Machine learning methods for solubility prediction in chemical experiments

Adnan Malik
201538993

Supervised by Bao Nguyen (School of Chemistry), Arief Gusnanto & Luisa Cutillo

Submitted in accordance with the requirements for the
module MATH5872M: Dissertation in Data Science and Analytics
as part of the degree of

Master of Science in Data Science and Analytics

The University of Leeds, School of Mathematics

September 2022

Abstract

Solubility Prediction of chemical compounds using Machine Learning has been an area of great discussion and research for the last few decades. It remains a very challenging task because measuring Solubility is a complex procedure, and the measured Solubility data is always expected to have some experimental errors. Finding a suitable Machine Learning algorithm has also been challenging, considering the unavoidable errors in experimental data and lack of relevant data.

In this research, we considered features that are most relevant for measuring the Solubility of a chemical compound. We built many conceptually different Machine Learning algorithms and focused on **finding the most pertinent algorithms for the job**. We also devise a way to acknowledge unavoidable errors in experimental data when assessing these models. Finally, we combined the predictions of the top-performing Machine Learning models for a final prediction. This research shows how better predictions are possible if **1. Only relevant data is considered 2. Models are assessed considering the unavoidable experimental errors, and 3. A consensus of best-performing models is taken for the final prediction**.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Solubility, defined as the total amount of a substance that can completely dissolve in a given solvent at a given temperature (Mittal, 2017) is considered a critical physicochemical property of a drug compound (Llinàs et al., 2008). Usually measured in LogS (logarithm of 'solubility measured in mol/l').

Solubility prediction of chemical compounds has become a field of intense research with the growth of the drug and chemical industry. It has been an area of interest for researchers for at least the last 60 years. The process of dissolution being complex in itself makes it difficult to get accurate estimates or predictions. However, the difficulties are being tackled with an increase in computational power and the introduction of accessible machine learning technologies. One of the recent studies by Boobier et al. (2020) performed well enough to show that the problem is difficult but not impossible.

1.1 Why Solublity Prediction?

Solubility plays a vital role in the drug development and designing process. To give a clear idea, let's discuss what important properties a drug must have before we can push it to production. When developing a drug, its ADMET properties are always looked up before pushing it into production for its consumers. ADMET stands for:

A (Absorption) - How well the drug is absorbed in the body.

D (Distribution) - How well the drug is distributed in the body.

M (Metabolism) - How well the drug breaks down in the body.

E (Excretion) - How well the drug exits the body

T (Toxicity) - The drug's toxicity level for the body.

It was found that all of these properties are affected by the poor Solubility of a drug compound. A study by Bergström et al. (2016) showed that the compound's Solubility is a limiting

factor for achieving good Absorption. To summarise, the drug compound must dissolve entirely in the body so that the body absorbs its contents, and only then can the drug be expected to benefit the body.

Another area where Solubility plays a vital role is in the Crystallisation process. Crystallisation is used to separate and purify compounds by forming crystals in the pharmaceutical, food, and fine chemical sectors (Su et al., 2015). It allows us to obtain a pure and clean crystal of the compounds from an impure solution. It has been established that a suitable solvent for crystallisation should have high Solubility for the solute (Karunanithi and Achenie, 2007).

The significant role that Solubility plays in these fields and other fields such as drug designing (Lipinski et al., 1997), protein engineering (Khurana et al., 2018) etc., gives a good understanding that we can reduce a high cost and labour if we can predict Solubility before manufacturing the compound.

1.2 Summarising the main challenges

In summary, three challenges need tackling for effectively using Machine Learning to predict the solubility of chemical compounds:

1. Lack of publicly available and relevant data - The solubility process is complex, making it challenging to collect data of significant relevance.

2. Unavoidable errors of $\pm$0.5–0.7 for LogS in the experimentally collected data (Palmer and Mitchell, 2014).

3. Difficulty in preparing and deciding models/methods for solubility prediction (Palmer and Mitchell, 2014).

A discussion on the challenges will follow in the chapter ahead.

1.3 This dissertation

This dissertation will focus on tackling the challenges mentioned above, primarily on developing robust and reliable machine learning models to predict solubility, considering that unavoidable experimental solubility errors exist.

The dissertation will use the data of Aqueous solubility by Boobier et al. (2020) for this purpose because the work of Boobier et al. (2020) has by far been one of the best, even though some issues still need to be addressed. They achieved better results by focusing more on the

dissolution process and gathering relevant data, which reduced the noise and the number of predictors in the dataset by a significant amount.

This dissertation aims to present work from the side of developing and employing relevant machine learning algorithms. We focus on building various models, aiming to cover most of the present-day machine algorithms. For this purpose, we divide our models into five different approaches, as can be seen below:

1. **Ordinary Linear Regression approaches**
    - Ordinary Least Squares
    - Ridge Regression
    - Lasso Regression
    - Support Vector Regression
2. **Latent-variable approaches**
    - Principle components regression
    - Partial least squares
3. **Ensemble Learning approaches**
    (a) Bagging algorithms
        - Bagging of Decision trees
        - Random forest
        - Extremely randomised trees
    (b) Boosting algorithms
        - Adaptive boosting
        - Gradient boosting
4. **Kernel-based approaches**
    - Non-linear Support Vector Regression
    - Gaussian Processes
5. **Artificial Neural Networks approach**
    - Multi-layer perceptron algorithm.

The variation in algorithms for some approaches is done to explore optimisation and model enhancement options and search for the best among each.

Finally, we aim to find the best-performing models to determine which machine learning algorithms are well suited for Solubility prediction. We then combine their results to get a better final prediction result.

Chapter 2

Literature Review

To understand how the research has been so far in this field, we will start with understanding some of the crucial variables and datasets that the researchers have prepared for predicting the Solubility of chemical compounds. Through this, we will get some idea about data availability and reliability challenges.

We will look at a brief history of Solubility Prediction. We will also look at the issues and challenges that researchers faced. And finally, we will look at the role machine learning played over the years in this field.

2.1 The search for relevant independent variables

When we look at the old research papers on this topic, we find that researchers started by searching for relevant variables that would help them estimate a compound's Solubility. So they tried to find relationships between Solubility and different molecular properties of compounds.

The earliest search for variables found that the octanol-water partition coefficient (denoted by 'P') significantly affects a compound's solubility (Hansch et al., 1968). Later, researchers found that melting point is also one of those variables which have a high impact on the Solubility of a compound (Yalkowsky and Valvani, 1980).

From the other relevant variables investigated, a relationship between Solubility and the "Molecular area of compounds" was found (Silla et al., 1992). And similarly, later, researchers found many relevant variables from the molecular properties of the compound. Other variables considered for predicting aqueous Solubility were molecular weight, solute and solvent interaction-based descriptors like solvent access surface area, solvation energy, dipole moment of the solution, Number of atoms, etc., (Boobier et al., 2020).

As Machine learning algorithms became the norm for Solubility prediction, the search for

relevant variables slowed down, but enough data accumulated. Sometimes when modelling for solubility prediction, the focus on selecting relevant features was overlooked by researchers as data shortage remained a huge problem and modelling required more data. The advantage of overlooking relevant features was that more data was available to train models. And the disadvantage was that model's reliability, and robustness suffered due to unnecessary noise from irrelevant data.

2.2 The famous Solubility Challenge

The most common Machine learning algorithm used until the 1990s was the Linear regression algorithm. After the 1990s, as computation became feasible and easy, we saw more advanced algorithms become the norm, such as Neural Networks (Hewitt et al., 2009).

Many models were developed and used to predict Solubility, but issues in accurately predicting aqueous Solubility persisted, which caused huge discussions among the researchers. The problem was in attaining highly accurate data, which is because measuring the Solubility of compounds is a very complex procedure, and we often get irreproducible values for solubilities. So, the difficulties in attaining reliable data and the lack of overall data remained a reason for the models to perform poorly.

In 2008, Llinàs et al. (2008) put forth a Solubility challenge that provided a dataset of 132 compounds with accurately measured solubilities. The challenge asked researchers in the field to use their methods and techniques to perform solubility prediction on this dataset. The challenge wanted the participants to blindly predict the solubilities of 32 compounds given the other 100 compounds.

The way Llinàs et al. (2008) produced an accurate dataset was because of their use of the 'Chasing Equilibrium' technique (CheqSol). CheqSol measurements of Solubility are very reliable, showing good agreement with the experimental values in different laboratories worldwide. We must know that a typical experimental error of ±0.5–0.7 for LogS is always expected, as was mentioned by Palmer and Mitchell (2014). Compared to that, data acquired through Cheqsol has shallow errors: The standard errors for the measured intrinsic solubilities were typically less than 0.05 logS (in mol/l) (Palmer and Mitchell, 2014).

This challenge had many entries and later concluded that it is not definitively evident which prediction methods were the best since they all performed equally well.

This challenge led to many discussions on the topic, and different researchers concluded different reasons for the model's performance on solubility prediction. Overall this challenge

turned out to be very beneficial in this field of study because researchers drew many conclusions about data and methods (Palmer and Mitchell, 2014), which we will mention in the next section. Another challenge was released by Llinas and Avdeef (2019), and is expected to bring more insights into this field.

2.3 The Actual Challenges

Most researchers believed that for the poor performance of machine learning models, only data quality is to be blamed. But other factors became evident after the solubility challenge (Llinàs et al., 2008). In other words, the solubility challenge revealed the real challenges in the field of solubility prediction.

As mentioned by Wang and Hou (2011), A big challenge for modeling aqueous Solubility is to collect more data. Secondly, the quality of collected data must be high with low experimental errors. Thirdly, One must collect relevant solubility data which are varied enough to cover most of the chemical space of drugs. To summarise, To develop accurate solubility prediction models, the key is to collect a sufficient number of high-quality experimental data, and the suspicious data must be verified.

Another big challenge, which many didn't expect, was the quality of models or methods employed in prediction. Palmer and Mitchell (2014) performed tests on two different datasets; the first had data from the literature that was expected to have experimental errors, and the second was data acquired from the CheqSol method, which is very accurate with comparatively lower error values. Looking at the two datasets, one would assume that the models will perform poorly on the data with higher errors, i.e. the literature data, and would perform much better on the CheqSol data. So, Palmer and Mitchell (2014) used two different models to test both data separately (i) a Random Forest model using all 2D and 3D descriptors; (ii) a multiple linear regression model using two variables: experimental melting point and experimental logP. The resulting models gave similar errors on both datasets, showing that both models performed equally well. The results made Palmer and Mitchell (2014) conclude that it is the deficiency in models and methods employed to predict Solubility and not the experimental errors in the data.

Researchers have been trying to solve both issues (data and models) by gathering more accurate data and building databases, as we will see in the next section. Also, advancement in computation has allowed us to use a variety of prediction models.

2.4 Some widely utilised Datasets

As mentioned previously, a lack of data has been a major challenge in solubility prediction (Wang and Hou, 2011). So researchers have always tried their best to gather as much data as possible and form relevant databases and datasets which can be used for solubility prediction. In this section, we will discuss some datasets and databases that were created and are well-known among researchers in this field.

Earlier in 1992, Yalkowsky and Dannenfelser (1992) released the Aquasol database, which contained solubility data for many compounds, being one of the earliest databases in this field. In the same year, they also released an external dataset (Yalkowsky and Banerjee, 1992).

Following soon after, Howard and Meylan (1999) released another famous database by the name PHYSPROP database, which contains chemical structures, names, and physical properties for over 41,000 chemical compounds.

These two became well-known among researchers, and many combined data from both to form their training and testing datasets. One good example of this is Huuskonen (2000), who prepared a dataset of 1297 organic compounds. The resultant dataset by Huuskonen (2000) is also widely known among researchers for evaluation and training models. This dataset is usually referred to as the Huuskonen dataset.

In 2001, Jain and Yalkowsky (2001) produced a dataset which used the Aquasol database (Yalkowsky and Dannenfelser, 1992) to obtain melting points and intrinsic aqueous solubilities of 580 compounds, and included data for other compounds from different literature sources. They combined many literature data into one to form their dataset. We can find many research papers mentioning this dataset as Yalkowsky's dataset for presenting an evaluation of their model's performance.

Among other widely used datasets is the dataset provided by Delaney (2004), a dataset that gave experimental Solubility of drug molecules (log mol(L^{-1}) at 25 °C). Another dataset that researchers put forth was data on the water solubility of 2615 compounds in un-ionised form measured at 25 ± 5 °C by Raevsky et al. (2004) in the same year. Both datasets are widely used for evaluating models.

Some of the other available databases which were used and are still being used include the Reaxyz online database. It was released in 2009 and acted as a tool for data retrieval from published works such as journals etc., (Lawson et al., 2014).

A fascinating database can be found in the Online Chemical modelling Environment (OCHEM)

released in 2011 (Sushko et al., 2011). It is also a web-based platform like "Reaxyz" that contains options even to build models for solubility prediction and is widely used for its Database. The Database includes options to query the data and search a variety of molecular descriptors. OCHEM has allowed and invited researchers in the field to use the data from their data storage platform and build machine learning models.

In 2019, AqSolDB was introduced by (Sorkun et al., 2019), which was labelled by some as the largest curated dataset available. Sorkun et al. (2019) combined many datasets from the literature available at the time to form their Database. The Database consists of 9982 unique compounds and contains representations, experimental aqueous solubilities, and 2D descriptor data of all of these compounds. The data in the AqSolDB is stored as CSVs (comma-separated values). This Database is a huge step forward in dealing with the issue of data shortage.

2.5 Use of machine learning for Solubility Prediction

Let us now see in this section the role that machine learning has played in the field of solubility prediction and discuss some of the studies performed over the years and their best findings. We will look at how changes began to appear in modelling and what types of models got introduced and used with time.

MLR (Multiple linear regression) and ANN (Artificial neural network) are the most commonly employed models in solubility prediction. Some of the other machine learning models which researchers widely used include PLS (Partial Least Squares), SVR (Support Vector Regression), RF (Random Forest), GP (Gaussian Processes), Bagging, Boosting, PCR (Principal component Regression), etc.

Along with the type difference between models, the machine learning models also differed sometimes in the metrics that researchers used for model evaluation. Almost all the researchers calculated Pearson's r^2 and RMSE scores as metrics to evaluate their models. However, some researchers (Boobier et al., 2020) considered Pearson's r^2 and RMSE to be not very reliable and argued that they do not take into consideration the expected experimental errors established by Palmer and Mitchell (2014).

2.5.1 Early machine learning challenges: MLR & ANN

As stated before, the earlier advancements in computation and machine learning included the widespread use of Linear Regression models. Linear models were widely used till the 1990s, and from the early 1990s, researchers introduced more advanced models for solubility prediction (Hewitt et al., 2009).

Everything gained momentum towards solving the challenge of solubility prediction with the introduction of the General solubility equation (GSE) by Yalkowsky and Valvani (1980). It used only two variables (partition coefficient and melting point) to estimate Solubility and was widely accepted for solubility estimation. The development of the GSE showed that the water solubility of non-electrolytes can be accurately calculated from its melting point and its octanol-water partition coefficient. This was a massive discovery at that time and, even though not entirely accurate in estimating the Solubility, worked very well. This opened opportunities for generating more solubility data, which researchers can utilise in machine learning models.

But the melting point and partition coefficients still needed to be calculated by experiments and it was not an easy task to get this data for many of the compounds. And so still, some problems persisted in gathering the solubility data.

To overcome this limitation, Jain and Yalkowsky (2010) came up with an alternate method of gathering solubility data called Prediction of Solubility from SCRATCH, where they did not use any experimentally measured quantity to estimate Solubility. The algorithm utilised predicted melting points and coefficients. It used two additive molecular descriptors, the enthalpy of melting and the aqueous activity coefficient: two non-additive molecular descriptors, the symmetry and flexibility of the compound. The melting point prediction was trained on over 2200 compounds, whereas the aqueous activity coefficient was trained on about 1640 compounds. Models were trained and validated using 10-fold cross-validation on a dataset of 883 compounds allowing easy access to predicted melting points and coefficients.

The results from SCRATCH were not as good as GSE, but it was still acceptable and gave a good overall prediction. Most importantly, it helped in covering up the issue of data shortage.

With some other inputs from researchers, solubility datasets and databases became available which many researchers utilised. This data availability then led to an increase in the use of machine learning models for Solubility predictions.

In 1998, Huuskonen et al. (1998) used an Artificial neural network to perform solubility predictions. It was one of the earliest use of ANN for solubility prediction. The dataset used contained aqueous solubility values for 211 compounds. The data was further divided into a training set (160) and a test set (n = 51). The artificial neural network contained 23 parameters in the input layer, 5 neurons in the hidden layer, and 1 in the output. An external dataset provided by Yalkowsky and Banerjee (1992) was used, and they achieved a score r^2 = 0.86 for testing the model. The model performed well and gave good results despite the small dataset size.

In the same year, another fascinating study was done by Mitchell and Jurs (1998). They used Convolutional neural networks (CNN) and Multiple Linear Regression (MLR). The dataset they used in the study consisted of 332 compounds from the AQUASOL database (Yalkowsky and Dannenfelser, 1992). The exciting part was that they turned chemical structures into numbers and then used them for modelling. They built three models. The first used Linear regression for feature selection and modelling, the second used Linear regression for feature selection but CNN for modelling, and the third used CNN for feature selection and modelling. They evaluated the model by calculating their RMSE values and found that the third model performed the best with the least RMSE value. This is one of the few studies using convolutional neural networks to model Solubility.

In 1999, the PHYSPROP database was released by Howard and Meylan (1999). The data available for modelling increased.

In 2000, Huuskonen (2000) performed modelling with MLR and ANN. He used data from the Aquasol database (Yalkowsky and Dannenfelser, 1992) and the PHYSPROP database (Howard and Meylan, 1999). The dataset was divided into training (884) and test-set (413). The parameters for the ANN were 30-12-1. On testing both models, he found that ANN (r^2=0.92) performed better than MLR (r^2=0.88). This is also where Huuskonen (2000) released the well-known Huuskonen dataset. This study along with Mitchell and Jurs (1998), suggested that linear models might be performing worse than non-linear models.

In 2001, Ran and Yalkowsky (2001) came up with an enhanced General solubility equation (GSE). They used a combination of data from the Aquasol database (Yalkowsky and Dannenfelser, 1992) and other available literature data. They evaluated the revised GSE against the original GSE and found that it gave 46% more accurate predictions. They used MLR to prepare a model over the data, compared the coefficients with the revised GSE, and found that coefficient values were very close between the two.

Seeing different models perform till now gives us an idea that non-linear models may perform better than linear models. An interesting concept of using the "wisdom of the crowd" also became common among researchers in the field. To apply the "wisdom of the crowd" approach, the researchers started to build many good-performing models and combine their output by taking the mean or median. A similar study was done by Tetko et al. (2001), who took an ensemble of many neural networks and got r^2 score of 0.90.

In 2002, A review of solubility prediction methods was done by Jorgensen and Duffy (2002) after considering all the techniques used before 2002. They mentioned that with whatever the

state of the models was at that time, progressing further will require a resolution of outliers in the data and increasing the overall data.

As can be seen, till now, mostly MLR and NN were common among the researchers in this field, with ANN outperforming MLR. The paper by Gao et al. (2002) brought something different and used Principle Component Regression (PCR) for predicting Solubility. They used a combination of aqueous solubility data from the Aquasol database Yalkowsky and Dannenfelser (1992), other available literature data, and some in-house measured solubility data. Their results with PCR were evaluated and compared to an ANN model, where their model outperformed the ANN model giving an r^2 score of 0.91. But we can still say that it's not that decisive which model is better than the other, as we can see that the ANN performance recorded by Huuskonen (2000) is not significantly different from the PCR model's performance reported here.

2.5.2 Increase in Data

The amount of data being used to prepare the models increased with time, and the researchers started using data from multiple sources in the hope of better predicting models. A similar approach was taken by Engkvist and Wrede (2002). In this study, they used two data sets, first taken from several published studies and the other taken from the PHYSPROP dataset Howard and Meylan (1999). They developed a Recurring neural network (RNN) for solubility prediction with 63 input neurons, 5 hidden neurons, and 1 output neuron. Their final model consisted of a training set of 3042 molecules, a test set of 309 molecules, and an external validation set of 307 molecules. The r^2 score was 0.91 for the training set, 0.89 for the test set, and 0.86 for the external validation set. They tested their model on the dataset provided by Huuskonen (2000) and achieved an r^2 score of 0.95, showing that Neural networks might be an answer to solving the solubility prediction challenge.

In 2003, Yan and Gasteiger (2003) prepared a Backpropagation Recurring neural network and an MLR to predict Solubility. They used data provided by Huuskonen (2000) for this. They found a similar result with the Neural network performing better than MLR, where NN gave an r^2 score of 0.92 and MLR gave an r^2 score of 0.82.

Researchers continued to employ different machine learning algorithms. An interesting approach was taken by Butina and Gola (2003), who developed a Cubist model (Decision tree model which uses linear regression rules for growing the tree) and a partial least squares (PLS) model. They used the data from Syracuse Research Corporation (SRC) Database. They found that the Cubist model performed better than the PLS.

A different machine learning model was likewise used in the same year by Lind and Malt-

seva (2003). They employed Support vector regression in their study, and they made use of the exact Huuskonen (2000) dataset. They found that their model gave RMSE = 0.68, an adequate evaluation of the log S values of the test.

In 2004, Yan et al. (2004) conducted experiments on modelling aqueous solubility prediction using MLR and Backpropagation Neural Network. They used the external dataset provided by Huuskonen (2000) as one dataset and took another dataset of 2827 compounds from Merck KGaA Company, Germany. Their results concluded that when predicting the aqueous Solubility of compounds, first, an unsupervised learning method must be applied to look at the descriptor space. Then a linear model (MLR) must be built on the data followed by the investigation of whether a non-linear model (NN) would perform better. This study suggested that there may be cases where linear models may perform better than non-linear models.

2.5.3 Need for relevant data

We saw till now that various models were being looked at, and an increase in datasets also took place. The aim of each machine learning-based solubility prediction was focused more around getting more data and building models. We saw that only a few focused on gathering relevant data or the data which makes sense from an actual chemistry point of view. We can summarise our findings until now by saying that data quantity and model types were the main focus of researchers for solubility prediction during this period. From the methods, Non-linear models seem to be outperforming linear models. And only a few paid attention to gathering relevant data.

A change occurred when Delaney (2004) used an approach that focused on getting relevant data to perform modelling. They used features that defined molecular structure, and they built Linear regression model with only 9 descriptors. Their results proved more stable than the GSE (Ran and Yalkowsky, 2001).

Another study that focused on reducing the descriptors and using only relevant descriptors was the study by Fröhlich et al. (2004). They used Support vector regression for the study and the external dataset by Huuskonen (2000) for evaluation. They concluded that with less than 2% of the original descriptors, one could achieve a model quality that is almost the same as when using all descriptors (from the descriptors that we being used at the time) and achieved to get r^2 score = 0.9.

In 2005, Delaney (2005) did a review of different machine learning models and stated the need for better data. Studies such as Delaney (2004), Fröhlich et al. (2004) and Delaney (2005) suggested the need to consider relevant descriptors (or predictors) for modelling rather than us-

ing many descriptors. Their studies also showed that we could achieve better performance with less data if it is relevant enough.

2.5.4 Variety in machine learning algorithms became the norm and Actual challenges started to unfold

In 2007, researchers found that Random forest models can perform better than PLS, SVR, and even ANN (Palmer et al., 2007). This study by Palmer et al. (2007) used datasets from Huuskonen (2000), Delaney (2004), and Guillory (2003). They showed that the Random forest model performed better than the other models and gave an r^2 score of 0.89. Another study that must be mentioned here is the study by Kovdienko et al. (2010), which also supported the findings of this study and showed that Random forest performs better than MLR. This gives us more confidence in our assumption that linear models may not be suitable for Solubility prediction.

A study was done by Schwaighofer et al. (2007) in 2007 which used Gaussian Processes for modelling. This study used many different datasets, including the data from Huuskonen (2000) and PHYSPROP (Howard and Meylan, 1999). On the Huuskonen dataset, their model gave an r^2 score of 0.92.

Another study in the same year supported using Gaussian Processes for solubility prediction by Schroeter et al. (2007). They built a GP regression model using the datasets from Huuskonen (2000). They built GP, RNN, SVM, and RandomForest models and found that GP gave the least RMSE score among all of 0.579 followed by SVM, which provided an RMSE score of 0.6. This study, along with the last one, shows that from among the models' Gaussian process regression might be a good candidate for modelling solubility.

In 2008, the famous solubility challenge Llinàs et al. (2008) was released, and they released their dataset.

In 2009, Bagging was used for solubility prediction but did not perform very well compared to other models. The data they used for their study was taken from the PHYSPROP database (Howard and Meylan, 1999) and from other available literature. The model gave an RMSE of 1.17.

Then in the year 2011, It was concluded after looking at the solubility prediction attempts by Wang and Hou (2011) that there is a need for more good quality data. New model types kept on being implemented to generate more robust models.

In 2014, Following up on the solubility challenge (Llinàs et al., 2008), Palmer and Mitchell

(2014) concluded that it's not the experimental error causing a considerable challenge in solubility prediction, but rather it is the modelling methods that are being used.

2.5.5 Collective intelligence approaches and flaws in the evaluation metrics

The "wisdom of the crowd" approach to modelling increased. This may be an outcome of the conclusion made by Palmer and Mitchell (2014), which made the researchers cease the search for a better model and start taking a consensus of many fair-performing models. A study by Boobier et al. (2017) took a similar approach. They used NN, RF, Bagging, KNN, Extra Trees (ET), AdaBoost, PLS, SGD, SVM, and Decision Trees for modelling Solubility. They used data prepared by their group using the Cheqsol method. They also compared all the different models and found that among all of these ANN or Multi-layer perceptron performed the best with an r^2 score of 0.71. They also compared their models with the human predictor, and the r^2 score of the human predictor was 0.72. Combining the predictions by averaging or taking the median gave promising results.

Then in 2019, Sorkun et al. (2019) released AqSolDB, which served as a reference database for many researchers.

In 2020, Boobier et al. (2020) came up with one of the most accurate modelling techniques. They used the data from Open Notebook Science Challenge aqueous solubility dataset and the Reaxys database (Lawson et al., 2014). While others usually focused on getting descriptors from 2D models of molecules, they generated descriptors from 3D models of molecules and used them for modelling. Using 3D models to extract descriptors is computationally expensive, but it gave them better results than when 2D models were used to extract descriptors. They also discarded some irrelevant descriptors. They used only 14 descriptors instead of many, which may not be very relevant. In addition to the reduction in the Number of descriptors, they also suggested that r^2 score and RMSE alone are not reliable metrics to assess the solubility predictions because an error is always expected in the experimental data (Palmer and Mitchell, 2014) and these metrics do not take this fact into account. Therefore, the study introduced two additional metrics which check the percentage of predictions under 0.7$\pm$LogS and 1.0$\pm$LogS. This study produced some exciting findings. They found that Linear models did not perform that well compared to non-linear models, which also supports our assumption. They found that taking an ensemble of the top performers gave more stable and accurate predictions (using the wisdom of the crowd). They compared their models against some benchmarked open-access commercial tools and found that their model gave more precise results than others. The method used by Boobier et al. (2020) performed very well compared to many other ways of predicting Solubility. It also suggested, along with other studies, that taking the wisdom of the crowd approach might be a much better way of achieving a good-performing model and getting robust

predictions.

The variations in modelling techniques are still being explored and used in search of a better-performing model. Ensemble learning methods were explored recently in 2021 by Bentéjac et al. (2021), and they found that they tend to perform better in solubility prediction. They gave an r^2 score of 0.979, again suggesting that non-linear modelling could be an answer to achieving a more accurate solubility prediction.

2.5.6 Key take-away points

From the review of all the research literature, we can list down the following take-away points:

1. Linear models might not be suitable for solubility prediction, and maybe if more focus is paid to developing non-linear models, we can expect even better results.

2. We should use only very relevant and good-quality descriptors (predictors) with as much data as possible. But, we must avoid using unnecessary variables to build a large dataset because it may lead to noise in our model and can affect modelling badly.

3. Another critical finding from this discussion is to pay attention to and employ the "wisdom of the crowd" approach, which seems to give much more promising and robust prediction outputs.

4. Customised evaluation metrics, which consider that a level of experimental error is expected in the data, may help in tuning machine learning models and enhancing their performance.

Chapter 3

Methodology

This chapter will discuss the Methodology employed to achieve our desired goals. As stated earlier in section 1.3, we will be using the dataset provided by Boobier et al. (2020), fit many models to it, perform hyper-parameters tuning, explore optimisation areas by employing different models of the same algorithm and finally select the best performing models for predictions.

3.1 The Architecture

In this Section, we will understand the skeleton of the whole process without getting into details (details will be covered in the subsequent sections), which will form a good foundation for us to understand further the details of the methods used.

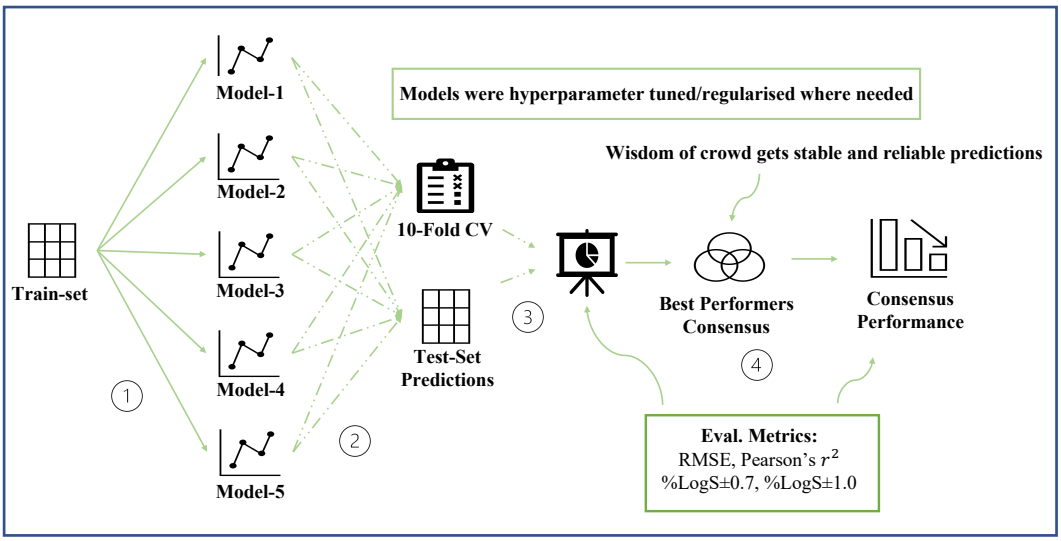


Figure 3.1: Architecture summarising the Methodology

Figure 3.1 above summarises the methodology without getting into details. It will make clear the flow of jobs that take place and how we arrive at our desired goal. Figure 3.1 assumes

that the data from the source is further split into a training set and a testing set. It considers a case where only five different machine learning algorithms are modelled. Let's understand what is going on.

Notice the numbers marked in Figure 3.1. They represent the steps involved.

1. So, starting from number 1, we have our training set and use it to model five different machine learning models (Model-1 to Model-5). Note, for our analysis. We will consider more than five models, as previously mentioned in Section 1.3.

2. Then, we perform 10-fold Cross-validation on each of the models using the training set, evaluate the model's performance and save the performance metrics. Similarly, we evaluate our model on the test set and store the evaluation metrics. In the coming sections, we will cover the details of cross-validation and the wisdom behind using a separate test set.

3. Now that we can compare our models using the evaluation metrics, we make the comparison and select the top performers.

4. Once we have the best performing models with us, we take a consensus (wisdom of the crowd approach) by aggregating their predictions on the test set to get the final predictions. Finally, we measure the performance of the consensus, which tells us how well our method has performed.

We must also understand that before splitting the data, it is checked and cleaned from Nulls, Outliers, and other discrepancies. Each model is also hyper-parameter-tuned and optimised.

3.2 Data Checks

The data, as mentioned previously, taken from the study of Boobier et al. (2020) needs to be checked for discrepancies. Some most common issues that we encounter when getting the data ready for modelling include:

- Dealing with **Missing values**. It is crucial to deal with Missing values in the dataset to reduce unnecessary noise affecting the model.

- Dealing with **Outliers** in the data. These values do not fit properly in the variable's description, such as finding a value of 9999 in a variable that contains people's heights.

- Dealing with **Highly correlated independent variables**. The presence of highly correlated variables in the model can inflate the estimated regression coefficients and make these estimates completely unreliable (Lee and Lee, 2020).

A simple way to deal with Missing Values and Outliers is to remove the record containing them from the dataset. But, in cases where the dataset is small, and each record contributes a

good portion of information to the model, such a decision might not be very feasible. Therefore in cases with less data, one option is to perform imputation (generally of the mean or median of the variable). Another exciting technique for imputation is using a machine learning algorithm to predict the missing values or a replacement for the outlier (considering the variable of interest to be the target). We must note that if there are no missing values or outliers, then the model needs no imputation or any form of alteration.

As for dealing with highly correlated features, the basic idea is that when two features are highly correlated, there will not exist much independent information between them. Intuitively this tells us that if two features are highly correlated, then removing one of the two will not cause us huge information loss. There are two ways that we have used to check for the high correlation between features:

1. **Pearson's correlation coefficient** (r):

   Pearson's correlation coefficient is a ubiquitous measure for correlation between two continuous variables. It is denoted by *r* and is given between two variables x and y by:

$$r = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2}\sqrt{\sum_{i=1}^{n}(y_i - \bar{y})^2}} \tag{3.1}$$

   Where,

   r = correlation coefficient

   n = number of samples

   x_i = values of the x-variable in a sample

   $\bar{x}$ = mean of the values of the x-variable

   y_i = values of the y-variable in a sample

   $\bar{y}$ = mean of the values of the y-variable

   The value of r varies from -1 to +1. Values close to 0 indicate "No correlation" (at 0) or "Very less correlation" (around 0). The correlation increases as we move away from 0, either in the positive or negative direction. In the positive direction, it will mean that a positive linear relationship may exist between the variables; in the negative direction, it will mean that a negative linear relationship may exist between the variables.

   In our study involving chemical compounds, any value of correlation-squared r^2 greater than 0.9 will be considered highly correlated, as was considered by Boobier et al. (2020) in their study. Simply put, we expect that any correlation-squared value more than this threshold value will mean that the two variables contain roughly the same information.

2. **Variance Inflation factor (VIF)**:

When a model is formed with one or more highly correlated independent variables, the estimated regression coefficients can inflate, and the model gives unreliable predictions Lee and Lee (2020). This is called the problem of multicollinearity.

Variance inflation factor (VIF) gives us a measure of multicollinearity between variables and measures how much variance of the estimated coefficients is inflated. In simple words, it provides us with a measure of inflation of coefficients. To understand how VIF is calculated, it is essential first to understand the Coefficient of Determination R^2, as it is used at the core of VIF.

Coefficient of determination (R^2) is a simple method of evaluating a regression model by comparing the variation explained by Regression to the actual total variation (*Coefficient of Determination*, 2008). To understand properly, let's look at the following two formulae:

$$R_i^2 = \frac{Explained\,Variation}{Total\,Variation} \tag{3.2}$$

$$R_i^2 = \frac{\sum_{i=1}^{n}(\hat{y}_i - \bar{y})^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2} \tag{3.3}$$

Where,

y_j = i^{th} actual target value

$\hat{y}_j$ = i^{th} predicted value by regression

$\bar{y}$ = Mean of the actual target values

The value of R^2 varies between 0 to 1. A value close to 1 means the model has performed well and gives us a measure of the amount of variation explained by the model.

Coming back to the discussion on calculating VIF. VIF is estimated for each independent variable by regressing each variable on all the other independent variables and calculating the R^2 value for each (Lee and Lee, 2020).

For an independent variable, the VIF score can be written as shown in equation 3.4 (Lee and Lee, 2020):

$$VIF_i = \frac{1}{1 - R_i^2} \tag{3.4}$$

Where,

i = the i^{th} independent variable

R^2 = The Coefficient of determination value calculated after regressing i^{th} independent variable on all the other independent variables

We use the VIF score to show us redundant independent variables. A VIF score greater than 10 can be considered significant enough to remove an independent variable (Belsley et al., 1980).

Understanding and implementing these concepts will help us build a clean and reliable dataset for Regression.

3.3 Dataset Splitting and Validation

After we have preprocessed our data by cleaning all the discrepancies present in them, we can start with training our models on this data. The issue to tackle after training our model will be to evaluate our model's predictions because our main aim is to develop a well-performing model. We want our model to generalise what it learns from the training data. A common issue which prevents our model from generalising well is Overfitting. Let's understand properly what Overfitting is and why dataset splitting can help us correctly evaluate our model and detect Overfitting.

3.3.1 Overfitting and Underfitting

In our case of solubility prediction, We perform training because we want our model to use the solubility data and understand how Solubility is generally affected by the given independent variables. This allows our model to predict any compound's Solubility, including those it has never seen before. Simply put, we want our model to learn the relationship pattern between independent and dependent variables from the data. What we don't want is that our model merely memorises the given data points and does not understand the relationship pattern between independent and dependent variables. If our trained model has simply memorised the data points, it will not be able to generalise this relationship on unseen data accurately. This issue of memorising the data is a common issue in machine learning called **Overfitting**, where a very complex model is developed. We often say that if we want to overcome Overfitting, we must reduce the complexity of our model by reducing the variance of estimated model parameters and introducing some bias in them.

Also, if our model only learns some pattern from the dataset and lacks a lot of information, it may go to the other extreme of Overfitting, called **Underfitting**. In Underfitting, the developed model is too simple and has a high bias in estimates to perform well, so it is needed in this case to increase the complexity of the model by increasing variance and reducing the bias.

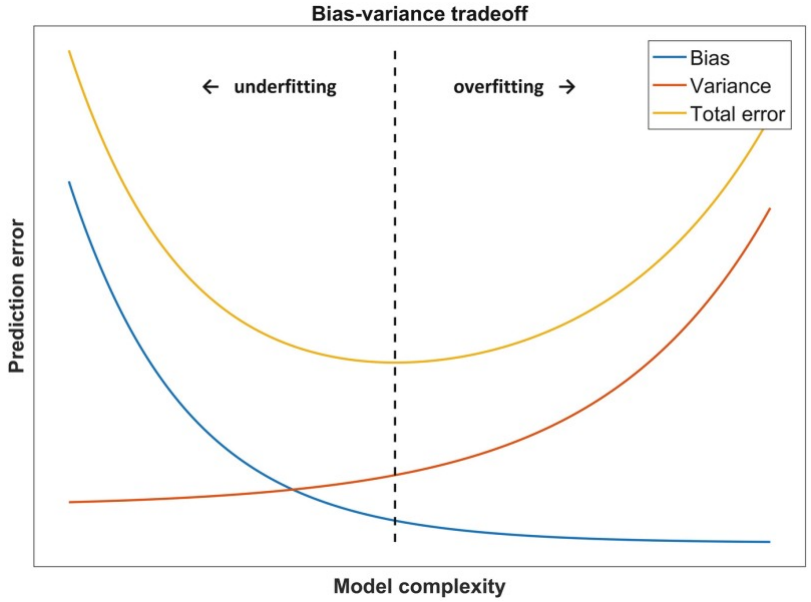


Figure 3.2: Bias-Variance trade-off

So there is a trade-off that we must consider between Model's Complexity, Overfitting and Underfitting to get a good-performing model, see figure 3.2[1].

3.3.2 Train-Test Splitting and Cross-Validation

Once we understand the commonality of Overfitting, the next question that comes to mind is, "How shall we check if our model is overfitting?". We can evaluate our model's predictions on training data and unseen data. If our model is Overfitting, it will perform very well on the training data (as the model memorises it) and very bad on the unseen data. We can calculate some evaluation metrics (covered in the next section) for evaluating our models on the unseen data and then tune our model to reduce its complexity and Overfitting based on the evaluation metrics. This is precisely why splitting a dataset into training, and testing sets come into the picture. The training set is taken to train the model, and the test set, which behaves like unseen data, is assumed to test the model's performance.

Another interesting approach to evaluate our model on unseen data is by using Cross-Validation. Cross-validation divides a dataset into K-folds or K subsets. One of the folds is kept as a testing fold, and the rest are used as the training folds, see figure 3.3[2]. The model is trained on the training folds and evaluated on the test fold. The process repeats for K-times making each fold a test fold one by one and others as training folds. This ensures that testing is always done on data hidden from the training set. Cross-validation is a prevalent practice in Machine learning for model evaluation and is highly encouraged in cases where the data is less.

[1] https://www.ncbi.nlm.nih.gov/books/NBK543534/figure/ch8.Fig3/?report=objectonly
[2] https://i0.wp.com/neptune.ai/wp-content/uploads/Cross-validation-k-fold.jpg

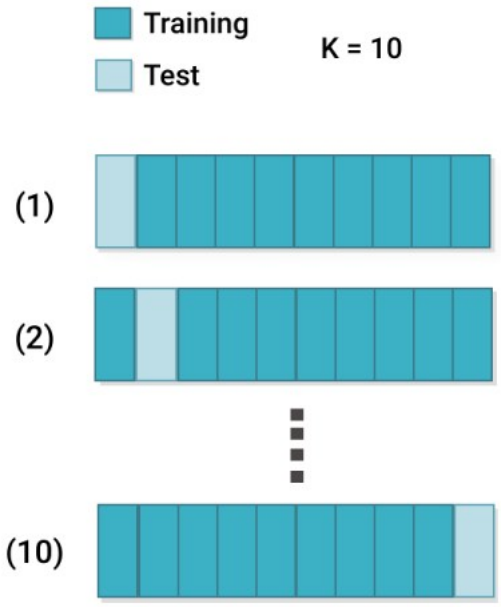


Figure 3.3: K-fold Cross Validation

It is always better to keep a portion of data entirely away as a test set and perform cross-validation only on the train set rather than the entire dataset. When we perform cross-validation on the full dataset, each fold is included with the training folds at some point. This way, the model gets exposed to the whole of the data and seeing all of the data may cause the model to develop bias and give unreliable predictions. Therefore, it is a much better and safer idea to keep a portion of data wholly hidden from the model for a final model evaluation and not rely only on cross-validation of an entire dataset for that.

In our analysis, We will do a train-test split with 80% data in the train-set and 20% in the test-set. We will train our models on the train set. Then use these trained models to get predictions for the train-set through cross-validation and evaluate those predictions. Then we use the same models to get predictions for the test set and evaluate those predictions. We are paying more attention to the test-set performance of the models.

3.4 Model Evaluation

We need to evaluate our model's predictions, which will give us confidence in our models and allow us to tune them to improve them. As discussed in the last section, we will evaluate our model on unseen data and see how well it predicts Solubility.

So once we have our model's predicted solubility values and the actual solubility values for those compounds, we can easily compare them and see how well our model has performed. To do this, we make use of evaluation metrics. Common ones include Root Mean Square Error (RMSE) and Pearson's correlation coefficient squared (r^2).

- **Root Mean Square Error (RMSE)**:

  Root Mean Square Error (RMSE) measures how far our predicted values are from the actual values. It calculates the difference between predicted and actual values called residuals for each input and the square root of the mean of those residuals. RMSE and Mean Square Error (MSE) are widely employed in Regression jobs to evaluate model predictions. Below, we can see MSE and RMSE equations:

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(\hat{y}_i - y_i)^2 \tag{3.5}$$

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(\hat{y}_i - y_i)^2} \tag{3.6}$$

  Where,

  N = Number of predictions or residuals

  $\hat{y}_i$ = i^{th} Predicted value

  y_i = i^{th} Actual value

  RMSE value ranges from 0 to infinity, and we expect our RMSE value to be as smaller as possible.

- **Pearson's correlation coefficient squared** (r^2):

  Pearson's correlation coefficient squared (r^2) comes from taking the square of Pearson's correlation coefficient and ranges from 0 to 1. Pearson's correlation coefficient is covered in detail in Section 3.2.

$$r^2 = \left(\frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2}\sqrt{\sum_{i=1}^{n}(y_i - \bar{y})^2}}\right)^2 \tag{3.7}$$

  It is often used as a Coefficient of determination (see Section 3.2) and to give a measure of variation explained by the model. However, this is not always the case, and it behaves

as the coefficient of determination only under certain conditions (Ozer, 1985).

Since it is derived from the Correlation coefficient, it will be a good indicator for us to judge our predictions. Our study will use this metric to understand how the predictions follow the actual values. Another reason to include it in our research is to compare our results with those of researchers in Solubility prediction, as it is one of the most employed metrics for model evaluation.

As mentioned in 2.3, an unexpected error in the experimental data is unavoidable and is always expected (Palmer and Mitchell, 2014). We can include metrics which evaluate the models taking into consideration this fact. So we include two additional metrics %P±0.7 and %P±1.0. Let's understand what they mean:

- **%P±0.7**:

  This metric indicates the percentage of predictions within ±0.7 of the experimental value. In other words, from all the predictions made, what percentage of predictions lies in the interval of LogS-0.7 - LogS+0.7 of the experimental values. The higher, the better, so it acts as a form of accuracy.

$$\%P \pm 0.7 = \left(\frac{\sum_{i=1}^{N} I_i}{N} * 100 \right) \% \tag{3.8}$$

$$I_i = \begin{cases} 1 & \text{if } LogS_i - 0.7 \leq P_i \leq LogS_i + 0.7 \\ 0 & \text{otherwise} \end{cases}$$

  Where,

  N = Total Number of experimental or predicted values

  $LogS_i$ = i^{th} experimental solubility value

  P_i = i^{th} predicted solubility value

- **%P±1.0**:

  This metric indicates the percentage of predictions within ±1.0 of the experimental value. In other words, from all the predictions made, what percentage of predictions lies in the interval of LogS-1.0 - LogS+1.0 of the experimental values. The higher, the better, so it acts as a form of accuracy.

$$\%P \pm 1.0 = \left(\frac{\sum_{i=1}^{N} I_i}{N} * 100 \right) \% \tag{3.9}$$

$$I_i = \begin{cases} 1 & \text{if } LogS_i - 1.0 \leq P_i \leq LogS_i + 1.0 \\ 0 & \text{otherwise} \end{cases}$$

Where,

N = Total Number of experimental or predicted values

$LogS_i$ = i^{th} experimental solubility value

P_i = i^{th} predicted solubility value

We must understand how we infer a model's performance using these metrics. We will give the primary focus to the metric %P$\pm$0.7 as it best covers the expected experimental errors. Then after this, other metrics will be prioritised and checked following %P$\pm$1.0, Pearson's r^2 and RMSE, respectively. This order of prioritising metrics is understood and learnt from Boobier et al. (2020).

3.5 Consensus predictions (Wisdom of the Crowd)

Wisdom of the crowd is a theory that roughly states that crowd estimates combined can get more accurate and reliable estimation than individual estimates, showing that crowd acting together might be more intelligent than individuals.

An interesting theorem called the **Condorcet's jury theorem** shows how the probability of correctly judging a matter by majority voting increases with an increase in the number of judges. The theory makes two essential assumptions. The first assumption is that each judge's probability of correctly judging a matter should be more than 0.5. And the second assumption is that each judge must be independent of each other or must not be influenced by each other's decisions. (Boland, 1989)

We can use Condorcet's jury theorem to employ many well-performing machine learning algorithms and combine their results. We must ensure that our models are diverse and that each model's prediction is not influenced by the other. This condition can be very well fulfilled when we apply utterly different machine learning algorithms to predict the Solubility and combine their predictions (by aggregation through mean or median). We can also think about it as reducing Overfitting by introducing some randomness to the overall model and reducing the model's complexity (see Section 3.3.1). This process of getting predictions is called Consensus prediction, where predictions from many different models are aggregated together and enhanced through this aggregation.

Note that this idea has been used in Ensemble learning methods within machine learning for a long time. In our study, We will employ this method in ensemble learning methods to get

the final predictions by combining predictions from the top-5 performing models. Using this idea for final predictions will allow us to achieve better, more stable and reliable results than individual models.

3.6 Machine Learning methods

The machine learning algorithms mentioned in Section 1.3 will be discussed in detail in this section, helping to understand how each algorithm is different from the other and how they work.

3.6.1 Ordinary Linear Regression algorithms

These methods work well if a linear relationship exists between the independent and dependent variables.

Ordinary Least Squares (OLS)

The Ordinary least squares approach to linear Regression focuses on finding a linear relationship between the independent and the dependent variables. It assumes that the dependent variable is a linear function of independent variables, where linearity is checked in the coefficients of independent variables rather than the variables themselves. Whether a certain data is suited for linear Regression or not can be checked by checking it for linearity. Checking simple correlations or plotting the data and getting ideas about the structure can help. But one of the highly stressed ways to check for linearity is by plotting a residual-predictions plot which may make things clearer (Osborne and Waters, 2002). Such a model is highly affected by multicollinearity in the data. For an input vector $X = (X_1, X_2, ..., X_p)$, we can write equation for the full model as formula 3.10 (Hastie et al., 2009) below:

$$f(X) = \beta_0 + \sum_{j=1}^{p} X_j \beta_j \tag{3.10}$$

Where,

p = Total Number of independent features (or predictors)

$f(X)$ = predicted value

X_j = j^{th} independent variable

β_0 = intercept value or bias

β_j = estimated coefficient value for the j^{th} indpendent variable.

As seen from the equation, predictions are made by finding a linear relationship between the dependent and independent variables. Now, Let's see how the ordinary least squares method

work.

The key idea behind OLS is to find the relevant Coefficients/Weights $\beta = (\beta_0, \beta_1, ..., \beta_p)$, which minimises the sum of squared residuals. When we have training data $(x_1, y_1), ..., (x_N, y_N)$, where x_i is the i^{th} observation of an independent variable X_j and y_i is the i^{th} value of the dependent variable Y, then the sum of squared residuals is given by:

$$R(\beta) = \sum_{i=1}^{N} (y_i - f(x_i))^2 \tag{3.11}$$

Where,

N = Total number of observations

The term $(y_i - f(x_i))$ is called the i^{th} residual.

In other words, we can say that equation 3.11 (Hastie et al., 2009) is the cost function we aim to reduce to perform a successful linear regression.

Ridge Regression

Ridge regression is an extension of the OLS-based linear regression approach covered in the last section. It also has the same assumptions as the OLS approach and works very well if a linear relationship exists between the independent and dependent variables. Ridge regression uses Regularised least squares (RLS) instead of the Ordinary least squares approach. Let's understand what that means.

The ordinary least squares approach is prone to Overfitting (refer to Section 3.3.1). Overfitting in linear Regression causes an increase in the estimated coefficient values and makes the model more complex. This is what we mean when we say that the model learns each variable by merely memorising the data. Intuitively we can think of two ways to deal with this problem; first, we decrease the coefficient values or shrink them; second, we drive some of the coefficients which are not contributing much to the model to 0. This is precisely what is done in the case of regularised least squares, reducing the model's complexity and, therefore, Overfitting.

The difference between OLS and RLS lies in the cost function they try to minimise. We have seen the cost function for OLS in equation 3.11. Now let's look at the regularised cost function used by Ridge regression.

$$Loss = \sum_{i=1}^{N} (y_i - f(x_i))^2 + \lambda \sum_{j=1}^{p} \beta_j^2 \tag{3.12}$$

$$Loss = R(\beta) + \lambda R_g(\beta)$$

Where,

p = Total number of independent variables (or predictors)

N = Total number of observations

y_i = i^{th} value for the dependent variable

$f(x_i)$ = predicted value for the i^{th} observation

β_j = Coefficient for the j^{th} independent variable

λ = Regularisation Coefficient

As can be seen above in equation 3.12[3], an additional term is added to the OLS loss function, this term is called the regulariser. And here, we have a hyperparameter λ (Regularisation coefficient) which can be used to tune the model for good. The additional term is a penalty term and penalises large values of coefficients leading to an overall shrink in the coefficients. That is how we achieve a regularised least squares model with reduced model complexity.

Usually, when two or more correlated variables exist, a large coefficient on one gets cancelled by a similar large negative coefficient on its correlated variable. This issue is also resolved when we introduce a size constraint on the coefficients. (Hastie et al., 2009)

Lasso Regression

Lasso regression is also a regularised least squares (RLS) approach to linear Regression. It has the same assumptions as the OLS approach and is very similar to Ridge regression.

As explained in the section about Ridge regression, the regularised least squares approach brings down the model's complexity by putting a constraint on the coefficients of the independent variables. Unlike Ridge regression, where the algorithm shrinks the coefficients, some coefficients are driven to 0. Let's have a look at it's cost function in equation 3.13[4] below:

$$Loss = \sum_{i=1}^{N}(y_i - f(x_i))^2 + \lambda \sum_{j=1}^{p} |\beta_j| \tag{3.13}$$

$$Loss = R(\beta) + \lambda R_g(\beta)$$

Where,

p = Total number of independent variables (or predictors)

N = Total number of observations

y_i = i^{th} value for the dependent variable

$f(x_i)$ = predicted value for the i^{th} observation

β_j = Coefficient for the j^{th} independent variable

[3]Concept from: (Hastie et al., 2009)

[4]Concept from: (Hastie et al., 2009)

λ = Regularisation Coefficient

As can be seen, the difference between Ridge regression and Lasso regression is only in the regularisation method. If we compare the regularisation term between ridge and lasso, we can see that in lasso, instead of squared coefficient value, we consider only the magnitude of the coefficients. Due to this, we may get a model with some independent variables not contributing to the model because the coefficients are driven to 0. Finally, We again control the hyperparameter λ (Regularisation Coefficient) to tune our model.

Support Vector Regression (SVR)

Support Vector Machines, commonly used for classification jobs can be used for Regression. Let's understand how linear support vector regression works with the diagram below:

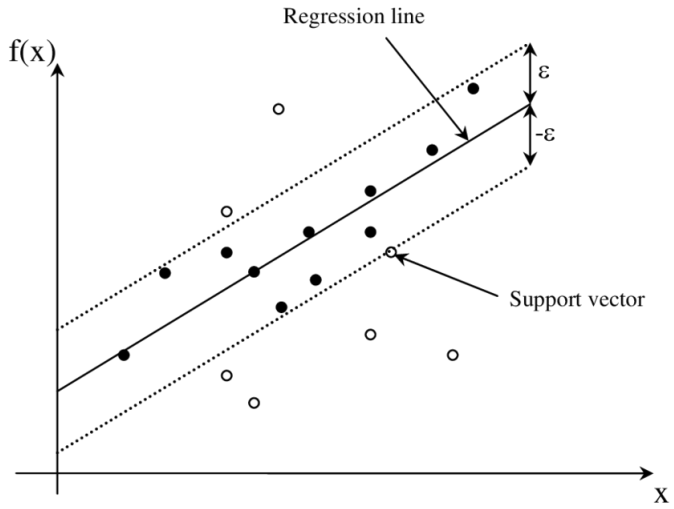


Figure 3.4: Linear Support vectors regression

As can be seen from figure 3.4[5], Support vector regression focuses on fitting the largest possible street covering all or most of the data-points. We can change the thickness of the street by changing the hyperparameter ϵ (also shown in the diagram above). This is done so that the data points in the street are maximised, and we omit outliers. The support vectors are the data points supporting the street's outer boundaries. Let's look at the cost function as given by Hastie et al. (2009) considering a simple 1-Dimensional linear SVR case:

$$H(\beta, \beta 0) = \sum_{i=0}^{N} V(y_i - f(x_i)) + \lambda ||\beta^2|| \tag{3.14}$$

[5]From: https://www.researchgate.net/profile/Hassen-Bouzgou-2/publication/316351306/figure/fig7/AS:485878301761550@1492853822
of-linear-SVM-regression-with-tube.png

$$V(r) = \begin{cases} 0 & \text{if } |r| < \epsilon \\ |r| - \epsilon & \text{otherwise} \end{cases}$$

Where,

y_i = i^{th} predicted value

β = estimated coefficient of independent variable

β_0 = intercept

λ = Regularisation coefficient

r = residual

ϵ = parameter which defines the thickness of the street

If we think about equation 3.14 (Hastie et al., 2009), we can see that if the residual (r) size is within ϵ then loss is considered 0. This is called the ϵ-insensitive nature of the algorithm, where any error of size less than ϵ is ignored. This also tells us that if we add more training points within the street region, the predictions remain unaffected, again showing that the model is ϵ-insensitive.

Support vectors also perform well, given that we can use hyperparameters to tune them. In this case, we can use two hyperparameters, ϵ and λ.

3.6.2 Latent-variable approaches

These approaches to machine learning generate a series of variables that are then utilised for modelling but do not exist by themselves. These variables are called latent variables or hidden variables. We will discuss two such approaches in this section, also clarifying the idea behind using such variables.

Principle Component Regression (PCR)

Principle Component Regression uses a linear combination of independent variables instead of the independent variables to perform regression. To understand how these linear combinations are calculated, we must understand Principle Component Analysis (PCA) which is the foundational concept of PCR.

Principle component Analysis (PCA) is a dimensionality reduction technique which projects the data on a plane in a manner that the maximum variance of the data is preserved in the projection. A simple example of 2-dimensional data is given in the figure 3.5[6] below:

[6]From: https://www.analyticsvidhya.com/wp-content/uploads/2016/03/2-1-e1458494877196.png

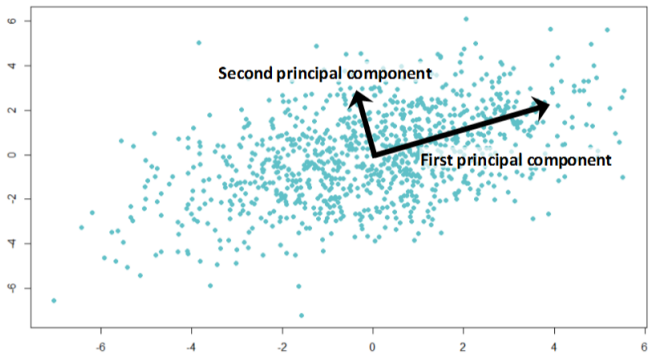


Figure 3.5: Principle Components in 2-dimensional data

PCA works by identifying the axes which preserve the maximum variance of the data when the data is projected on them, as shown in figure 3.5 above. This 2-dimensional example found these two lines or axes to project the data on. These axes are called principal components. For a case with more than two dimensions, it will project the data on as many axes as the Number of dimensions. Thus, the principle components are nothing but linear combinations of the independent variables. So in the case of n-independent variables, we may write the first component as:

$$z_1 = \beta_1 x_1 + \beta_2 x_2 + ... + \beta_n x_n \tag{3.15}$$

Where,

$x_1, ..., x_n$ = n independent variables

$\beta_1, ..., \beta_n$ = Coefficients for each indpendent variables after projection on the first axis.

We must note that this method is scale variant, meaning its behaviour changes with the scale of the independent variables. This also makes intuitive sense because if the scales of different variables are different, the variance captured will not be reliable. It is important to scale the independent variables to the same scale before using this method.

The idea behind Principle component regression is to use these components in place of independent variables to predict the dependent variable. The idea is to reduce dimensionality for the regression job by considering only those components that can explain most of the independent variables' data variance (say 95% or more). This is usually less than the number of independent variables, and this reduction in dimensionality causes our model complexity to remain low and avoids Overfitting. Another benefit of using PCA for Regression is that the principle components are uncorrelated and prevent multicollinearity. In our study, we treat the Number of components like a hyperparameter, find the number of components that gives the best results, and then fit a linear regression model over those components to get the predictions.

Partial Least Sqaures (PLS)

The partial least squares method of Regression forms a linear combination of independent variables considering the relationship between each independent variable and the dependent variable. PLS focuses on covering as much explained variance in the dependent variable as possible. Since PLS prioritises preserving dependent variable variance, the preserved independent variable variance may remain lesser than that in PCR for the same number of components. Like PCR, this method is also scale variant and demands that variables be scaled before processing. To summarise PLS, we can say that it forms components with high variance and a high correlation with the dependent variable (Hastie et al., 2009). Let's see how it works.

Hastie et al. (2009) explained mathematically that for a dataset containing standardised (scaled to have mean 0 and standard deviation 1) independent variables $x_1, x_2, ..., x_n$ and a dependent variable y, PLS starts by computing the inner-product $\hat{\varphi}_{1i} = \langle x_i, y \rangle$ for each i. This gives us the strength of effect that the independent variable has on the dependent variable. Then the first partial least squares component or direction is calculated by $z_1 = \sum_i \hat{\varphi}_{1i} x_i$, showing the dependent variable's role in constructing a component. Then the dependent variable y is regressed on z_1, and a coefficient $\hat{\theta}_1$ is obtained. Finally, the independent variables are orthogonalised with respect to z_1. The process is repeated until $M \leq n$ components or directions are obtained. Because we orthogonalised the variables, we get a sequence of orthogonal components, meaning they would be free of correlation among themselves. Finally, we use these components in our regression model instead of the independent variables.

A difference that we can observe between PLS and PCR is that PCR does not consider any relationship with the dependent variable when constructing the components. PCR may lose out on some of the predictive power if an independent variable has a high correlation with the dependent variable but has low variance. However, PLS may cover the predictive power well in such cases. But in cases where the opposite is true, PCR may give promising results. In the end, they both are dimensionality reduction techniques which help reduce the model's complexity and give us an uncorrelated set of variables for training. In our study, we find the number of components which offers the best results and then fit a linear regression model over those components to get the predictions.

3.6.3 Ensemble Learning approaches

Ensemble learning methods use the concept of collective intelligence or the wisdom of the crowd approach (see Section 3.5) to get predictions. A considerable benefit of such an approach is that it reduces overall model complexity, reduces the chances of Overfitting, makes the overall model more reliable and gives us better predictions (see Section 3.3.1 and 3.5).

Bagging Approaches

Bagging of Decision trees

Bagging is one of the widely used ensemble learning algorithms and finds its application as a foundational concept in some of the other ensemble learning algorithms. Let's understand how it works, figure 3.6[7] below.

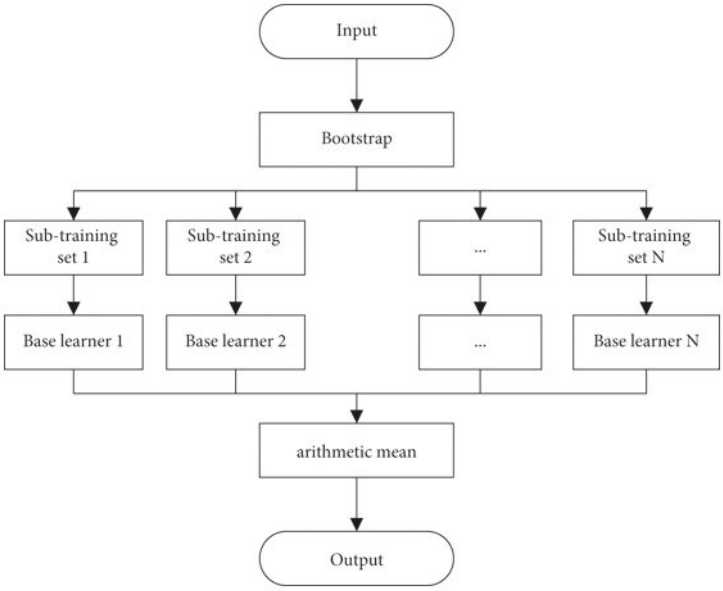


Figure 3.6: Steps in a general Bagging algorithm

In a bagging algorithm, small subsets of data are formed by randomly sampling records from the input training dataset with replacement[8], see figure 3.6. These subsets of data are the sub-training sets, usually called bootstrap samples. Then different machine learning models (referred to as Base learners in figure 3.6) are trained on these samples, as shown in the figure. Finally, each trained model is used to predict a given data, and then the predictions from all the models are averaged as a final prediction. This process of taking a particular model type, training over bootstrap samples, and aggregating predictions of many such models is called Bagging. In our study, we will be Bagging decision trees. Let's discuss how a decision tree works for Regression, and then we will discuss how bagging many decision trees gives better results.

We should understand that decision-trees for Regression generally use Mean Squared Error (MSE) as a measure for node impurity to decide the order of features (independent variables) to split on and also to determine the threshold value at which the feature is to be split, see Equation 3.5 for MSE. In other words, when making decisions on each node, a decision tree looks for a feature and its threshold value, which, if used for splitting the feature space, will give the lowest

[7]From: https://www.researchgate.net/publication/355764051/figure/fig2/AS:1085298373079102@1635766704425/The-flowchart-of-bagging-ensemble-learning-model.jpg

[8]With Replacement means we can sample that same record again after being sampled once.

MSE value compared to other features. The common loss function for a decision tree, when used for Regression, can be defined as:

$$\arg\min_{i,t}[MSE(y|x_i \leq t) + MSE(y|x_i > t)] \tag{3.16}$$

In equation 3.16[9], the i^{th} Independent variable is selected to split on, at the threshold value t. This splitting results in two split regions, the first one contains data for $x_i \leq t$, and the other one contains data for $x_i > t$. Predictions are made by averaging each partitioned region. MSE is calculated for both and then added together. This is the cost function we aim to minimise in a decision tree when used for Regression by selecting appropriate values for i and t.

Returning to the discussion on Bagging, when we consider bootstrap samples instead of the entire data for training decision trees, the splitting threshold (t) varies with each tree, and we get entirely different decision trees. This is interesting and aligns with the concept of the wisdom of the crowd, where each model is expected to be independent and as diverse as possible to perform better collectively.

Another way to look at it is that individual decision-trees tend to have high variance and are prone to Overfitting, so a small perturbation in data can alter the whole decision-tree (Dietterich and Kong, 1995). Now, when we train many decision trees over different bootstrap samples, each decision tree can learn different possible data variations. A form of randomness is introduced in our overall bagged model, reducing the model's variance and reducing Overfitting. Thus, when we take aggregation of their predictions, a bagged model tends to give more reliable and better predictions.

Random forest

The random forest algorithm is an extension of the "bagging of decision trees" algorithm. We can still refer to the same diagram in figure 3.6 to understand the general workflow of the random forest algorithm. The difference lies in how it creates its bootstrap samples compared to the general decision-tree bagging approach. The rest of the workflow is explained in detail in the previous section. Let's understand how random forest differs in the bootstrapping stage.

In the general decision-tree bagging approach, random subsets are taken only from the records (observations) and all features (independent variables) are included in each subset. However, In random forest, along with taking a random subset of the records, we also take a random subset of the features. And again, sampling is done with replacement. So, in the case of random forest, one bootstrap sample may have only a subset of features and not all of them. Then decision trees are trained on each subset. Now, each decision tree will select its best splitting feature

[9]Concept from: (Hastie et al., 2009)

and threshold only from the subset of features in the bootstrap sample; therefore, every decision tree will differ even more. This method introduces diversity, randomness and independence in each trained decision tree.

We must note that in decision-tree bagging approaches, we aim to reduce the model's complexity. As mentioned earlier, decision trees have high variance (Dietterich and Kong, 1995). So, as more variance reduces, we can expect better results. By making decision trees more diverse and independent, random forest helps in reducing variance even further and reducing the model's complexity.

Extremely Randomised Trees

This is another bagging approach, often referred to as the Extra-Trees algorithm. As the name suggests, it can be understood as a more randomised version of the random forest approach introducing more randomness and reducing the model's variance. But it has significant differences with the general decision-tree bagging and the random forest algorithms.

As mentioned by Geurts et al. (2006), The Extremely-randomised trees train many decision trees on the training data (not on the bootstrap samples) and then, to get a final prediction, it combines their results by aggregation. It randomly samples features and the threshold to split on for training the decision trees. We can see differences between this algorithm from the two previously discussed. A similarity between random forest and extremely-randomised trees is that the features are sampled at random in both. Whereas both differ in the way, they select their splitting-threshold point.

As mentioned previously, this method further reduces the model's complexity and enhances overall model performance. However, it must be noted that there is a trade-off between variance and bias (see Section 3.2), and we must make sure that we don't make our model too simple, as that can again make predictions unreliable.

Boosting Approaches

Adaptive Boosting

Boosting is another type of Ensemble learning method. We know now that Bagging works by the parallel formation of many models and aggregating their results to get the final result. Boosting works in a slightly different manner. Let's understand the workflow of a boosting algorithm,

Boosting algorithm trains weak learners[10] sequentially with an improvement of the successive learners by conditioning them on the errors made by the previous weak learner. Finally,

[10] a simple machine learning model, with high bias and low variance

on a given data, the boosting algorithm takes a weighted aggregation of all the weak learners' (forming an overall strong learner) predictions to give a final prediction. So, we can say that a boosting algorithm focuses on building many simple models (high on bias and low on variance) and then increases their complexity (reduces bias and increases variance) by their weighted aggregation. To understand better refer to Section 3.3.1.

Let's see how a simple boosting algorithm works as explained by Drucker (1997):

1. It assigns weights to each sample (a record) of the data in a way that the sum of all the sample weights is equal to 1. Initially, equal weights are assigned to each sample.

2. A weak decision tree is built on this data, and predictions are made on the training set itself, giving $\hat{y}_i$ for $i = 1, 2, 3, ..., n$, where n is the Number of samples/records.

3. Loss is calculated for each sample. In our study we consider linear loss, given by equation 3.17 (Drucker, 1997):

$$L_i = \frac{|\hat{y}_i - y_i|}{D} \tag{3.17}$$

   Where,

   y_i = i^{th} dependent variable value

   $D = sup|\hat{y}_i - y_i|$ for $i = 1, 2, 3, ..., n$

   Note that "sup" stands for supremum, which takes the smallest number greater than or equal to any number in the set.

4. The loss average is taken to allot confidence (weight or coefficient) value to the weak learner.

5. Sample weights are updated using this confidence value. This will determine the chances of the following learner picking a particular sample. If a sample's prediction was bad previously, the sample weight is increased so that it is prioritised by the next learner to learn. And if a sample's prediction was acceptable, its weight is reduced, making it a lower priority for the next weak learner to learn.

6. Whole process is repeated, and many such learners are formed. Finally, an aggregation of their predictions is weighted by their confidence value for the final prediction.

The steps mentioned above summarise Adaptive boosting, which uses a single node decision-tree (stump) as its weak learner and successive learners adapt based on the previous learner's performance.

Gradient Boosting

Gradient boosting is another widely used boosting algorithm that generally performs very well. As a boosting algorithm, it follows the same workflow of sequentially training weak learners and then combining them to build an overall strong learner. A primary difference between Gradient boosting and Adaptive boosting is that Gradient boosting uses a differential loss function and employs gradient descent optimisation for minimising loss.

Let's understand the algorithm in a summary, as explained by Hastie et al. (2009) and Bentéjac et al. (2021), simplified below:

1. The algorithm starts by building a very simple decision tree. And by using this model, we get initial predictions.

2. For each record in the data, using the predicted values, we calculate pseudo-residuals. Usually calculated by equation 3.18 (Hastie et al., 2009) below:

$$r_{im} = -\left[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}\right]_{f=f_m(initially)} \tag{3.18}$$

   Where,

   r_{im} = i^{th} residual obtained from m^{th} model.

   y_i = i^{th} value of dependent variable

   $f(x_i)$ = i^{th} predicted value

3. Now, the next decision tree is trained taking these pseudo-residuals as the target. let's denote the new model as $h_{m+1}(x)$.

4. Current model is updated using weighted addition of the new model, can be seen below in equation 3.19 (Bentéjac et al., 2021):

$$f(x) = f_m(x) + \lambda(\rho_{m+1} h_{m+1}(x)) \tag{3.19}$$

   Where,

   λ = Learning rate, it can be used as a hyperparameter to tune the model's performance.

   ρ_{m+1} = coefficient or weight of the $(m+1)^{th}$ model

   Note, the value of ρ_{m+1} is calculated by optimising the cost function shown below, as understood from Bentéjac et al. (2021):

$$\rho_{m+1} = \arg\min_{\rho} \sum_{i=1}^{N} L(y_i, f_m(x_i) + \rho h_{m+1}(x_i))]$$

These steps are repeated until we get a good-performing model with accurate predictions. It can be seen from equation 3.19 that the final model is nothing but an application of gradient descent optimisation. Since the algorithm uses gradient descent optimisation, it is much smoother and can achieve better loss-function minimisations at times.

3.6.4 Kernel-based approaches

These are those algorithms which make use of kernels to capture complex information from the data. They allow non-linearity to be introduced and therefore make it easy for the model to capture non-linear information from the data (Hastie et al., 2009).

Non-linear Support Vector Regression

This method follows the same concept and background as the linear SVR, mentioned in Section 3.6.1. The difference is that it uses basis functions to transform its inputs so that we can introduce non-linearity, as seen in the matrix form in Equation 3.20 (Hastie et al., 2009) below. Capturing non-linear patterns from data allows the model to perform much better as the model can fetch more information from the data.

$$f(x) = h(x)^T\beta + \beta_0 \tag{3.20}$$

It is found that the basis function exists only in inner products on expanding and simplifying this equation. So, instead of using basis functions, we use kernels which give us these inner products. Equation 3.21 (Hastie et al., 2009) shows equation for kernel 'K':

$$K(x, x') = \langle h(x), h(x') \rangle \tag{3.21}$$

Kernels aim to find similarities between points, and therefore many variations of kernels exist. In our study, we use the Radial basis function (RBF) which is given below in Equation 3.22 (Hastie et al., 2009):

$$K(x, x') = \exp(-\gamma||x - x'||^2) \tag{3.22}$$

γ in Equation 3.22 acts as a hyperparameter. Apart from γ, all the hyperparameters used for this model will remain very similar to those used in Linear Support Vector Regression.

Gaussian Process Regression

We know that one can fit infinitely many functions for a given data set. For better Regression, we want to select the function which explains the data perfectly. Gaussian process (GP) is a bayesian approach to Regression that gives us a probability distribution over such functions. Then from that distribution, we finally sample a function which best explains our data and gives

the best predictions.

As mentioned, GP is a distribution over functions. It is a set of random variables where the joint distribution of random variables' subsets is a joint Gaussian distribution. Functions are considered as the random variables and can be written as shown in Equation 3.23 (Melo, 2012):

$$f(x) \sim GP(m(x), K(x, x')) \tag{3.23}$$

Where '$m(x)$' is mean and is commonly assumed to be zero (Bishop and Nasrabadi, 2006), and 'K' stands for the kernel; refer to the last section where it is discussed in some detail. Let's understand how it solves the problem of Regression using bayesian as explained by Melo (2012), understanding summarised below:

1. Take a training data $(X, Y) = (x_i, y_i)_{i=1}^{N}$, where we assume that $y_i = f(x_i) + \epsilon$ (gaussian noise with variance σ) .

2. Now, we assume Gaussian Process (GP) for y. This gives us a gaussian prior distribution[11], $y \sim \mathcal{N}(0, K(X, X) + \sigma^2 I)$, where I is the Identity Matrix.

3. Now, test data is considered X_*. Because of our GP assumption, We can take joint distribution for our random variables y and y_* (prediction values at X_*), Equation 3.24 (Melo, 2012):

$$\begin{bmatrix} y \\ y_* \end{bmatrix} \sim N\left(0, \begin{bmatrix} K(X, X) + \sigma^2 I & K(X, X_*) \\ K(X_*, X) & K(X_*, X_*) \end{bmatrix}\right) \tag{3.24}$$

4. Conditional probability is taken for the Posterior distribution. This is what we want, the probability distribution for predicted values after observing the test input data, Equation 3.25 (Melo, 2012):

$$y_* | y \sim \mathcal{N}(m_*, \sigma_*) \tag{3.25}$$

   Finally, we get our prediction mean (m_*) and standard deviation value (σ_*). Predictions can be sampled from this distribution for each test data point.

We use Radial Basis Function in this study. The function is further expanded and defined for gaussian process by GPy (https://github.com/SheffieldML/GPy), Formula 3.26 (Duvenaud, 2013) below:

$$K(x, x') = \sigma^2 \exp\left(-\frac{(x - x')^2}{2\ell^2}\right) \tag{3.26}$$

Where,

[11] Prior distribution means a distribution which shows our beliefs about the random variable before observing test data. In this case, it means that this distribution is our belief regarding the predictions of test data before actually observing the test data

σ^2 = Variance of the radial function.

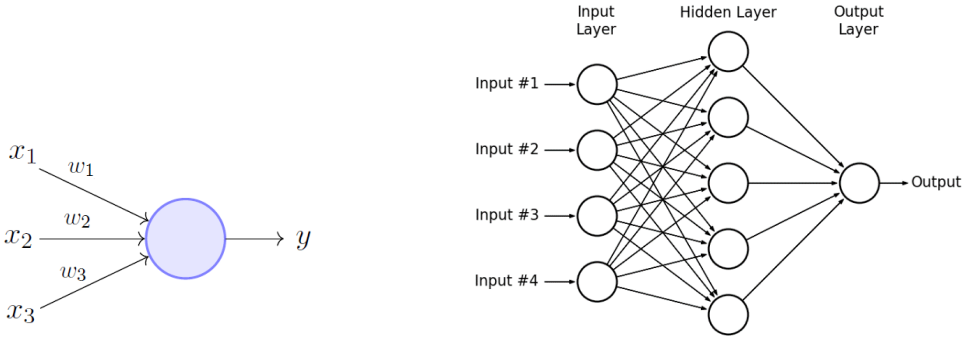
l = lengthspace controls the wiggles in our function.

Both σ^2 and l are hyperparameters for us in Gaussian Processes Regression.

3.6.5 Artificial Neural Networks Approach

Multi-layer Perceptron

The idea of artificial neural networks is inspired by the natural neural networks which run in the human body. This idea is used to design a Perceptron which acts as an artificial neuron, and we can use a combination of perceptrons to give us an artificial neural network.



(a) A Single Perceptron

(b) Multi-layer Perceptron Network

Figure 3.7: Constructing an Artificial Neural Network

As can be seen from Figure 3.7a[12], Perceptron (single neuron) takes in a couple of inputs, and the output is simply a weighted aggregation of those inputs. Usually, the output from Perceptron is modified by passing it through an activation function. An Activation function transforms the output and makes it more flexible to learn complex patterns from the data[13]; this process is called Activation.

Figure 3.7b[14] shows a Neural Network made of many layers of perceptrons connected together. It has an input layer which takes in the inputs, a hidden layer (could be more than one) where inputs are further processed by weighting and an output layer from where we take the final output. For Regression, Activation of the hidden layer can help the model learn complex patterns. After the weights are learnt, and necessary hidden layer activation is applied, the neural network model is ready to perform predictions. Let's discuss briefly how weights are learnt, and model is trained.

[12]From: https://www.researchgate.net/profile/Nicola-Fronzetti/publication/335609766/figure/fig1/AS:799455209078784@156761638349
Minsky-Papert-1969.ppm

[13]For example, a non-linear activation function is very helpful in making the model learn non-linear patterns

[14]From: https://www.researchgate.net/profile/Mohamed-Zahran-16/publication/303875065/figure/fig4/AS:371118507610123@1465492
hypothetical-example-of-Multilayer-Perceptron-Network.png

In training, the model minimises a loss function by finding appropriate weights. It uses Stochastic Gradient Descent (SGD) optimisation algorithm to find the gradient of the loss function for each weight and then use that gradient to update respective weights, see Equation 3.27 (Pedregosa et al., 2011) below.

$$w_{new} \leftarrow w_{old} - \eta(\alpha \frac{\partial R(w)}{\partial w} + \frac{\partial Loss}{\partial w}) \tag{3.27}$$

Where,

η = Learning rate, a hyperparameter.

$R(w)$ = Regularisation term, L2-norm (Pedregosa et al., 2011)

α = Regularisation coefficient, a hyperparameter.

We will be considering Mean Squared Error (MSE) as the loss function for Multi-layer Perceptron in this study, see Equation 3.5 for reference.

3.7 Python 3 packages for the Machine Learning algorithms

Algorithm	**Abbreviation**	**Package**
Ordinary Least Squares	OLS	sklearn.linear_model.LinearRegression
Ridge Regression	RR	sklearn.linear_model.Ridge
Lasso Regression	LaR	sklearn.linear_model.Lasso
Support Vector Regression	SVR	sklearn.svm.LinearSVR
Principle Component Regression	PCR	sklearn.decomposition.PCA
Partial Least Squares	PLS	sklearn.cross_decomposition.PLSRegression
Bagging (decision-trees)	BAG	sklearn.ensemble.BaggingRegressor
Random Forest	RF	sklearn.ensemble.RandomForestRegressor
Extremely Randomised Trees	ET	sklearn. Ensemble.ExtraTreesRegressor
Adaptive Boosting	ADAB	sklearn.ensemble.AdaBoostRegressor
Gradient Boosting	GRADB	sklearn.ensemble.GradientBoostingRegressor
Non-linear Support Vector Regression	KSVR	sklearn.svm.SVR
Gaussian Process Regression	GP	GPy.models.GPRegression
Multi-layer Perceptron	MLP	sklearn.neural_network.MLPRegressor

Table 3.1: Python 3 packages and methods for the machine learning algorithms

Table 3.1 lists down the main machine learning Python 3 packages and their methods used in this study. Two primary packages used are Scikit-Learn (Pedregosa et al., 2011) and GPy (https://github.com/SheffieldML/GPy).

Chapter 4

Results and Discussion

Methodologies were implemented in Python 3. To see the full Python 3 implementation, Please check `https://github.com/AdnanMalik0` and request private repository access by emailing at **mm21am@leeds.ac.uk** OR **adnanmallick29@gmail.com**. This section covers the results obtained and what they concluded. Discussions and Comparisons are shown as we go ahead section by section.

4.1 Data

The data was taken from Boobier et al. (2020), who generated most of the variables using 3D modelling of molecules. The full dataset contained 900 records and 15 columns[1]. The data-dictionary can be seen in Table 4.1 (Boobier et al., 2020) below:

Variable	Description
Compound	Compound identifier
G_sol	Gibbs free energy of optimised solution structure (Hartrees)
DeltG_sol	Solvation energy (Hartrees)
volume	Molar volume ($cm^3.mol$)
sol_dip	Dipole moment of solution structure (Debye)
O_charges	Sum of charges on solution structure oxygen atoms
C_charges	Sum of charges on solution structure carbon atoms
Most_neg	Charge on most negative atom of solution structure
Most_pos	Charge on most positive atom of solution structure
Het_charges	Sum of charges on solution structure non-hydrogen/carbon atoms
MW	Molar volume ($cm^3.mol$)
SASA	Solvent Accessible Surface Area ($Å^2$)
LogS	Logarithm of Solubility (mol^{-l})
Lsolu_Hsolv	Energy gap between solute LUMO and solvent HOMO (eV)
Lsolu_Hsolu	Energy gap between solvent LUMO and solute HOMO (eV)

Table 4.1: Data Dictionary

[1]Apart from 'Compound' and 'LogS', each column represents the generated descriptors

All variables except 'Compound' and 'LogS' are the Independent Variables, and 'LogS' is the Dependent Variable we aim to predict.

4.1.1 Data Discrepencies

Data was found free of any Missing Values (NA, null, None, <blank>, etc.). Let's look at the Histograms of each variable to develop an understanding of them and find any data discrepancies, see Figure 4.1. This is a beneficial step as a Histogram Plot helps us to understand the data and plan our next steps according to that understanding.

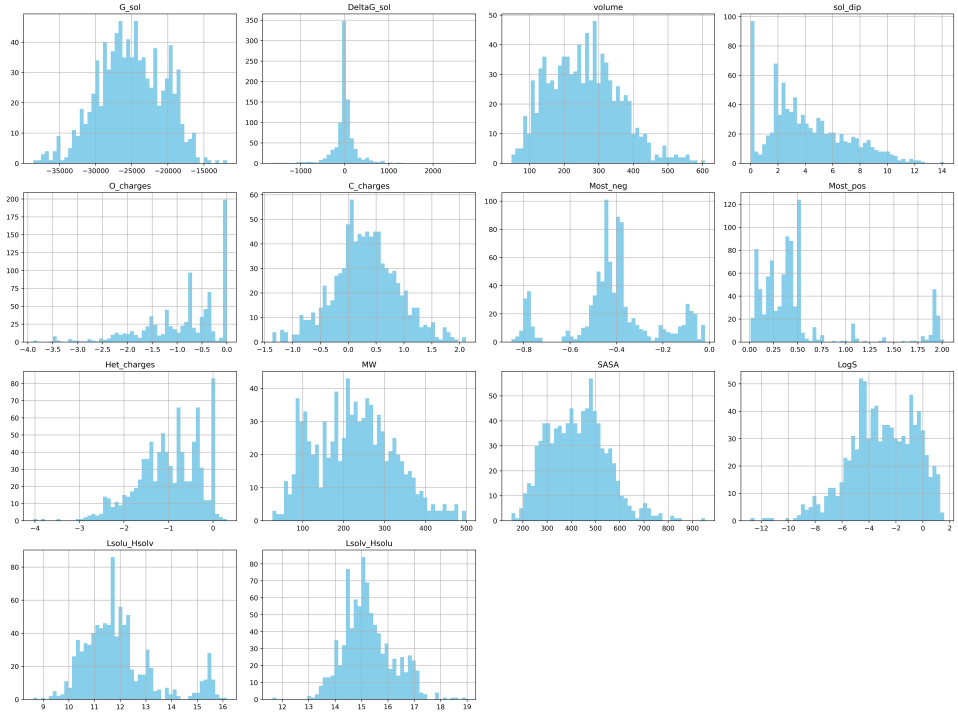


Figure 4.1: Histogram of the variables

Upon looking at Figure 4.1, we can see that the scale varies highly between each variable, and this indicates that we should consider scaling our data for machine learning algorithms to perform better, especially those which are sensitive to feature scaling. Looking for any special value (such as outliers), we did not discard any data-point because each was considered for its relevance in chemistry by Boobier et al. (2020), and the histograms did not suggest anything significant to oppose that.

4.1.2 Correlations among indpendent variables

As we saw, our data contains 13 independent variables and a dependent variable (LogS). Here, we use Pearson's r^2 and Variance Inflation Factor (VIF) to find high correlations and issues of multicollinearity among the independent variables, see figure below.

Feature-1	Feature-2	r^2
G_sol	volume	0.9386
G_sol	MW	0.9148
G_sol	SASA	0.9263
volume	MW	0.8868
volume	SASA	0.9592

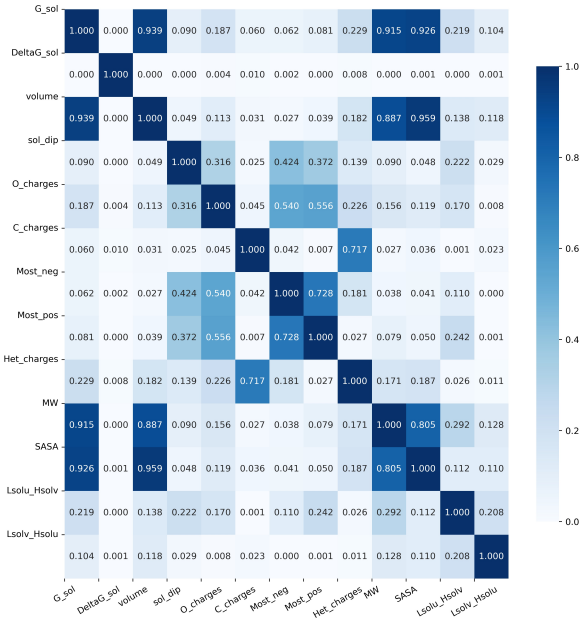


Figure 4.2: Pearson's r^2 Correlation for independent variables

From Figure 4.2, we intend to find all correlated pairs with r^2 value roughly more than 0.9 (Boobier et al., 2020). We can solve the high correlation problem by removing one from each pair. As can be seen from the figure that 'G_sol' shows a high correlation with three different independent variables. The other one is 'volume', which also shows a high correlation with three different independent variables (G_sol being one of them). Now, let's see what the Variance Inflation Factor (VIF) looks like for the variable and how it changes when we remove 'volume' and 'G_sol' one by one.

	feature	VIF
0	G_sol	55.477487
1	DeltaG_sol	1.108167
2	volume	61.561986
3	sol_dip	2.085399
4	O_charges	4.608036
5	C_charges	8.156544
6	Most_neg	7.069496
7	Most_pos	8.290925
8	Het_charges	10.435531
9	MW	27.351018
10	SASA	53.267101
11	Lsolu_Hsolv	3.186777
12	Lsolv_Hsolu	2.381385

(a) All features

	feature	VIF
0	G_sol	55.409652
1	DeltaG_sol	1.107254
2	sol_dip	2.085370
3	O_charges	4.601918
4	C_charges	7.994726
5	Most_neg	6.988946
6	Most_pos	7.802351
7	Het_charges	10.380055
8	MW	17.373417
9	SASA	25.751757
10	Lsolu_Hsolv	3.125059
11	Lsolv_Hsolu	2.379572

(b) No 'volume'

	feature	VIF
0	DeltaG_sol	1.029306
1	sol_dip	2.082639
2	O_charges	4.165492
3	C_charges	7.080361
4	Most_neg	6.963510
5	Most_pos	7.688017
6	Het_charges	9.855356
7	MW	8.745510
8	SASA	6.900846
9	Lsolu_Hsolv	2.959558
10	Lsolv_Hsolu	2.379506

(c) No 'volume' & 'G_sol'

Figure 4.3: Variance Inflation Factor for the independent variables

From Figure 4.3a, we can see that variables had large VIF values when we included 'G_sol' and 'volume' in the dataset. We removed 'volume' first and then checked VIF (see Figure 4.3b), we found high VIF (>10) in 'G_sol' and others. Then we removed 'G_sol' and found that VIF values for the remaining variables dropped significantly to values less than 10, **So we keep the resulting dataset as our final dataset (see Figure 4.3c)**. This helped us deal with the issue of

multicollinearity in our dataset. It also shows us that much of the information contained within 'volume' and 'G_sol' can still be accessed from the remaining independent variables because of their high correlation they had with the remaining variables.

4.1.3 Checking relationship between independent and dependent variables

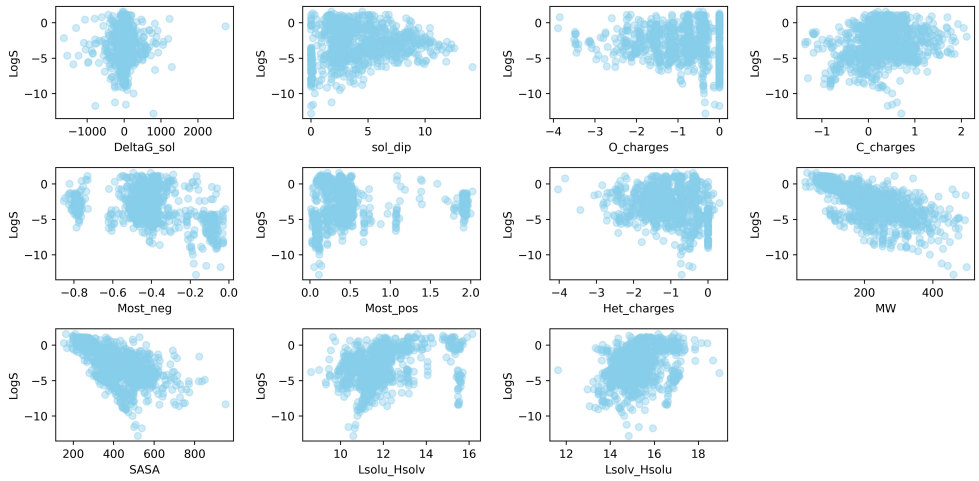


Figure 4.4: Scatter-plots between dependent and independent variables

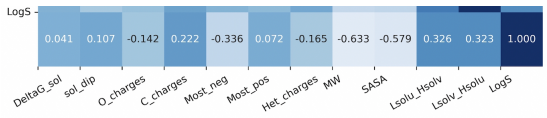


Figure 4.5: Pearson's r between dependent and independent variables

Looking at Figure 4.4, we can say that it's hard to tell whether the independent variables relate linearly to the dependent variable, except 'SASA' and 'MW', which do show some form of pattern resembling a negative linear relationship. This can also be seen from Figure 4.5, where 'SASA', 'MW' and 'Most_neg' seems to have comparatively higher negative correlation (r) value than others. Others do not show any clear linear relationship, which suggests that linear relationships might be challenging to find. However, the opposite is also true, which means that non-linear models might have the upper hand over linear models.

4.2 Model's Outputs

After getting the data ready for modelling, we split data into a training set (80%) and a test set (20%). We trained the model over the training set. Model parameters were tuned using Grid-search[2] except in Latent variables models (PCR & PLS) where we selected the best number of

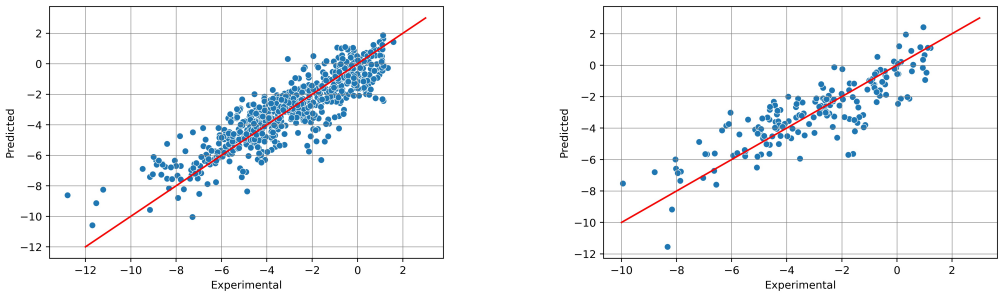
[2]Grid-search uses cross-validation to find hyperparameter values which best minimises the Mean Squared Error (MSE).

components to optimise the model and Gaussian Process Regression (GP) where optimisation is done based on log-likelihood and prior. Finally, the trained model was evaluated by Cross-Validation and then evaluated on the test set. This section discusses the model's performance on cross-validation and the test set.

4.2.1 Ordinary Linear Regression Models

Ordinary Least Squares (OLS)

We created the model with the default hyperparameters, and there was no avenue for hyperparameter tuning. The following plots and tables summarise the results of OLS.



(a) Cross-validation predictions

(b) Test-set predictions

Figure 4.6: Predictions versus Experimental plots (OLS)

RMSE	r^2	%P±0.7	%P±1.0
1.1699	0.7764	49.86	65.0

Table 4.2: CV results (OLS)

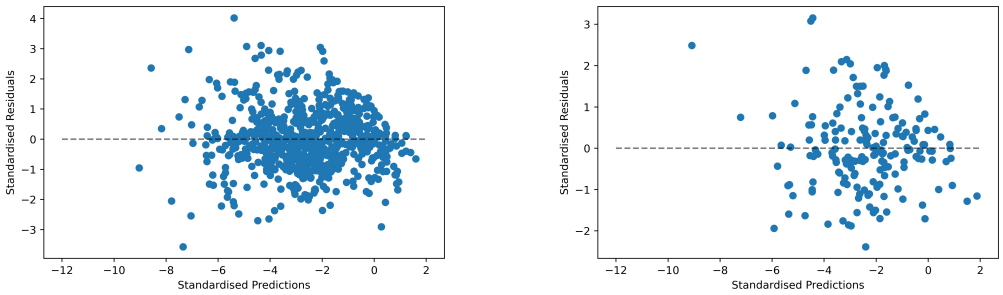
RMSE	r^2	%P±0.7	%P±1.0
1.2728	0.7217	49.44	58.88

Table 4.3: Test-set results (OLS)

Figure 4.6 shows that the predictions are somewhat scattered, with some points far from the experimental value. From tables 4.2 & 4.3, model seems to give better predictions on CV and worse on the test-set suggesting slight overfitting. Considering %P±0.7 from both the tables, overall performance is less than 50%, which means that 50% predictions were acceptable.

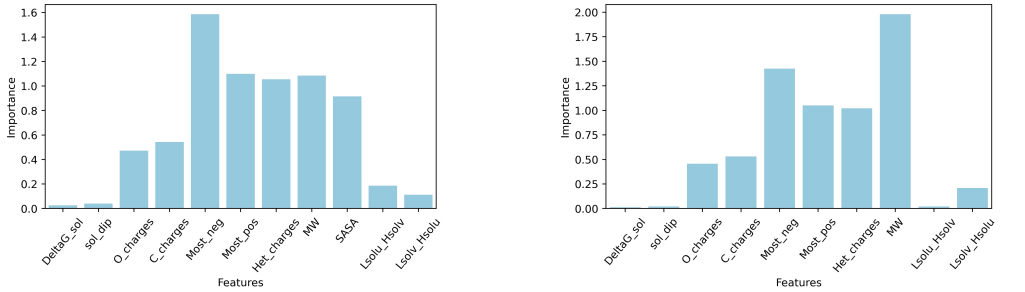
Let's check if our independent variables have a linear relationship with the dependent variable using a residual-predictions plot for both (CV and test-set predictions). As shown by Osborne and Waters (2002)[3], a residual-predictions plot with points scattered randomly around the residual=0 line means that there is indeed a linear relationship between independent and dependent variable. In our case, as seen from Figure 4.7, it is clear that the data points are not randomly distributed but instead look clustered in one area. This clustering indicates that linear models might not be an appropriate answer to our solubility prediction problem.

[3]Osborne and Waters (2002) showed that to check linearity between independent and dependent variables, a residuals-predictions plot is a preferable method



(a) CV residual-predictions (b) Test-set residual-predictions

Figure 4.7: Residuals-Predictions plot (OLS)



(a) Features importance (OLS) (b) Features importance without SASA (OLS)

Figure 4.8: Features Importance plots (OLS)

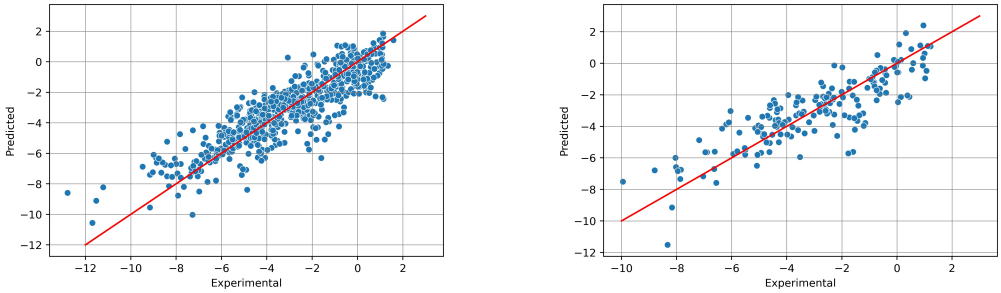
Before proceeding, we need to mention a very important discussion about the model's coefficients and correlation between variables. Using absolute values of model's coefficients[4], Figure 4.8a shows variable importance. 'MW' or 'SASA' were expected to have higher importance because of the higher correlation they had with 'LogS', but instead, we see 'Most_neg' having higher importance. The reason behind this is because there is somewhat high correlation between 'MW' and 'SASA' of 0.80 (Figure 4.2) which causes several issues (see Sections 3.2 & 3.6.1) and also makes our model's coefficients uninterpretable. When we remove one of these two variables (removing 'SASA' as it has comparatively high VIF) and retrain our model, we see that suddenly 'MW' became more important than 'Most_neg', see Figure 4.8b. This proves our point about correlation and its effect on the model's coefficients. Even though we see such a behaviour, we did not discard 'SASA' from the dataset because it contained some relevant independent information for Solubility prediction (discussed in detail in the coming sections, more specifically in Section 4.4). Since it makes interpretability almost impossible, we will not discuss model coefficients for any linear model.

[4]To see model's coefficients, check Appendix

Ridge Regression (RR)

We trained the Ridge Regression model over the data. Hyperparameter λ (denoted as α in Scikit-Learn) was tuned, and the value which gave the best results was found to be 'α=**0.99**'.

Figure 4.9 shows no significant difference between this model and OLS, with only a slight improvement due to regularisation. Similar can be seen from Tables 4.4 & 4.5. However, %P±0.7 here is exactly 50% (better than OLS). Other metrics have also improved slightly. As mentioned in the last section about model coefficients and correlation, the issue of uninterpretable coefficients persists with RR because it shares the same assumptions as OLS. However, we observed a shrink in coefficients in RR as compared to OLS [5]. Also, using the residual plot from the last section, we can say that the minimal enhancement in performance is most probably due to the lack of linearity.



(a) Cross-validation predictions

(b) Test-set predictions

Figure 4.9: Predictions versus Experimental plots (RR)

RMSE	r^2	%P±0.7	%P±1.0
1.1697	0.7765	50.0	64.86

Table 4.4: CV results (RR)

RMSE	r^2	%P±0.7	%P±1.0
1.2712	0.7222	49.44	60.0

Table 4.5: Test-set results (RR)

Lasso Regression (LaR)

We trained the Lasso Regression model over the data. Hyperparameter λ (denoted as α in Scikit-Learn) was tuned, and the value which gave the best results was found to be 'α=**0.01**'.

RMSE	r^2	%P±0.7	%P±1.0
1.1717	0.7760	50.83	65.0

Table 4.6: CV results (LaR)

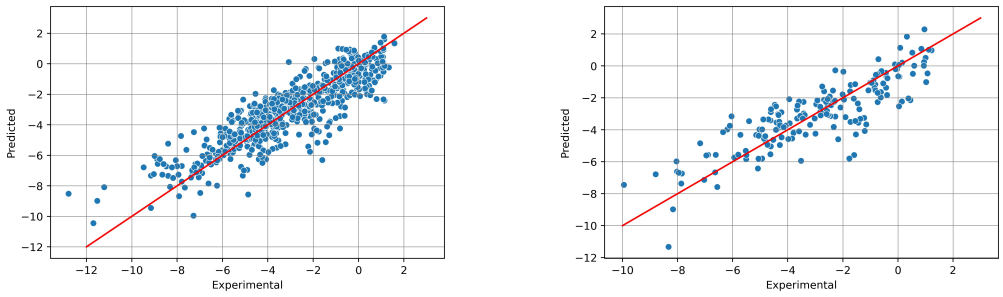
RMSE	r^2	%P±0.7	%P±1.0
1.2623	0.7249	48.86	61.11

Table 4.7: Test-set results (LaR)

Figure 4.10 again shows no significant difference. The coefficient values of the variables dropped significantly[6]. We see a slight better value for %P±0.7 in CV Table 4.10a but the

[5] To see the model's coefficients, check Appendix

[6] To see the model's coefficients, check Appendix



(a) Cross-validation predictions

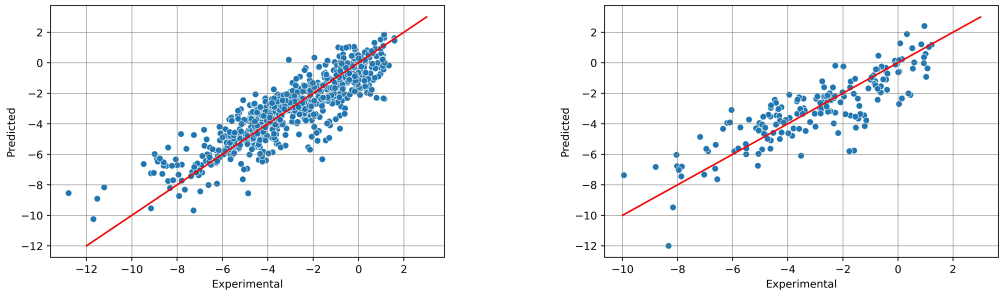
(b) Test-set predictions

Figure 4.10: Predictions versus Experimental plots (LaR)

performance drops in the test-set 4.10b. LaR gives better RMSE on the test set than OLS and RR, but we cannot conclude it to be the best performing because of comparatively lower values for %P±0.7. The performance again did not improve much. Even regularisation did not improve results much, as seen in RR and LaR, most likely due to a lack of linear relationship between independent and dependent variables (see Figure 4.7).

Support Vector Regression (SVR)

We trained the linear Support Vector Regression model on the data. Hyperparameter ϵ was tuned, and the value which gave the best results was found to be 'ϵ=**0.54**', with no regularisation (λ).



(a) Cross-validation predictions

(b) Test-set predictions

Figure 4.11: Predictions versus Experimental plots (SVR)

RMSE	r^2	%P±0.7	%P±1.0
1.1702	0.7764	49.58	65.41

Table 4.8: CV results (SVR)

RMSE	r^2	%P±0.7	%P±1.0
1.2875	0.7178	50.0	61.11

Table 4.9: Test-set results (SVR)

Overall, this model also did not give any significant difference in its results. An interesting bit here is that this model gave better results on test-set than CV (Tables 4.8 & 4.9) in terms of %P±0.7. However, other metrics gave similar results to the last three linear models.

All the linear models seem to perform similarly. A huge reason that no significant improvement was observed even after using regularisation and SVR is the lack of linear relationship between independent and dependent variables, as shown by residual-prediction plots (Figure 4.7).

4.2.2 Latent-variable Models

Principle Component Regression (PCR)

We performed Principle Component Analysis on the training data and got eleven principle components for all the eleven descriptors (independent variables). See the variance explained by each below in Table 4.10.

n	1	2	3	4	5	6	7	8	9	10	11
Var.	0.389	0.168	0.159	0.09	0.073	0.042	0.036	0.02	0.007	0.005	0.004

Table 4.10: Number of Components (n) and their respective Explained Variance (PCR)

As seen from Table 4.10 above, the first seven components can explain variance more than 95%. But as stated in Section 3.6.2, we shouldn't decide on a number of components for our model based only on the amount of independent-variable variance it explains because there might be some variables with a low variance which have some correlation with the dependent variable and is not correctly represented in the high-variance components. And, we may lose some predictive power if we discard many low-variance components. Therefore, we prepare many models with increasing components and evaluate them by cross-validation to get the final best number of components.

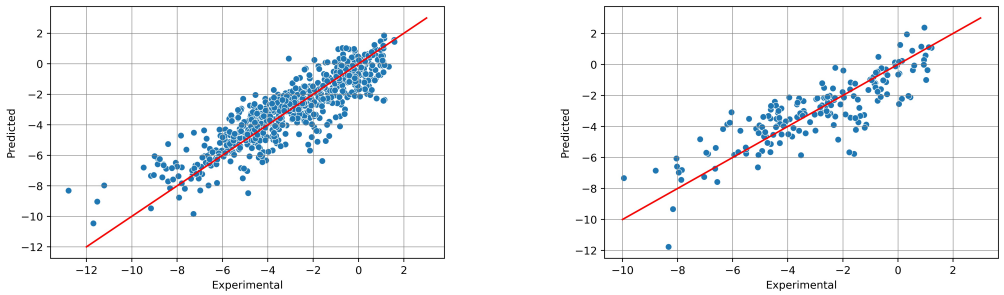
n	RMSE	Pearson-rsquare	%P±0.7	%P±1.0
1	2.431	0.0148	19.58	27.08
2	1.6928	0.5268	33.19	45.27
3	1.6085	0.5721	38.75	51.80
4	1.6115	0.5704	38.47	52.22
5	1.3852	0.6880	40.83	54.02
6	1.3551	0.6977	41.11	56.25
7	1.3128	0.7192	42.77	58.61
8	1.2791	0.7331	45.41	61.80
9	1.1992	0.7652	49.30	63.75
10	1.1671	0.7777	50.13	65.41
11	1.1699	0.7764	49.86	65.0

Table 4.11: Model's CV results with increasing Number of Components' n' (PCR)

Based on Table 4.11 and prioritising %P$\pm$0.7, we can select for our final model the **first 10 components** as model gave best results. This also proves our point about not only relying on the independent variable's variance. Now, after fitting a linear regression model on these

components, let's consider the model's performance.

From Figure 4.12 and Tables 4.12 & 4.13, We can say that the performance based on %P±0.7 is similar but somewhat better than the previously discussed Linear models. The fact that %P±0.7 did not change between CV and Test-set suggests some stability in the model. This could be because of using principal components that are not correlated (Section 3.6.2) and therefore make the model perform better in some way. But, overall model's performance still did not show a significant enhancement making us conclude again that this is probably because of the lack of linear relationship between dependent and independent variables.



(a) Cross-validation predictions

(b) Test-set predictions

Figure 4.12: Predictions versus Experimental plots (PCR)

RMSE	r^2	%P±0.7	%P±1.0
1.1671	0.7777	50.13	65.41

Table 4.12: CV results (PCR)

RMSE	r^2	%P±0.7	%P±1.0
1.2836	0.7178	50.0	58.88

Table 4.13: Test-set results (PCR)

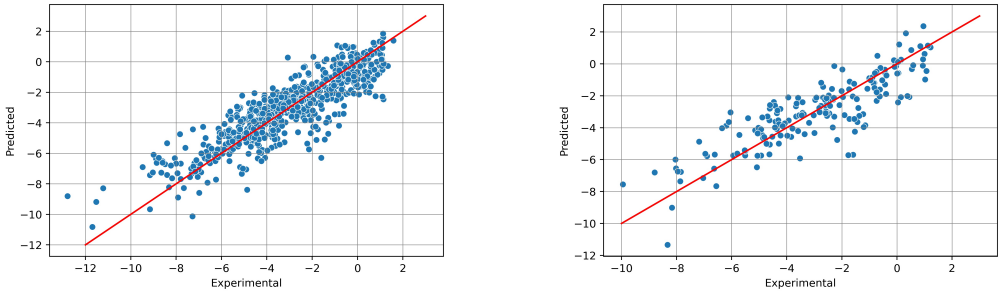
Partial Least Squares (PLS)

To perform Partial Least Squares regression, we prepare many models with increasing components and evaluate them using cross-validation to get the final best number of components.

Based on the Table 4.14, we can select for our final model **the first 7 components**. We get the predictions on CV and Test-set (Figure 4.13) and their evaluation results (Tables 4.15 & 4.16). We can see that the performance is not very different, and the metric falls around the same range as the previous models. An interesting thing to note is that PLS performed better than PCR on the CV, but on Test-set, the performance dropped even below the PCR performance. But since the overall difference between the two is minimal, concluding that one of them performs better might not be correct. And the performance is very similar to that of Ordinary linear regression models. Until now, this suggests that using the Linear model is limiting our predictions from enhancement, as previously discussed.

n	RMSE	Pearson-rsquare	%P±0.7	%P±1.0
1	1.4543	0.653267	40.97	58.47
2	1.3620	0.6956	44.58	58.88
3	1.2627	0.7401	45.55	60.27
4	1.2251	0.7556	47.36	63.61
5	1.2022	0.7641	50.27	64.58
6	1.1775	0.7737	49.44	65.13
7	1.1678	0.7772	51.11	64.86
8	1.1691	0.7766	50.41	65.13
9	1.1696	0.7766	50.00	65.13
10	1.1699	0.7764	49.86	65.13
11	1.1699	0.7764	49.86	65.0

Table 4.14: Model's CV results with increasing Number of Components' n' (PCR)



(a) Cross-validation predictions

(b) Test-set predictions

Figure 4.13: Predictions versus Experimental plots (PLS)

RMSE	r^2	%P±0.7	%P±1.0
1.1678	0.7772	51.11	64.86

Table 4.15: CV results (PLS)

RMSE	r^2	%P±0.7	%P±1.0
1.2714	0.7220	48.88	59.44

Table 4.16: Test-set results (PLS)

4.2.3 Ensemble Learning Models

Bagging Models

Bagging of Decision-trees (BAG)

To perform the Bagging of decision trees, the number of decision trees was taken as a hyperparameter (n_estimators). The hyperparameter was tuned, and the value that gave the best results was found to be **'n_estimators=500'**. We trained the model, and feature importance can be seen in Figure 4.14 below:

From Figure 4.14, We can see that 'SASA' seems to have the highest importance. It also proves that it would have been a considerable loss to remove it from the dataset, as discussed in

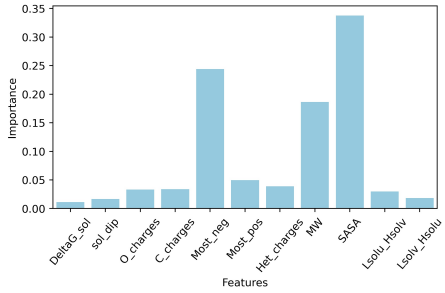
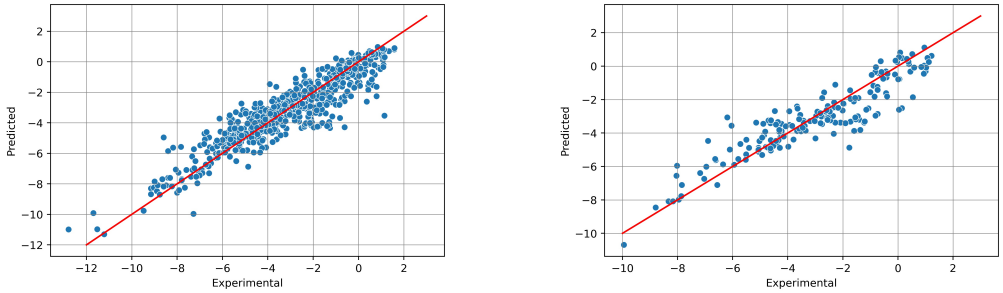


Figure 4.14: Features Importance (BAG)

Section 4.2.1 under OLS. And the plot gives a good idea about other important variables.



(a) Cross-validation predictions

(b) Test-set predictions

Figure 4.15: Predictions versus Experimental plots (BAG)

RMSE	r^2	%P±0.7	%P±1.0
0.9452	0.8539	61.94	77.50

Table 4.17: CV results (BAG)

RMSE	r^2	%P±0.7	%P±1.0
0.9975	0.8324	60.55	73.33

Table 4.18: Test-set results (BAG)

From Figure 4.15, it is clear that the predictions are less scattered than they were when we considered linear models. As can be seen from Tables 4.17 & 4.18, The model performed quite well and gave 10% increase on the %P±0.7 value as compared to linear model. This shows that the data contained much non-linear information, accessed by our decision-tree Bagging model. Bagging many decision-tree showed huge improvement. The predictions on Test-set are slightly worse than CV's, but the model seems stable and gave similar results.

Random Forest (RF)

For this model, we tuned three hyperparameters. We tuned n_estimators (number of decision trees), max_features (number of the independent variable in each tree), and max_depth (depth of each tree). Tuned values were found to be **'n_estimators=100'**, **'max_features=7'** and **'max_depth=50'**. After training, the feature's importance in the model was found, see Figure 4.16:

From Figure 4.16, we can see that 'SASA' again seems to have the highest importance. We

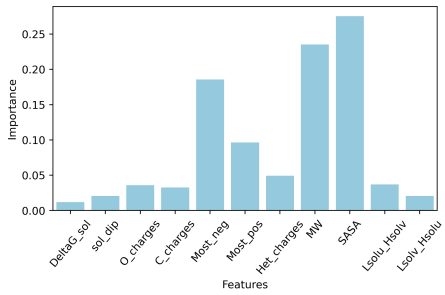
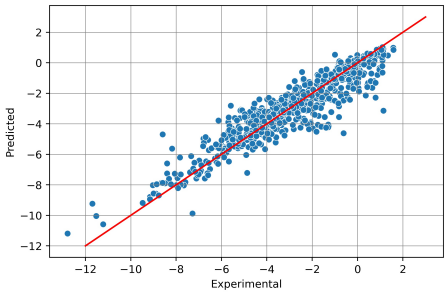
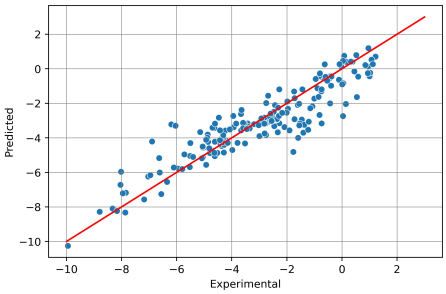


Figure 4.16: Features Importance (RF)

can see that 'MW' follows 'SASA' as the second most important feature, unlike in the BAG algorithm. This could be because of the considerable difference between both algorithms (Section 3.6.3).



(a) Cross-validation predictions



(b) Test-set predictions

Figure 4.17: Predictions versus Experimental plots (RF)

RMSE	r^2	%P±0.7	%P±1.0
0.9520	0.8529	60.55	76.11

Table 4.19: CV results (RF)

RMSE	r^2	%P±0.7	%P±1.0
1.0014	0.8320	61.66	74.44

Table 4.20: Test-set results (RF)

From Figure 4.17 and Tables 4.19 & 4.20, the predictions look very similar to BAG predictions. But, we can say that RF overall performs slightly better than BAG based on their performance by %P±0.7 on unseen test-data, see Table 4.20 & 4.18. The slightly better performance can be explained by the additional randomness that RF introduces into the model, which reduces variance; for details, see Section 3.6.3.

Extremely Randomised Trees (ET)

For this model, the number of decision trees was considered a hyperparameter. Tuned value for number of decision-trees was found to be **'n_estimators=100'**. We trained the model, and the feature's importance can be seen in Figure 4.18.

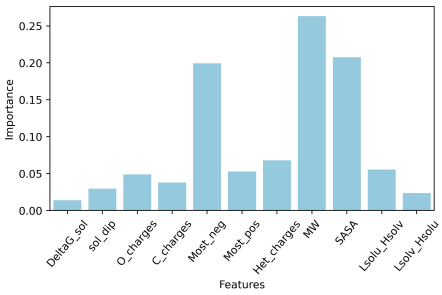
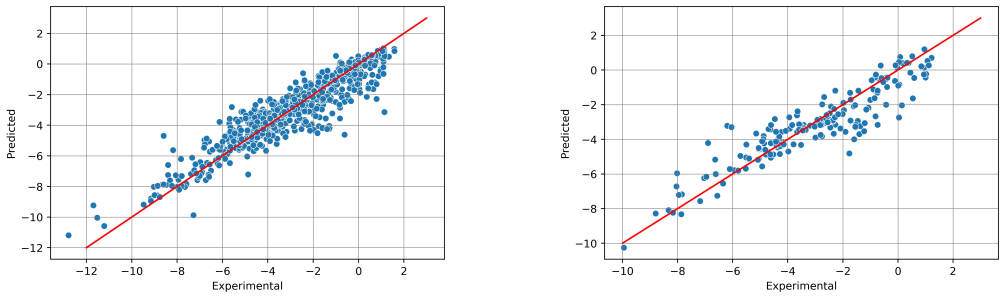


Figure 4.18: Features Importance (ET)

From Figure 4.18, we can see that 'MW' is considered the most important feature, followed by 'SASA'. Overall, we can say that, until now, considering all the bagging models, the top-3 features have been 'SASA', 'MW', and 'Most_neg'.



(a) Cross-validation predictions

(b) Test-set predictions

Figure 4.19: Predictions versus Experimental plots (ET)

RMSE	r^2	%P±0.7	%P±1.0
0.9057	0.8652	64.30	79.02

Table 4.21: CV results (ET)

RMSE	r^2	%P±0.7	%P±1.0
0.9655	0.8429	63.88	75.55

Table 4.22: Test-set results (ET)

From Figure 4.19, the predictions are more concentrated near the line of best fit. As can be seen from Tables 4.21 & 4.22, ET performed the best among all Bagging approaches. It also makes sense because ET allows more randomness into the model, causing a further reduction in variance and making the model perform better.

Boosting Models

Adaptive Boosting (ADAB)

For Adaptive Boosting, the number of decision trees (n_estimators) and the maximum depth of decision trees (max_depth) were taken as hyperparameters. Tuned values for hyperparameters were found to be **'n_estimators=1000'** and **max_depth=5**. We trained the model, and the feature's importance can be seen in Figure 4.20.

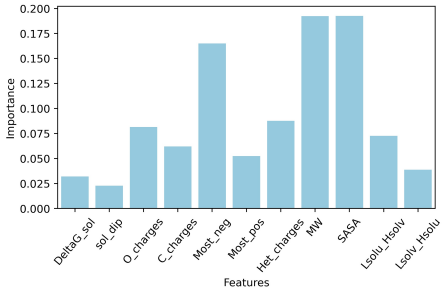
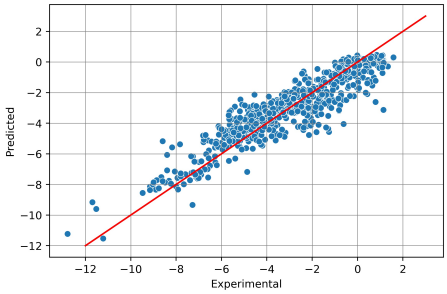
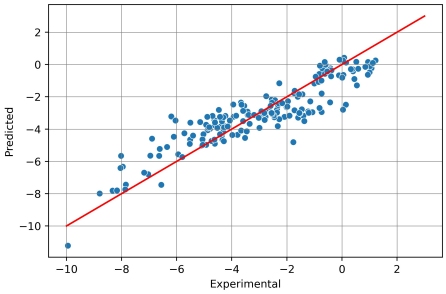


Figure 4.20: Features Importance (ADAB)

From Figure 4.20, It can be seen that just as bagging algorithms' MW', 'SASA' and 'Most_neg' remain the most important features.



(a) Cross-validation predictions



(b) Test-set predictions

Figure 4.21: Predictions versus Experimental plots (ADAB)

RMSE	r^2	%P±0.7	%P±1.0
0.9856	0.8446	57.22	72.91

Table 4.23: CV results (ADAB)

RMSE	r^2	%P±0.7	%P±1.0
1.030	0.8249	53.33	68.33

Table 4.24: Test-set results (ADAB)

From Figure 4.21, the predictions are better than linear models and looks like ADAB is able to capture some relevant information but the performance is not as good as the Bagging models (Tables 4.23 & 4.24). A possible reason could be the difficulty in reducing the loss and learning the samples. Since ADAB tries to calculate loss by taking misclassified samples and updating the weights of the samples based on the loss, the process might not always be very smooth. Thus, the algorithm may allot incorrect sample weights, causing a deficiency in learning the sample.

Gradient Boosting (GRADB)

For Gradient Boosting, the number of decision-trees (n_estimators), learning rate λ (learning_rate) and the maximum depth of decision-trees (max_depth) were taken as hyperparameters. Tuned values for hyperparameters were found to be **'n_estimators=300'**, **max_depth=3** and **learning_rate=0.1**. We trained the model, and the feature's importance can be seen in Figure

4.22.

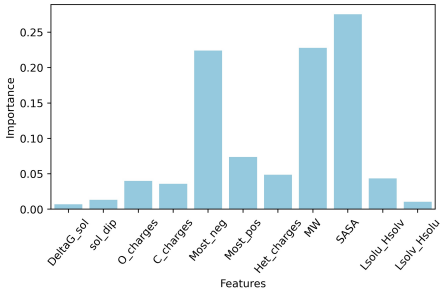
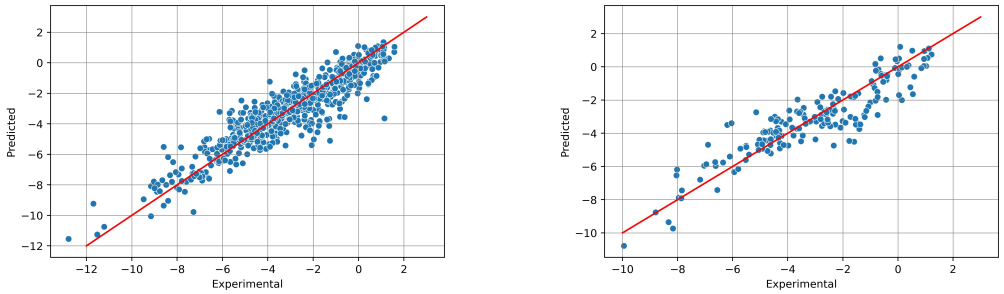


Figure 4.22: Features Importance (GRADB)

From Figure 4.22, It can be seen that 'MW', 'SASA' and 'Most_neg' remain the most important features. The model gave other features less importance as compared to the top three.



(a) Cross-validation predictions

(b) Test-set predictions

Figure 4.23: Predictions versus Experimental plots (GRADB)

RMSE	r^2	%P±0.7	%P±1.0
0.9197	0.8588	62.63	77.50

Table 4.25: CV results (GRADB)

RMSE	r^2	%P±0.7	%P±1.0
0.9971	0.8298	56.11	73.88

Table 4.26: Test-set results (GRADB)

From Figure 4.23, the predictions are better than ADAB and seem to come closer to the line of fit. From Tables 4.23 & 4.24, We can see that GRADB performed better than ADAB. A possible reason could be because it uses Gradient Descent and incorporates learning rate to find a minimum point of the loss function, which makes it smoother and more capable of reducing the loss.

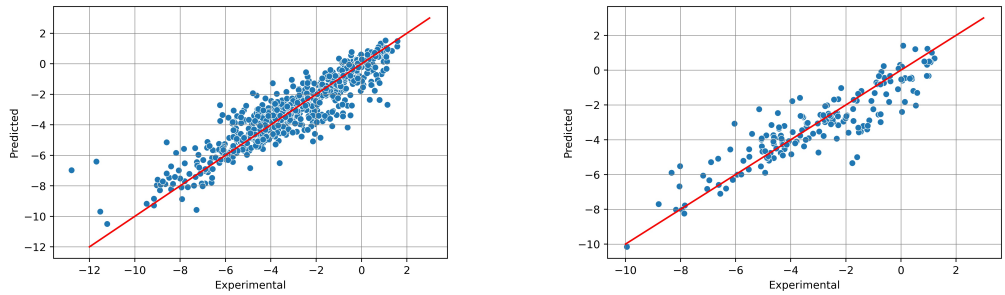
4.2.4 Kernel-based Models

Non-linear Support Vector Regression (KSVR)

For Non-linear Support Vector Regression, thickness of street ϵ (epsilon), regularisation coefficient λ (C) and radial basis function's coefficient (γ) were taken as hyperparameters. Tuned values for hyperparameters were found to be **'epsilon=0.069'**, **C=5** and γ**=0.0909**. We trained

the model, and the performance can be seen below. In KSVR, we transform our features into a high-dimensional space through kernel transformation, so retrieving the feature's importance is not feasible.

The predictions can be seen from Figure 4.24. Indeed, KSVR performed much better than linear Support Vector Regression (SVR). We can say that transforming the input using kernel allowed the model to learn more complex patterns in the data, which may have been non-linear and, therefore, inaccessible to the linear model. From Tables 4.27 & 4.28, we see that the model performed better than most of the models discussed so far. So, allowing a non-linear transformation of the inputs supports our assumptions about linear models being unfit for solubility predictions.



(a) Cross-validation predictions

(b) Test-set predictions

Figure 4.24: Predictions versus Experimental plots (KSVR)

RMSE	r^2	%P±0.7	%P±1.0
0.9825	0.8401	64.16	77.50

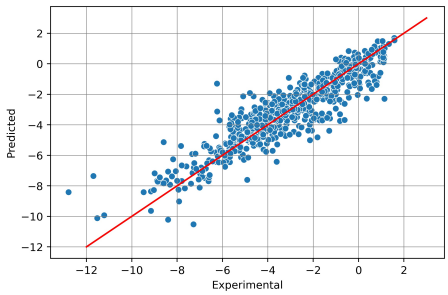
Table 4.27: CV results (KSVR)

RMSE	r^2	%P±0.7	%P±1.0
1.0704	0.8044	61.66	72.22

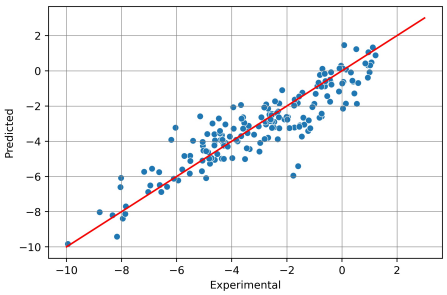
Table 4.28: Test-set results (KSVR)

Gaussian Process Regression (GP)

We performed Gaussian Process Regression, and we optimised values for Radial Basis Function's variance (rbf.variance), lengthscale (rbf.lengthscale) and Gaussian Noise variance (noise_var) through GPy (https://github.com/SheffieldML/GP) module which uses log-likelihood and priors. We used the complete training set to optimise our parameters and found optimised values to be **rbf.variance=15.1466**, **rbf.lengthscale=3.780** and **noise_var=0.6335**. These parameters were further optimised with cross-validation for each training fold to achieve maximum performance. However, the same parameter values were used when performing predictions on the Test-set. Since we transform our features into a high- dimensional space through kernel transformation, so retrieving the feature's importance is not feasible.



(a) Cross-validation predictions



(b) Test-set predictions

Figure 4.25: Predictions versus Experimental plots (GP)

RMSE	r^2	%P±0.7	%P±1.0
0.9721	0.8447	62.08	76.38

Table 4.29: CV results (GP)

RMSE	r^2	%P±0.7	%P±1.0
1.0451	0.8150	57.22	70.55

Table 4.30: Test-set results (GP)

From Figure 4.25 and Tables 4.29 & 4.30, we can see that the model performed quite well. It performed better than all linear models and gave acceptable predictions. A reason why the bayesian approach performed better is that it is known to work well with limited data and makes use of kernels to learn complex patterns. But it may not always perform better than frequentist[7] approaches, and that depends. Suppose the frequentist models can learn all necessary patterns from the data and develop a good understanding of the independent variable. In that case, they may outperform the bayesian approach, as is indicated in our study.

4.2.5 Artificial Neural Networks Model

Multi-layer Perceptron (MLP)

For Multi-layer Perceptron, the hidden layer's size (hidden_layer_sizes) was considered a hyperparameter. After performing Grid-search cross-validation, the tuned value was found to be **hidden_layer_sizes=310**. To minimise the loss function (MSE), we used stochastic gradient descent (SGD) as the optimiser (**solver='sgd'**).

RMSE	r^2	%P±0.7	%P±1.0
0.9210	0.8595	65.27	78.47

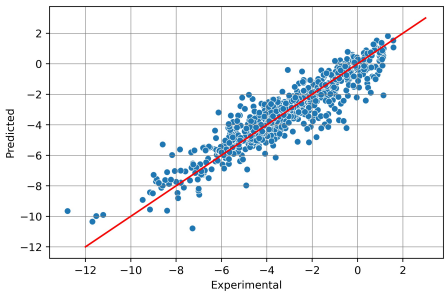
Table 4.31: CV results (MLP)

RMSE	r^2	%P±0.7	%P±1.0
1.0890	0.8027	59.44	72.22

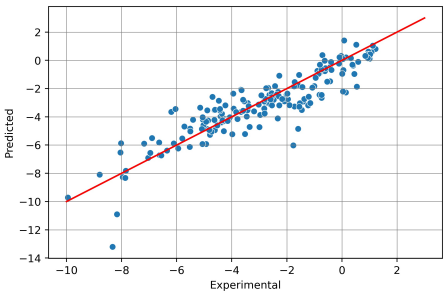
Table 4.32: Test-set results (MLP)

From Figure 4.26, we can see that the predictions are somewhat concentrated near the line of best fit. From Tables 4.31 & 4.32, we can see that MLP performed very good and is among the best performers. Neural Network generally performs well because it focuses on tuning

[7]Frequentist approach means where 'Frequency' is used as a measure of Probability. All the models apart from Gaussian Process Regression discussed in this study follow the frequentist approach and highly depend on the amount of data.



(a) Cross-validation predictions



(b) Test-set predictions

Figure 4.26: Predictions versus Experimental plots (MLP)

the weights by utilising gradients (SGD), making the optimisation process smoother. Building multiple layers can transform the input and extract minute complex patterns from it.

4.3 Consensus Prediction Results

We consider the top-5 best-performing models based on the evaluation metrics. We give the highest priority to %P±0.7 as mentioned in Section 3.4 because this is what we aim to maximise in the final predictions and get as many accurate predictions as possible.

From the results and discussion, we can conclude that Extremely Randomised Trees (ET) is the best performer. We list down the top-5 best-performing models for consensus predictions:

1. Bagging of Decision-trees (BAG)
2. Random Forest (RF)
3. Extremely Randomised Trees (ET)
4. Non-linear Support Vector Regression (KSVR)
5. Multi-layer Perceptron (MLP)

Consensus predictions are taken by aggregating predictions from the listed models using Mean and Median. The consensus performance on the test-set can be seen below, giving us a good understanding of the consensus's performance on the unseen data[8].

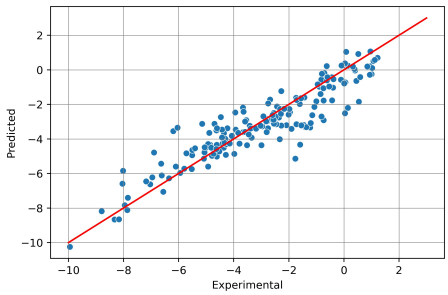
RMSE	r^2	%P±0.7	%P±1.0
0.9668	0.8427	65.0	76.11

Table 4.33: Consensus results (MEAN)

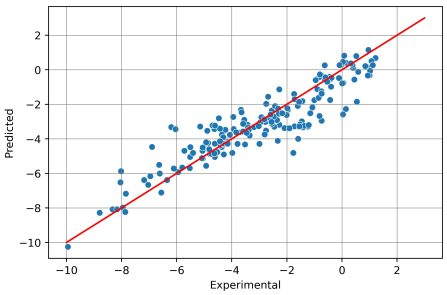
RMSE	r^2	%P±0.7	%P±1.0
0.9637	0.8439	63.33	74.44

Table 4.34: Consensus results (MEDIAN)

[8]If you want to see how consensus performed on CV (train-set), check Appendix



(a) Test-set predictions (MEAN)



(b) Test-set predictions (MEDIAN)

Figure 4.27: Predictions versus Experimental plots (CONSENSUS)

From Figure 4.27 and Tables 4.33 & 4.34, We can see that consensus predictions are by far the best, proving the theory of Collective Intelligence (Wisdom of the Crowd) as discussed earlier.

4.4 Removing and Retaining features enhanced performance

This section will show that removing highly correlated features ('G_sol' & 'volume') enhanced our final output. It will also show that retaining 'SASA', which had a somewhat high correlation but not as high as 'G_sol' or 'volume', actually helped in getting better performance, proving the point made in Section 4.2.1 under OLS about not discarding 'SASA' from the dataset.

RMSE	r^2	%P±0.7	%P±1.0
0.9709	0.8437	61.11	75.0

Table 4.35: Cons. (MEAN), all features

RMSE	r^2	%P±0.7	%P±1.0
0.9738	0.8412	60.55	74.44

Table 4.36: Cons. (MEDIAN), all features

Consensus performance of the top-5 best performing models on the Test-set is shown above (Tables 4.35 & 4.36) for a case where we considered all features during modelling and no correlated features were removed. Prioritising %P±0.7, we can see that the performance is worse than what we got after removing the highly correlated features' G_sol' and 'volume' in our consensus. This proves that removing highly correlated features helped reduce unnecessary complexity from our models and allowed better results.

RMSE	r^2	%P±0.7	%P±1.0
1.008	0.8296	60.55	75.0

Table 4.37: Cons. (MEAN), no 'SASA'

RMSE	r^2	%P±0.7	%P±1.0
1.0180	0.8273	60.55	73.88

Table 4.38: Cons. (MEDIAN), no 'SASA'

Consensus performance of top-5 best performing models on the Test-set is shown above (Tables 4.37 & 4.38) for a case where 'SASA' along with 'G_sol' & 'volume' was removed during modeling. Prioritising %P±0.7, we can see again that the performance is worse than what

we got in our consensus predictions where the dataset retained 'SASA' and had only 'G_sol' & 'volume' removed. This shows us that even if there is slightly higher correlation[9] between two variables, we can still expect to extract some relevant independent information from them. Removing them may cause a loss of information.

These checks are made on the Test-set as it gives us a good understanding of how the models performed on the unseen data. Refer to GitHub repository[10] to check how these results were derived.

4.5 Conclusion

Based on the Results and Discussion so far, we can summarise and conclude our findings with the following points:

- It is essential that we consider correct evaluation metrics which take into account the unavoidable experimental errors, as we did by considering %P$\pm$0.7 & %P$\pm$1.0 for this study. This gives us more confidence in our models and results.
- Linear models performed severely compared to non-linear models for Solubility Prediction due to a lack of linear relationship between dependent and independent variables.
- From among the Ordinary Linear models, the performance was slightly better for the regularised models (RR & LaR) than the non-regularised models. This could be due to a reduction in variance because of regularisation.
- PCR gave slightly better performance than PLS. PCR focuses more on preserving independent variable variance than PLS and does not consider the correlation between dependent and independent variables. So when there are no or few independent variables which show a high correlation with the dependent variable, preserving more independent variable variance may preserve more information, as was our case.
- Bagging approaches proved to be the best for Solubility prediction. Extremely Randomised Trees was the best model among the Ensemble methods and proved to be the best model for Solubility Prediction. This is because it allowed more randomness and reduced variance in the model.
- Boosting algorithms gave good results, with Gradient Boosting outperforming Adaptive Boosting as it uses a differentiable loss function and gradient descent to find the minimum point for the loss function.

[9] Not to be confused with those with very high correlations (>0.9), as was with 'G_sol' and 'volume'
[10] Please ask for repository access by emailing at **mm21am@leeds.ac.uk** OR **adnanmallick29@gmail.com**

- Kernel-based methods showed that non-linear transformation of inputs could allow more information to be extracted from the data and give better results.
- Artificial Neural Networks performed very well. They could find accurate predictions by tuning the model with a complex multi-layer network, which helps learn intricate data patterns.
- Consensus of the top-5 models performed even better than individual performance of models. They showed that consensus predictions might be a good answer for better results.

Linear models do not seem to be a good fit for Solubility Prediction. And we can say that Bagging algorithms, Non-linear Support Vector Regression algorithms and Artificial Neural Networks are other reliable ways to approach Solubility Prediction.

This study was limited in performance due to some lack of data. We can say that with more varied and accurate training data, we can expect even better results.

Bibliography

Belsley, D. A., Kuh, E. and Welsch, R. E. (1980), 'Regression diagnostics: Identifying influential data and sources of collinearity', *Wiley Series in Probability and Mathematical Statistics* .

Bentéjac, C., Csörgő, A. and Martínez-Muñoz, G. (2021), 'A comparative analysis of gradient boosting algorithms', *Artificial Intelligence Review* **54**(3), 1937–1967.

Bergström, C. A., Charman, W. N. and Porter, C. J. (2016), 'Computational prediction of formulation strategies for beyond-rule-of-5 compounds', *Advanced Drug Delivery Reviews* **101**, 6–21. Understanding the challenges of beyond-rule-of-5 compounds.

Bishop, C. M. and Nasrabadi, N. M. (2006), *Pattern recognition and machine learning*, Vol. 4, Springer.

Boland, P. J. (1989), 'Majority systems and the condorcet jury theorem', *Journal of the Royal Statistical Society: Series D (The Statistician)* **38**(3), 181–189.

Boobier, S., Hose, D. R., Blacker, A. J. and Nguyen, B. N. (2020), 'Machine learning with physicochemical relationships: solubility prediction in organic solvents and water', *Nature communications* **11**(1), 1–10.

Boobier, S., Osbourn, A. and Mitchell, J. B. (2017), 'Can human experts predict solubility better than computers?', *Journal of cheminformatics* **9**(1), 1–14.

Butina, D. and Gola, J. M. (2003), 'Modeling aqueous solubility', *Journal of chemical information and computer sciences* **43**(3), 837–841.

Coefficient of Determination (2008), Springer New York, New York, NY, pp. 88–91.

Delaney, J. S. (2004), 'Esol: estimating aqueous solubility directly from molecular structure', *Journal of chemical information and computer sciences* **44**(3), 1000–1005.

Delaney, J. S. (2005), 'Predicting aqueous solubility from structure', *Drug discovery today* **10**(4), 289–295.

Dietterich, T. G. and Kong, E. B. (1995), Machine learning bias, statistical bias, and statistical variance of decision tree algorithms, Technical report, Citeseer.

Drucker, H. (1997), Improving regressors using boosting techniques, *in* 'ICML', Vol. 97, Citeseer, pp. 107–115.

Duvenaud, D. (2013), 'The kernel cookbook'. [Online]. [Accessed 21 August 2022]. Available from: `https://www.cs.toronto.edu/~duvenaud/cookbook/`.

Engkvist, O. and Wrede, P. (2002), 'High-throughput, in silico prediction of aqueous solubility based on one-and two-dimensional descriptors', *Journal of chemical information and computer sciences* **42**(5), 1247–1249.

Fröhlich, H., Wegner, J. K. and Zell, A. (2004), 'Towards optimal descriptor subset selection with support vector machines in classification and regression', *QSAR & Combinatorial Science* **23**(5), 311–318.

Gao, H., Shanmugasundaram, V. and Lee, P. (2002), 'Estimation of aqueous solubility of organic compounds with qspr approach', *Pharmaceutical research* **19**(4), 497–503.

Geurts, P., Ernst, D. and Wehenkel, L. (2006), 'Extremely randomized trees', *Machine learning* **63**(1), 3–42.

Guillory, J. K. (2003), 'Handbook of aqueous solubility data by samuel h. yalkowsky and yan he. crc press, boca raton, fl. 2003. xii+ 1496 pp. 18× 26 cm. isbn 0-89493-1532-8. 299.95.′.

Hansch, C., Quinlan, J. E. and Lawrence, G. L. (1968), 'Linear free-energy relationship between partition coefficients and the aqueous solubility of organic liquids', *The Journal of Organic Chemistry* **33**(1), 347–350.

Hastie, T., Tibshirani, R., Friedman, J. H. and Friedman, J. H. (2009), *The elements of statistical learning: data mining, inference, and prediction*, Vol. 2, Springer.

Hewitt, M., Cronin, M. T., Enoch, S. J., Madden, J. C., Roberts, D. W. and Dearden, J. C. (2009), 'In silico prediction of aqueous solubility: the solubility challenge', *Journal of chemical information and modeling* **49**(11), 2572–2587.

Howard, P. and Meylan, W. (1999), 'Physical/chemical property database (physprop)'.

Huuskonen, J. (2000), 'Estimation of aqueous solubility for a diverse set of organic compounds based on molecular topology', *Journal of Chemical Information and Computer Sciences* **40**(3), 773–777.

Huuskonen, J., Salo, M. and Taskinen, J. (1998), 'Aqueous solubility prediction of drugs based on molecular topology and neural network modeling', *Journal of chemical information and computer sciences* **38**(3), 450–456.

Jain, N. and Yalkowsky, S. H. (2001), 'Estimation of the aqueous solubility i: Application to organic nonelectrolytes', *Journal of Pharmaceutical Sciences* **90**(2), 234–252.

Jain, P. and Yalkowsky, S. H. (2010), 'Prediction of aqueous solubility from scratch', *International Journal of Pharmaceutics* **385**(1), 1–5.

Jorgensen, W. L. and Duffy, E. M. (2002), 'Prediction of drug solubility from structure', *Advanced drug delivery reviews* **54**(3), 355–366.

Karunanithi, A. T. and Achenie, L. E. (2007), Chapter 4 - solvent design for crystallization of pharmaceutical products, *in* K. M. Ng, R. Gani and K. Dam-Johansen, eds, 'Chemical Product Design: Toward a Perspective Through Case Studies', Vol. 23 of *Computer Aided Chemical Engineering*, Elsevier, pp. 115–147.

Khurana, S., Rawi, R., Kunji, K., Chuang, G.-Y., Bensmail, H. and Mall, R. (2018), 'DeepSol: a deep learning framework for sequence-based protein solubility prediction', *Bioinformatics* **34**(15), 2605–2613.

Kovdienko, N. A., Polishchuk, P. G., Muratov, E. N., Artemenko, A. G., Kuz'min, V. E., Gorb, L., Hill, F. and Leszczynski, J. (2010), 'Application of random forest and multiple linear regression techniques to qspr prediction of an aqueous solubility for military compounds', *Molecular informatics* **29**(5), 394–406.

Lawson, A. J., Swienty-Busch, J., Géoui, T. and Evans, D. (2014), The making of reaxys—towards unobstructed access to relevant chemistry information, *in* 'The Future of the History of Chemical Information', ACS Publications, pp. 127–148.

Lee, C. F. and Lee, J. C. (2020), *Handbook Of Financial Econometrics, Mathematics, Statistics, And Machine Learning (In 4 Volumes)*, World Scientific.

Lind, P. and Maltseva, T. (2003), 'Support vector machines for the estimation of aqueous solubility', *Journal of chemical information and computer sciences* **43**(6), 1855–1859.

Lipinski, C. A., Lombardo, F., Dominy, B. W. and Feeney, P. J. (1997), 'Experimental and computational approaches to estimate solubility and permeability in drug discovery and development settings', *Advanced Drug Delivery Reviews* **23**(1), 3–25. In Vitro Models for Selection of Development Candidates.

Llinas, A. and Avdeef, A. (2019), 'Solubility challenge revisited after ten years, with multilab shake-flask data, using tight (sd 0.17 log) and loose (sd 0.62 log) test sets', *Journal of Chemical Information and Modeling* **59**(6), 3036–3040.

Llinàs, A., Glen, R. C. and Goodman, J. M. (2008), 'Solubility challenge: can you predict solubilities of 32 molecules using a database of 100 reliable measurements?', *Journal of chemical information and modeling* **48**(7), 1289–1303.

Melo, J. (2012), 'Gaussian processes for regression: a tutorial', *Technical Report* .

Mitchell, B. E. and Jurs, P. C. (1998), 'Prediction of aqueous solubility of organic compounds from molecular structure', *Journal of chemical information and computer sciences* **38**(3), 489–496.

Mittal, B. (2017), Chapter 2 - pharmacokinetics and preformulation, *in* B. Mittal, ed., 'How to Develop Robust Solid Oral Dosage Forms from Conception to Post-Approval', Academic Press, pp. 17–37.

Osborne, J. W. and Waters, E. (2002), 'Four assumptions of multiple regression that researchers should always test', *Practical assessment, research, and evaluation* **8**(1), 2.

Ozer, D. J. (1985), 'Correlation and the coefficient of determination.', *Psychological bulletin* **97**(2), 307.

Palmer, D. S. and Mitchell, J. B. (2014), 'Is experimental data quality the limiting factor in predicting the aqueous solubility of druglike molecules?', *Molecular Pharmaceutics* **11**(8), 2962–2972.

Palmer, D. S., O'Boyle, N. M., Glen, R. C. and Mitchell, J. B. (2007), 'Random forest models to predict aqueous solubility', *Journal of chemical information and modeling* **47**(1), 150–158.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M. and Duchesnay, E. (2011), 'Scikit-learn: Machine learning in Python', *Journal of Machine Learning Research* **12**, 2825–2830.

Raevsky, O. A., Raevskaja, O. E. and Schaper, K.-J. (2004), 'Analysis of water solubility data on the basis of hybot descriptors. part 3. solubility of solid neutral chemicals and drugs', *QSAR & Combinatorial Science* **23**(5), 327–343.

Ran, Y. and Yalkowsky, S. H. (2001), 'Prediction of drug solubility by the general solubility equation (gse)', *Journal of chemical information and computer sciences* **41**(2), 354–357.

Schroeter, T. S., Schwaighofer, A., Mika, S., Ter Laak, A., Suelzle, D., Ganzer, U., Heinrich, N. and Müller, K.-R. (2007), 'Estimating the domain of applicability for machine learning qsar models: a study on aqueous solubility of drug discovery molecules', *Journal of Computer-aided molecular design* **21**(9), 485–498.

Schwaighofer, A., Schroeter, T., Mika, S., Laub, J., Ter Laak, A., Sülzle, D., Ganzer, U., Heinrich, N. and Müller, K.-R. (2007), 'Accurate solubility prediction with error bars for electrolytes: A machine learning approach', *Journal of chemical information and modeling* **47**(2), 407–424.

Silla, E., Tuñón, I., Villar, F. and Pascual-Ahuir, J. (1992), 'Molecular surface calculations on organic compounds: Molecular area-aqueous solubility relationships', *Journal of Molecular Structure: THEOCHEM* **254**, 369–377.

Sorkun, M. C., Khetan, A. and Er, S. (2019), 'Aqsoldb, a curated reference set of aqueous solubility and 2d descriptors for a diverse set of compounds', *Scientific data* **6**(1), 1–8.

Su, Q., Nagy, Z. K. and Rielly, C. D. (2015), 'Pharmaceutical crystallisation processes from batch to continuous operation using msmpr stages: Modelling, design, and control', *Chemical Engineering and Processing: Process Intensification* **89**, 41–53.

Sushko, I., Novotarskyi, S., Körner, R., Pandey, A. K., Rupp, M., Teetz, W., Brandmaier, S., Abdelaziz, A., Prokopenko, V. V., Tanchuk, V. Y. et al. (2011), 'Online chemical modeling environment (ochem): web platform for data storage, model development and publishing of chemical information', *Journal of computer-aided molecular design* **25**(6), 533–554.

Tetko, I. V., Tanchuk, V. Y., Kasheva, T. N. and Villa, A. E. (2001), 'Estimation of aqueous solubility of chemical compounds using e-state indices', *Journal of chemical information and computer sciences* **41**(6), 1488–1493.

Wang, J. and Hou, T. (2011), 'Recent advances on aqueous solubility prediction', *Combinatorial chemistry & high throughput screening* **14**(5), 328–338.

Yalkowsky, S. and Dannenfelser, R. (1992), 'Aquasol database of aqueous solubility, college of pharmacy, university of arizona', *Tucson, AZ* **189**.

Yalkowsky, S. H. and Banerjee, S. (1992), *Aqueous solubility: Methods of estimation for organic compounds*, Marcel Dekker.

Yalkowsky, S. H. and Valvani, S. C. (1980), 'Solubility and partitioning i: Solubility of nonelectrolytes in water', *Journal of Pharmaceutical Sciences* **69**(8), 912–922.

Yan, A. and Gasteiger, J. (2003), 'Prediction of aqueous solubility of organic compounds based on a 3d structure representation', *Journal of chemical information and computer sciences* **43**(2), 429–434.

Yan, A., Gasteiger, J., Krug, M. and Anzali, S. (2004), 'Linear and nonlinear functions on modeling of aqueous solubility of organic compounds by two structure representation methods', *Journal of computer-aided molecular design* **18**(2), 75–87.

Appendix A

Additional Information

Some additional information that may interest the reader. Please note that the complete analysis was done in Python 3, and notebooks with brief explanations of each step are available at my GitHub repository.

For accessing the full code notebook, Please check `https://github.com/AdnanMalik0` and request repository access by emailing at **mm21am@leeds.ac.uk** OR **adnanmallick29@gmail.com**.

A.1 %P±0.7 & %P±1.0 definition in Python 3

Code below shows the function definition to get %P±0.7 & %P±1.0; the concept was first introduced by Boobier et al. (2020):

```
def within_range(predictions_list, actual_list, within_value):
    x=0
    for i in range(len(list2)):
        if (predictions_list[i]-within_value)<= actual_list[i] <=
        (predictions_list[i]+within_value):
            x+=1
    return((float(x)/(len(list2)))*100)
```

Code below shows how we can obtain the outputs for %P±0.7 & %P±1.0:

```
%P±0.7 = within_range(predictions_list, actual_list, 0.7)
%P±1.0 = within_range(predictions_list, actual_list, 1.0)
```

A.2 Coefficients of Linear Regressions

Below you will find the regression estimated coefficient values for the Ordinary Least Squares, Ridge Regression and Lasso Regression models.

A.2.1 Ordinary Least Squares Coefficients

Variable	Coefficient value
DeltaG_sol	0.02522276
sol_dip	-0.03951519
O_charges	-0.47277027
C_charges	-0.54272197
Most_neg	-1.58566049
Most_pos	-1.09924604
Het_charges	-1.05522159
MW	-1.08569106
SASA	-0.91508408
Lsolu_Hsolv	0.18565449
Lsolv_Hsolu	0.11188965

Table A.1: OLS variable coefficients

A.2.2 Ridge Regression Coefficients

Variable	Coefficient value
DeltaG_sol	0.02489549
sol_dip	-0.03693133
O_charges	-0.4736126
C_charges	-0.52506368
Most_neg	-1.56676351
Most_pos	-1.07786421
Het_charges	-1.04112563
MW	-1.08585749
SASA	-0.91235613
Lsolu_Hsolv	0.19191034
Lsolv_Hsolu	0.10641225

Table A.2: RR variable coefficients

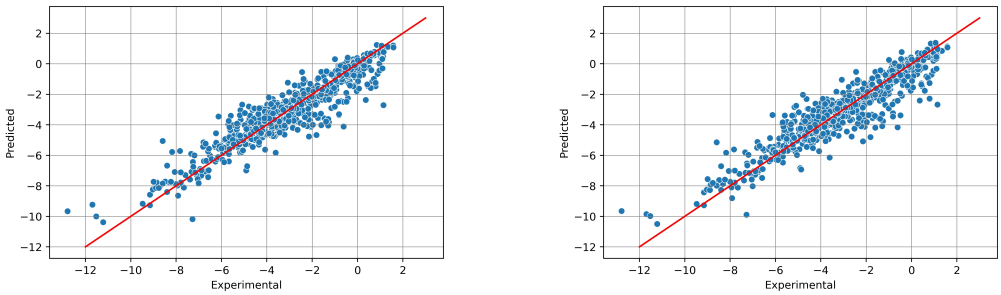
A.2.3 Lasso Regression Coefficients

Variable	Coefficient value
DeltaG_sol	0.01233747
sol_dip	-0.00896499
O_charges	-0.47143076
C_charges	-0.38257663
Most_neg	-1.48062686
Most_pos	-0.97109055
Het_charges	-0.91199435
MW	-1.10494999
SASA	-0.87540955
Lsolu_Hsolv	0.2161295
Lsolv_Hsolu	0.06343979

Table A.3: LaR variable coefficients

A.3 Consensus Performance on Cross-validation

This section can be considered part of Section 4.3 but was omitted previously because the aim there was to compare performance on the unseen dataset (Test-set performance). Consensus performance on Train-set by Cross-validation can be seen below:



(a) CV predictions (MEAN)

(b) CV predictions (MEDIAN)

Figure A.1: Predictions versus Experimental plots (CONSENSUS-CV)

RMSE	r^2	%P±0.7	%P±1.0
0.8754	0.8743	67.63	81.11

Table A.4: Consensus CV results (MEAN)

RMSE	r^2	%P±0.7	%P±1.0
0.9007	0.8660	67.22	80.13

Table A.5: Consensus CV results (MEDIAN)