Physical Sciences Data Infrastructure (PSDI)

Project: Modelling Phase Transitions: Characterising Henry's Law
PSDI Internship Programme 2024 Report
24th June - 30th August 2024

Project Student: Joshua Cheung

Report Date: 30/08/2024

PSDI-Intern-Series-2024:Report_1

Project: Modelling Phase Transitions: Characterising Henry's Law
PSDI-Intern-Series-2024:Report_1
Report Date: 30/08/2024
DOI: DOI HERE
Published by University of Southampton

Physical Sciences Data Infrastructure
PSDI acknowledges the funding support by the EPSRC grants EP/X032701/1, EP/X032663/1 and EP/W032252/1

Title: *Physical Sciences Data Infrastructure Phase 1b*
Principal Investigator: *Professor Simon Coles*
Other Investigator: *Professor Jeremy Frey*
Co-Investigators: *Dr Nicola Knight & Dr Samantha Kanza*

Contents

1 Project Details **1**
- 1.1 Project Student . . . 1
- 1.2 Project Supervisors . . . 1
- 1.3 Project Description . . . 1

2 Lay Summary **2**

3 Introduction **2**

4 Methodology **4**
- 4.1 Obtaining the Data . . . 4
  - 4.1.1 Solubility . . . 4
  - 4.1.2 Henry's law . . . 4
- 4.2 Dataset Building . . . 5
- 4.3 Data Processing . . . 6
  - 4.3.1 Scaling Henry's law constant . . . 6
  - 4.3.2 Feature Engineering . . . 6
  - 4.3.3 Feature Selection and Principal Component Analysis (PCA) . . . 8
- 4.4 Machine Learning . . . 9
  - 4.4.1 Algorithms . . . 9
  - 4.4.2 Model Training . . . 11
  - 4.4.3 Model Evaluation . . . 12

5 Results and Analysis **13**
- 5.1 Final Predictions . . . 13
  - 5.1.1 Solubility . . . 13
  - 5.1.2 Henry's law constant . . . 14
- 5.2 Representation of Functional Groups in the Datasets . . . 16

6 Limitations and Challenges **16**

7 Conclusion **16**

8 Outputs, Data & Software Links **17**

References **17**

9 Appendix **22**
- 9.1 Feature Selection with SelectKBest . . . 22
- 9.2 Principal Component Analysis (PCA) . . . 23
- 9.3 Scaling Results . . . 23
- 9.4 Model Hyper-parameters . . . 24

1 Project Details

Project Name	Modelling Phase Transitions: Characterising Henry's Law
Project Dates	24th June - 30th August 2024
Website	https://github.com/joooshc/PSDI_HLC

1.1 Project Student

Name and Title	Joshua Cheung
University Department Name	School of Chemistry and Chemical Engineering
Work Email	jc10g22@soton.ac.uk

1.2 Project Supervisors

Name and Title	Professor Jeremy Frey
University Department Name	School of Chemistry and Chemical Engineering
Work Email	J.G.Frey@soton.ac.uk

Name and Title	Dr. Joanna Grundy
University Department Name	School of Electronics and Computer Science
Work Email	J.Grundy@soton.ac.uk

1.3 Project Description

Aqueous solubility is a highly important property in a range of scientific areas, ranging from mass production of chemicals in industry, to drug design [1] and flow synthesis [2]. Measured as $\log(\text{mol dm}^{-3})$ and denoted as $\log S$, it describes the amount of a solute that is dissolved in a solvent. Related to this is Henry's law constant (k_H), in units of $\frac{\text{mol}}{\text{atm}}$, which describes the proportion of a gas that is dissolved in a liquid [3].

Despite its importance, existing computational methods for predicting Henry's law constant are largely restricted to semi-empirical methods, and there is a significant lack of experimentation with machine learning. During the past 15 years, interest has started to arise in the use of machine learning to predict solubility [4], and this project hopes to expand this to the prediction of Henry's law constant.

In this project, data was curated from several sources of data to create a master dataset consisting of $\log S$, k_H, and a corresponding temperature, as well as chemical identifiers. A diverse assortment of data preparation methods were assessed, including normalization, feature selection based on variance, and SelectKBest. Various machine learning algorithms, such as LightGBM and K-Nearest Neighbours (KNN), were trialled for predicting $\log S$ and k_H values.

The overall objective of this project was to experiment with machine learning for the prediction of $\log S$ and k_H, and explore the links between them.

2 Lay Summary

The main focus of this project was to use machine learning models to predict values for solubility ($\log S$) and Henry's law constant (k_H), exploring the links between the two properties. These properties are highly important in a variety of fields, such as drug design [1], modelling environmental systems [5], understanding decompression sickness [6], as well as flow chemistry [2].

Data was curated and processed to form a dataset with over 21 000 datapoints, which was used to train and test an assortment of machine learning models. Overall, the most accurate model for both properties was the LightGBM Gradient Boosting Regressor [7], which combines the results from sub-models to learn progressively and improve results.

Many data-related challenges were faced, ranging from data scarcity, to subpar data quality (e.g. missing identifiers, missing temperatures, extreme values). The vast majority of the datapoints were at standard temperature (25° C), and all were assumed to be at standard pressure. It is highly likely that the models will not perform well at other temperatures. Regardless, two machine learning models were successfully used to predict solubility and Henry's law constant for a wide range of compounds in aqueous solution.

3 Introduction

Henry's law, discovered in 1803 by William Henry [8], suggests that the solubility of a gas in a given solvent is 'proportional to its partial pressure in the gas phase' [3]. This statement is valid in the limits of infinite dilution of the solute when the gas is considered ideal [9]. For an ideal-dilute solution (a solution where the solvent follows Raoult's law and the solute follows Henry's law), Henry's constant can be calculated using the following equation, where x_A is the mole fraction of the solute, k_H is Henry's law constant with dimensions of pressure, and p_A is the pressure [10].

$$p_A = x_A k_H \tag{1}$$

The name 'constant', is a misnomer; it is not a true constant, and is dependent on temperature. In truth, Henry's law constant is actually a proportionality constant, as is seen in the format of the equation. As a general rule, it has a somewhat quadratic trend where it increases at low temperatures to a maximum, then decreases at high temperatures [11].

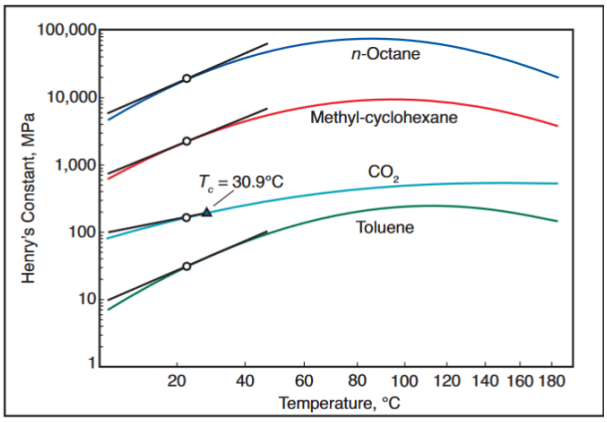

Figure 1: The temperature dependence of some organic solutes in water. The x axis is linear when plotted as 1/T. Figure taken from *Smith et al* [11].

Since Henry's law extrapolates to infinite dilution, it cannot be measured directly, increasing complexity.

Henry's law constant can also be defined in relation to the liquid and gas phase abundances, which is described as Henry solubility [9]. In this work, the terms 'Henry solubility' and 'Henry's law

constant' are used interchangeably. The unit used was $\frac{\text{mol}}{\text{m}^3\ \text{Pa}}$.

$$K_{Hs} = \frac{Q_l}{Q_g} \tag{2}$$

Solubility describes the composition of a saturated solution in terms of the quantity of a defined solute in a solvent [12]. In 1887, François Marie Raoult published an article detailing a series of experiments in which he measured the dependence of the vapour pressure of a solution on the quantity of a given solute in solution [13,14]. Rault's law states that 'the ratio of the partial vapour pressure of each component to its vapour pressure when present as the pure liquid, is approximately equal to the mole fraction of A in the liquid mixture.' [10]

$$p_A = x_A p_A^* \tag{3}$$

Henry's law constant can be approximated for a case where the solute is almost immiscible by combining Raoult's law and Henry's law [11]:

$$H = \frac{(1 - x_w)p_A}{x_A} \tag{4}$$

Here (equation 4), x_w is the mole fraction of water in the organic phase, x_A is the mole fraction of the solute in the aqueous phase, and p_A is the vapour pressure of the solute.

In cases where the solute is significantly more miscible, the derivation originates from equation 1, but requires analysis of the phase equilibrium. This consists of fitting a liquid-activity model to the data and extrapolating to infinite dilution [11].

$$k_H = \gamma_A^\infty p_A \tag{5}$$

More broadly speaking, both Henry's law constant (solubility) and solubility refer to the ability of a solute (gas or liquid) to dissolve in a given solvent.

Solubility, and Henry Solubility, are highly important chemical properties for a wide range of areas. Arguably, the most crucial of these areas is drug solubility. The solubility of a drug must be known to ensure that the correct amount of the drug is circulated throughout the body [1]. As the human body is up to 75% water by body weight [15], it's essential that a drug must be able to dissolve in water to correctly be absorbed in the body to have a therapeutic effect. Furthermore, there are many environmental issues relating to the solubility of pollutants, such as water soluble polymers in freshwater [16], and the solvation of excess CO_2 in the oceans [5].

Existing work relating to the prediction of solubility is primarily limited to Quantitative Structure Property Relationship (QSPR) methods, which attempt to find a relationship between properties and solubility with an equation of the form $f(x) =$. Within the last 15 years, several studies that use machine learning to predict solubility have been published, using a range of different techniques and features. Generally, the most common descriptors used appear to be molecular fingerprints and 2D descriptors. These studies tend to show promising results when working with an internal dataset, but when trialled against an external dataset, their predictions reduce in accuracy and precision [4][1].

Whilst general solubility appears to be moderately well explored, there is a scarcity of work relating to Henry Solubility. There are fewer datasets available, with only *Sander et al* [3] seeming to be readily available. This is in comparison to the numerous solubility datasets available, such as AqSolDB [17], *Lowe et al* [18], and OChem [19]. To my knowledge, the only study completed using machine learning to predict aqueous Henry's law constants is *Machine Learning Approach for the Estimation of Henry's law constant Based on Molecular Descriptors* by *Ullah et al* [20]. This study also used the Sander dataset, and similar methods to what has been completed in this project. Other methods to predict Henry's law constant include molecular dynamics simulations,

[1]This study provides a much more comprehensive review of work that has previously been completed in this area.

QSPR studies, and semi-empirical methods, the latter of which are less accurate due to their nature. Whilst molecular dynamics simulations are highly accurate, they are not a viable method for mass prediction of constants due to the time and computing complexity [21].

There is seemingly a lack of work that directly links and predicts both Henry solubility and general solubility. This project aims to explore the link between the two properties via development of machine learning models to predict them.

Within this project, data was collected and curated from a range of sources to create a dataset usable for machine learning. Chemical descriptors were obtained using RDKit [22], and processed using various sklearn [23] functions, such as MinMaxScaler and RobustScaler. Seven different machine learning models were trialled: LightGBM regressor, Kernel Ridge Regression, Random Forest, AdaBoost, Support Vector Regression, Linear Regression, and a Neural Network. The best model was selected and used to generate final predictions, which were analysed against a test set.

This report details the methodology used, the results obtained and the accompanying analysis, limitations and challenges faced during the course of the project, and the conclusion. Further results obtained in the project are available in the appendix.

4 Methodology

Broadly, there were 3 primary steps involved in the methodology of this project:

1. Data Curation
2. Data Processing for Machine Learning
3. Machine Learning (testing and training models)

In step 1, data was obtained from a multiple sources and combined into one main dataset, using InChI as identifiers to link the compound data. 2D molecular descriptors were added to the dataset using RDKit [22].Following this, in step 2, the molecular descriptors were processed using various scaling and filtration methods to reduce the dimensionality of the data and alter the distribution of the features. Finally, 7 different machine learning models were trained and tested on the data.

4.1 Obtaining the Data

4.1.1 Solubility

Data was gathered from the IUPAC Solubility Data Series (SDS) [24], AqSolDB [17], the Dortmund Data Bank (DDB) [25], and *Transparency in Modeling through Careful Application of OECD's QSAR/QSPR Principles via a Curated Water Solubility Data Set* [18][2].

Where necessary (IUPAC SDS and DDB), data was converted from mole fractions to $\log S$ The equation used is as follows, where x_A is the mole fraction of the solute:

$$\log S = \log\left(\frac{x_A}{0.018 \times (1 - x_A)}\right) \tag{6}$$

A total of 11 703 datapoints were obtained, with 10 965 unique compounds.

4.1.2 Henry's law

Henry's law constants were obtained from the *Compilation of Henry's law constants* by *R. Sander* [3]. The SI unit Fortran90 file was processed in Python 3.12 and converted into a comma separated variable (csv) file for integration into the main dataset.

[2]This section is brief, as the miscibility dataset from 2023 [26] was used as a base, and the only new data is from Lowe et al [18]

The .f90 file was treated as a plain text file and parsed to find the relevant data. In the downloaded dataset, CAS numbers, InChI Keys, and names were used as identifiers, with the program attempting to resolve first via InChI Key, then CAS, then name.

InChI keys were converted to InChI using CIRpy (v1.02) [27], a python interface for the Chemical Identifier Resolver [28]. However, this failed to resolve approximately a quarter of datapoints (3 124). For an unknown reason, compounds labeled with 'MCM:', e.g. 'MCM:C8BCOH' could not be resolved using the InChI key. Additionally, no CAS numbers were given for these compounds. It was discovered that they are from the *Master Chemical Mechanism v3.3* site [29]. This site includes InChI and SMILES for each data entry, and was scraped to obtain corresponding InChI for each constant. This resolved an additional 2 534 compounds. The remaining compounds were fetched by trying InChI key, CAS, or name with PubChemPy (v1.0.4), a python wrapper for the PubChem PUG REST API. [30] Overall, only 73 compounds remained unresolved.

A total of 45 221 datapoints were collected (excluding those labelled as 'C', 'X', '?', or 'E'[3]), with 10 173 unique compounds. Out of these, 40 862 of the corresponding temperatures were missing. These were assumed to be standard temperature, 25°C. For compounds where there were multiple values for Henry's constant at the same temperature, the mean was taken. Overall, 12 225 datapoints remained in the processed dataset.

4.2 Dataset Building

InChI were added as identifiers for all the compounds, and used to join all the sub-datasets into one larger dataset, with 18 006 unique compounds, and 47 619 datapoints overall. However, where a compound appeared in multiple datasets, there was one entry per dataset per compound, leading to duplicated datapoints. Duplicated datapoints were combined and values were averaged, e.g. when there were multiple values for solubility at the same temperature for a given compound, leading to an overall dataset size of 21 291 $\times$ 9. Two data source columns were added: one for solubility and one for Henry's law constant. These contained a name or DOI for the data source.

Next, RDKit descriptors [22] were added, further increasing the dataset size to 21 291 $\times$ 219. The following sub-datasets were created:

Dataset	Rows	Cols	Description
Master	47 619	9	Master dataset containing all data. Some datapoints may be duplicated due to different data sources.
Cleaned Master	21 291	9	Master dataset with duplicated datapoints processed
Cleaned Master + RDKit	21 291	219	RDKit descriptors added as features for Machine Learning

Table 1: The unscaled master datasets created in this project

[3]Further information about what these codes mean can be found in the source paper [3]

Dataset	Rows	Cols	Comps	Description
Henry's constant	9 506	219	7 624	Dataset with RDKit descriptors filtered to compounds that only have a k_H entry and no $\log S$ data
$\log S$	9 140	219	8 594	Dataset with RDKit descriptors filtered to compounds that only have a $\log S$ entry and no k_H data
Henry's constant and $\log S$	2563	219	2 563	Dataset with RDKit descriptors filtered to compounds that have data for $\log S$ and k_H

Table 2: The unscaled sub-datasets created from the master dataset.

4.3 Data Processing

4.3.1 Scaling Henry's law constant

Initially, algorithms were tested with unaltered target data. However, the distribution of values for k_H are highly skewed towards 0, with approximately 250 values greater than 10^{15}. With such a wide range of values, scores were low, and application of different scaling methods on the features did not change the models performance significantly, if at all. Several algorithms, such as a linear regression model, assume that the prediction errors are normally distributed, and therefore have a skew of 0 [31, 32]. These algorithms perform significantly worse when the data has a high amount of skew, and/or deviates from a Gaussian distribution. The unscaled target data had a skew of 110.

A natural log transform was tested on k_H, decreasing the skew from 110 to 0.95. This can be seen in 2. Ultimately, values greater than 10^{15} were removed before applying the log transform, and the scaled data had a skew of 0.51, and a range of 64.6.

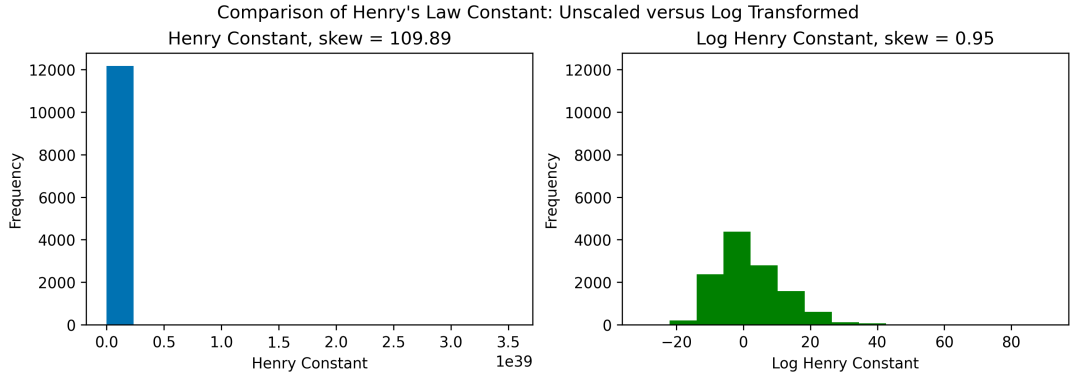

Figure 2: A comparison of distribution of values of Henry's law constant unaltered, versus log transformed.

4.3.2 Feature Engineering

First, all missing values must be filtered out from the dataset, as the majority of machine learning algorithms cannot operate on data that contains NaNs. Next, features were scaled.

Whilst tree-based models such as Random Forest are generally robust to lack of scaling, many others such as KNN are highly reliant on the data having the same scale [33]. Decreasing skew, ensuring the data lies within the same scale and removing extreme values can all help to improve a model's performance.

Each dataset was processed with functions from sklearn.preprocessing [34] as follows:

1. If Henry's constant is the target, log transform Henry's constant and replace the raw values[4]
2. Replace ∞ and $-\infty$ values with NaNs
3. Remove columns where more than 1% of values are NaNs
4. Remove columns where less than 1% of values are unique
5. Normalize features (MinMax or Normalize functions)
6. Scale data (log1p, MaxAbsScaler, PowerTransformer, RobustScaler or StandardScaler)
7. Outliers removed from target column ($\log S$ or k_H)

The Z score was used as a metric to remove outliers from the target feature (e.g. $\log S$), removing anything with a score greater than 3.

$$Z = \frac{x - \bar{x}}{\sigma} \tag{7}$$

Scaling Method	**Transformation**
MinMax	$x_{\text{std}} = \frac{x - x_{\text{min}}}{x_{\text{max}} - x_{\text{min}}}$, $y = x_{\text{std}}(x_{\text{min}} - x_{\text{max}}) + x_{\text{min}}$ [35]
Normalize	$y = \frac{x}{\|\|x\|\|_2}$, $\|\|x\|\|_2 = \sqrt{x_1^2 + x_2^2 + ... + x_n^2}$ [32]
log1p	$y = \ln(1 + x)$ [36]
MaxAbsScaler	$y = \frac{X}{\|x_{\text{max}}\|}$ [37]
PowerTransformer	$y = \begin{cases} \frac{(x+1)^\lambda - 1}{\lambda} & \lambda \neq 0, x \geq 0 \\ \log(x+1) & \lambda = 0, x \geq 0 \\ -\frac{(-x+1)^{2-\lambda} - 1}{2-\lambda} & \lambda \neq 2, x < 0 \\ -\log(-x+1) & \lambda = 2, x < 0 \end{cases}$ [38, 39]
RobustScaler	$y = \frac{x_i - Q_2 x}{Q_3(x) - Q_1 x}$ [40]
StandardScaler	$y = \frac{x - \mu}{\sigma}$ [41]

Table 3: The methods used to normalise and transform the features, where X is the input and y is the scaled data.

Each scaled dataset variation was tested using the default LightGBM regressor [7] and a 5 fold GroupShuffleSplit with defined seed, and scores were compared to find the best scaling method. The scaling method for each dataset was chosen based on the R^2 score (explained later in section 4.4.3).

A total of 2 normalisation techniques and 7 scaling methods were tested on the features, leading to a total of 15 dataset variations (14 scaling combinations and 1 unscaled).

For both datasets, training with completely unscaled and unnormalised data gave the best results (available in section 9). This suggests that there are features which are naturally larger than the others, e.g. molecular weight, which could be contributing more to the model. These larger features are effectively weighted, with a greater importance compared to when normalised. When normalisation is applied, the feature no longer has this artificial weighting, and contributes less to the model, causing a reduction in score.

Due to the importance of scaling, the best scaling method was chosen disregarding the results for the unscaled datasets.

[4]The reasoning for this is explained in 5

4.3.3 Feature Selection and Principal Component Analysis (PCA)

Feature Selection As the data was still highly dimensional, with over 200 different features, feature selection was used to avoid the curse of dimensionality. This is the concept that the number of samples required to predict something with a given level of accuracy increases exponentially with the dimensionality of the function. This can especially become an issue for algorithms like KNN, where the number of possible neighbours increases exponentially with the number of dimensions [42].

With such a large number of features, many of which were sparse, it would be hard for a model to get a good representation of the population of training data without further processing. Such features that only contain 1 distinct value for all compounds will not have any overall contribution to the model [43].

Two different methods for feature selection were trialled: SelectKBest [44] and removing any features that have a percentage of distinct values that is lower than a given threshold. SelectKBest scores all the features using the ANOVA (Analysis of Variation) F-test, and selects the top k features, ranked by score. The F-score compares the variation between the sample means to the variation within the samples.

By default, the scoring function was set to 'f_classif' and this was not changed. This function calculates the F-score. This choice in scoring function was likely a contributing factor to its poor performance (table 9), where using the maximum number of features gave the best results, even when many of those features only contained a singular value. Ideally, this should have been changed to a regression scorer, such as 'f_regression'. The equations [45] for 'f_classif' are given below (sample frequency, followed by F-score):

$$s^2 = \frac{\sum(x - \bar{x})^2}{(n-1)} \tag{8}$$

$$F = \frac{s_a^2}{s_b^2} \tag{9}$$

The final method that was trialled, and later chosen, for feature selection was based off calculating the percentage of values in the column that were unique, and dropping any columns where this percentage was below a given threshold (1%). This proved to be the most effective and straightforward to use, since it was not reliant on a more complex equation or statistical test.

$$\text{Percentage Unique} = \frac{\text{Number of Unique Values}}{\text{Total Number of Values}} \times 100 \tag{10}$$

Figure 3: A heat-map comparison of the $\log k_H$ dataset before and after scaling and feature selection.

Fig. 3 displays the difference in the data distribution before and after scaling. There are no longer missing values in the dataset, and the overall range of values is smaller, yet there is more distinction between them. Additionally, all the features are now on the same scale.

Principal Component Analysis (PCA) The sklearn implementations of Kernel PCA [46] and standard PCA [47] were tested across a range of components from 1 to the total number of features. PCA is an unsupervised method of dimensionality reduction in which features that are linearly dependent, or correlated, are reduced into a lower dimensional linear subspace. It attempts to maintain as much relevant information as possible. However, it assumes that data with low variance is unimportant; in this way, it is not suitable for all datasets and models [32, 48, 49].

Kernel PCA is a variant of PCA that is designed to deal with non-linearity in data [50]. Using kernels can sometimes handle complex data better than only using Euclidean distances [51]. It was trialled after standard PCA was found to have a highly detrimental effect on model performance, reducing the R^2 by 0.5 or more (results available in appendix 9.2).

A variety of different kernels were trialled, using a bisection algorithm to determine the highest R^2 output with a given number of components. However, the best number of components was consistently the total number of features in the dataset, and it still yielded worse scores than using no PCA or kernel PCA whatsoever. For this reason, PCA was ultimately discarded.

4.4 Machine Learning

4.4.1 Algorithms

The No Free Lunch Theorem [52] approach was taken for deciding on which algorithms to use. After determining the best scaling method for the data, 7 different regression algorithms were trialled[5]. Each algorithm and scaling pair was optimized using a parameter grid search, with the sklearn function GridSearchCV [54]. Post-optimization, each model was run again with a 5 fold GridShuffleSearch, and scored with MSE and R^2. The model for each dataset with the highest R^2 and lowest MSE was used to make the final predictions.

[5]A neural network [53] was also tested but removed due to time constraints.

Algorithm	MSE	$\mathbf{R^2}$
LightGBM	**1.01**	**0.80**
Kernel Ridge Regression	1.06	0.79
KNN	1.23	0.76
Random Forest	1.28	0.75
AdaBoost	1.53	0.70
Support Vector Regression	1.65	0.68

Table 4: Algorithm testing on v0.6.0 version of the $\log S$ dataset.

Algorithm	MSE	$\mathbf{R^2}$
LightGBM	**5.04**	**0.89**
Kernel Ridge Regression	5.92	0.87
Random Forest	8.74	0.81
KNN	9.31	0.81
AdaBoost	0.73	0.73
Support Vector Regression	68.92	-0.45

Table 5: Algorithm testing on v0.6.0 version of the $\log k_H$ dataset.

Hyperparameters are available in the appendix, section 9.4.

Ensemble Methods Ensemble methods are algorithms that aggregates a set of models (regressors or classifiers) on a subset of the training data, then train further models using a weighted vote or average of their predictions [55]. This method relies on the idea that a collection of models, or learners, has a greater overall accuracy than a single model. They tend to avoid overfitting, and are not as affected by the curse of dimensionality as other methods [56], like support vector regression. Two types of ensemble learning methods were trialled: Gradient Boosting and Random Forest.

Multiple different variations of gradient boosting exist. Here, the AdaBoost and the LightGBM regressor gradient boosting algorithms were chosen. Gradient boosting is an iterative method in which a sample of data is selected, a set of models are trained on it, and the next iteration of models attempts to learn from its predecessors using the residuals. Models are trained sequentially, sampling with replacement from the training set [57]. Using this method, some models may perform significantly better on some parts of the dataset but worse on others, but when aggregated, give a much better result overall.

Adaboost AdaBoost is a type of gradient boosting regressor that varies from standard boosting methods in that it is adaptive, hence the name. Instead of the new models learning from the previous ones, the worst scoring predictions have their selection probability increased so they are more likely to appear in the next iteration of the training set. In other words, the weight of the worst scoring datapoints is adjusted. This weighting is what separates AdaBoost from standard boosting algorithms [58].

LightGBM Regressor The LightGBM boosting regressor is a variant of the Gradient Boosting Decision Tree algorithm that uses a combination of Gradient-based One-Side Sampling and Exclusive Feature Bundling, which together decrease the training time compared to a traditional gradient boosting algorithm by 'up to over 20 times' [59]. This model was chosen for testing due to its previous performance in models relating to miscibility.

Random Forest Regressor The LightGBM implementation of the Random Forest regressor was used [7]. This algorithm is a type of regressor, containing a collection of tree-structured classifiers $\{h(\mathbf{x}, \Theta_k), k = 1, \dots\}$ that depend on a random vector Θ so that the tree predictor $h(\mathbf{x}, \Theta)$, has numerical values instead of classes.

It is noted that the type of randomness used in this algorithm may be more suitable for classification problems than regression, however the Random Forest algorithm cannot overfit due to the Law of Large Numbers [60].

Single Models

Support Vector Regression (SVR) The sklearn implementation (based on libsvm [61]) of SVR [62] was used in this project. A Support Vector Regression model is a generalisation of a Support Vector Machine, with the introduction of an ϵ tube, where ϵ is the tolerance margin. The support vectors are defined as any points which lie outside of the ϵ tube - the larger ϵ is, the more support vectors there are. SVR is different to other algorithms in that it uses a symmetrical loss function, measuring the absolute loss [63].

Some of the key advantages of this algorithm is that it is said to be good at generalisation, and have high precision/accuracy. Conversely, one of the disadvantages of this algorithm implementation in particular is its high fit time complexity (more than quadratic), which means it took a long time to train each model. It relies on kernel functions, and aims to find a function that fits the target with an error of no greater than the ϵ tube width for each target datapoint. The algorithm iterates until a specified convergence criterion is met [64].

Kernel Ridge Regression Kernel Ridge Regression is a variant of Support Vector Regression that uses a squared error loss instead of ϵ-insensitive loss [65]. It replaces all datapoints with their feature vector (eq. 11), which can cause the number of dimensions to increase infinitely higher than the number of datapoints [66]. The kernel trick is then used to process the data without having to complete a calculation for every feature vector [67]. The kernel trick is the application of a kernel function, e.g. the sigmoid kernel, to data to transform it to a higher dimensional space [68]. This is followed by performing computations on it within that higher dimensional space, yet without having to do an infinite amount of work.

$$\mathbf{x}_i \rightarrow \Phi_i = \Phi(\mathbf{x}_i) \tag{11}$$

K-Nearest Neighbour Regressor (KNN) KNN is an algorithm that is most often used for classification problems. It assumes that like datapoints are close to one another using a given space metric, e.g. Euclidean, or Manhattan. It is a type of 'lazy learning' method, which can be quite memory intensive depending on the size of the dataset used. Despite its simplicity to implement, it is prone to issues such as the curse of dimensionality and overfitting [69].

Linear Regression A linear regression is one of the simplest possible machine learning algorithms. It fits a line of best fit to the targets, and minimises the residual sum of squares between the target value and the predicted value [70]. Linear regression was only used on the first version of the dataset and not continued further due to results.

4.4.2 Model Training

Preliminary Testing The full dataset for each target variable was separated into a training set and a validation set, with a randomly generated 85:15 split by InChI. Models were trained and tested on the training set using a variation of KFold, GroupShuffleSplit [71], with 5 folds. GroupShuffleSplit splits the dataset based on a group, InChI, rather than solely by target variable like standard KFold. Using this method ensured that a compound could not appear in both the training and test set, minimising data leakage. By iterating through the folds, a prediction was made for every entry in the training set. This method was used for determining what algorithm and what scaling method to use.

Obtaining Results The dataset containing both $\log S$ and k_H was used as the test set. The $\log S$ dataset was used as the training set for $\log S$, and used as the prediction set for k_H, and vice versa for the Henry's constant set.

Dataset	Usage
Henry's constant	Training Henry's law model, prediction set for missing $\log S$ values
$\log S$	Training $\log S$ model, prediction set for missing Henry's constant values
Henry's constant and $\log S$	Test set for both models

Table 6: Datasets used to obtain the results and create the final output dataset.

The datasets were scaled according using the best scoring scaling method (MinMax $\rightarrow$ RobustScaler for both), and the best scoring algorithm was trained and used to predict the missing values. The predicted values were added to a copy of the Cleaned Master dataset and the data source columns were updated accordingly. Henry's law constant was also unscaled.

Additionally, predictions were also made for the test set, and metrics were calculated and used to evaluate overall model performance.

4.4.3 Model Evaluation

Models were evaluated based on R^2 and Mean Squared Error (MSE).

Mean Squared Error:

$$\text{MSE} = \frac{\Sigma(y_i - \hat{y}_i)^2}{n} \tag{12}$$

Mean squared error is a metric of the Euclidean distance between the predicted y value and the true y value. The greater the error, the more inaccurate the model is.

R^2 Score (Coefficient of Determination):

$$R^2 = 1 - \frac{\Sigma_i(y_i - \hat{y}_i)^2}{\Sigma_i(y_i - \bar{y})^2} \tag{13}$$

The R^2 score measures how well the predicted y values and the true y values correlate to each other. [72] A perfect correlation would have an R^2 value of 1, no correlation scores 0, and perfect negative correlation would be -1.

One of the key disadvantages of both these metrics is that they are very easily skewed by extreme values. In particular, the presence of a few points can cause a large increase in R^2. Additionally, R^2 values tend to be higher when there is a large amount of variance in the values it is comparing [33].

5 Results and Analysis

5.1 Final Predictions

5.1.1 Solubility

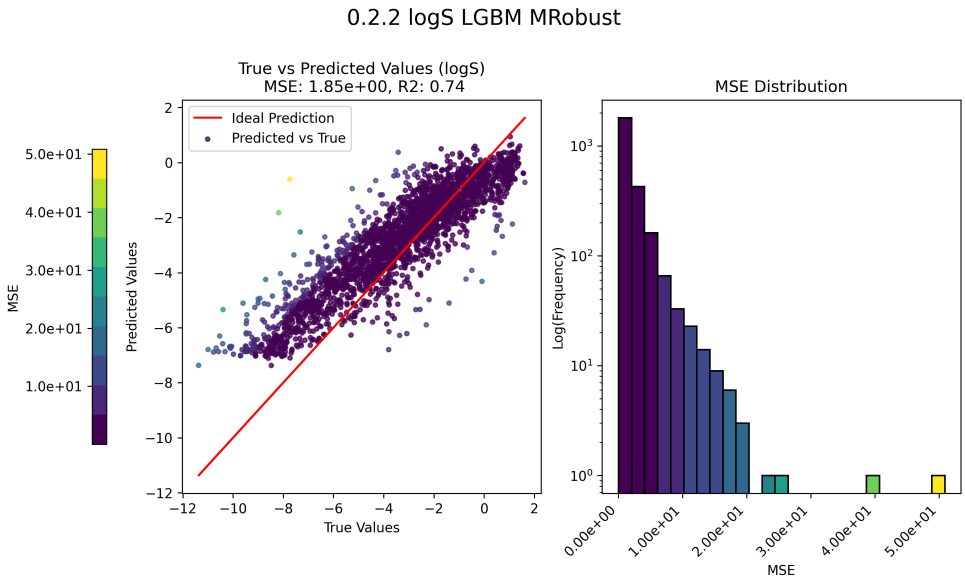

Figure 4: From left to right: the scatter plot of true versus predicted $\log S$ values for the test set (v0.2.2), and the distribution of the associated MSE for each prediction

Metric	$\log S$
$\mathbf{R^2}$	0.73
MSE	1.85
% MSE $>$ **1**	45.52%

Table 7: The final scores for the $\log S$ test set (v0.2.2) predictions

Overall, the model to predict solubility performed well, consistently obtaining a moderately low MSE and a high R^2 score. In the later versions of the dataset, the scores worsen in comparison to earlier iterations, e.g. $R^2 = 0.84$ in table 11 with v0.4.2 of the dataset. This is likely due to changes in the distribution of data in the dataset as more datapoints were added, and data processing methods were adjusted and debugged. It is likely that scores would improve again if hyper-parameter optimization was performed again to reflect the changes in the data.

In terms of error by compound, it appears that the vast majority of compounds were predicted well (note that the y-axis is log scaled in the histogram in Fig. 4), with a few major outliers.

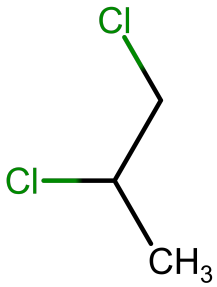

Figure 5: The least accurately predicted compound, 1,2-dichloropropane [73], in the $\log S$ test set (MSE = 50.87)

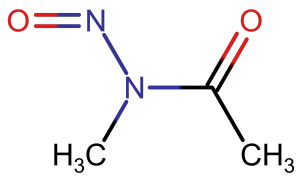

Figure 6: The most accurately predicted compound, N-Methyl-N-nitrosoacetamide [73], in the $\log S$ test set (MSE = 6.01e-07)

Currently, it is unknown why these compounds in particular were handled particularly well or badly by the model. This is something to be explored in the future.

5.1.2 Henry's law constant

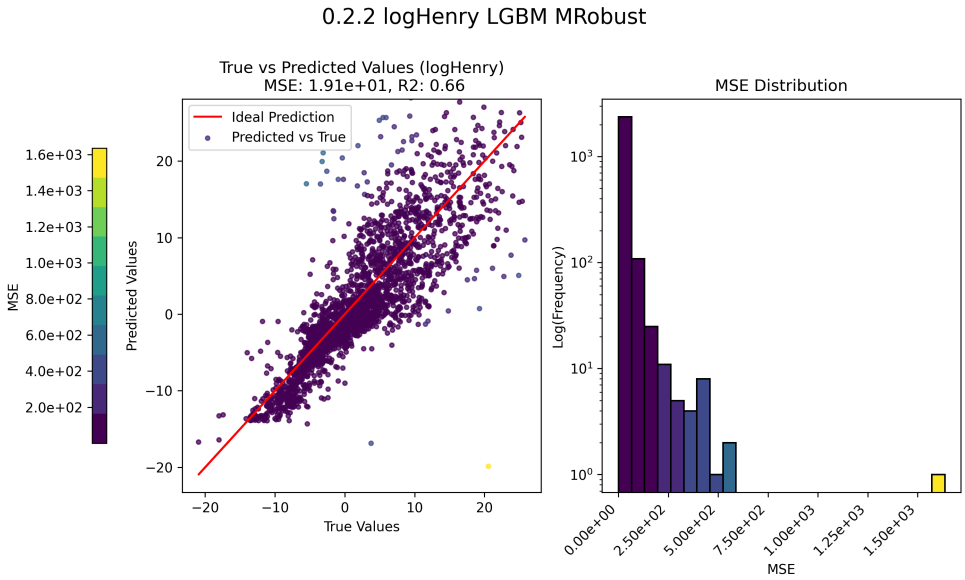

Figure 7: From left to right: the scatter plot of true versus predicted $\log k_H$ values for the test set (pred-set v0.2.2, dataset v0.7.1), and the distribution of the associated MSE for each prediction

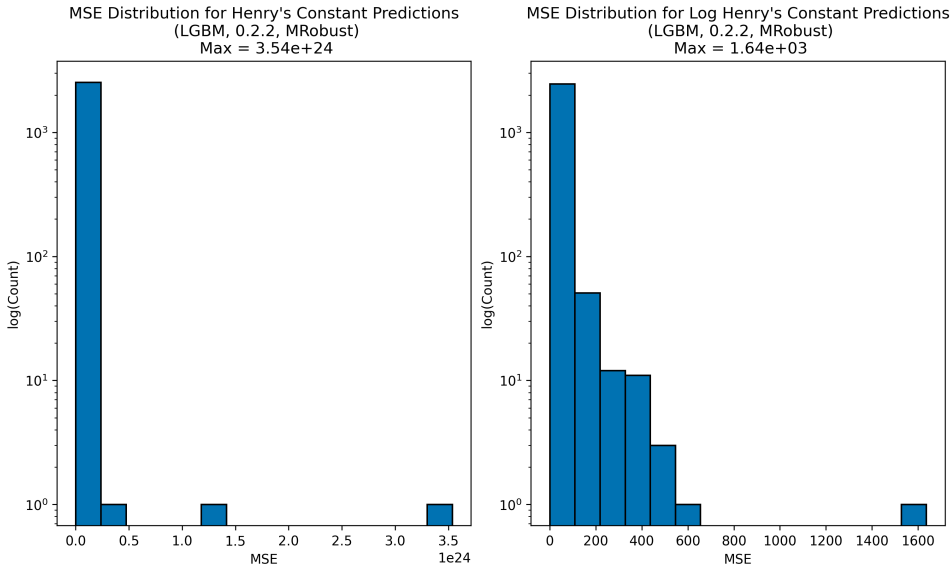

Figure 8: The distribution of MSE values for each compound before and after the reverse log transform was applied to the predictions (pred-set v0.2.2, dataset v0.7.1)

Metric	Log Transformed	Transform Reversed
$\mathbf{R^2}$	0.66	-68.67
MSE	19.11	2.13×10^{21}
% MSE $>$ 1	76.53%	52.93%

Table 8: A comparison of scores before and after reversing the log transform on the target variable (k_H), predset v0.2.2, dataset v0.7.1

The top and worst 10 predictions were compared for the transformed and reverse transformed target variable. There was no overlap between the 10 worst predicted compounds, but an overlap of 3 compounds for the best predicted compounds.

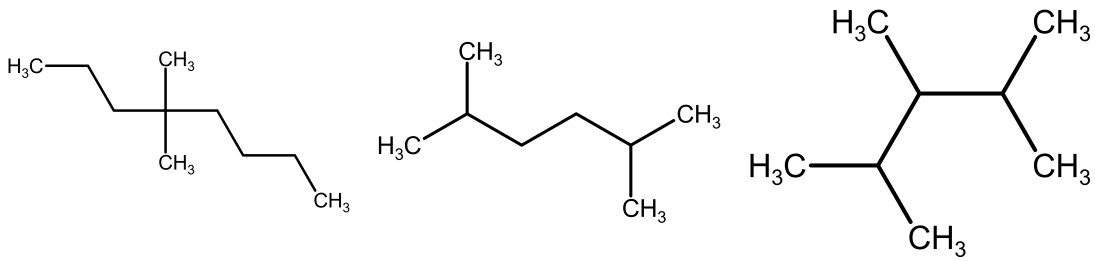

Figure 9: The compounds that are in the top 10 most accurate predictions for log transformed and reverse transformed Henry's law constant

All of the compounds in the overlap are hydrocarbons and alkanes, meaning that they only contain hydrogen and oxygen. The alkane homologous series has a moderately fixed trend and chemical properties, so it is unsurprising that the overlapping compounds are all the same type. One of the reasons for this could be an overabundance of molecules with hydrocarbon fragments in the training set. This is analysed further in the following section (5.2).

5.2 Representation of Functional Groups in the Datasets

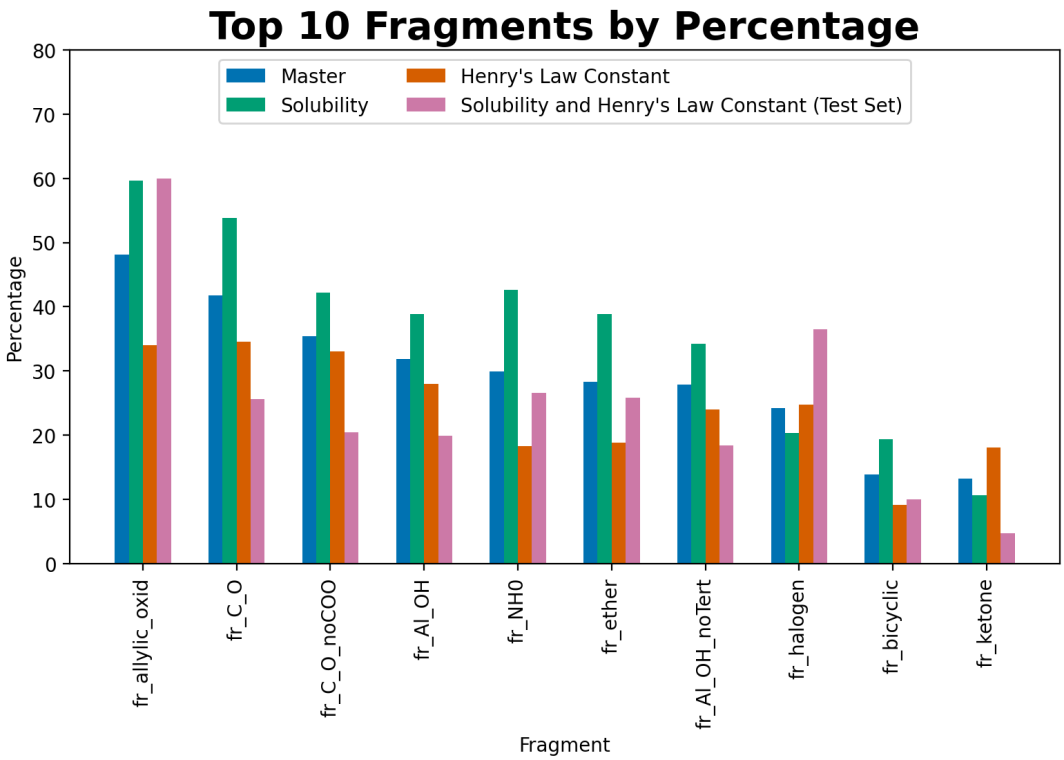

Figure 10: A comparison of the frequency percentage of the top 10 fragments in the master dataset, and the other datasets (v0.7.1)

Ideally, the percentage of compounds in a dataset with a specific fragment, e.g. *fr_C_O*, should be the same as the test set. However, this is often not the case, with some fragments appearing nearly twice as often in the test set as the training set, like *fr_allylic_oxid* in the Henry's law Dataset. This may have contributed to the poor scores when using the test set for certain subsets of compounds, as the test set was not randomly chosen. Further work is required to determine why particular compound and functional groups were predicted well, or poorly, by the models. Due to time constraints, this analysis has not been completed.

6 Limitations and Challenges

There were many challenges throughout the course of the project, which were mostly data related. For Henry's law constant, there was only one dataset widely available. Out of 45 221 datapoints, 90% (40 862) datapoints had no corresponding temperature. Those with no listed corresponding temperature were assumed to be at 25° C. Additionally, many had unknown CAS or InChI keys, which had to be resolved using various different python libraries. One library in particular, CIRpy, had a tendency to replace unresolved identifiers with water. Whilst this was simple to fix, as all datasets should not contain water, it was not identified until dataset version 0.6.0. The original scope of the project was too broad, and lead to time management issues.

7 Conclusion

Two models were developed simultaneously using the same processes to predict solubility and Henry's law constant. A dataset was compiled, containing both target variables, and a test set of the

intersection of the datasets was set aside. The models were trained on the data that only contained the specific target variable (9 140 and 9 506 datapoints respectively), and tested on the shared data (2 563 datapoints). Both models achieved an acceptable level of accuracy and precision, with approximately 50% of predictions in each test set scoring lower than 1 for MSE. For an unknown reason, the model to predict Henry's law constant performs exceptionally well for hydrocarbons, with MSE in the range of $10^{-5} - 10^{-6}$. Due to limited time available, the links between which features contribute best to predictions for each property were not investigated, and this aim was not fulfilled. This work will hopefully be completed in the future.

Flaws were identified in the way that the final models were developed, and there are definite opportunities for further enhancement of their performance. Furthermore, like in many machine learning projects, there were significant challenges with data, both in terms of quantity and quality. Data sanitation could be improved, however there is only so much data that can be gathered on the internet as a whole.

To further investigate what features contribute to the models' prediction ability, recursive feature elimination could be used to rank feature importance. This could be compared to the feature ranking by SelectKBest (with corrected scoring to use a regression based scorer rather than classification). To explore which compounds had the most accurate predictions, further analysis of fragments could be performed, and how it links with other features in the dataset. An uneven distribution of functional groups could be combated by methods such as oversampling, and data augmentation.

Further expansion could be completed by the inclusion of melting and boiling point data, testing how they work as features and targets for $\log S$ and k_H. Additionally, the links between phase diagrams and calculation of k_H could be explored. Another possible route to explore would be to trial different representations of the molecules, rather than solely using the base RDKit 2D descriptors.

8 Outputs, Data & Software Links

All the data, scripts, results, and other outputs are available at https://github.com/joooshc/PSDI_HLC. Anything containing solubility data has been removed due to copyright issues with DDB.

References

[1] Savjani KT, Gajjar AK, Savjani JK. Drug solubility: importance and enhancement techniques. ISRN Pharm. 2012 Jul;2012:195727.

[2] Burange AS, Osman SM, Luque R. Understanding flow chemistry for the production of active pharmaceutical ingredients. iScience. 2022;25(3):103892. Available from: https://www.sciencedirect.com/science/article/pii/S2589004222001626.

[3] Sander R. Compilation of Henry's law constants (version 5.0.0) for water as solvent. Atmospheric Chemistry and Physics. 2023;23(19):10901-2440. Available from: https://acp.copernicus.org/articles/23/10901/2023/.

[4] Llompart P, Minoletti C, Baybekov S, Horvath D, Marcou G, Varnek A. Will we ever be able to accurately predict solubility? Scientific Data. 2024 Mar;11(1):303. Available from: https://doi.org/10.1038/s41597-024-03105-6.

[5] Vallero DA. Chapter 34 - Engineering aspects of climate change. In: Letcher TM, editor. Climate Change (Third Edition). third edition ed. Elsevier; 2021. p. 771-97. Available from: https://www.sciencedirect.com/science/article/pii/B9780128215753000347.

[6] Chandan G, Cascella M. Gas Laws and Clinical Application; 2023. Online. Available from: https://www.ncbi.nlm.nih.gov/books/NBK546592/.

[7] Microsoft. lightgbm.LGBMRegressor; 2024. Version 4.5.0. Online. Available from: https://lightgbm.readthedocs.io/en/latest/pythonapi/lightgbm.LGBMRegressor.html.

[8] Henry W, Banks J. III. Experiments on the quantity of gases absorbed by water, at different temperatures, and under different pressures. Philosophical Transactions of the Royal Society of London. 1803;93:29-274. Available from: https://royalsocietypublishing.org/doi/abs/10.1098/rstl.1803.0004.

[9] Sander R, Acree WE, Visscher AD, Schwartz SE, Wallington TJ. Henry's law constants (IUPAC Recommendations 2021). Pure and Applied Chemistry. 2022;94(1):71-85. Available from: https://doi.org/10.1515/pac-2020-0302 [cited 2024-08-28].

[10] Atkins P, de Paula J, Keeler J. Atkins' Physical Chemistry. 12th ed. Oxford University Press; 2023.

[11] Harvey A, Smith F. Avoid Common Pitfalls when using Henry's Law. Chemical Engineering Progress; 2007. Available from: https://tsapps.nist.gov/publication/get_pdf.cfm?pub_id=50449.

[12] solubility; 2019. Available from: https://doi.org/10.1351/goldbook.S05740.

[13] Williams R. This Month In Physics History: François-Marie Raoult and Raoult's Law: May 23, 1887. American Physical Society; 2011. Online. Available from: https://www.aps.org/archives/publications/apsnews/201105/physicshistory.cfm.

[14] Leicester HM, Klickstein HS. A Source Book in Chemistry, 1400-1900. No. v. 1 in A Source Book in Chemistry, 1400-1900. McGraw-Hill; 1952. Available from: https://books.google.co.uk/books?id=zXZKfpRHLCcC.

[15] Popkin BM, D'Anci KE, Rosenberg IH. Water, hydration, and health. Nutrition Reviews. 2010 08;68(8):439-58. Available from: https://doi.org/10.1111/j.1753-4887.2010.00304.x.

[16] Robison-Smith C, Masud N, Tarring EC, Ward BD, Cable J. A class of their own? Water-soluble polymer pollution impacting a freshwater host-pathogen system. Science of The Total Environment. 2024;907:168086. Available from: https://www.sciencedirect.com/science/article/pii/S004896972306713X.

[17] Sorkun MC, Khetan A, Er S. Aqueous Solubility Database. Scientific Data. 2019. Available from: https://doi.org/10.1038/s41597-019-0151-1.

[18] Lowe CN, Charest N, Ramsland C, Chang DT, Martin TM, Williams AJ. Transparency in Modeling through Careful Application of OECD's QSAR/QSPR Principles via a Curated Water Solubility Data Set. Chemical Research in Toxicology. 2023 Mar;36(3):465-78. Available from: https://doi.org/10.1021/acs.chemrestox.2c00379.

[19] Sushko I, Novotarskyi S, Körner R, Pandey AK, Rupp M, Teetz W, et al. Online chemical modeling environment (OCHEM): web platform for data storage, model development and publishing of chemical information. Journal of Computer-Aided Molecular Design. 2011 Jun;25(6):533-54. Available from: https://doi.org/10.1007/s10822-011-9440-2.

[20] Ullah A, Shaheryar M, Lim HJ. Machine Learning Approach for the Estimation of Henry's Law Constant Based on Molecular Descriptors. Atmosphere. 2024;15(6). Available from: https://www.mdpi.com/2073-4433/15/6/706.

[21] Zhang W, Wang Y, Ren S, Hou Y, Wu W. Novel Strategy of Machine Learning for Predicting Henry's Law Constants of CO2 in Ionic Liquids. ACS Sustainable Chemistry & Engineering. 2023 Apr;11(15):6090-9. Available from: https://doi.org/10.1021/acssuschemeng.3c00874.

[22] RDKit. rdkit.Chem.Descriptors module; 2024. Online. Available from: https://www.rdkit.org/docs/source/rdkit.Chem.Descriptors.html.

[23] Pedregosa F, Varoquaux G, Gramfort A, Michel V, Thirion B, Grisel O, et al. Scikit-learn: Machine learning in Python. Journal of machine learning research. 2011;12(Oct):2825-30.

[24] IUPAC. Salomon M, editor. Solubility Data Series; 2023. Contains data from 1979 to present. Accessed in 2023. Online.

[25] DDBST - Dortmund Data Bank Software & Separation Technology GmbH. DDB Version 2023; 2023. Accessed in 2023, 2024 version now requires login to access. Online.

[26] Cancado DNT, Basha TFMA, Cheung J. Machine Learning Models to Predict Miscibility; 2023. Unpublished.

[27] Swain M. CIRpy; 2015. Online. Available from: https://cirpy.readthedocs.io/en/latest/index.html.

[28] Nicklaus MC. Chemical Identifier Resolver; 2016. Online.

[29] Rickard A. Master Chemical Mechanism, MCM v3.3.1; 2023. Online. Available from: https://mcm.york.ac.uk/MCM.

[30] Swain M. PubChemPy; 2017. Online. Available from: https://github.com/mcs07/PubChemPy/tree/master.

[31] Boehmke B, Greenwell B. 3.2 Target Engineering. In: Hands-On Machine Learning with R. Taylor & Francis; 2020. Available from: https://bradleyboehmke.github.io/HOML/.

[32] Zheng A, Casari A. Feature Engineering for Machine Learning: Principles and Techniques for Data Scientists. Roumeliotis R, Bleiel J, editors. O'Reilly Media; 2018.

[33] Kuhn M, Johnson K. Feature Engineering and Selection: A Practical Approach for Predictive Models. Taylor & Francis; 2019. Available from: http://www.feat.engineering/.

[34] Scikit-learn. sklearn.preprocessing; 2024. Version 1.5.1. Online. Available from: https://scikit-learn.org/stable/api/sklearn.preprocessing.html.

[35] Scikit-learn. MinMaxScaler; 2024. Version 1.5.1. Online. Available from: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html#sklearn.preprocessing.MinMaxScaler.

[36] Misc. numpy.log1p; 2024. V2.1.0, accessed 2024-08-23. Online. Available from: https://numpy.org/doc/stable/reference/generated/numpy.log1p.html.

[37] Hvitfeldt E. 1.3 Scaling Issues. In: Feature Engineering A-Z; 2024. Commit 66ef79, accessed 2024-08-23. Available from: https://feaz-book.com/numeric-maxabs.

[38] Hvitfeldt E. 5.1 Yeo-Johnson. In: Feature Engineering A-Z; 2024. Commit 66ef79, accessed 2024-08-23. Available from: https://feaz-book.com/numeric-yeojohnson.

[39] Misc. scipy.stats.yeojohnson; 2024. V1.14.1 (stable), accessed 2024-08-24. Online. Available from: https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.yeojohnson.html.

[40] Hvitfeldt E. 10.1 Robust Scaling. In: Feature Engineering A-Z; 2024. Commit 66ef79, accessed 2024-08-23. Available from: https://feaz-book.com/numeric-robust.

[41] Scikit-learn. StandardScaler; 2024. Version 1.5.1. Online. Available from: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html.

[42] Chen L. In: LIU L, ÖZSU MT, editors. Curse of Dimensionality. Boston, MA: Springer US; 2009. p. 545-6. Available from: https://doi.org/10.1007/978-0-387-39940-9_133.

[43] Altman N, Krzywinski M. The curse(s) of dimensionality. Nature Methods. 2018 Jun;15(6):399-400. Available from: https://doi.org/10.1038/s41592-018-0019-x.

[44] Scikit-learn. SelectKBest; 2024. Version 1.5.1. Online. Available from: https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.SelectKBest.html.

[45] Mahbobi M, Tiemann TK. Chapter 6. F-Test and One-Way ANOVA. In: Introductory Business Statistics with Interactive Spreadsheets - 1st Canadian Edition; 2022. V1.03. Available from: https://opentextbc.ca/introductorybusinessstatistics/chapter/f-test-and-one-way-anova-2/.

[46] Scikit-learn. KernelPCA; 2024. Version 1.5.1. Online. Available from: https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.KernelPCA.html.

[47] Scikit-learn. PCA; 2024. Version 1.5.1. Online. Available from: https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html.

[48] IBM. What is principal component analysis (PCA)?; 2023. Online. Available from: https://www.ibm.com/topics/principal-component-analysis.

[49] Sewell M. Principal Component Analysis; 2008. Online. Available from: http://www.stats.org.uk/pca/.

[50] Schölkopf B, Smola A, Müller KR. Kernel principal component analysis. In: Gerstner W, Germond A, Hasler M, Nicoud JD, editors. Artificial Neural Networks — ICANN'97. Berlin, Heidelberg: Springer Berlin Heidelberg; 1997. p. 583-8.

[51] Briscik M, Dillies MA, Déjean S. Improvement of variables interpretability in kernel PCA. BMC Bioinformatics. 2023 Jul;24(1):282. Available from: https://doi.org/10.1186/s12859-023-05404-y.

[52] Wolpert DH, Macready WG. No free lunch theorems for optimization. IEEE Transactions on Evolutionary Computation. 1997;1(1):67-82.

[53] Paszke A, Gross S, Massa F, Lerer A, Bradbury J, Chanan G, et al. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In: Advances in Neural Information Processing Systems 32. Curran Associates, Inc.; 2019. p. 8024-35. Available from: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[54] Scikit-learn. GridSearchCV; 2024. Version 1.5.1. Online. Available from: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html.

[55] Dietterich TG. Ensemble Methods in Machine Learning. In: Multiple Classifier Systems. Berlin, Heidelberg: Springer Berlin Heidelberg; 2000. p. 1-15.

[56] IBM JM, Kavlakoglu E. What is ensemble learning?; 2024. Online. Available from: https://www.ibm.com/topics/ensemble-learning.

[57] IBM. What is boosting?; n.d. Online. Available from: https://www.ibm.com/topics/boosting.

[58] Drucker H. Improving Regressors using Boosting Techniques. In: Proceedings of the Fourteenth International Conference on Machine Learning. ICML '97. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.; 1997. p. 107-15.

[59] Ke G, Meng Q, Finley T, Wang T, Chen W, Ma W, et al. Lightgbm: A highly efficient gradient boosting decision tree. Advances in neural information processing systems. 2017;30:3146-54.

[60] Breiman L. Random Forests. Machine Learning. 2001 Oct;45(1):5-32. Available from: https://doi.org/10.1023/A:1010933404324.

[61] Chang CC, Lin CJ. LIBSVM: A library for support vector machines. ACM transactions on intelligent systems and technology (TIST). 2011;2(3):1-27.

[62] Scikit-learn. SVR; 2024. Version 1.5.1. Online. Available from: https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVR.html.

[63] Awad M, Khanna R. In: Support Vector Regression. Berkeley, CA: Apress; 2015. p. 67-80. Available from: https://doi.org/10.1007/978-1-4302-5990-9_4.

[64] MathWorks. Understanding Support Vector Machine Regression; 2024. Online. Available from: https://uk.mathworks.com/help/stats/understanding-support-vector-machine-regression.html.

[65] Scikit-learn. 1.3 Kernel Ridge Regression; 2024. Version 1.5.1. Online. Available from: https://scikit-learn.org/stable/modules/kernel_ridge.html.

[66] Welling M. Kernel ridge Regression; n.d. Online. Available from: https://web2.qatar.cmu.edu/~gdicaro/10315-Fall19/additional/welling-notes-on-kernel-ridge.pdf.

[67] Haugh M. Kernel Ridge Regression for OR & FE: Support Vector Machines (and the Kernel Trick); n.d. Online. Available from: http://www.columbia.edu/~mh2078/MachineLearningORFE/SVMs_MasterSlides.pdf.

[68] IBM. What are SVMs?; 2023. Online. Available from: https://www.ibm.com/topics/support-vector-machine.

[69] IBM. What is the k-nearest neighbors (KNN) algorithm?; n.d. Online. Available from: https://www.ibm.com/topics/knn.

[70] Scikit-learn. LinearRegression; 2024. Version 1.5.1. Online. Available from: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html.

[71] Scikit-learn. GroupShuffleSplit; 2024. Version 1.5.1. Online. Available from: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GroupShuffleSplit.html.

Num Components	$\mathbf{R^2}$	MSE
1	-0.74	78.76
61	0.15	38.40
91	0.18	37.06
106	0.19	36.83
114	0.20	36.39
122	0.20	36.38

Table 10: The effect of PCA on model performance, dataset version 0.5.3.

[72] Newcastle University. Coefficient of Determination, R-squared; 2022. Available from: https://www.ncl.ac.uk/webtemplate/ask-assets/external/maths-resources/statistics/regression-and-correlation/coefficient-of-determination-r-squared.html.

[73] CAS Common Chemistry. 1,2-Dichloropropane; n.d. (retrieved 2024-08-26) (CAS RN: 78-87-5). Licensed under the Attribution-Noncommercial 4.0 International License (CC BY-NC 4.0). Online.

[74] Van Rossum G, Drake FL. Python 3 Reference Manual. Scotts Valley, CA: CreateSpace; 2009.

[75] Hunter JD. Matplotlib: A 2D graphics environment. Computing in science & engineering. 2007;9(3):90-5.

[76] Harris CR, Millman KJ, van der Walt SJ, Gommers R, Virtanen P, Cournapeau D, et al. Array programming with NumPy. Nature. 2020;585:357-62.

9 Appendix

9.1 Feature Selection with SelectKBest

Num Features (k)	$\mathbf{R^2}$	MSE
1	0.00	54.5
98	0.86	7.48
146	0.87	6.93
170	0.87	6.93
182	0.88	6.39
188	**0.88**	**6.39**
195	0.88	6.41

Table 9: A table comparing the results for SelectKBest on the $\log k_H$ (v0.4.2) dataset. The highest scoring result is highlighted in bold.

9.2 Principal Component Analysis (PCA)

9.3 Scaling Results

Normalisation	**Scaling**	**MSE**	$\mathbf{R^2}$
None	None	0.77	0.86
MinMax	**RobustScaler()**	**0.88**	**0.84**
MinMax	normalize	0.90	0.83
Normalize	normalize	0.90	0.83
Normalize	PowerTransformer(standardize=False)	0.95	0.82
MinMax	StandardScaler()	0.96	0.82
Normalize	MaxAbsScaler()	1.06	0.80
Normalize	RobustScaler()	1.44	0.73
Normalize	StandardScaler()	2.36	0.56
MinMax	MaxAbsScaler()	4.86	0.09
MinMax	MinMax	4.97	0.07
Normalize	MinMax	4.97	0.07
MinMax	PowerTransformer(standardize=False)	9.88	-0.85

Table 11: A comparison of the scaling methods trialled on the $\log S$ dataset (v0.4.2) sorted by R^2, testing using the base LightGBM regressor. The scaling method chosen for the rest of the modelling is highlighted in bold.

Normalisation	**Scaling**	**MSE**	$\mathbf{R^2}$
None	None	5.64	0.90
MinMax	**RobustScaler()**	**6.41**	**0.88**
MinMax	StandardScaler()	6.83	0.87
Normalize	normalize	7.37	0.86
MinMax	normalize	7.37	0.86
Normalize	MaxAbsScaler()	7.64	0.86
Normalize	PowerTransformer(standardize=False)	8.30	0.85
Normalize	RobustScaler()	16.27	0.70
Normalize	MinMax	40.63	0.25
MinMax	MaxAbsScaler()	40.63	0.25
MinMax	MinMax	40.63	0.25
MinMax	PowerTransformer(standardize=False)	60.59	-0.11
Normalize	StandardScaler()	67.25	-0.24

Table 12: A comparison of the scaling methods trialled on the k_H dataset (v0.3.2) sorted by R^2, testing using the base LightGBM regressor. The scaling method chosen for the rest of the modelling is highlighted in bold.

9.4 Model Hyper-parameters

Parameter	$\log S$	$\log \mathbf{k_H}$
n_estimators	200	100
learning_rate	0.1	0.2
loss	exponential	exponential

Table 13: The hyper-parameters used for AdaBoost Regressor

Parameter	$\log S$	$\mathbf{k_H}$
boosting_type	gbdt	gbdt
n_estimators	600	600
num_leaves	31	31
max_bin	255	244
num_threads	4	4
learning_rate	0.1	0.05

Table 14: The hyper-parameters used for LightGBM Regressor

Parameter	$\log S$	$\mathbf{k_H}$
boosting_type	rf	rf
n_estimators	400	600
num_leaves	50	31
max_bin	255	255
num_threads	4	4
learning_rate	0.1	0.05
bagging_freq	4	1
bagging_fraction	0.8	0.8

Table 15: The hyper-parameters used for Random Forest

Parameter	$\log S$	$\mathbf{k_H}$
kernel	laplacian	laplacian
alpha	0.05	0.04

Table 16: The hyper-parameters used for Kernel Ridge Regression

Parameter	$\log S$	$\mathbf{k_H}$
n_neighbors	9	5
weights	distance	distance
p	1	1

Parameter	$\log S$	$\mathbf{k_H}$
kernel	linear	linear
C	1.0	1.0
epsilon	0.1	0.4

Table 17: The hyper-parameters used for model X