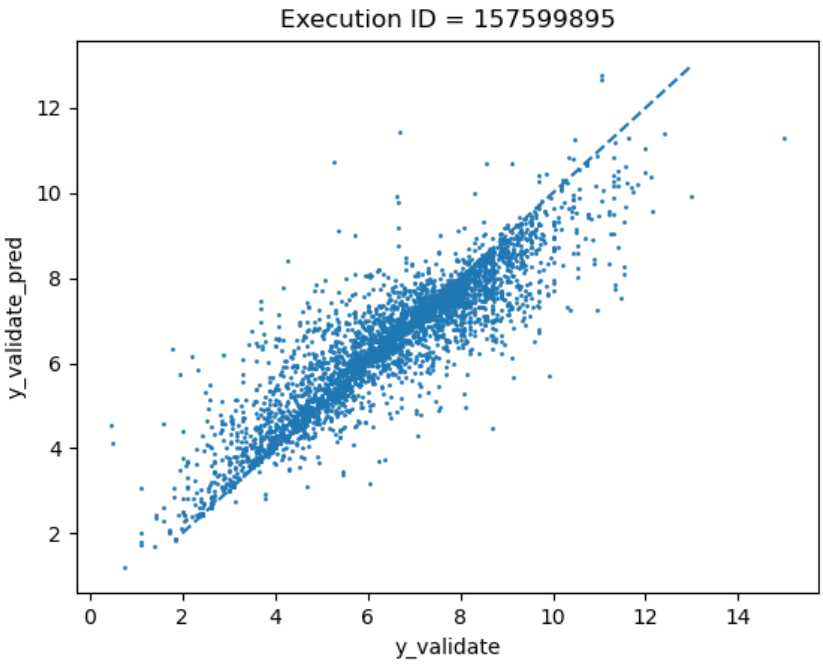


Figure 5: (Sample) Visualizing accuracy. $R^2 \approx 0.805$.

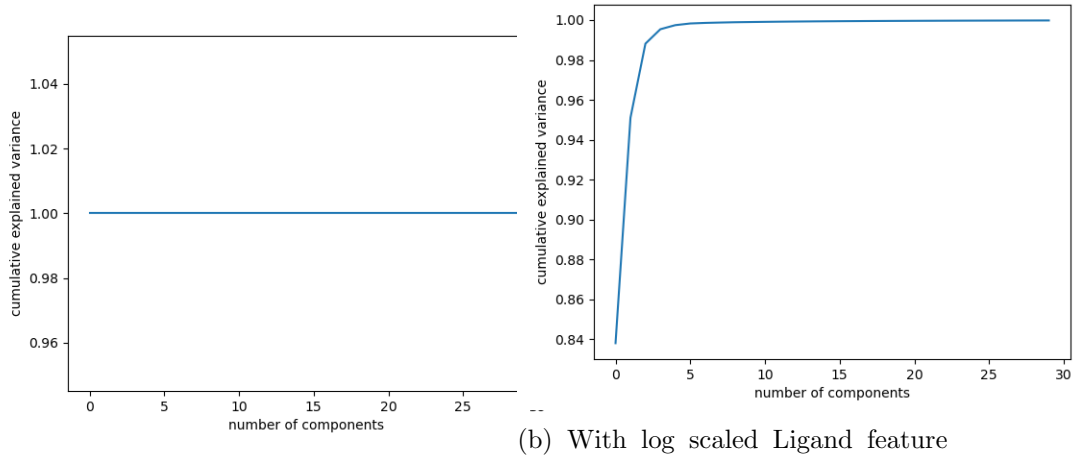
6 Machine Learning Models

The following machine learning models were used:

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

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

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



(a) With original Ligand feature IPC

(b) With log scaled Ligand feature IPC

Figure 6: Cumulative PCA of ligand features

6.1 Data Preprocessing

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

6.1.1 Data cleaning

Using principal 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 which had extremely small and extremely huge values e.g $k * 10^{39}$ where $k \in (0, 1]$. Hence, this feature was scaled using a logarithm function. This was done for numerical safety during the training of the models. Figure 6 illustrates the effect of this scaling on the cumulative PCA.

- There were a lot of NaN (Not a number) and zeros in the data. Perhaps these descriptors did not make sense for some of the ligands. As the presence of 0s was harmless, they were not changed in the input data. However, the NaN values were removed before the data was input into the model. Figure 7 shows cumulative PCA of the protein features.

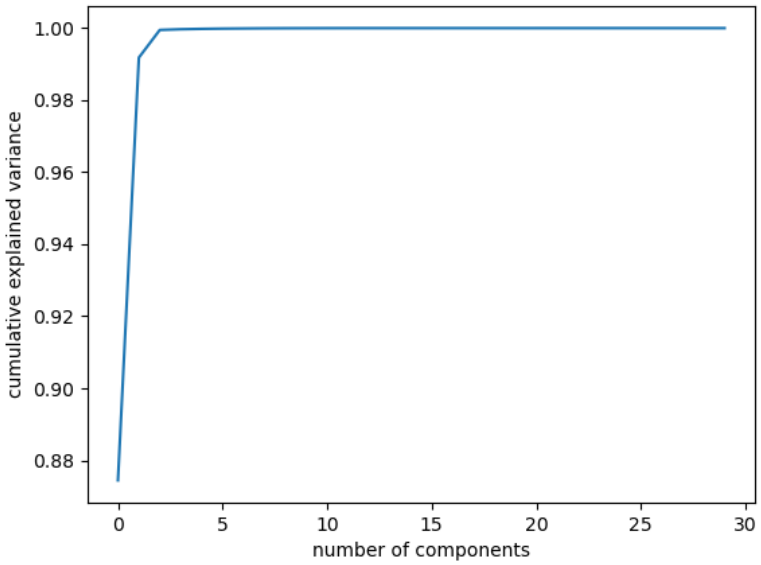


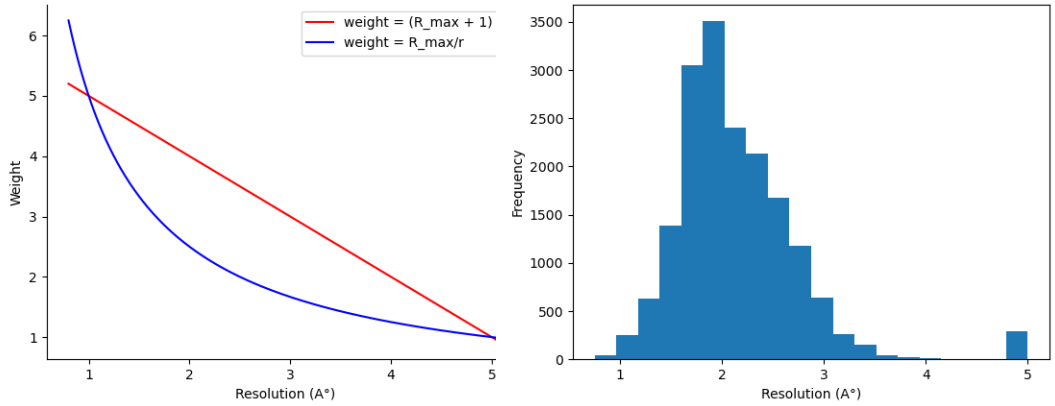
Figure 7: cumulative PCA of protein features

The ligand feature IPC must be scaled before it is input into the model. This is a limitation of the models studied in this project.

6.1.2 Dealing with measurement resolution

Atomic structures produced using X-ray crystallography always have a particular resolution associated with them (Å units). The structural detail of the 3D image is inversely proportional to the resolution. As shown in a small excerpt of the PDB Databank INDEX file, a resolution measurement for each of the complexes (or data points) is also present in the data.

```
# ==============================================================================
# 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, -log Kd/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)
```



(a) Weight calculation formulae (b) Resolution distribution

Figure 8: Weight calculation and resolution distribution

```
1w8l  1.80  2004   0.49  Ki=320mM      // 1w8l.pdf (1P3)
```

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

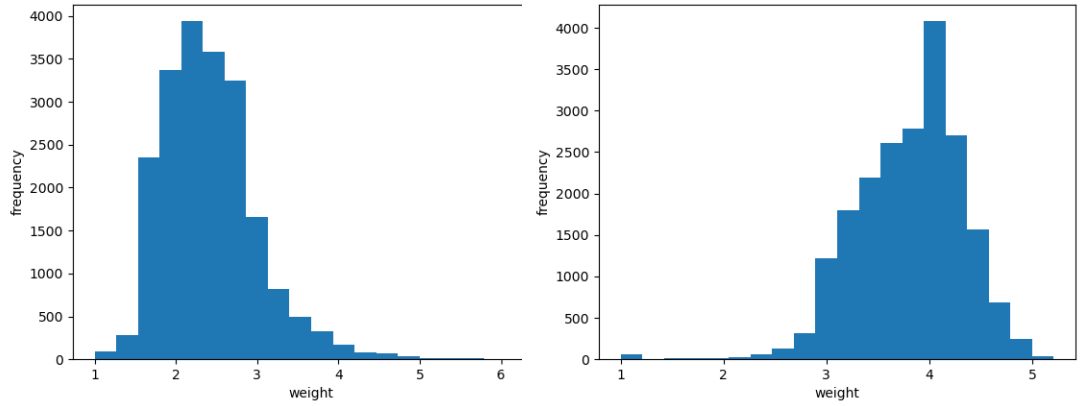
- Hyperbolic formula: $W_i = \frac{\max \mathrm{R}_{1\ldots\mathrm{n}}}{R_i}$
- Linear formula: $W_i = (\max \mathrm{R}_{1\ldots\mathrm{n}} + 1) - R_i$

Here, R_i is a resolution of a given data point. It was found that $\max \mathrm{R}_{1\ldots\mathrm{n}} \approx 5$ Å in the data. 1 was added to the max resolution in the linear formula to avoid data points with 0 weight. Figure 8a depicts the formulae. Figure 8b shows the distribution of data points as a function of resolution. The resolution of the measurements ranges from 1 Å to 5 Å approximately.

Figure 9 shows the distribution of the weights obtained from the data for the linear and hyperbolic case. Figure 9b shows that linear weighting of data points results in data points with a higher weight as compared to hyperbolic weighting. This makes intuitive sense. As shown in 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.

The calculated weights were used in two ways during the model training

- Duplicating the data points.



(a) Hyperbolic weighting (b) Linear weighting

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

- Using the weights directly in the fit function.

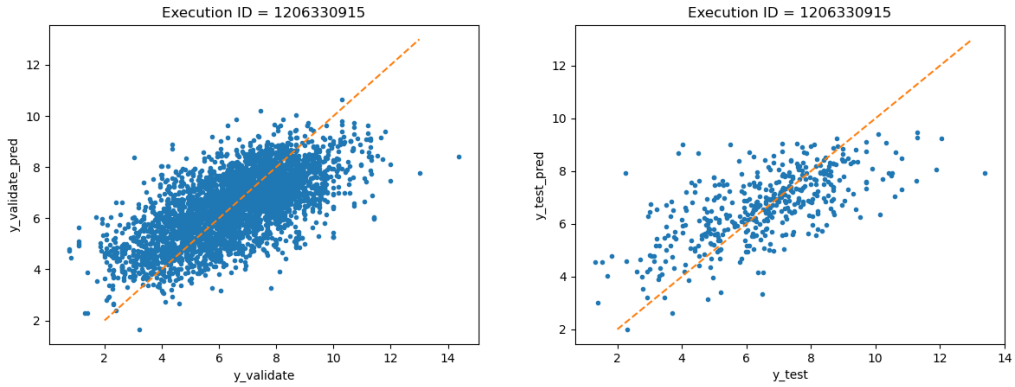
After the duplication, $\approx$ 39000 data points were obtained for the linear weighting formula and $\approx$ 62000 for the hyperbolic weighting.

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. It was the cheapest model (computationally) that was used to fit the data. Table 1 shows the overview of the analysis with this model. Figure 10 shows the best results obtained with linear regression.

No. features	Feature selection	Weighting	Training	Validation	Testing
457	-	-	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[1]	Hyperbolic	$\approx$0.373	$\approx$0.393	$\approx$0.402
40	Pearson Correlation	Hyperbolic	0.288	0.276	0.286
40	Spearman Correlation	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



(a) Validation Accuracy (b) Test Accuracy

Figure 10: Linear Model using Hyperbolic weighting and all 457 features.

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 different data for training and validation. What is important for the validity of the model is that the results remain in a reasonable range.

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

6.2.1 Score function of genetic feature selection

Because the linear regression model was computationally cheap, refitting the model multiple times in the genetic algorithm was affordable. There were two objectives at hand

- Getting the best performing model.
- Reducing the number of input features to make the model simple and explainable.

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

The above objectives were 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. On the other hand, removing a feature that is‘ correlated with the output degrades the model.

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

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

Figure 11 provides a graphical depiction of the score 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 one does not need to keep the values of both components (i.e. *features eliminated* and R^2 score) in a similar range. Hence one need not worry about scaling any of these components.

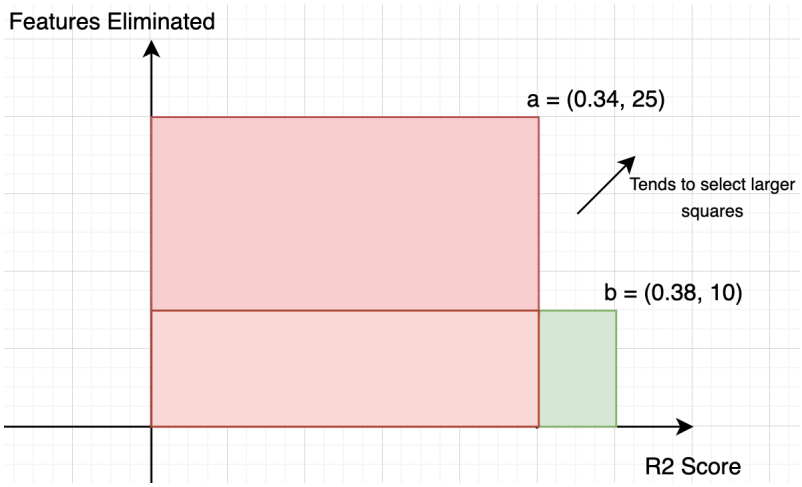


Figure 11: Genetic Algorithm score function representation.

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, a local minimum could be obtained 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.

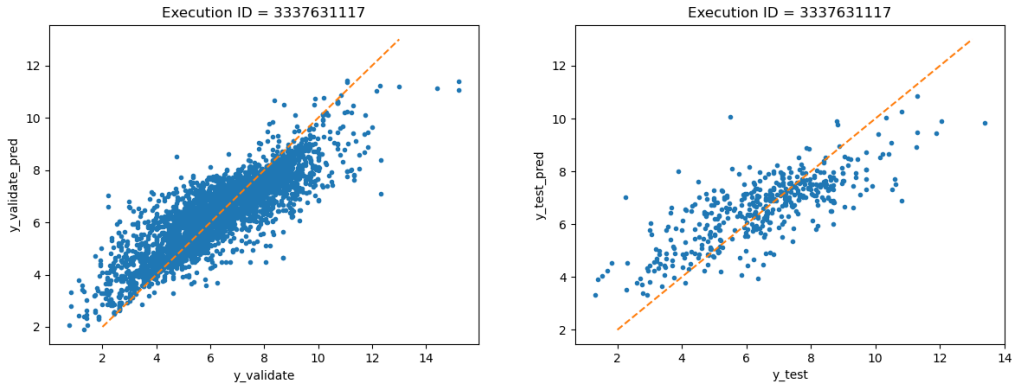
6.3 Random Forest Regression

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.



(a) Validation Accuracy (b) Testing Accuracy

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

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, one 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, these models were not considered to be the best models.

No. Features	Feature Selection	Weighting	Training	Validation	OOB score	Testing
457	-	-	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 Correlation	Hyperbolic	0.930	0.677	0.672	0.880
40	Pearson Correlation	Hyperbolic	0.925	0.651	0.646	0.870
229	Genetic Elitism	Hyperbolic	0.958	0.791	0.792	0.537
394	Genetic Normal	Hyperbolic	0.960	0.808	0.791	0.543
176	**manual**	**Hyperbolic**	**0.947**	**0.737**	**0.734**	**0.579**

Table 2: Random Forest Regression R^2 Score table

6.3.1 Dealing with correlated features

In Section 4.2 it was 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 our trained model would be unusable by someone who could not measure one of these selected features (or if the measurements were incorrect).

For this reason, the stochasticity of the random forest was used as an 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 is known as the split criterion. The model was forced to randomly select only 20% of the features for deciding on the split criterion. More specifically, the following values were used

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

These hyperparameters were determined after some empirical testing. The number of estimators had to be increased as the stochasticity of our ensemble was higher. 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

The model was trained by duplicating the data according to their weights. The following results were obtained after training - Training R^2 score = 0.995, Validation R^2 score = 0.8298 and Out Of Bag (OOB) score = 0.977.

The following issues were found 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.
- As training this model is costly, duplicating the data points further increases the cost of training.

Hence, it was concluded that the duplication of data based on their weights is a wrong strategy for this model.

When the weights were directly used in the fit function, no noticeable improvement was seen in our model. 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. For example in a machine with 4 CPUs, fitting a linear model on our data takes ≈ 0.51 seconds whereas fitting the random forest model takes ≈ 25.86 seconds. Hence the same strategy cannot be used for feature selection as used in the linear models. The following strategies were used to determine the importance of features in the model.

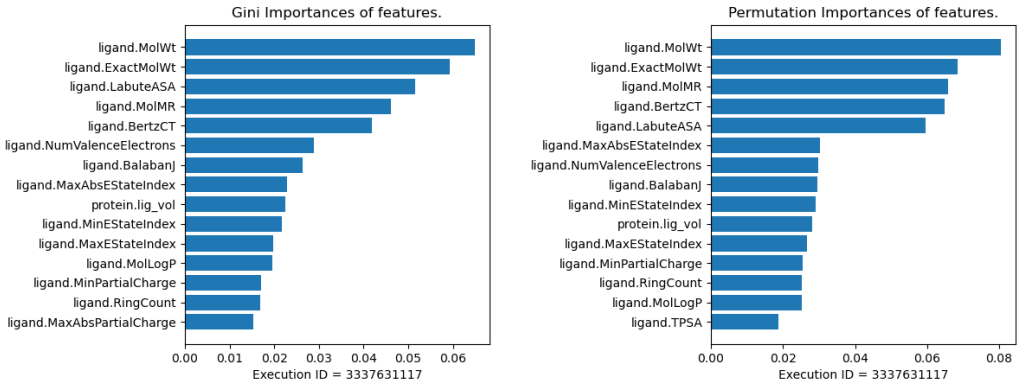
- **Gini Importance (or) Decrease in impurity**. This is calculated by the model itself. For every feature, it calculates how much decrease in the split criterion the feature contributes in the entire ensemble (i.e in every node it's used in all the trees). The disadvantage is that they cannot be reliable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

- **Permutation Importance** - This is a model agnostic method. To calculate the importance of a feature x, it takes the given data and shuffles the values of x. Now the values of feature x do not correspond to the correct data points, hence there is a reduction in the prediction accuracy. The importance of the feature is proportional to the reduction in accuracy. The disadvantage of this method 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, a combination of permutation importance and genetic algorithms was used to find out the importance of the selected features. This is because it was expensive to refit the model. However, it was cheap to evaluate the predicted results of the model after shuffling the features.

The following 2 strategies were used with the genetic algorithms -



(a) Gini importance

(b) Permutation importance

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

- **Using specific scoring function**. In Section 6.2.1, the intuition behind the scoring function was discussed. In the random forest model, however, this scoring function selected a lot of features (> 400). Hence the following scoring function was used to reduce the number of features.

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

  A stronger signal was used for the number of eliminated features. This scoring function, however, turned out to be too strong and none 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 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. Support vector regression was trained with the RBF kernel.

No. Features	Feature Selection	Weighting	Training	Validation	Testing
457	-	-	0.310	0.308	0.356
457	-	Hyperbolic	0.325	0.326	0.371
457	**-**	**Linear**	**0.336**	**0.334**	**0.374**
40	Spearman Correlation	Linear	0.256	0.286	0.262
40	Pearson Correlation	Linear	0.199	0.200	0.199
40	Spearman Correlation	Hyperbolic	0.252	0.266	0.254
40	Pearson Correlation	Hyperbolic	0.169	0.169	0.169
176	Manual	Linear	0.306	0.298	0.358

Table 3: Table showing R^2 Scores for SVR

Data duplication strategy was not used for this model because the training time complexity is $O(n^2)$ where n is the number of data points. However, the data point weights were used 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 quality data as compared to the validation data. However, this varies very highly depending on the random seed used. What must be observed is that the approximate range in which the values are obtained is small.

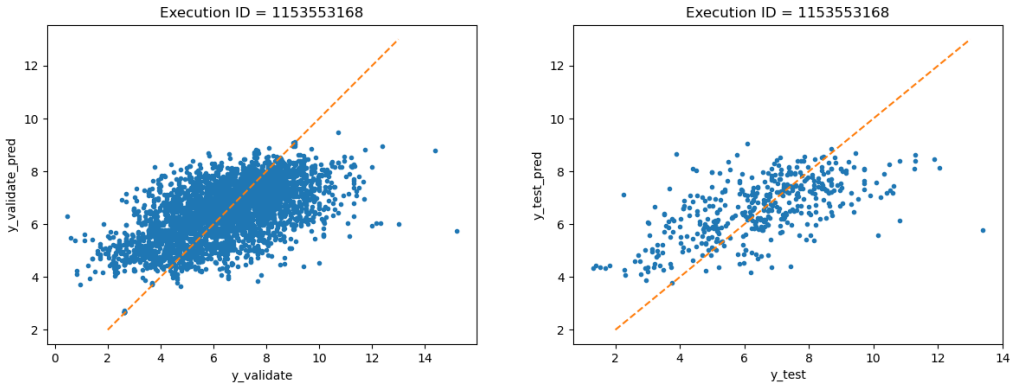
As the performance of the model was not on par with other models, the costly genetic algorithm for feature selection was not run on SVR. Moreover, the time it took to get the R^2 score (validation and testing) was very high and was not suitable to be used in a permutation-based genetic algorithm as discussed 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 different strategies to improve the performance of the model were analyzed. 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, one could reduce the complexity of our tree by using a hyperplane that is not axis aligned.



(a) Validation accuarcy

(b) Testing accuracy

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

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

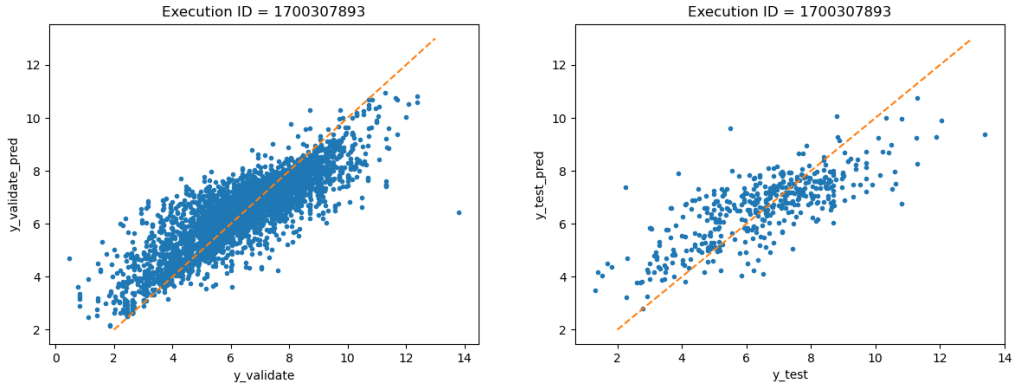
This can be accomplished by calculating the eigenvector of our data and transforming our feature space such that the basis vectors are the eigenvectors of the data. Rotation forests use this concept to reduce the complexity of their trees and to get a better quality model. They were first proposed for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

The training R^2 score was comparable to that of the random forest regressor. However, a considerable improvement in the accuracy was not seen. One of the reasons, is that rotation forests are shown to be better than random forests only for the continuous feature values. However, the data set has both real-valued and discrete values (although the discrete values are encoded as real-values for our model). Hence it was concluded that this model is not the best model for the Protein-Ligand binding affinity prediction.

Table 4 gives an overview of the performance of Rotation forests. The implemented model does not have an option to train weighted data. The models trained by using features selected by output correlation were not considered 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



(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.

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

7 Discussion

In this project, the protein-ligand binding affinity was determined using spatial features extracted by RDKit and D-Pocket. The 2D/3D features made the data more accessible to simple machine learning models. It was crucial because the complex models that directly get the 3D features from the data (E.g., 3D convolutions) require expensive to accumulate data.

Most of the time was spent on 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. It was found that the 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, genetic algorithms were run on this model to get the best ≈ 50 features. This could not be accomplished in the case of Random Forest Regression models.

Our study found that genetic algorithms were not very helpful in improving the performance of our model. However, they were successful in eliminating around 200 features for the Random Regression Model and around 400 features in the Linear Regression model without hampering the performance to a large degree.

One issue with feature selection algorithms is that they make the model depend on the selected features. This would hamper the model generalization. A single unavailable feature or a single feature measured incorrectly would render our model useless. This hard constraint may not be good when the data collection process is expensive and error-prone.

The Random Forest Regression model 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, the feature correlation within families can be used as an advantage to improve the robustness of the model. By increasing the randomness of our feature selection during the data split the model is forced to rely less on a specific feature. Hence a wrong measurement or a corrupted feature would not hamper our model significantly.

Since the random forest model gives an R^2 score of > 0.5 consistently, the binding affinity predicted by the model for any new PL complex can be

taken as a reference for the 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, the 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, hyper-parameter optimization could not be performed in conjunction with the genetic feature selection algorithms as it was 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  0  1  0  1
    6.0470   60.1670    7.5640  S 0  0  0  0  1  0  4
    .......
    .......
   -2.6338   67.4589    8.1225  H 0  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>
```

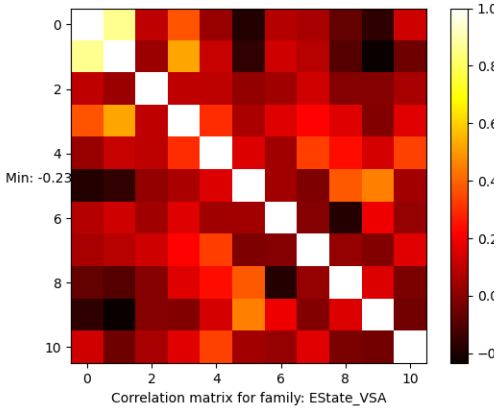
```
363.3

> <NUM_HB_ATOMS>
7

> <NUM_ROTOR>
1

> <XLOGP2>
1.31
```

9.3 MOL2 File Format

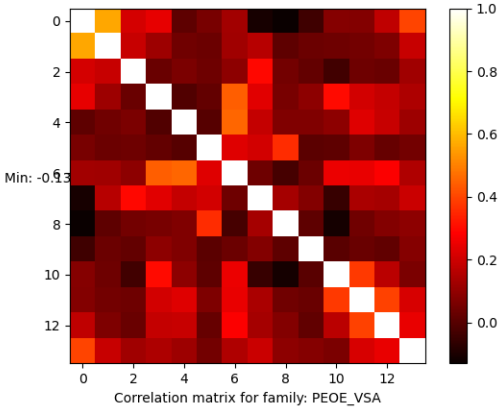
```
###
### Created by X-TOOL on Fri Sep 26 17:34:18 2014
###

@<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

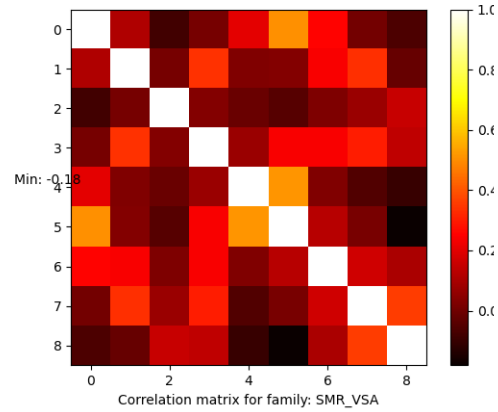
This section shows the correlation heat maps of the most interesting feature families in the our dataset.



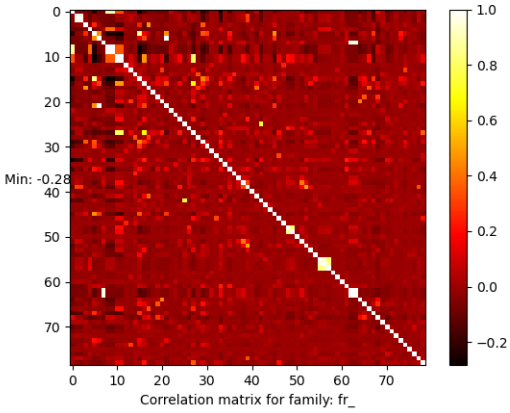
(a) Family EState_VSA



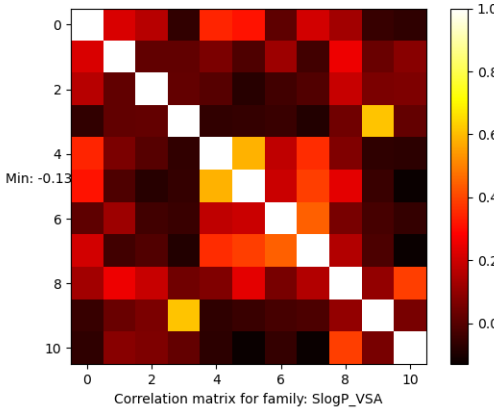
(b) Family PEOE_VSA



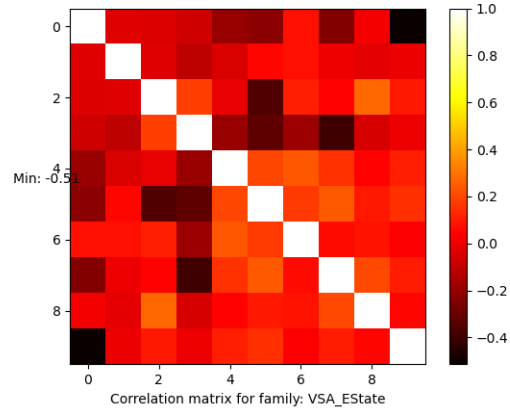
(c) Family SMR_VSA



(d) Family fr_



(e) Family SlogP_VSA



(f) Family VSA_EState

Figure 16: Correlation Heat Maps