MSc Project - Binding Affinity Prediction of Protein-Ligand Complex

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

Github Repository [13]

Supervisors: Simon Bray & Alireza Khanteymoori

September 27, 2021

Contents

1 Abstract

2 Introduction

2.1 Biological Background

Proteins are the workhorses of our body. They are necessary for many essential 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). They can also act as stimulating agents. In addition to this, the ligands can also start/stop the protein from doing its function. The correct functioning of these protein-ligand complexes is thus 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 [9].

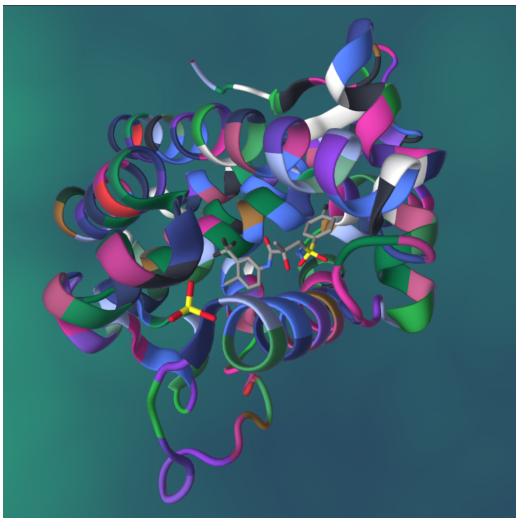


Figure 1: Haemoglobin transporter protein [27].

When one finds a target drug candidate, 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? To answer these questions, biologists and pharmacists conduct wet-lab experiments that are expensive [8].

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 drug candidates 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? It is determined computationally by building a machine learning model that trains on data collected over the past decades. The goal of the project is to reduce drug discovery costs using the ML model.

2.2 Understanding Binding Affinity

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 the dissociation constant, K_i to inhibition constant, and IC_{50} 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. 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 dissociation rate constant and k_1 is the association rate constant. Similarly, K_i and IC_{50} are defined using concentration albeit non-trivially [21] [9].

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** [15], 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

the binding affinity between a protein and ligands. Moreover, it contains information on protein-protein complexes. [14]

This project aims to study the protein-ligand binding affinity. Hence, only the binding affinity data between the proteins and the ligands should be extracted from **Protein Data Bank**. A dataset called **PDBBind Data Bank** already does this [25]. Using the curated protein-ligand affinity data present in the PDBBind Data bank, the project builds a machine learning model that learns to predict this affinity.

3 Methods

3.1 Problem Formulation

3.1.1 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.

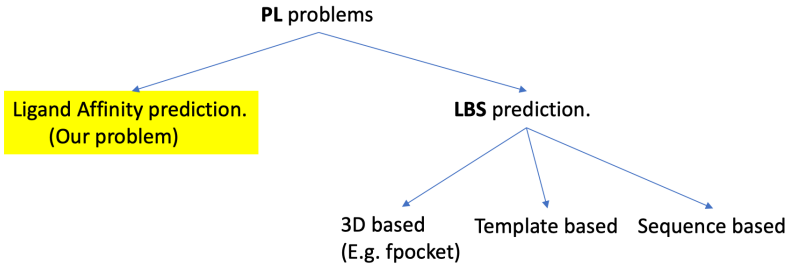


Figure 2: Protein-Ligand problem classifcation.

The 3D structure of proteins and ligands strongly influences how they bind. 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 [9].

Any potential binding location in the 3D structure of a protein is called a pocket. The other parts of the protein do not have a direct influence during the binding. Hence only pocket features are extracted from the data. A 3D-based LBS prediction package called *fpocket* is used to find the potential pockets. A submodule in the package, called *dpocket*, is used to extract the binding pocket's features.

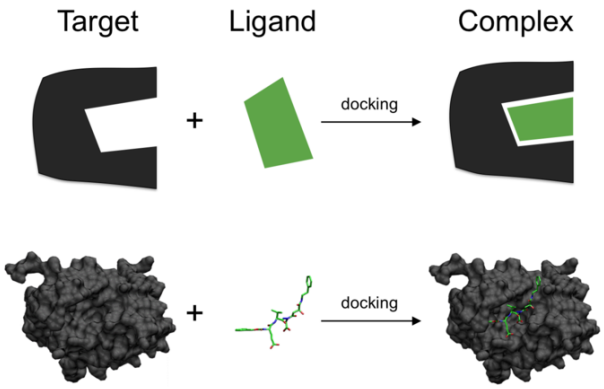


Figure 3: Lock and Key hypothesis in molecular docking [24].

The features of proteins and ligands are kept separate till the training phase as much as possible. In other words, the features are not concatenated before the preprocessing of the ML pipeline (For example, before the output correlation calculation). It helps the model's input become plug and play. Hence, the binding affinity between any protein and any ligand is predictable by the trained model. However, one must note that the concatenation of features is done during the feature selection phase using genetic algorithms because the training of models is a requirement for running them (Section 3.3.3).

3.1.2 File formats overview

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 essential 3D structural information required for predicting the binding affinity. These formats use the **XYZ format** internally to represent the 3D structure of their molecules.

XYZ format: XYZ format is a chemical file format that represents the geometry of a molecule. It specifies the atoms in a molecule along with their cartesian coordinates in the X, Y, and Z-axes. Hence it is named *XYZ format*. The following text illustrates the format. Section 7.1 gives a detailed example. [30]

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

The unit of distance used is this format is Angstrom (Å). $1\,\text{Å} = 10^{-10}$ m. [30]

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. 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 [26] [7]. Please see [2] for an example pdb file.

Structure Data File (SDF) Format: SDF format file is a Chemical Table file (CT File) that contains a structure of the molecule in the XYZ format. It contains information like atomic bonds, connectivity information, molecular weight, and molecular formula. [29] Section 7.2 illustrates the SDF file format.

Mol2 format: Similar to the SDF format, Mol2 also represents the 3D structure of molecules in the XYZ 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 are used in the project because features of more ligands could be extracted using the RDKit feature extractor. Section 7.3 illustrates the Mol2 file format in more detail.

SMILES format: SMILES is an acronym for Simplified Molecular-Input Line-Entry System. It represents a molecule using an ASCII string [23]. Using the 3D data in SDF and Mol2 formats, a graph representing the 3D molecular structure can be created (where nodes are atoms and edges are atomic bonds). Using graph traversal algorithms, 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. Nevertheless, it is a useful format to represent the molecules compactly [28].

3.2 Extraction of features

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

3.2.1 Ligand Features using RDKit

RDKit is an open-source cheminformatics software library [16]. The core data structures and algorithms of RDKit are written in C++, and the Python wrappers are generated using the *Boost.Python* package. Using the module *RDKit.Chem.Descriptors*, 402 features were extracted for each ligand [4]. Each descriptor value is encoded as a real-valued number, hence the ligand feature space is $\mathbf{R}^{402}$ before any feature elimination.

3.2.2 Protein Features using fpocket/dpocket descriptors

Fpocket stands for "Find pocket" whereas *Dpocket* stands for "Describe pocket"[10]. *Fpocket* uses 3D Voronoi tessalation and the concepts of "Alpha Spheres" to find out pockets in the protein structure [11] [31]. The descriptors (features) of all pockets in the protein are extracted from the given protein PDB file. For every pocket, 55 descriptors are obtained in total. They are taken as real-valued numbers, $\mathbf{R}$. Hence, the input space for protein features is $\mathbf{R}^{55}$.

The descriptors of the ligand-binding pockets are extracted from *Dpocket*. *Dpocket* is provided with the protein PDB file and the ligand ID of the PL complex as input. It generates three 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 the pockets that are the binding pockets based on a criterion. The ligand can bind at different locations (pockets) in the protein 3D structure. Hence, there may be features of more than one pocket in this file.

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

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

In the project, dpout_fpocketnp.txt files are not used as they contain the non-binding pockets. The pocket descriptors of the dpout_fpocketp.txt file are preferred because explicitly defined pockets of **dpout_explicitp.txt** are heavily biased towards the ligand.

3.3 Feature selection

3.3.1 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. It 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.

3.3.2 Feature Family analysis

The features extracted from the PDB files 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 matrices of 2 families, AUTOCORR2d_ and Chi. As can be seen in the heatmap, there is a heavy correlation between features within these families. Hence a strategy to deal with this is also required [22]. The correlation heat maps for the other families are shown in section 7.5.

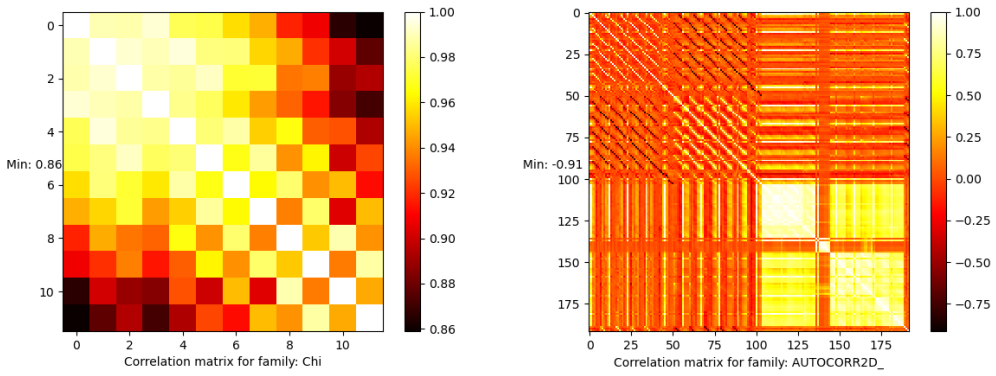


Figure 4: Correlation Heat Map.

3.3.3 Feature Selection Strategies

The features of the proteins and the ligands were selected separately. It helps us make the model plug-and-play as discussed in section 3.1.1. The best combination (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 subsequent subsections, a few feature selection strategies considered in this project are discussed.

Selection by output correlation: The *pearson* and *spearman* correlations of each feature were calculated against the output variable [19]. An assumption that the features were either linearly related to the output (in the case of Pearson correlation) or had a monotonic relationship (in the case of Spearman correlation) was made. 20 Ligand descriptors that were the most correlated to the output variable were selected. Similarly, the best 20 protein descriptors that were the most correlated to the output variable were selected.

Using genetic algorithms: Since the feature space is a non-continuous problem of combinatorial complexity, genetic feature selection algorithms [12] [6] were also studied. Each feature was represented by a binary number, 1 for its inclusion 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 (in other words, an individual in a population). A "population"

of n chromosomes is maintained during the algorithm. For each generation, the best chromosomes are selected as parents. The chromosomes of the next generation are generated by doing crossover and mutation of the selected 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 4]

Algorithm 1 Selection of features for the model using genetic algorithm [6].

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 ← *children*
    end for
    return best(*population*)
end procedure

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.

3.4 Testing

3.4.1 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. One can set the seed of the pseudorandom algorithms used in the training process to accomplish this.

The second approach is used in this project. Every script, when executed, reports an execution ID. It 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.

3.4.2 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. [20]

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

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

A perfect model would have all the points on the $y = x$ line. It 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.

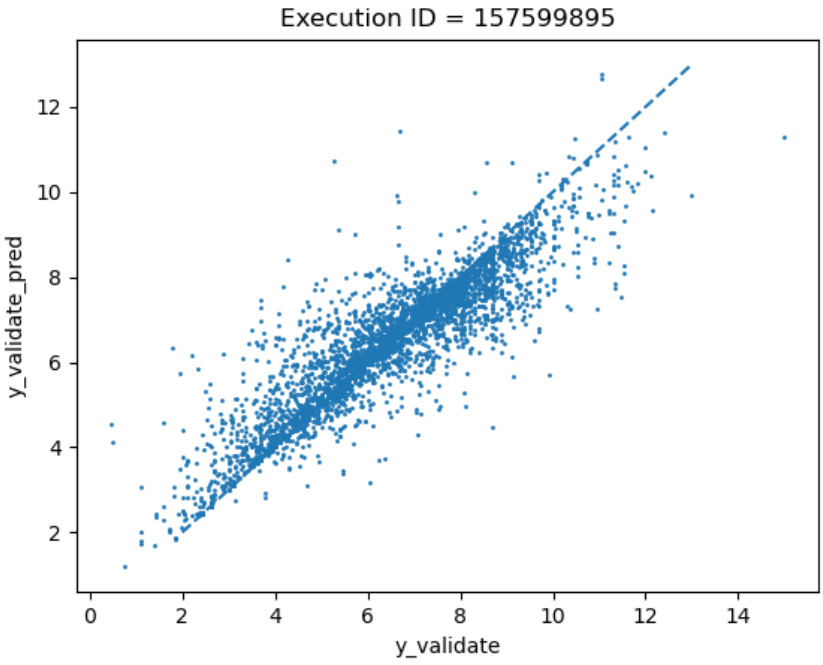


Figure 5: (Sample) Visualizing accuracy. $R^2 \approx 0.805$.

4 Results

The following machine learning models were used in the project:

- 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.

4.1 Data Preprocessing

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

4.1.1 Data cleaning

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

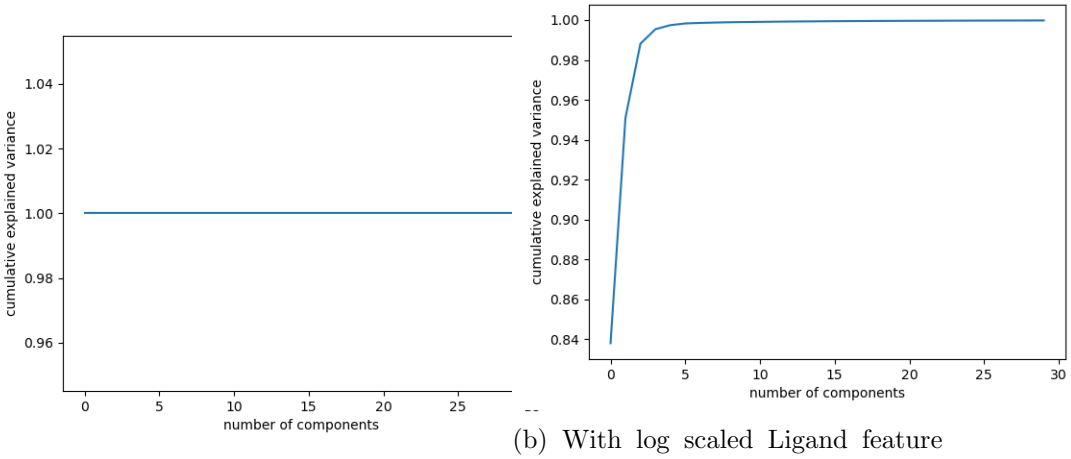
- There was a ligand feature named IPC which had extremely small and extremely huge values e.g $k * 10^{39}$ where $k \in (0, 1]$. Hence, this feature was scaled using a logarithm function. This scaling was required for the numerical safety of the model training process. Figure 6 illustrates the effect 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 zeros was harmless, they were kept unchanged 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. The ligand feature IPC must be scaled before it is input into the model. This requirement is a limitation of the models studied in this project.

4.1.2 Dealing with measurement resolution

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



(a) With original Ligand feature IPC.

(b) With log scaled Ligand feature IPC.

Figure 6: Cumulative PCA of ligand features.

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)
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

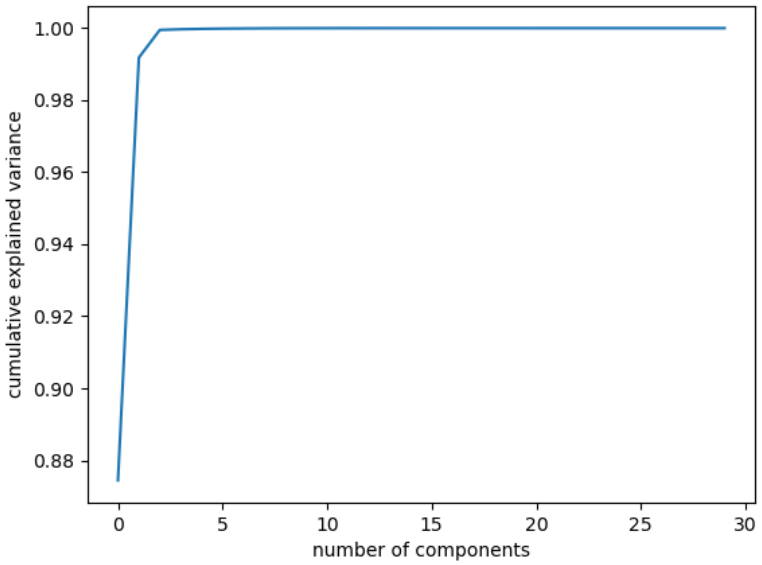


Figure 7: cumulative PCA of protein features.

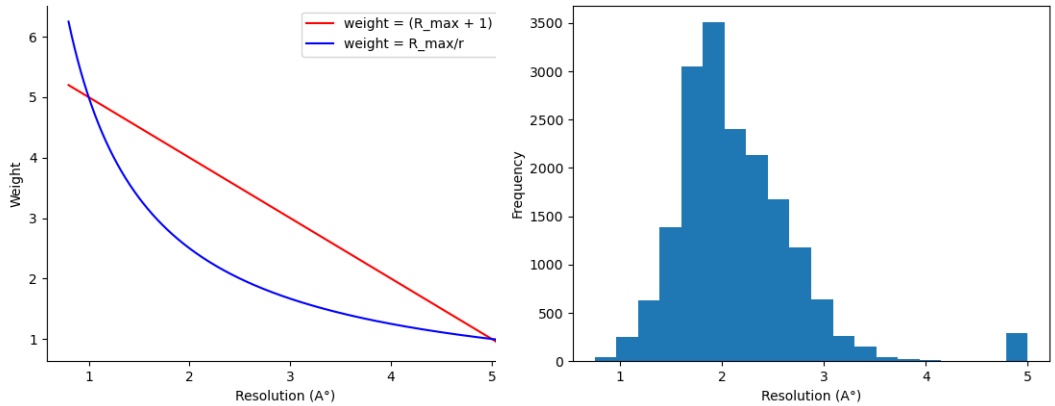
to avoid data points with 0 weight. Figure 8a depicts the formulae. Figure 8b shows the distribution of data points as a function of resolution. The resolution of the measurements ranges from 1 Å to 5 Å approximately.

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 on average as compared to hyperbolic weighting. It makes intuitive sense. As shown in Figure 8b, most of the data points have a resolution of around 2 Å. Since linear weighting around 2 Å is higher than the hyperbolic weighting (See Figure 8a), the linearly weighted data points will have a higher weight on average.

The calculated weights are used in the following two ways during the model training:

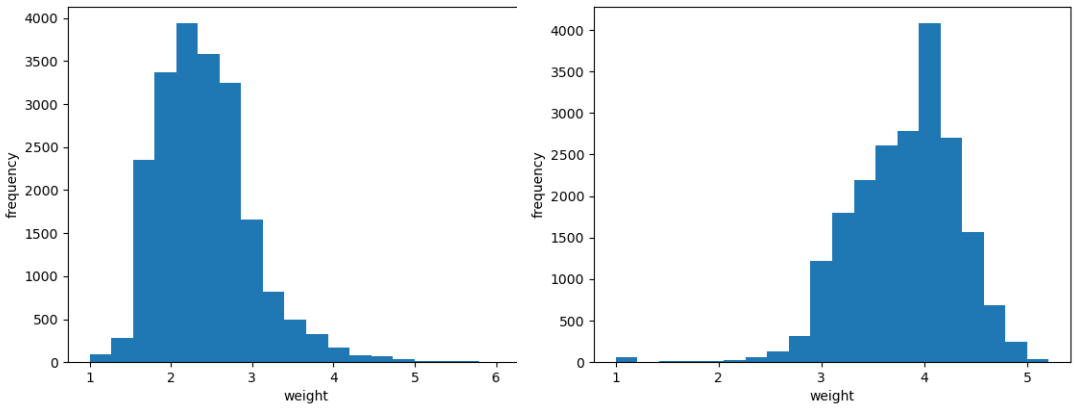
- Duplicating the data points.
- Using the weights directly in the fit function.

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



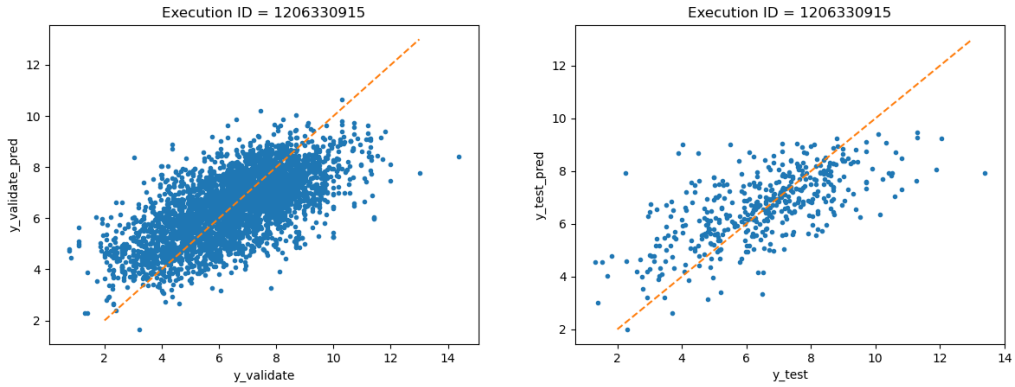
(a) Weight calculation formulae. (b) Resolution distribution.

Figure 8: Weight calculation and resolution distribution.



(a) Hyperbolic weighting. (b) Linear weighting.

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



(a) Validation Accuracy.

(b) Test Accuracy.

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

4.2 Linear regression

Linear regression fits a linear model to a given data [18]. It tries to minimize the square of errors between the predicted output value and the actual output value. It is the cheapest ML model (computationally) that fits the data in the project. 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	≈0.373	≈0.393	≈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.

The reason for the low R^2 score is that linear models assume strong linearity between the input variables and the output variable. It 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.

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

However, this is not always the case. What is essential for the validity of the model is that the results remain in a reasonable range (See section 7.4).

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

4.2.1 Score function of genetic feature selection

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

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

The above objectives were not always against each other. Removing a feature that is not correlated with the output variable may improve the model performance. On the contrary, removing a feature that is correlated with the output variable degrades its performance.

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. The genetic algorithm tries to improve the score. It will try to eliminate more features as well as try to get a higher R^2 score. In the example below, the genetic algorithm prefers point a over b. It is because a gives almost the same R^2 score but eliminates more features when compared with b. Another advantage of this score function is that one does not need to keep the values of both components (*features eliminated* and R^2 score) in a similar range. Hence one need not worry about scaling any of these components.

4.2.2 Initialization strategies of population

The following two strategies were used to initialize the population for the genetic algorithm:

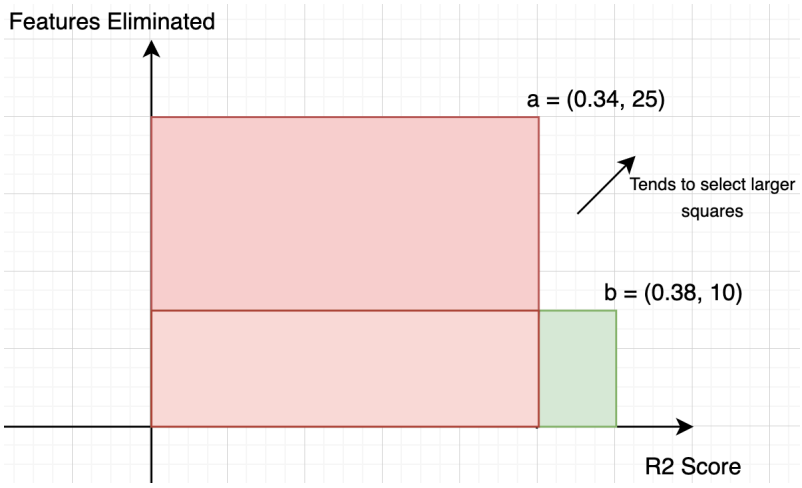


Figure 11: Genetic Algorithm score function representation.

- **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 one 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. However, both the initialization strategies gave similar results.

Table 1 illustrates the results for the linear regression model.

4.3 Random Forest Regression

Random forest regression is an ensemble ML model of regression trees [5]. When training each tree, the algorithm finds out which feature value can

divide the data into two groups to yield the lowest sum of squared values on both sides. The criterion can be different as well. An example of this is the lowest absolute error. Subsequently, each of the two data groups is split recursively till a stopping criterion. E.g., a set number of data points n must be present in every tree leaf.

For each tree, a subset of the data is used for training. The subset of data is sampled with replacement. This sampling with replacement is known as bagging. After training several trees in the ensemble, each tree becomes an expert in a subset of the data. These "experts" are then used to predict the output of new given inputs. The average of the predicted outputs is the result of the whole random forest regressor.

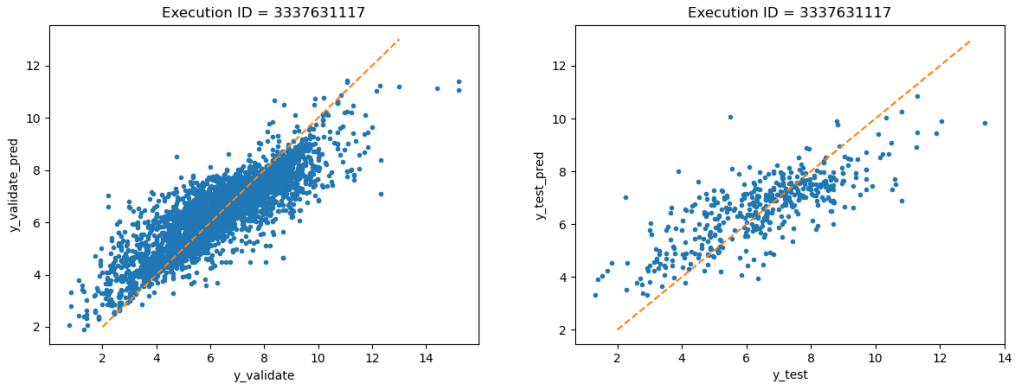
The random forest regression model is a non-linear ML model. One advantage of this is that there is no need for any assumption about the data. Moreover, random forests can easily handle categorical features together with the real-valued features. On the negative side, the function represented by the ensemble cannot be easily understood and is interpreted as a black-box function at best.

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

Table 2 gives an overview of the random forest model. The table shows that the features selected by output correlation (Spearman and Pearson correlations) seem to generalize well to the test data (See section 7.4). However, one cannot consider the models with these features good because their validation errors and the corresponding test errors are not in the same range. Moreover, the correlated feature selection process assumes that the data features have a linear (or monotonic) correlation with the output variable. However, this assumption may not be correct with the given data set. Hence, these models are 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.



(a) Validation Accuracy.

(b) Testing Accuracy.

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

4.3.1 Dealing with correlated features

In Section 3.3.2 it was discussed that within some feature families, there are heavily correlated features. One way to deal with this would be to take one feature per family and exclude the rest. However, this would not be a good strategy for generalization. It is because our trained model would be unusable by someone who could not measure one of these selected features (or if the measurements of some of the selected features were incorrect).

For this reason, the stochasticity of the random forest regression models can be 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 at each node in the tree. The list of constructed tuples becomes a decision criterion when predicting new outputs. The random forest model was forced to randomly select 20% of the features for deciding the (feature, value) tuple at each tree node for every tree in the ensemble. More specifically, the following hyperparameter values are used by the model:

- 400 Trees (n_estimators)
- Using 20% of the features for selecting a (feature, value) tuple at each tree node (max_features). (Increased the stochasticity of the model)
- A requirement that at least 2 data points be present in each leaf of each tree in the ensemble. (min_samples_leaf)

These hyperparameters were determined after some empirical testing. The number of estimators had to be increased as the stochasticity of the ensemble was higher. The random forest model would hence be dependent on the entire family of features. It would not rely on a single feature heavily.

4.3.2 Dealing with measurement resolution

The model was initially trained by duplicating the data according to their weights. The following results were obtained after the 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 data duplication strategy:

- The out-of-bag error (OOB score) was erroneously high here. It 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. It 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 improvement was seen in our model. It is because the data is already skewed towards the ≈2 Å data points. Figure 8b illustrates this skew.

4.3.3 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 the data takes ≈ 0.51 seconds. On the other hand, 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.

One way to deal with this is to determine the importance of features in the data. Firstly, a random forest model is fit using all the features as input. Subsequently, the following strategies were used to determine the importance of features in the model:

- **Gini Importance (or) Decrease in impurity**. It is a type of importance calculated by the model itself. For every feature x, the model calculates how much x contributes to the reduction of the entropy when the training data is used. Whenever x is used by a tree to split the training data (in the feature, value tuple), it contributes to the reduction in entropy. The reduction in entropy at all the nodes in the ensemble where x is used is added up. This sum quantifies the importance of the feature x. The disadvantage of this method is that it cannot be reliable when the features have a high cardinality. Figure 13a illustrates this importance.

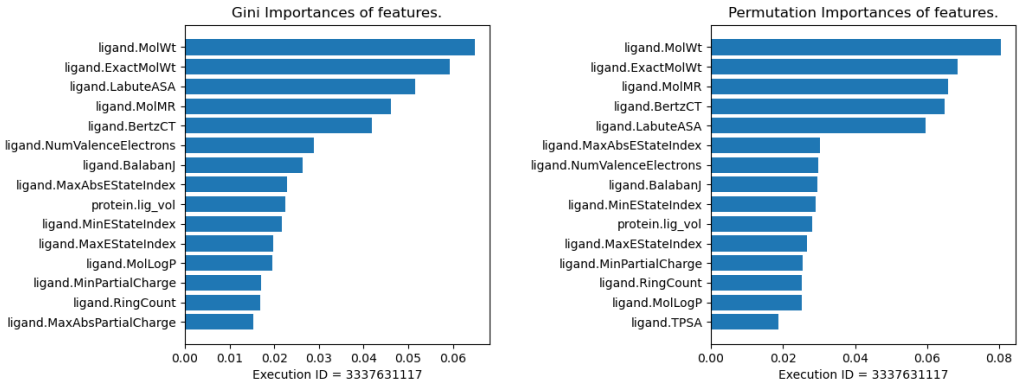
- **Permutation Importance** - This is a model agnostic method. To calculate the importance of a feature x, it takes the given data and shuffles the values of x (in other words, shuffles the values of the whole column referred by x). Now each value of the feature x does not correspond to the correct data points, hence there is a reduction in the prediction accuracy. The importance of the feature is proportional to the reduction in the prediction accuracy. The disadvantage of this method is that since each importance of each feature is evaluated 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.

4.3.4 Permutation Importance and genetic algorithm

To overcome the issue with the above feature importance calculations, a combination of permutation importance and the genetic algorithm was used to obtain the importance of features. The reason for using this combination is that it was expensive to refit the random forest model repeatedly. However, it was computationally cheap to evaluate the predicted results of the model after shuffling the features. For example, the time it takes to get the R^2 score for the validation data is ≈ 0.0004 seconds.

The following 2 strategies were used with the genetic algorithms:

- **Using specific scoring function**. In Section 4.2.1, the intuition behind the scoring function was discussed. In the random forest model, however, this scoring function selected many features (> 400). Hence, the following scoring function was used to reduce the number of se-



(a) Gini importance.

(b) Permutation importance.

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

lected 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 in the end.

- **Elitism**. Elitism is a strategy used in genetic algorithms to select the top few elements from each generation for the next generation in addition to the newly created children. This strategy keeps the best performing subset of the population always in the current generation. Hence, it tries to improve them further by either cross-over with other good chromosomes or through mutation.

4.4 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 [3]. Any error higher than this minimum error is penalized during the training of the model. Support vector regression used in our project was trained with the RBF kernel.

Data duplication strategy could not be used for this model because its training time complexity was $O(n^2)$ where n is the number of data points.

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.

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 may be because the testing data is of better quality as compared to the validation data. What must be observed is that the approximate range in which the values are obtained is small (See Section 7.4).

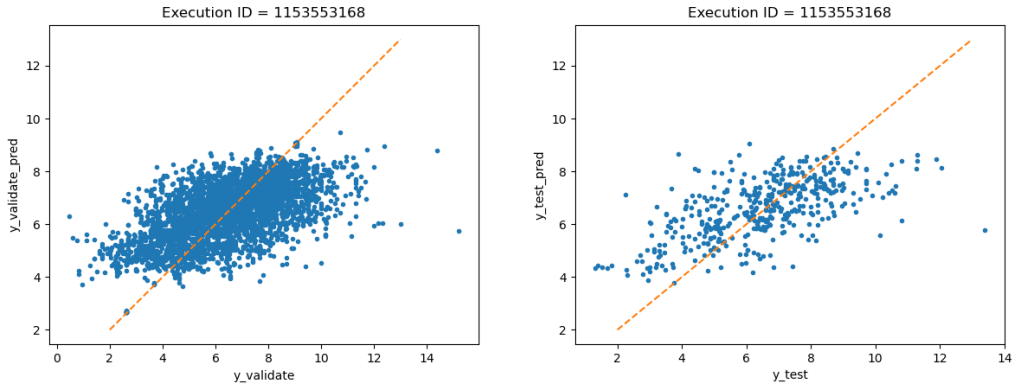
As the performance of this model was not on par with other models, the costly genetic algorithm for feature selection was not run on SVR. Moreover, the time it took to get the R^2 score (validation and testing) was very high and which made the model not suitable to be used in a permutation-based genetic algorithm as discussed in the random forest section. For example, the fitting for the model on a computer with 4 CPUs takes ≈ 99.07 seconds, and testing the model on validation data ≈ 27.01 seconds on the same computer. Hence, 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 methods.

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

4.5 Rotation Forests

In Section 4.3 various strategies were analyzed to improve the performance of the random forest regression models [1]. While fitting a tree, the random forest model tries to get a list of (feature, value) tuples that best divides the training data. However, each (feature, value) tuple only represents an axis-aligned hyperplane.

If there are some real-valued features in the data set, one could reduce



(a) Validation accuarcy.

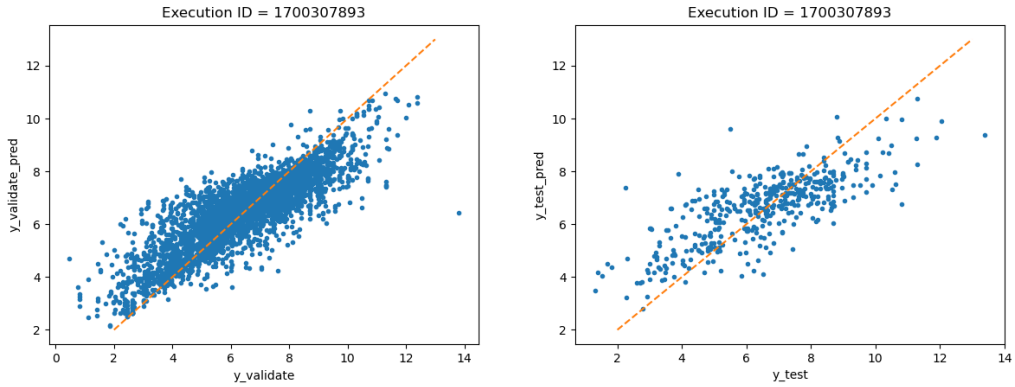
(b) Testing accuracy.

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

the complexity of the fitted tree by using hyperplanes that are not axis aligned. For example, consider approximation of the lines $y = kx$ where $k \in \mathbf{R}$. To approximate these using axis-aligned hyperplanes, one would require a deep tree of high complexity. If the data is transformed such that the line $y = x$ transforms (or rotates) to $y = 0$, only 1 (feature, value) tuple is needed to approximate each of these lines.

This transformation can be accomplished by first computing the eigenvector of the training data. The input feature space can then be transformed (rotated) such that the basis vectors of the new feature space are the eigenvectors of the training data. Rotation forests use this computation to reduce the complexity of their trees and obtain a better quality model [17]. They were first proposed and built for the classification problem. The implementation of the rotation forests was hence adapted for the regression case in this project.

The training R^2 score for the rotation forests was comparable to that of the random forest regressor. However, a considerable improvement in the accuracy was not observed. One of the reasons, is that rotation forests are shown to be better than random forests only for the continuous feature space. The data set in this project contained both real-valued features and discrete-valued features (although the discrete values are encoded as real-values for the model). Moreover, the training time of this model was the highest of all models in the project. The implementation did not support



(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.

multithreading/multiprocessing and it took ≈ 25 minutes 29 seconds to train the model. Hence it can be 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 4.3.

No. Features	**Selected**	**Training**	**Validation**	**Testing**
457	-	0.967	0.756	0.543
40	Person Correlation	0.948	0.596	0.878
40	Spearman Correlation	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.

5 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 protein-ligand complexes (E.g., 3D convolutions) require data that is expensive to accumulate.

Most of the time was spent in analyzing and studying the most suitable linear model (Linear Regression) and the most suitable non-linear model (Random Forest Regression). 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 as well as discrete. The discrete values were also represented input as real numbers to our model.
- There are feature families in the ligand descriptors that have highly correlated members.

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.

It was also found that genetically selected features were not very helpful in improving the performance of our model. Nevertheless, genetic algorithms were successful in eliminating around 200 features for the Random Regression Model and around 400 features for the Linear Regression model without hampering the performance to a large degree.

One issue with the feature selection algorithms is that they make the model depend on the selected features. This dependency would hamper the model's generalization. A single unavailable feature or a single incorrectly measured feature 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 the specific properties of the protein-ligand binding affinity data. Random forest regression can deal with both the real-valued and discrete input features because the

main aim while fitting a tree (in the ensemble) is to reduce the entropy of the divided data. Since all the discrete features in the data were input 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 the (feature, value) tuple 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 the 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 used in this project do have limitations. Firstly, random forest regression models cannot be explained easily. With hundreds of trees in the forest, the model at best can only be treated as a Black Box. Because of its complexity, it was not possible to show why some features were more important when compared to others (E.g., in the feature importance calculation). Secondly, the model relies heavily on the ligand features. Hence, it assumes that the binding affinity only slightly depends on the protein features. On the contrary, the protein and the ligand features are both equally responsible for the binding affinity. Also, hyperparameter optimization could not be performed in conjunction with the genetic feature selection algorithms as it was prohibitively expensive.

6 Conclusion

In this project, methods and preprocessing techniques to predict the protein-ligand binding affinity were studied. In conclusion, using random forest regression for this problem is suggested because of the ease of its training and its generalization capability. Random forest regression can also be used as a baseline to compare any new models.

In further work, there is scope for improving the feature selection process of the model. The random forest regression model used in the project depends heavily on the ligand features. Further investigation on new models that do not have this limitation would be helpful. One way could be to build a model with all protein features and one family of ligand features. This procedure can be repeated for other families of ligand features. The most

important features in each ligand family can be used as surrogates for the whole family. In addition to this, more study needs to be conducted to create explainable models. It would be both accepted by the Cheminformatics community and would also be helpful to further the understanding of the protein-ligand binding affinity.

References

[1] A. Bagnall, M. Flynn, J. Large, J. Line, A. Bostrom, and G. Cawley. Is rotation forest the best classifier for problems with continuous features?, 2020.

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

[3] Basak, Pal, and Patranabis. Support vector regression, 2007.

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

[5] Leo Breiman. Random forests, 2001.

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

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

[8] Joseph A. DiMasi, Henry G. Grabowski, and Ronald W. Hansen. Innovation in the pharmaceutical industry: New estimates of r and d costs, 2016.

[9] Xing Du, Yi Li, Yuan-Ling Xia, Shi-Meng Ai, Jing Liang, Peng Sang, Xing-Lai Ji, and Shu-Qun Liu. Insights into protein–ligand interactions: Mechanisms, models, and methods, 2016.

[10] Le Guilloux, Schmidtke, and Tuffery. Fpocket: An open source platform for ligand pocket detection, 2009.

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

[12] John H. Holland. Genetic algorithms, 1960.

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

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

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

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

[17] Rodríguez, Kuncheva, and Alonso. Rotation forest: A new classifier ensemble method, 2006.

[18] Schneider, Hommel, and Blettner. Linear regression analysis: part 14 of a series on evaluation of scientific publications, 2010.

[19] Schober, Boer, and Schwarte. Correlation coefficients: Appropriate use and interpretation, 2018.

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

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

[22] Tu, Kellett, Clerehugh, and Gilthorpe. Problems of correlations between explanatory variables in multiple regression analyses in the dental literature (referred till "diagnosis of multicollinearity"), 2005.

[23] David Weininger. Smiles, a chemical language and information system. 1. introduction to methodology and encoding rules, 1988.

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

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

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

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

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

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

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

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

7 Appendix

7.1 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
```

7.2 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  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
```

7.3 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
```

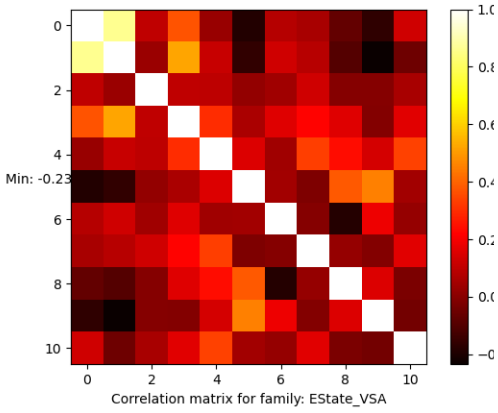
7.4 Testing results better than validation results

It is observed on a few occasions that the testing results were better than the validation results. The reason for this anomaly is that the number of data points in the test set was very less ($\approx$ 500) as compared to the training ($\approx$ 15000)/validation ($\approx$ 2000) sets. This difference was due to a mistake in the early test/train split in which the testing data and the training/validation data had similar PL complexes. To test completely unseen data points, some of the complexes from the test set had to be removed. This created an odd situation in which there were more noisy data points in the training/validation set than the testing set. It resulted in the testing set sometimes outperforming the validation/training sets. As the test set should not be touched till the end of the project, this bug was found only at the final stage of the project. However, this issue was only seen in some of the test cases.

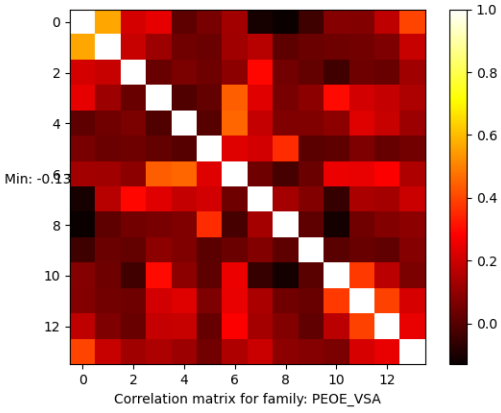
This issue can be solved when the new version of the PDB databank is released. Testing can then can be performed on the new complexes released in the newest version of the PDB databank.

7.5 Correlation Heat Maps

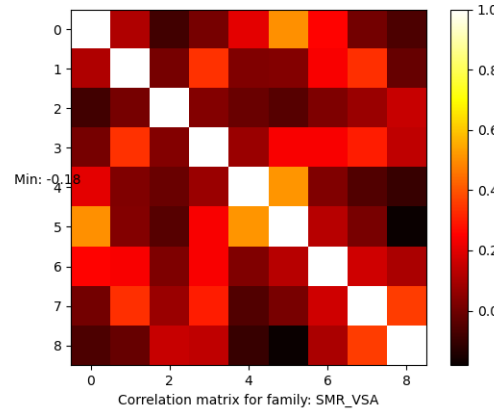
This section shows the correlation heat maps of the most interesting feature families in the our dataset.



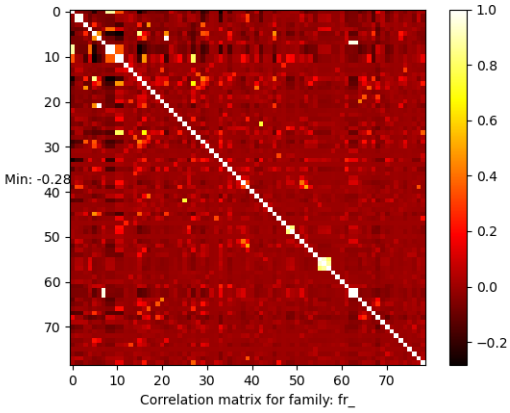
(a) Family EState_VSA.



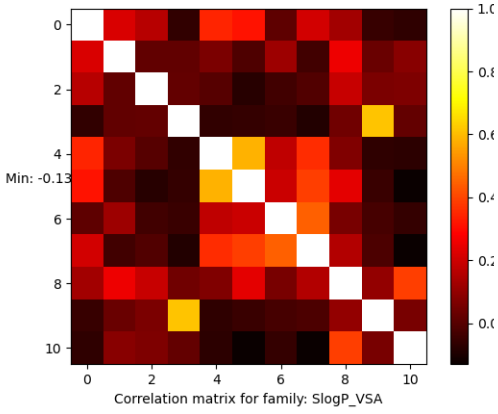
(b) Family PEOE_VSA.



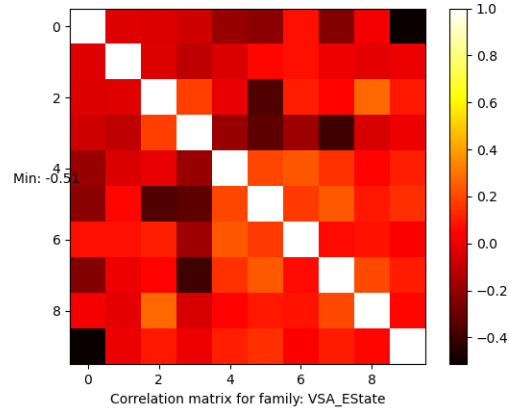
(c) Family SMR_VSA.



(d) Family fr_.



(e) Family SlogP_VSA.



(f) Family VSA_EState.

Figure 16: Correlation Heat Maps.